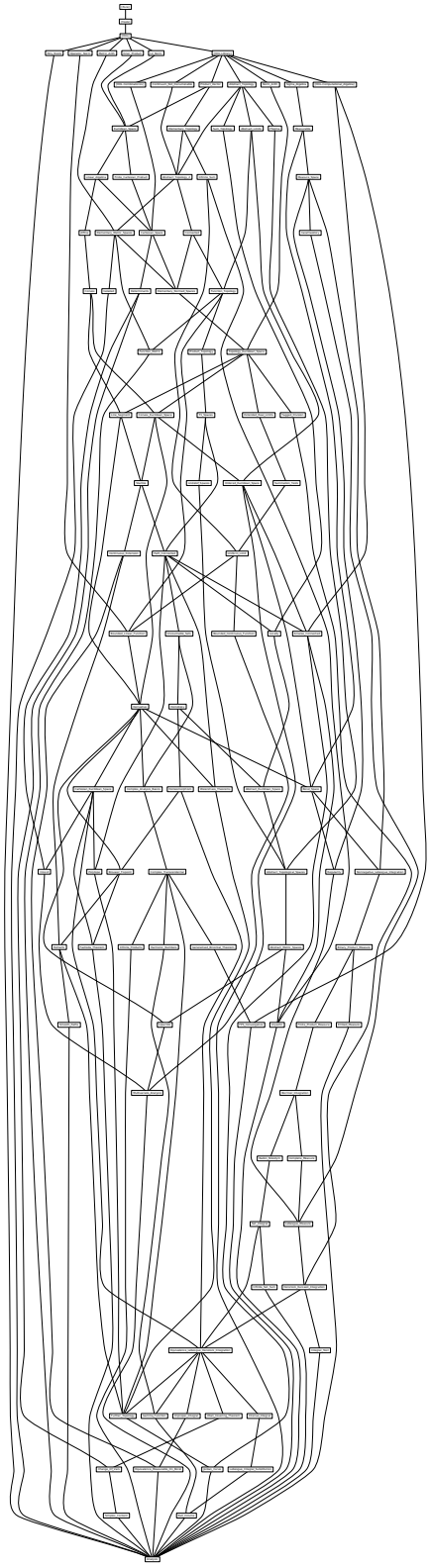
Analysis

September 11, 2023

Contents



Chapter 1

Linear Algebra

theory *L2_Norm*
imports *Complex_Main*
begin

1.1 L2 Norm

definition *L2_set* :: $('a \Rightarrow real) \Rightarrow 'a\ set \Rightarrow real$ **where**
$L2\_set\ f\ A = sqrt\ (\sum i{\in}A.\ (f\ i)^2)$

lemma *L2_set_cong*:
$[\![A = B;\ \bigwedge x.\ x \in B \Longrightarrow f\ x = g\ x]\!] \Longrightarrow L2\_set\ f\ A = L2\_set\ g\ B$
unfolding *L2_set_def* **by** *simp*

lemma *L2_set_cong_simp*:
$[\![A = B;\ \bigwedge x.\ x \in B =simp=> f\ x = g\ x]\!] \Longrightarrow L2\_set\ f\ A = L2\_set\ g\ B$
unfolding *L2_set_def simp_implies_def* **by** *simp*

lemma *L2_set_infinite* [*simp*]: $\neg\ finite\ A \Longrightarrow L2\_set\ f\ A = 0$
unfolding *L2_set_def* **by** *simp*

lemma *L2_set_empty* [*simp*]: $L2\_set\ f\ \{\} = 0$
unfolding *L2_set_def* **by** *simp*

lemma *L2_set_insert* [*simp*]:
$[\![finite\ F;\ a \notin F]\!] \Longrightarrow$
$L2\_set\ f\ (insert\ a\ F) = sqrt\ ((f\ a)^2 + (L2\_set\ f\ F)^2)$
unfolding *L2_set_def* **by** (*simp add*: *sum_nonneg*)

lemma *L2_set_nonneg* [*simp*]: $0 \leq L2\_set\ f\ A$
unfolding *L2_set_def* **by** (*simp add*: *sum_nonneg*)

lemma *L2_set_0'*: $\forall\ a{\in}A.\ f\ a = 0 \Longrightarrow L2\_set\ f\ A = 0$
unfolding *L2_set_def* **by** *simp*

lemma *L2_set_constant*: *L2_set* (λx. *y*) *A* = *sqrt* (*of_nat* (*card A*)) * |*y*|
unfolding *L2_set_def* **by** (*simp add*: *real_sqrt_mult*)

lemma *L2_set_mono*:
assumes ⋀*i*. *i* ∈ *K* ⟹ *f i* ≤ *g i*
assumes ⋀*i*. *i* ∈ *K* ⟹ *0* ≤ *f i*
shows *L2_set f K* ≤ *L2_set g K*
unfolding *L2_set_def*
by (*simp add*: *sum_nonneg sum_mono power_mono assms*)

lemma *L2_set_strict_mono*:
assumes *finite K* **and** *K* ≠ {}
assumes ⋀*i*. *i* ∈ *K* ⟹ *f i* < *g i*
assumes ⋀*i*. *i* ∈ *K* ⟹ *0* ≤ *f i*
shows *L2_set f K* < *L2_set g K*
unfolding *L2_set_def*
by (*simp add*: *sum_strict_mono power_strict_mono assms*)

lemma *L2_set_right_distrib*:
0 ≤ *r* ⟹ *r* * *L2_set f A* = *L2_set* (λx. *r* * *f x*) *A*
unfolding *L2_set_def*
apply (*simp add*: *power_mult_distrib*)
apply (*simp add*: *sum_distrib_left* [*symmetric*])
apply (*simp add*: *real_sqrt_mult sum_nonneg*)
done

lemma *L2_set_left_distrib*:
0 ≤ *r* ⟹ *L2_set f A* * *r* = *L2_set* (λx. *f x* * *r*) *A*
unfolding *L2_set_def*
apply (*simp add*: *power_mult_distrib*)
apply (*simp add*: *sum_distrib_right* [*symmetric*])
apply (*simp add*: *real_sqrt_mult sum_nonneg*)
done

lemma *L2_set_eq_0_iff*: *finite A* ⟹ *L2_set f A* = *0* ⟷ (∀ x∈*A*. *f x* = *0*)
unfolding *L2_set_def*
by (*simp add*: *sum_nonneg sum_nonneg_eq_0_iff*)

proposition *L2_set_triangle_ineq*:
L2_set (λi. *f i* + *g i*) *A* ≤ *L2_set f A* + *L2_set g A*
proof (*cases finite A*)
case *False*
thus *?thesis* **by** *simp*
next
case *True*
thus *?thesis*
proof (*induct set*: *finite*)
case *empty*
show *?case* **by** *simp*

```
  next
    case (insert x F)
    hence sqrt ((f x + g x)^2 + (L2_set (λi. f i + g i) F)^2) ≤
          sqrt ((f x + g x)^2 + (L2_set f F + L2_set g F)^2)
      by (intro real_sqrt_le_mono add_left_mono power_mono insert
                L2_set_nonneg add_increasing zero_le_power2)
    also have
      ... ≤ sqrt ((f x)^2 + (L2_set f F)^2) + sqrt ((g x)^2 + (L2_set g F)^2)
      by (rule real_sqrt_sum_squares_triangle_ineq)
    finally show ?case
      using insert by simp
  qed
qed

lemma L2_set_le_sum [rule_format]:
  (∀ i∈A. 0 ≤ f i) ⟶ L2_set f A ≤ sum f A
  apply (cases finite A)
  apply (induct set: finite)
  apply simp
  apply clarsimp
  apply (erule order_trans [OF sqrt_sum_squares_le_sum])
  apply simp
  apply simp
  apply simp
  done

lemma L2_set_le_sum_abs: L2_set f A ≤ (∑ i∈A. |f i|)
  apply (cases finite A)
  apply (induct set: finite)
  apply simp
  apply simp
  apply (rule order_trans [OF sqrt_sum_squares_le_sum_abs])
  apply simp
  apply simp
  done

lemma L2_set_mult_ineq: (∑ i∈A. |f i| * |g i|) ≤ L2_set f A * L2_set g A
  apply (cases finite A)
  apply (induct set: finite)
  apply simp
  apply (rule power2_le_imp_le, simp)
  apply (rule order_trans)
  apply (rule power_mono)
  apply (erule add_left_mono)
  apply (simp add: sum_nonneg)
  apply (simp add: power2_sum)
  apply (simp add: power_mult_distrib)
  apply (simp add: distrib_left distrib_right)
  apply (rule ord_le_eq_trans)
```

apply (*rule L2_set_mult_ineq_lemma*)
apply *simp_all*
done

lemma *member_le_L2_set*: ⟦*finite A*; *i* ∈ *A*⟧ ⟹ *f i* ≤ *L2_set f A*
unfolding *L2_set_def*
by (*auto intro*!: *member_le_sum real_le_rsqrt*)

end

1.2 Inner Product Spaces and Gradient Derivative

theory *Inner_Product*
imports *Complex_Main*
begin

1.2.1 Real inner product spaces

Temporarily relax type constraints for *open*, *uniformity*, *dist*, and *norm*.

setup ‹*Sign.add_const_constraint*
(***const_name*** ‹*open*›, *SOME* ***typ*** ‹'*a*::*open set* ⇒ *bool*›)›

setup ‹*Sign.add_const_constraint*
(***const_name*** ‹*dist*›, *SOME* ***typ*** ‹'*a*::*dist* ⇒ '*a* ⇒ *real*›)›

setup ‹*Sign.add_const_constraint*
(***const_name*** ‹*uniformity*›, *SOME* ***typ*** ‹('*a*::*uniformity* × '*a*) *filter*›)›

setup ‹*Sign.add_const_constraint*
(***const_name*** ‹*norm*›, *SOME* ***typ*** ‹'*a*::*norm* ⇒ *real*›)›

class *real_inner* = *real_vector* + *sgn_div_norm* + *dist_norm* + *uniformity_dist*
+ *open_uniformity* +
fixes *inner* :: '*a* ⇒ '*a* ⇒ *real*
assumes *inner_commute*: *inner x y* = *inner y x*
and *inner_add_left*: *inner* (*x* + *y*) *z* = *inner x z* + *inner y z*
and *inner_scaleR_left* [*simp*]: *inner* (*scaleR r x*) *y* = *r* ∗ (*inner x y*)
and *inner_ge_zero* [*simp*]: *0* ≤ *inner x x*
and *inner_eq_zero_iff* [*simp*]: *inner x x* = *0* ⟷ *x* = *0*
and *norm_eq_sqrt_inner*: *norm x* = *sqrt* (*inner x x*)
begin

lemma *inner_zero_left* [*simp*]: *inner 0 x* = *0*
using *inner_add_left* [*of 0 0 x*] **by** *simp*

lemma *inner_minus_left* [*simp*]: *inner* (− *x*) *y* = − *inner x y*
using *inner_add_left* [*of x* − *x y*] **by** *simp*

lemma *inner_diff_left*: *inner* $(x - y)$ z = *inner* x z − *inner* y z
using *inner_add_left* [*of* $x - y$ z] **by** *simp*

lemma *inner_sum_left*: *inner* $(\sum x{\in}A.\ f\ x)$ y = $(\sum x{\in}A.$ *inner* $(f\ x)$ $y)$
by (*cases finite A*, *induct set*: *finite*, *simp_all add*: *inner_add_left*)

lemma *all_zero_iff* [*simp*]: $(\forall u.$ *inner* x $u = 0) \longleftrightarrow (x = 0)$
by *auto* (*use inner_eq_zero_iff* **in** *blast*)

Transfer distributivity rules to right argument.

lemma *inner_add_right*: *inner* x $(y + z)$ = *inner* x y + *inner* x z
using *inner_add_left* [*of* y z x] **by** (*simp only*: *inner_commute*)

lemma *inner_scaleR_right* [*simp*]: *inner* x $(scaleR\ r\ y) = r * (inner\ x\ y)$
using *inner_scaleR_left* [*of* r y x] **by** (*simp only*: *inner_commute*)

lemma *inner_zero_right* [*simp*]: *inner* x $0 = 0$
using *inner_zero_left* [*of* x] **by** (*simp only*: *inner_commute*)

lemma *inner_minus_right* [*simp*]: *inner* x $(-\ y) = -$ *inner* x y
using *inner_minus_left* [*of* y x] **by** (*simp only*: *inner_commute*)

lemma *inner_diff_right*: *inner* x $(y - z)$ = *inner* x y − *inner* x z
using *inner_diff_left* [*of* y z x] **by** (*simp only*: *inner_commute*)

lemma *inner_sum_right*: *inner* x $(\sum y{\in}A.\ f\ y) = (\sum y{\in}A.$ *inner* x $(f\ y))$
using *inner_sum_left* [*of f A x*] **by** (*simp only*: *inner_commute*)

lemmas *inner_add* [*algebra_simps*] = *inner_add_left inner_add_right*
lemmas *inner_diff* [*algebra_simps*] = *inner_diff_left inner_diff_right*
lemmas *inner_scaleR* = *inner_scaleR_left inner_scaleR_right*

Legacy theorem names

lemmas *inner_left_distrib* = *inner_add_left*
lemmas *inner_right_distrib* = *inner_add_right*
lemmas *inner_distrib* = *inner_left_distrib inner_right_distrib*

lemma *inner_gt_zero_iff* [*simp*]: $0 <$ *inner* x $x \longleftrightarrow x \neq 0$
by (*simp add*: *order_less_le*)

lemma *power2_norm_eq_inner*: $(norm\ x)^2$ = *inner* x x
by (*simp add*: *norm_eq_sqrt_inner*)

Identities involving real multiplication and division.

lemma *inner_mult_left*: *inner* $(of\_real\ m * a)$ $b = m * (inner\ a\ b)$
by (*metis real_inner_class.inner_scaleR_left scaleR_conv_of_real*)

lemma *inner_mult_right*: *inner* a $(of\_real\ m * b) = m * (inner\ a\ b)$
by (*metis real_inner_class.inner_scaleR_right scaleR_conv_of_real*)

lemma *inner_mult_left'*: *inner* (*a* ∗ *of_real m*) *b* = *m* ∗ (*inner a b*)
by (*simp add*: *of_real_def*)

lemma *inner_mult_right'*: *inner a* (*b* ∗ *of_real m*) = (*inner a b*) ∗ *m*
by (*simp add*: *of_real_def real_inner_class.inner_scaleR_right*)

lemma *Cauchy_Schwarz_ineq*:
(*inner x y*)^2 ≤ *inner x x* ∗ *inner y y*
proof (*cases*)
assume *y* = *0*
thus *?thesis* **by** *simp*
next
assume *y*: *y* ≠ *0*
let *?r* = *inner x y* / *inner y y*
have *0* ≤ *inner* (*x* − *scaleR ?r y*) (*x* − *scaleR ?r y*)
by (*rule inner_ge_zero*)
also have ... = *inner x x* − *inner y x* ∗ *?r*
by (*simp add*: *inner_diff*)
also have ... = *inner x x* − (*inner x y*)^2 / *inner y y*
by (*simp add*: *power2_eq_square inner_commute*)
finally have *0* ≤ *inner x x* − (*inner x y*)^2 / *inner y y* **.**
hence (*inner x y*)^2 / *inner y y* ≤ *inner x x*
by (*simp add*: *le_diff_eq*)
thus (*inner x y*)^2 ≤ *inner x x* ∗ *inner y y*
by (*simp add*: *pos_divide_le_eq y*)
qed

lemma *Cauchy_Schwarz_ineq2*:
|*inner x y*| ≤ *norm x* ∗ *norm y*
proof (*rule power2_le_imp_le*)
have (*inner x y*)^2 ≤ *inner x x* ∗ *inner y y*
using *Cauchy_Schwarz_ineq* **.**
thus |*inner x y*|^2 ≤ (*norm x* ∗ *norm y*)^2
by (*simp add*: *power_mult_distrib power2_norm_eq_inner*)
show *0* ≤ *norm x* ∗ *norm y*
unfolding *norm_eq_sqrt_inner*
by (*intro mult_nonneg_nonneg real_sqrt_ge_zero inner_ge_zero*)
qed

lemma *norm_cauchy_schwarz*: *inner x y* ≤ *norm x* ∗ *norm y*
using *Cauchy_Schwarz_ineq2* [*of x y*] **by** *auto*

subclass *real_normed_vector*
proof
fix *a* :: *real* **and** *x y* :: *'a*
show *norm x* = *0* ⟷ *x* = *0*
unfolding *norm_eq_sqrt_inner* **by** *simp*
show *norm* (*x* + *y*) ≤ *norm x* + *norm y*

```
    proof (rule power2_le_imp_le)
      have inner x y ≤ norm x * norm y
        by (rule norm_cauchy_schwarz)
      thus (norm (x + y))² ≤ (norm x + norm y)²
        unfolding power2_sum power2_norm_eq_inner
        by (simp add: inner_add inner_commute)
      show 0 ≤ norm x + norm y
        unfolding norm_eq_sqrt_inner by simp
    qed
  have sqrt (a² * inner x x) = |a| * sqrt (inner x x)
    by (simp add: real_sqrt_mult)
  then show norm (a *R x) = |a| * norm x
    unfolding norm_eq_sqrt_inner
    by (simp add: power2_eq_square mult.assoc)
qed

end

lemma square_bound_lemma:
  fixes x :: real
  shows x < (1 + x) * (1 + x)
proof −
  have (x + 1/2)² + 3/4 > 0
    using zero_le_power2[of x+1/2] by arith
  then show ?thesis
    by (simp add: field_simps power2_eq_square)
qed

lemma square_continuous:
  fixes e :: real
  shows e > 0 ⟹ ∃ d. 0 < d ∧ (∀ y. |y − x| < d ⟶ |y * y − x * x| < e)
  using isCont_power[OF continuous_ident, of x, unfolded isCont_def LIM_eq,
rule_format, of e 2]
  by (force simp add: power2_eq_square)

lemma norm_le: norm x ≤ norm y ⟷ inner x x ≤ inner y y
  by (simp add: norm_eq_sqrt_inner)

lemma norm_lt: norm x < norm y ⟷ inner x x < inner y y
  by (simp add: norm_eq_sqrt_inner)

lemma norm_eq: norm x = norm y ⟷ inner x x = inner y y
  apply (subst order_eq_iff)
  apply (auto simp: norm_le)
  done

lemma norm_eq_1: norm x = 1 ⟷ inner x x = 1
  by (simp add: norm_eq_sqrt_inner)
```

lemma *inner_divide_left*:
fixes *a* :: *'a* :: {*real_inner*,*real_div_algebra*}
shows *inner* (*a* / *of_real m*) *b* = (*inner a b*) / *m*
by (*metis* (*no_types*) *divide_inverse inner_commute inner_scaleR_right mult.left_neutral mult.right_neutral mult_scaleR_right of_real_inverse scaleR_conv_of_real times_divide_eq_left*)

lemma *inner_divide_right*:
fixes *a* :: *'a* :: {*real_inner*,*real_div_algebra*}
shows *inner a* (*b* / *of_real m*) = (*inner a b*) / *m*
by (*metis inner_commute inner_divide_left*)

Re-enable constraints for *open*, *uniformity*, *dist*, and *norm*.

setup ‹*Sign.add_const_constraint*
(***const_name*** ‹*open*›, *SOME* ***typ*** ‹*'a*::*topological_space set* ⇒ *bool*›)›

setup ‹*Sign.add_const_constraint*
(***const_name*** ‹*uniformity*›, *SOME* ***typ*** ‹(*'a*::*uniform_space* × *'a*) *filter*›)›

setup ‹*Sign.add_const_constraint*
(***const_name*** ‹*dist*›, *SOME* ***typ*** ‹*'a*::*metric_space* ⇒ *'a* ⇒ *real*›)›

setup ‹*Sign.add_const_constraint*
(***const_name*** ‹*norm*›, *SOME* ***typ*** ‹*'a*::*real_normed_vector* ⇒ *real*›)›

lemma *bounded_bilinear_inner*:
bounded_bilinear (*inner*::*'a*::*real_inner* ⇒ *'a* ⇒ *real*)
proof
fix *x y z* :: *'a* **and** *r* :: *real*
show *inner* (*x* + *y*) *z* = *inner x z* + *inner y z*
by (*rule inner_add_left*)
show *inner x* (*y* + *z*) = *inner x y* + *inner x z*
by (*rule inner_add_right*)
show *inner* (*scaleR r x*) *y* = *scaleR r* (*inner x y*)
unfolding *real_scaleR_def* **by** (*rule inner_scaleR_left*)
show *inner x* (*scaleR r y*) = *scaleR r* (*inner x y*)
unfolding *real_scaleR_def* **by** (*rule inner_scaleR_right*)
show ∃ *K*. ∀ *x y*::*'a*. *norm* (*inner x y*) ≤ *norm x* ∗ *norm y* ∗ *K*
proof
show ∀ *x y*::*'a*. *norm* (*inner x y*) ≤ *norm x* ∗ *norm y* ∗ *1*
by (*simp add*: *Cauchy_Schwarz_ineq2*)
qed
qed

lemmas *tendsto_inner* [*tendsto_intros*] =
bounded_bilinear.tendsto [*OF bounded_bilinear_inner*]

lemmas *isCont_inner* [*simp*] =
bounded_bilinear.isCont [*OF bounded_bilinear_inner*]

lemmas *has_derivative_inner* [*derivative_intros*] =
bounded_bilinear.FDERIV [*OF bounded_bilinear_inner*]

lemmas *bounded_linear_inner_left* =
bounded_bilinear.bounded_linear_left [*OF bounded_bilinear_inner*]

lemmas *bounded_linear_inner_right* =
bounded_bilinear.bounded_linear_right [*OF bounded_bilinear_inner*]

lemmas *bounded_linear_inner_left_comp* = *bounded_linear_inner_left*[*THEN bounded_linear_compose*]

lemmas *bounded_linear_inner_right_comp* = *bounded_linear_inner_right*[*THEN bounded_linear_compose*]

lemmas *has_derivative_inner_right* [*derivative_intros*] =
bounded_linear.has_derivative [*OF bounded_linear_inner_right*]

lemmas *has_derivative_inner_left* [*derivative_intros*] =
bounded_linear.has_derivative [*OF bounded_linear_inner_left*]

lemma *differentiable_inner* [*simp*]:
f differentiable (*at x within s*) $\Longrightarrow$ *g differentiable at x within s* $\Longrightarrow$ (λx. *inner* (*f x*) (*g x*)) *differentiable at x within s*
unfolding *differentiable_def* **by** (*blast intro*: *has_derivative_inner*)

1.2.2 Class instances

instantiation *real* :: *real_inner*
begin

definition *inner_real_def* [*simp*]: *inner* = (*)

instance
proof
fix *x y z r* :: *real*
show *inner x y* = *inner y x*
unfolding *inner_real_def* **by** (*rule mult.commute*)
show *inner* (*x* + *y*) *z* = *inner x z* + *inner y z*
unfolding *inner_real_def* **by** (*rule distrib_right*)
show *inner* (*scaleR r x*) *y* = *r* * *inner x y*
unfolding *inner_real_def real_scaleR_def* **by** (*rule mult.assoc*)
show *0* $\leq$ *inner x x*
unfolding *inner_real_def* **by** *simp*
show *inner x x* = *0* $\longleftrightarrow$ *x* = *0*
unfolding *inner_real_def* **by** *simp*
show *norm x* = *sqrt* (*inner x x*)
unfolding *inner_real_def* **by** *simp*
qed

end

lemma
shows *real_inner_1_left*[*simp*]: *inner 1 x = x*
and *real_inner_1_right*[*simp*]: *inner x 1 = x*
by *simp_all*

instantiation *complex* :: *real_inner*
begin

definition *inner_complex_def*:
*inner x y = Re x * Re y + Im x * Im y*

instance
proof
fix *x y z* :: *complex* **and** *r* :: *real*
show *inner x y = inner y x*
unfolding *inner_complex_def* **by** (*simp add*: *mult.commute*)
show *inner (x + y) z = inner x z + inner y z*
unfolding *inner_complex_def* **by** (*simp add*: *distrib_right*)
show *inner (scaleR r x) y = r * inner x y*
unfolding *inner_complex_def* **by** (*simp add*: *distrib_left*)
show *0 ≤ inner x x*
unfolding *inner_complex_def* **by** *simp*
show *inner x x = 0 ⟷ x = 0*
unfolding *inner_complex_def*
by (*simp add*: *add_nonneg_eq_0_iff complex_eq_iff*)
show *norm x = sqrt (inner x x)*
unfolding *inner_complex_def norm_complex_def*
by (*simp add*: *power2_eq_square*)
qed

end

lemma *complex_inner_1* [*simp*]: *inner 1 x = Re x*
unfolding *inner_complex_def* **by** *simp*

lemma *complex_inner_1_right* [*simp*]: *inner x 1 = Re x*
unfolding *inner_complex_def* **by** *simp*

lemma *complex_inner_i_left* [*simp*]: *inner* i *x = Im x*
unfolding *inner_complex_def* **by** *simp*

lemma *complex_inner_i_right* [*simp*]: *inner x* i *= Im x*
unfolding *inner_complex_def* **by** *simp*

lemma *dot_square_norm*: *inner x x = (norm x)*^{2}

by (*simp only*: *power2_norm_eq_inner*)

lemma *norm_eq_square*: *norm x* = *a* ⟷ *0* ≤ *a* ∧ *inner x x* = a^2
by (*auto simp add*: *norm_eq_sqrt_inner*)

lemma *norm_le_square*: *norm x* ≤ *a* ⟷ *0* ≤ *a* ∧ *inner x x* ≤ a^2
apply (*simp add*: *dot_square_norm abs_le_square_iff*[*symmetric*])
using *norm_ge_zero*[*of x*]
apply *arith*
done

lemma *norm_ge_square*: *norm x* ≥ *a* ⟷ *a* ≤ *0* ∨ *inner x x* ≥ a^2
apply (*simp add*: *dot_square_norm abs_le_square_iff*[*symmetric*])
using *norm_ge_zero*[*of x*]
apply *arith*
done

lemma *norm_lt_square*: *norm x* < *a* ⟷ *0* < *a* ∧ *inner x x* < a^2
by (*metis not_le norm_ge_square*)

lemma *norm_gt_square*: *norm x* > *a* ⟷ *a* < *0* ∨ *inner x x* > a^2
by (*metis norm_le_square not_less*)

Dot product in terms of the norm rather than conversely.

lemmas *inner_simps* = *inner_add_left inner_add_right inner_diff_right inner_diff_left inner_scaleR_left inner_scaleR_right*

lemma *dot_norm*: *inner x y* = ((*norm* (*x* + *y*))^2 − (*norm x*)^2 − (*norm y*)^2) / *2*
by (*simp only*: *power2_norm_eq_inner inner_simps inner_commute*) *auto*

lemma *dot_norm_neg*: *inner x y* = (((*norm x*)^2 + (*norm y*)^2) − (*norm* (*x* − *y*))^2) / *2*
by (*simp only*: *power2_norm_eq_inner inner_simps inner_commute*)
(*auto simp add*: *algebra_simps*)

lemma *of_real_inner_1* [*simp*]:
inner (*of_real x*) (*1* :: *'a* :: {*real_inner*, *real_normed_algebra_1*}) = *x*
by (*simp add*: *of_real_def dot_square_norm*)

lemma *summable_of_real_iff*:
summable (λ*x*. *of_real* (*f x*) :: *'a* :: {*real_normed_algebra_1*,*real_inner*}) ⟷ *summable f*
proof
assume ∗: *summable* (λ*x*. *of_real* (*f x*) :: *'a*)
interpret *bounded_linear* λ*x*::*'a*. *inner x 1*
by (*rule bounded_linear_inner_left*)
from *summable* [*OF* ∗] **show** *summable f* **by** *simp*
qed (*auto intro*: *summable_of_real*)

1.2.3 Gradient derivative

definition
gderiv ::
[*'a*::*real_inner* ⇒ *real*, *'a*, *'a*] ⇒ *bool*
((*GDERIV* (_)/ (_)/ :> (_)) [*1000*, *1000*, *60*] *60*)
where
GDERIV f x :> D ⟷ *FDERIV f x :>* (λ*h*. *inner h D*)

lemma *gderiv_deriv* [*simp*]: *GDERIV f x :> D* ⟷ *DERIV f x :> D*
by (*simp only*: *gderiv_def has_field_derivative_def inner_real_def mult_commute_abs*)

lemma *GDERIV_DERIV_compose*:
⟦*GDERIV f x :> df*; *DERIV g* (*f x*) :> *dg*⟧
⟹ *GDERIV* (λ*x*. *g* (*f x*)) *x :> scaleR dg df*
unfolding *gderiv_def has_field_derivative_def*
apply (*drule* (*1*) *has_derivative_compose*)
apply (*simp add*: *ac_simps*)
done

lemma *has_derivative_subst*: ⟦*FDERIV f x :> df*; *df* = *d*⟧ ⟹ *FDERIV f x :>*
d
by *simp*

lemma *GDERIV_subst*: ⟦*GDERIV f x :> df*; *df* = *d*⟧ ⟹ *GDERIV f x :> d*
by *simp*

lemma *GDERIV_const*: *GDERIV* (λ*x*. *k*) *x :> 0*
unfolding *gderiv_def inner_zero_right* **by** (*rule has_derivative_const*)

lemma *GDERIV_add*:
⟦*GDERIV f x :> df*; *GDERIV g x :> dg*⟧
⟹ *GDERIV* (λ*x*. *f x* + *g x*) *x :> df* + *dg*
unfolding *gderiv_def inner_add_right* **by** (*rule has_derivative_add*)

lemma *GDERIV_minus*:
GDERIV f x :> df ⟹ *GDERIV* (λ*x*. − *f x*) *x :>* − *df*
unfolding *gderiv_def inner_minus_right* **by** (*rule has_derivative_minus*)

lemma *GDERIV_diff*:
⟦*GDERIV f x :> df*; *GDERIV g x :> dg*⟧
⟹ *GDERIV* (λ*x*. *f x* − *g x*) *x :> df* − *dg*
unfolding *gderiv_def inner_diff_right* **by** (*rule has_derivative_diff*)

lemma *GDERIV_scaleR*:
⟦*DERIV f x :> df*; *GDERIV g x :> dg*⟧
⟹ *GDERIV* (λ*x*. *scaleR* (*f x*) (*g x*)) *x*
:> (*scaleR* (*f x*) *dg* + *scaleR df* (*g x*))
unfolding *gderiv_def has_field_derivative_def inner_add_right inner_scaleR_right*
apply (*rule has_derivative_subst*)

apply (*erule* (*1*) *has_derivative_scaleR*)
apply (*simp add*: *ac_simps*)
done

lemma *GDERIV_mult*:
⟦*GDERIV f x :> df*; *GDERIV g x :> dg*⟧
⟹ *GDERIV* (λ*x*. *f x* ∗ *g x*) *x :> scaleR* (*f x*) *dg* + *scaleR* (*g x*) *df*
unfolding *gderiv_def*
apply (*rule has_derivative_subst*)
apply (*erule* (*1*) *has_derivative_mult*)
apply (*simp add*: *inner_add ac_simps*)
done

lemma *GDERIV_inverse*:
⟦*GDERIV f x :> df*; *f x* ≠ *0*⟧
⟹ *GDERIV* (λ*x*. *inverse* (*f x*)) *x :> −* (*inverse* (*f x*))^2 ∗$_R$ *df*
by (*metis DERIV_inverse GDERIV_DERIV_compose numerals*(*2*))

lemma *GDERIV_norm*:
assumes *x* ≠ *0* **shows** *GDERIV* (λ*x*. *norm x*) *x :> sgn x*
unfolding *gderiv_def norm_eq_sqrt_inner*
by (*rule derivative_eq_intros* | *force simp add*: *inner_commute sgn_div_norm norm_eq_sqrt_inner assms*)+

lemmas *has_derivative_norm* = *GDERIV_norm* [*unfolded gderiv_def*]

bundle *inner_syntax* **begin**
notation *inner* (**infix** · *70*)
end

bundle *no_inner_syntax* **begin**
no_notation *inner* (**infix** · *70*)
end

end

1.3 Cartesian Products as Vector Spaces

theory *Product_Vector*
imports
Complex_Main
HOL−Library.Product_Plus
begin

lemma *Times_eq_image_sum*:
fixes *S* :: *'a* :: *comm_monoid_add set* **and** *T* :: *'b* :: *comm_monoid_add set*
shows *S* × *T* = {*u* + *v* |*u v*. *u* ∈ (λ*x*. (*x*, *0*)) ‘ *S* ∧ *v* ∈ *Pair 0* ‘ *T*}
by *force*

1.3.1 Product is a Module

locale *module_prod* = *module_pair* **begin**

definition *scale* :: *'a* ⇒ *'b* × *'c* ⇒ *'b* × *'c*
  where *scale a v* = (*s1 a* (*fst v*), *s2 a* (*snd v*))

lemma *scale_prod*: *scale x* (*a*, *b*) = (*s1 x a*, *s2 x b*)
  by (*auto simp*: *scale_def*)

sublocale *p*: *module scale*
proof qed (*simp_all add*: *scale_def*
  m1.scale_left_distrib m1.scale_right_distrib m2.scale_left_distrib m2.scale_right_distrib)

lemma *subspace_Times*: *m1.subspace A* ⟹ *m2.subspace B* ⟹ *p.subspace* (*A* ×
B)
  unfolding *m1.subspace_def m2.subspace_def p.subspace_def*
  by (*auto simp*: *zero_prod_def scale_def*)

lemma *module_hom_fst*: *module_hom scale s1 fst*
  by *unfold_locales* (*auto simp*: *scale_def*)

lemma *module_hom_snd*: *module_hom scale s2 snd*
  by *unfold_locales* (*auto simp*: *scale_def*)

end

locale *vector_space_prod* = *vector_space_pair* **begin**

sublocale *module_prod s1 s2*
  rewrites *module_hom* = *Vector_Spaces.linear*
  by *unfold_locales* (*fact module_hom_eq_linear*)

sublocale *p*: *vector_space scale* **by** *unfold_locales* (*auto simp*: *algebra_simps*)

lemmas *linear_fst* = *module_hom_fst*
  and *linear_snd* = *module_hom_snd*

end

1.3.2 Product is a Real Vector Space

instantiation *prod* :: (*real_vector*, *real_vector*) *real_vector*
begin

definition *scaleR_prod_def*:
  scaleR r A = (*scaleR r* (*fst A*), *scaleR r* (*snd A*))

lemma *fst_scaleR* [*simp*]: *fst* (*scaleR r A*) = *scaleR r* (*fst A*)
  unfolding *scaleR_prod_def* **by** *simp*

lemma *snd_scaleR* [*simp*]: *snd* (*scaleR r A*) = *scaleR r* (*snd A*)
unfolding *scaleR_prod_def* **by** *simp*

proposition *scaleR_Pair* [*simp*]: *scaleR r* (*a*, *b*) = (*scaleR r a*, *scaleR r b*)
unfolding *scaleR_prod_def* **by** *simp*

instance
proof
fix *a b* :: *real* **and** *x y* :: *'a* × *'b*
show *scaleR a* (*x* + *y*) = *scaleR a x* + *scaleR a y*
by (*simp add*: *prod_eq_iff scaleR_right_distrib*)
show *scaleR* (*a* + *b*) *x* = *scaleR a x* + *scaleR b x*
by (*simp add*: *prod_eq_iff scaleR_left_distrib*)
show *scaleR a* (*scaleR b x*) = *scaleR* (*a* ∗ *b*) *x*
by (*simp add*: *prod_eq_iff*)
show *scaleR 1 x* = *x*
by (*simp add*: *prod_eq_iff*)
qed

end

lemma *module_prod_scale_eq_scaleR*: *module_prod.scale* (∗*R*) (∗*R*) = *scaleR*
apply (*rule ext*) **apply** (*rule ext*)
apply (*subst module_prod.scale_def*)
subgoal by *unfold_locales*
by (*simp add*: *scaleR_prod_def*)

interpretation *real_vector?*: *vector_space_prod scaleR*::_⇒_⇒*'a*::*real_vector scaleR*::_⇒_⇒*'b*::*real_vector*
rewrites *scale* = ((∗*R*)::_⇒_⇒(*'a* × *'b*))
and *module.dependent* (∗*R*) = *dependent*
and *module.representation* (∗*R*) = *representation*
and *module.subspace* (∗*R*) = *subspace*
and *module.span* (∗*R*) = *span*
and *vector_space.extend_basis* (∗*R*) = *extend_basis*
and *vector_space.dim* (∗*R*) = *dim*
and *Vector_Spaces.linear* (∗*R*) (∗*R*) = *linear*
subgoal by *unfold_locales*
subgoal by (*fact module_prod_scale_eq_scaleR*)
unfolding *dependent_raw_def representation_raw_def subspace_raw_def span_raw_def*
extend_basis_raw_def dim_raw_def linear_def
by (*rule refl*)+

1.3.3 Product is a Metric Space

instantiation *prod* :: (*metric_space*, *metric_space*) *dist*
begin

definition *dist_prod_def*[*code del*]:

dist x y = sqrt ((dist (fst x) (fst y))^2 + (dist (snd x) (snd y))^2)

instance ..
end

instantiation *prod* :: (*uniformity*, *uniformity*) *uniformity* **begin**

definition [*code del*]: ‹(*uniformity* :: (('*a* × '*b*) × ('*a* × '*b*)) *filter*) =
filtermap (λ((*x1*,*x2*),(*y1*,*y2*)). ((*x1*,*y1*),(*x2*,*y2*))) (*uniformity* ×$_F$ *uniformity*)›

instance..
end

Uniform spaces

instantiation *prod* :: (*uniform_space*, *uniform_space*) *uniform_space*
begin
instance
proof *standard*
fix *U* :: ‹('*a* × '*b*) *set*›
show ‹*open U* ⟷ (∀ *x*∈*U*. ∀$_F$ (*x'*, *y*) *in uniformity*. *x'* = *x* ⟶ *y* ∈ *U*)›
proof (*intro iffI ballI*)
fix *x* **assume** ‹*open U*› **and** ‹*x* ∈ *U*›
then obtain *A B* **where** ‹*open A*› ‹*open B*› ‹*x* ∈ *A*×*B*› ‹*A*×*B* ⊆ *U*›
by (*metis open_prod_elim*)
define *UA* **where** ‹*UA* = (λ(*x'*::'*a*,*y*). *x'* = *fst x* ⟶ *y* ∈ *A*)›
from ‹*open A*› ‹*x* ∈ *A*×*B*›
have ‹*eventually UA uniformity*›
unfolding *open_uniformity UA_def* **by** *auto*
define *UB* **where** ‹*UB* = (λ(*x'*::'*b*,*y*). *x'* = *snd x* ⟶ *y* ∈ *B*)›
from ‹*open A*› ‹*open B*› ‹*x* ∈ *A*×*B*›
have ‹*eventually UA uniformity*› ‹*eventually UB uniformity*›
unfolding *open_uniformity UA_def UB_def* **by** *auto*
then have ‹∀$_F$ ((*x'1*, *y1*), (*x'2*, *y2*)) *in uniformity* ×$_F$ *uniformity*. (*x'1*,*x'2*) = *x* ⟶ (*y1*,*y2*) ∈ *U*›
apply (*auto intro*!: *exI*[*of*_ *UA*] *exI*[*of*_ *UB*] *simp add*: *eventually_prod_filter*)
using ‹*A*×*B* ⊆ *U*› **by** (*auto simp*: *UA_def UB_def*)
then show ‹∀$_F$ (*x'*, *y*) *in uniformity*. *x'* = *x* ⟶ *y* ∈ *U*›
by (*simp add*: *uniformity_prod_def eventually_filtermap case_prod_unfold*)
next
assume *asm*: ‹∀ *x*∈*U*. ∀$_F$ (*x'*, *y*) *in uniformity*. *x'* = *x* ⟶ *y* ∈ *U*›
show ‹*open U*›
proof (*unfold open_prod_def*, *intro ballI*)
fix *x* **assume** ‹*x* ∈ *U*›
with *asm* **have** ‹∀$_F$ (*x'*, *y*) *in uniformity*. *x'* = *x* ⟶ *y* ∈ *U*›
by *auto*
then have ‹∀$_F$ ((*x'1*, *y1*), (*x'2*, *y2*)) *in uniformity* ×$_F$ *uniformity*. (*x'1*,*x'2*) = *x* ⟶ (*y1*,*y2*) ∈ *U*›

by (*simp add*: *uniformity_prod_def eventually_filtermap case_prod_unfold*)
then obtain *UA UB* **where** ‹*eventually UA uniformity*› **and** ‹*eventually UB uniformity*›
and *UA_UB_U*: ‹*UA* (*a1*, *a2*) ⟹ *UB* (*b1*, *b2*) ⟹ (*a1*, *b1*) = *x* ⟹ (*a2*, *b2*) ∈ *U*› **for** *a1 a2 b1 b2*
apply *atomize_elim* **by** (*simp add*: *case_prod_beta eventually_prod_filter*)
have ‹*eventually* (λ*a*. *UA* (*fst x*, *a*)) (*nhds* (*fst x*))›
using ‹*eventually UA uniformity*› *eventually_mono eventually_nhds_uniformity* **by** *fastforce*
then obtain *A* **where** ‹*open A*› **and** *A_UA*: ‹*A* ⊆ {*a*. *UA* (*fst x*, *a*)}› **and** ‹*fst x* ∈ *A*›
by (*metis* (*mono_tags*, *lifting*) *eventually_nhds mem_Collect_eq subsetI*)
have ‹*eventually* (λ*b*. *UB* (*snd x*, *b*)) (*nhds* (*snd x*))›
using ‹*eventually UB uniformity*› *eventually_mono eventually_nhds_uniformity* **by** *fastforce*
then obtain *B* **where** ‹*open B*› **and** *B_UB*: ‹*B* ⊆ {*b*. *UB* (*snd x*, *b*)}› **and** ‹*snd x* ∈ *B*›
by (*metis* (*mono_tags*, *lifting*) *eventually_nhds mem_Collect_eq subsetI*)
have ‹*x* ∈ *A* × *B*›
by (*simp add*: ‹*fst x* ∈ *A*› ‹*snd x* ∈ *B*› *mem_Times_iff*)
have ‹*A* × *B* ⊆ *U*›
using *A_UA B_UB UA_UB_U* **by** *fastforce*
show ‹∃ *A B*. *open A* ∧ *open B* ∧ *x* ∈ *A* × *B* ∧ *A* × *B* ⊆ *U*›
using ‹*A* × *B* ⊆ *U*› ‹*open A*› ‹*open B*› ‹*x* ∈ *A* × *B*› **by** *auto*
qed
qed
next
show ‹*eventually E uniformity* ⟹ *E* (*x*, *x*)› **for** *E* **and** *x* :: ‹'*a* × '*b*›
apply (*simp add*: *uniformity_prod_def eventually_filtermap case_prod_unfold eventually_prod_filter*)
by (*metis surj_pair uniformity_refl*)
next
show ‹*eventually E uniformity* ⟹ ∀⇩*F* (*x*::'*a*×'*b*, *y*) *in uniformity*. *E* (*y*, *x*)› **for** *E*
apply (*simp only*: *uniformity_prod_def eventually_filtermap case_prod_unfold eventually_prod_filter*)
apply (*erule exE*, *erule exE*, *rename_tac Pf Pg*)
apply (*rule_tac x*=‹λ(*x*,*y*). *Pf* (*y*,*x*)› **in** *exI*)
apply (*rule_tac x*=‹λ(*x*,*y*). *Pg* (*y*,*x*)› **in** *exI*)
by (*auto simp add*: *uniformity_sym*)
next
show ‹∃ *D*. *eventually D uniformity* ∧ (∀ *x y z*. *D* (*x*::'*a*×'*b*, *y*) ⟶ *D* (*y*, *z*) ⟶ *E* (*x*, *z*))›
if ‹*eventually E uniformity*› **for** *E*
proof −
from *that*
obtain *EA EB* **where** ‹*eventually EA uniformity*› **and** ‹*eventually EB uniformity*›
and *EA_EB_E*: ‹*EA* (*a1*, *a2*) ⟹ *EB* (*b1*, *b2*) ⟹ *E* ((*a1*, *b1*), (*a2*,

b2))› **for** *a1 a2 b1 b2*
by (*auto simp add*: *uniformity_prod_def eventually_filtermap case_prod_unfold eventually_prod_filter*)
obtain *DA* **where** ‹*eventually DA uniformity*› **and** *DA_EA*: ‹*DA* (*x*,*y*) ⟹ *DA* (*y*,*z*) ⟹ *EA* (*x*,*z*)› **for** *x y z*
using ‹*eventually EA uniformity*› *uniformity_transE* **by** *blast*
obtain *DB* **where** ‹*eventually DB uniformity*› **and** *DB_EB*: ‹*DB* (*x*,*y*) ⟹ *DB* (*y*,*z*) ⟹ *EB* (*x*,*z*)› **for** *x y z*
using ‹*eventually EB uniformity*› *uniformity_transE* **by** *blast*
define *D* **where** ‹*D* = (λ((*a1*,*b1*),(*a2*,*b2*)). *DA* (*a1*,*a2*) ∧ *DB* (*b1*,*b2*))›
have ‹*eventually D uniformity*›
using ‹*eventually DA uniformity*› ‹*eventually DB uniformity*›
by (*auto simp add*: *uniformity_prod_def eventually_filtermap case_prod_unfold eventually_prod_filter D_def*)
moreover have ‹*D* ((*a1*, *b1*), (*a2*, *b2*)) ⟹ *D* ((*a2*, *b2*), (*a3*, *b3*)) ⟹ *E* ((*a1*, *b1*), (*a3*, *b3*))› **for** *a1 b1 a2 b2 a3 b3*
using *DA_EA DB_EB D_def EA_EB_E* **by** *blast*
ultimately show *?thesis*
by *auto*
qed
qed
end

lemma (**in** *uniform_space*) *nhds_eq_comap_uniformity*: *nhds x* = *filtercomap* (λ*y*. (*x*, *y*)) *uniformity*
proof −
have ∗: *eventually P* (*filtercomap* (λ*y*. (*x*, *y*)) *F*) ⟷
eventually (λ*z*. *fst z* = *x* ⟶ *P* (*snd z*)) *F* **for** *P* :: *'a* ⇒ *bool* **and** *F*
unfolding *eventually_filtercomap*
by (*smt* (*verit*) *eventually_elim2 fst_conv prod.collapse snd_conv*)
thus *?thesis*
unfolding *filter_eq_iff*
by (*subst* ∗) (*auto simp*: *eventually_nhds_uniformity case_prod_unfold*)
qed

lemma *uniformity_of_uniform_continuous_invariant*:
fixes *f* :: *'a* :: *uniform_space* ⇒ *'a* ⇒ *'a*
assumes *filterlim* (λ((*a*,*b*),(*c*,*d*)). (*f a c*, *f b d*)) *uniformity* (*uniformity* ×$_F$ *uniformity*)
assumes *eventually P uniformity*
obtains *Q* **where** *eventually Q uniformity* ⋀*a b c*. *Q* (*a*, *b*) ⟹ *P* (*f a c*, *f b c*)
using *eventually_compose_filterlim*[*OF assms*(*2*,*1*)] *uniformity_refl*
by (*fastforce simp*: *case_prod_unfold eventually_filtercomap eventually_prod_same*)

class *uniform_topological_monoid_add* = *topological_monoid_add* + *uniform_space* +
assumes *uniformly_continuous_add'*:
filterlim (λ((*a*,*b*), (*c*,*d*)). (*a* + *c*, *b* + *d*)) *uniformity* (*uniformity* ×$_F$ *uniformity*)

lemma *uniformly_continuous_add*:
uniformly_continuous_on UNIV (λ(*x* :: *'a* :: *uniform_topological_monoid_add*,*y*).
x + *y*)
using *uniformly_continuous_add'*[**where** *?'a* = *'a*]
by (*simp add*: *uniformly_continuous_on_uniformity case_prod_unfold uniformity_prod_def filterlim_filtermap*)

lemma *filterlim_fst*: *filterlim fst F* (*F* ×$_F$ *G*)
by (*simp add*: *filterlim_def filtermap_fst_prod_filter*)

lemma *filterlim_snd*: *filterlim snd G* (*F* ×$_F$ *G*)
by (*simp add*: *filterlim_def filtermap_snd_prod_filter*)

class *uniform_topological_group_add* = *topological_group_add* + *uniform_topological_monoid_add*
+
assumes *uniformly_continuous_uminus'*: *filterlim* (λ(*a*, *b*). (−*a*, −*b*)) *uniformity*
uniformity
begin

lemma *uniformly_continuous_minus'*:
filterlim (λ((*a*,*b*), (*c*,*d*)). (*a* − *c*, *b* − *d*)) *uniformity* (*uniformity* ×$_F$ *uniformity*)
proof −
have *filterlim* ((λ((*a*,*b*), (*c*,*d*)). (*a* + *c*, *b* + *d*)) ∘ (λ((*a*,*b*), (*c*,*d*)). ((*a*, *b*), (−*c*,
−*d*))))
uniformity (*uniformity* ×$_F$ *uniformity*)
unfolding *o_def* **using** *uniformly_continuous_uminus'*
by (*intro filterlim_compose*[*OF uniformly_continuous_add'*])
(*auto simp*: *case_prod_unfold intro*!: *filterlim_Pair*
filterlim_fst filterlim_compose[*OF _ filterlim_snd*])
thus *?thesis*
by (*simp add*: *o_def case_prod_unfold*)
qed

end

lemma *uniformly_continuous_uminus*:
uniformly_continuous_on UNIV (λ*x* :: *'a* :: *uniform_topological_group_add*.
−*x*)
using *uniformly_continuous_uminus'*[**where** *?'a* = *'a*]
by (*simp add*: *uniformly_continuous_on_uniformity*)

lemma *uniformly_continuous_minus*:
uniformly_continuous_on UNIV (λ(*x* :: *'a* :: *uniform_topological_group_add*,*y*).
x − *y*)
using *uniformly_continuous_minus'*[**where** *?'a* = *'a*]
by (*simp add*: *uniformly_continuous_on_uniformity case_prod_unfold uniformity_prod_def filterlim_filtermap*)

lemma *real_normed_vector_is_uniform_topological_group_add* [*Pure.intro*]:
OFCLASS('*a* :: *real_normed_vector*, *uniform_topological_group_add_class*)
proof
show *filterlim* (λ((*a*::'*a*,*b*), (*c*,*d*)). (*a* + *c*, *b* + *d*)) *uniformity* (*uniformity* ×$_F$ *uniformity*)
unfolding *filterlim_def le_filter_def eventually_filtermap case_prod_unfold*
proof *safe*
fix *P* :: '*a* × '*a* ⇒ *bool*
assume *eventually P uniformity*
then obtain ε **where** ε: ε > 0 ⋀*x y*. *dist x y* < ε ⟹ *P* (*x*, *y*)
by (*auto simp*: *eventually_uniformity_metric*)
define *Q* **where** *Q* = (λ(*x*::'*a*,*y*). *dist x y* < ε / 2)
have *Q*: *eventually Q uniformity*
unfolding *eventually_uniformity_metric Q_def* **using** ‹ε > 0›
by (*meson case_prodI divide_pos_pos zero_less_numeral*)
have *P* (*a* + *c*, *b* + *d*) **if** *Q* (*a*, *b*) *Q* (*c*, *d*) **for** *a b c d*
proof −
have *dist* (*a* + *c*) (*b* + *d*) ≤ *dist a b* + *dist c d*
by (*simp add*: *dist_norm norm_diff_triangle_ineq*)
also have ... < ε
using *that* **by** (*auto simp*: *Q_def*)
finally show *?thesis*
by (*intro* ε)
qed
thus ∀$_F$ *x in uniformity* ×$_F$ *uniformity*. *P* (*fst* (*fst x*) + *fst* (*snd x*), *snd* (*fst x*) + *snd* (*snd x*))
unfolding *eventually_prod_filter* **by** (*intro exI*[*of* _ *Q*] *conjI Q*) *auto*
qed
next
show *filterlim* (λ((*a*::'*a*), *b*). (−*a*, −*b*)) *uniformity uniformity*
unfolding *filterlim_def le_filter_def eventually_filtermap*
proof *safe*
fix *P* :: '*a* × '*a* ⇒ *bool*
assume *eventually P uniformity*
then obtain ε **where** ε: ε > 0 ⋀*x y*. *dist x y* < ε ⟹ *P* (*x*, *y*)
by (*auto simp*: *eventually_uniformity_metric*)
show ∀$_F$ *x in uniformity*. *P* (*case x of* (*a*, *b*) ⇒ (− *a*, − *b*))
unfolding *eventually_uniformity_metric*
by (*intro exI*[*of* _ ε]) (*auto intro*!: ε *simp*: *dist_norm norm_minus_commute*)
qed
qed

instance *real* :: *uniform_topological_group_add* ..
instance *complex* :: *uniform_topological_group_add* ..

lemma *cauchy_seq_finset_iff_vanishing*:
uniformity = *filtercomap* (λ(*x*,*y*). *y* − *x* :: '*a* :: *uniform_topological_group_add*)

(*nhds 0*)
proof −
have *filtercomap* (λx. (*0*, *case x of* (x, y) ⇒ y − (x :: 'a))) *uniformity* ≤ *uniformity*
apply (*simp add*: *le_filter_def eventually_filtercomap*)
using *uniformity_of_uniform_continuous_invariant*[*OF uniformly_continuous_add'*]
by (*metis diff_self eq_diff_eq*)
moreover
have *uniformity* ≤ *filtercomap* (λx. (*0*, *case x of* (x, y) ⇒ y − (x :: 'a))) *uniformity*
apply (*simp add*: *le_filter_def eventually_filtercomap*)
using *uniformity_of_uniform_continuous_invariant*[*OF uniformly_continuous_minus'*]
by (*metis* (*mono_tags*) *diff_self eventually_mono surjective_pairing*)
ultimately show *?thesis*
by (*simp add*: *nhds_eq_comap_uniformity filtercomap_filtercomap*)
qed

Metric spaces

instantiation *prod* :: (*metric_space*, *metric_space*) *uniformity_dist* **begin**
instance
proof
show ‹*uniformity* = (*INF* e∈{*0* <..}. *principal* {(x::'a×'b, y). *dist x y* < e})›
proof (*subst filter_eq_iff*, *intro allI iffI*)
fix *P* :: ‹('a × 'b) × ('a × 'b) ⇒ *bool*›

have *1*: ‹∃ e∈{*0*<..}.
{(x,y). *dist x y* < e} ⊆ {(x,y). *dist x y* < a} ∧
{(x,y). *dist x y* < e} ⊆ {(x,y). *dist x y* < b}› **if** ‹a>0› ‹b>0› **for** *a b*
apply (*rule bexI*[*of* _ ‹*min a b*›])
using *that* **by** *auto*
have *2*: ‹*mono* (λP. *eventually* (λx. *P* (*Q x*)) *F*)› **for** *F* :: ‹'z *filter*› **and** *Q* :: ‹'z ⇒ 'y›
unfolding *mono_def* **using** *eventually_mono le_funD* **by** *fastforce*
have ‹∀⇩F ((*x1*::'a,*y1*),(*x2*::'b,*y2*)) *in uniformity* ×⇩F *uniformity*. *dist x1 y1* < *e*/*2* ∧ *dist x2 y2* < *e*/*2*› **if** ‹e>0› **for** *e*
by (*auto intro*!: *eventually_prodI exI*[*of* _ ‹*e*/*2*›] *simp*: *case_prod_unfold eventually_uniformity_metric that*)
then have *3*: ‹∀⇩F ((*x1*::'a,*y1*),(*x2*::'b,*y2*)) *in uniformity* ×⇩F *uniformity*. *dist* (*x1*,*x2*) (*y1*,*y2*) < *e*› **if** ‹e>0› **for** *e*
apply (*rule eventually_rev_mp*)
by (*auto intro*!: *that eventuallyI simp*: *case_prod_unfold dist_prod_def sqrt_sum_squares_half_less*)
show ‹*eventually P* (*INF* e∈{*0*<..}. *principal* {(x, y). *dist x y* < e}) ⟹ *eventually P uniformity*›
apply (*subst* (*asm*) *eventually_INF_base*)
using *1 3* **apply** (*auto simp*: *uniformity_prod_def case_prod_unfold eventually_filtermap 2 eventually_principal*)
by (*smt* (*verit*, *best*) *eventually_mono*)

```
  next
    fix P :: ‹('a × 'b) × ('a × 'b) ⇒ bool›
    assume ‹eventually P uniformity›
    then obtain P1 P2 where ‹eventually P1 uniformity› ‹eventually P2 uniformity›
      and P1P2P: ‹P1 (x1, y1) ⟹ P2 (x2, y2) ⟹ P ((x1, x2), (y1, y2))› for x1 y1 x2 y2
       by (auto simp: eventually_filtermap case_prod_beta eventually_prod_filter uniformity_prod_def)
    from ‹eventually P1 uniformity› obtain e1 where ‹e1>0› and e1P1: ‹dist x y < e1 ⟹ P1 (x,y)› for x y
      using eventually_uniformity_metric by blast
    from ‹eventually P2 uniformity› obtain e2 where ‹e2>0› and e2P2: ‹dist x y < e2 ⟹ P2 (x,y)› for x y
      using eventually_uniformity_metric by blast
    define e where ‹e = min e1 e2›
    have ‹e > 0›
      using ‹0 < e1› ‹0 < e2› e_def by auto
    have ‹dist (x1,x2) (y1,y2) < e ⟹ dist x1 y1 < e1› for x1 y1 :: 'a and x2 y2 :: 'b
      unfolding dist_prod_def e_def apply auto
      by (smt (verit, best) real_sqrt_sum_squares_ge1)
    moreover have ‹dist (x1,x2) (y1,y2) < e ⟹ dist x2 y2 < e2› for x1 y1 :: 'a and x2 y2 :: 'b
      unfolding dist_prod_def e_def apply auto
      by (smt (verit, best) real_sqrt_sum_squares_ge1)
    ultimately have *: ‹dist (x1,x2) (y1,y2) < e ⟹ P ((x1, x2), (y1, y2))› for x1 y1 x2 y2
      using e1P1 e2P2 P1P2P by auto

    show ‹eventually P (INF e∈{0<..}. principal {(x, y). dist x y < e})›
      apply (rule eventually_INF1[where i=e])
      using ‹e > 0› * by (auto simp: eventually_principal)
  qed
qed
end

declare uniformity_Abort[where 'a='a :: metric_space × 'b :: metric_space, code]

instantiation prod :: (metric_space, metric_space) metric_space
begin

proposition dist_Pair_Pair: dist (a, b) (c, d) = sqrt ((dist a c)^2 + (dist b d)^2)
  unfolding dist_prod_def by simp

lemma dist_fst_le: dist (fst x) (fst y) ≤ dist x y
  unfolding dist_prod_def by (rule real_sqrt_sum_squares_ge1)
```

```
lemma dist_snd_le: dist (snd x) (snd y) ≤ dist x y
  unfolding dist_prod_def by (rule real_sqrt_sum_squares_ge2)

instance
proof
  fix x y :: 'a × 'b
  show dist x y = 0 ⟷ x = y
    unfolding dist_prod_def prod_eq_iff by simp
next
  fix x y z :: 'a × 'b
  show dist x y ≤ dist x z + dist y z
    unfolding dist_prod_def
    by (intro order_trans [OF _ real_sqrt_sum_squares_triangle_ineq]
        real_sqrt_le_mono add_mono power_mono dist_triangle2 zero_le_dist)
next
  fix S :: ('a × 'b) set
  have ∗: open S ⟷ (∀ x∈S. ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ S)
  proof
    assume open S show ∀ x∈S. ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ S
    proof
      fix x assume x ∈ S
      obtain A B where open A open B x ∈ A × B A × B ⊆ S
        using ‹open S› and ‹x ∈ S› by (rule open_prod_elim)
      obtain r where r: 0 < r ∀ y. dist y (fst x) < r ⟶ y ∈ A
        using ‹open A› and ‹x ∈ A × B› unfolding open_dist by auto
      obtain s where s: 0 < s ∀ y. dist y (snd x) < s ⟶ y ∈ B
        using ‹open B› and ‹x ∈ A × B› unfolding open_dist by auto
      let ?e = min r s
      have 0 < ?e ∧ (∀ y. dist y x < ?e ⟶ y ∈ S)
      proof (intro allI impI conjI)
        show 0 < min r s by (simp add: r(1) s(1))
      next
        fix y assume dist y x < min r s
        hence dist y x < r and dist y x < s
          by simp_all
        hence dist (fst y) (fst x) < r and dist (snd y) (snd x) < s
          by (auto intro: le_less_trans dist_fst_le dist_snd_le)
        hence fst y ∈ A and snd y ∈ B
          by (simp_all add: r(2) s(2))
        hence y ∈ A × B by (induct y, simp)
        with ‹A × B ⊆ S› show y ∈ S ..
      qed
      thus ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ S ..
    qed
  next
    assume ∗: ∀ x∈S. ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ S show open S
    proof (rule open_prod_intro)
      fix x assume x ∈ S
      then obtain e where 0 < e and S: ∀ y. dist y x < e ⟶ y ∈ S
```

```
    using ∗ by fast
  define r where r = e / sqrt 2
  define s where s = e / sqrt 2
  from ‹0 < e› have 0 < r and 0 < s
    unfolding r_def s_def by simp_all
  from ‹0 < e› have e = sqrt (r^2 + s^2)
    unfolding r_def s_def by (simp add: power_divide)
  define A where A = {y. dist (fst x) y < r}
  define B where B = {y. dist (snd x) y < s}
  have open A and open B
    unfolding A_def B_def by (simp_all add: open_ball)
  moreover have x ∈ A × B
    unfolding A_def B_def mem_Times_iff
    using ‹0 < r› and ‹0 < s› by simp
  moreover have A × B ⊆ S
  proof (clarify)
    fix a b assume a ∈ A and b ∈ B
    hence dist a (fst x) < r and dist b (snd x) < s
      unfolding A_def B_def by (simp_all add: dist_commute)
    hence dist (a, b) x < e
      unfolding dist_prod_def ‹e = sqrt (r^2 + s^2)›
      by (simp add: add_strict_mono power_strict_mono)
    thus (a, b) ∈ S
      by (simp add: S)
  qed
  ultimately show ∃ A B. open A ∧ open B ∧ x ∈ A × B ∧ A × B ⊆ S by
fast
 qed
qed
qed

end

declare [[code abort: dist::('a::metric_space*'b::metric_space)⇒('a*'b) ⇒ real]]

lemma Cauchy_fst: Cauchy X ⟹ Cauchy (λn. fst (X n :: 'a::metric_space ×
'b::metric_space))
  unfolding Cauchy_def by (fast elim: le_less_trans [OF dist_fst_le])

lemma Cauchy_snd: Cauchy X ⟹ Cauchy (λn. snd (X n :: 'a::metric_space ×
'b::metric_space))
  unfolding Cauchy_def by (fast elim: le_less_trans [OF dist_snd_le])

lemma Cauchy_Pair:
  assumes Cauchy X and Cauchy Y
  shows Cauchy (λn. (X n :: 'a::metric_space, Y n :: 'a::metric_space))
proof (rule metric_CauchyI)
  fix r :: real assume 0 < r
  hence 0 < r / sqrt 2 (is 0 < ?s) by simp
```

obtain M **where** M: $\forall\, m{\geq}M.\ \forall\, n{\geq}M.\ dist\ (X\ m)\ (X\ n) < ?s$
using *metric_CauchyD* [*OF* ‹*Cauchy X*› ‹$0 < ?s$›] **..**
obtain N **where** N: $\forall\, m{\geq}N.\ \forall\, n{\geq}N.\ dist\ (Y\ m)\ (Y\ n) < ?s$
using *metric_CauchyD* [*OF* ‹*Cauchy Y*› ‹$0 < ?s$›] **..**
have $\forall\, m{\geq}max\ M\ N.\ \forall\, n{\geq}max\ M\ N.\ dist\ (X\ m,\ Y\ m)\ (X\ n,\ Y\ n) < r$
using $M\ N$ **by** (*simp add*: *real_sqrt_sum_squares_less dist_Pair_Pair*)
then show $\exists\, n0.\ \forall\, m{\geq}n0.\ \forall\, n{\geq}n0.\ dist\ (X\ m,\ Y\ m)\ (X\ n,\ Y\ n) < r$ **..**
qed

Analogue to *uniformly_continuous_on_def* for two-argument functions.

lemma *uniformly_continuous_on_prod_metric*:
fixes f :: ‹('a::*metric_space* × 'b::*metric_space*) ⇒ 'c::*metric_space*›
shows ‹*uniformly_continuous_on* $(S{\times}T)\ f \longleftrightarrow (\forall\, e{>}0.\ \exists\, d{>}0.\ \forall\, x{\in}S.\ \forall\, y{\in}S.\ \forall\, x'{\in}T.\ \forall\, y'{\in}T.\ dist\ x\ y < d \longrightarrow dist\ x'\ y' < d \longrightarrow dist\ (f\ (x,\ x'))\ (f\ (y,\ y')) < e)$›
proof (*unfold uniformly_continuous_on_def*, *intro iffI impI allI*)
fix e :: *real*
assume ‹$e > 0$› **and** ‹$\forall\, e{>}0.\ \exists\, d{>}0.\ \forall\, x{\in}S.\ \forall\, y{\in}S.\ \forall\, x'{\in}T.\ \forall\, y'{\in}T.\ dist\ x\ y < d \longrightarrow dist\ x'\ y' < d \longrightarrow dist\ (f\ (x,\ x'))\ (f\ (y,\ y')) < e$›
then obtain d **where** ‹$d > 0$› **and** d: ‹$\forall\, x{\in}S.\ \forall\, y{\in}S.\ \forall\, x'{\in}T.\ \forall\, y'{\in}T.\ dist\ x\ y < d \longrightarrow dist\ x'\ y' < d \longrightarrow dist\ (f\ (x,\ x'))\ (f\ (y,\ y')) < e$›
by *auto*
show ‹$\exists\, d{>}0.\ \forall\, x{\in}S{\times}T.\ \forall\, y{\in}S{\times}T.\ dist\ y\ x < d \longrightarrow dist\ (f\ y)\ (f\ x) < e$›
apply (*rule exI*[*of* _ d])
using ‹$d{>}0$› d[*rule_format*] **apply** *auto*
by (*smt* (*verit*, *del_insts*) *dist_fst_le dist_snd_le fst_conv snd_conv*)
next
fix e :: *real*
assume ‹$e > 0$› **and** ‹$\forall\, e{>}0.\ \exists\, d{>}0.\ \forall\, x{\in}S{\times}T.\ \forall\, x'{\in}S{\times}T.\ dist\ x'\ x < d \longrightarrow dist\ (f\ x')\ (f\ x) < e$›
then obtain d **where** ‹$d > 0$› **and** d: ‹$\forall\, x{\in}S{\times}T.\ \forall\, x'{\in}S{\times}T.\ dist\ x'\ x < d \longrightarrow dist\ (f\ x')\ (f\ x) < e$›
by *auto*
show ‹$\exists\, d{>}0.\ \forall\, x{\in}S.\ \forall\, y{\in}S.\ \forall\, x'{\in}T.\ \forall\, y'{\in}T.\ dist\ x\ y < d \longrightarrow dist\ x'\ y' < d \longrightarrow dist\ (f\ (x,\ x'))\ (f\ (y,\ y')) < e$›
proof (*intro exI conjI impI ballI*)
from ‹$d > 0$› **show** ‹$d\ /\ 2 > 0$› **by** *auto*
fix $x\ y\ x'\ y'$
assume [*simp*]: ‹$x \in S$› ‹$y \in S$› ‹$x' \in T$› ‹$y' \in T$›
assume ‹$dist\ x\ y < d\ /\ 2$› **and** ‹$dist\ x'\ y' < d\ /\ 2$›
then have ‹$dist\ (x,\ x')\ (y,\ y') < d$›
by (*simp add*: *dist_Pair_Pair sqrt_sum_squares_half_less*)
with d **show** ‹$dist\ (f\ (x,\ x'))\ (f\ (y,\ y')) < e$›
by *auto*
qed
qed

Analogue to *isUCont_def* for two-argument functions.

lemma *isUCont_prod_metric*:
fixes *f* :: ‹('*a*::*metric_space* × '*b*::*metric_space*) ⇒ '*c*::*metric_space*›
shows ‹*isUCont f* ⟷ (∀ *e*>*0*. ∃ *d*>*0*. ∀ *x*. ∀ *y*. ∀ *x'*. ∀ *y'*. *dist x y* < *d* ⟶ *dist x' y'* < *d* ⟶ *dist* (*f* (*x*, *x'*)) (*f* (*y*, *y'*)) < *e*)›
using *uniformly_continuous_on_prod_metric*[*of UNIV UNIV*]
by *auto*

This logically belong with the real vector spaces but we only have the necessary lemmas now.

lemma *isUCont_plus*[*simp*]:
shows ‹*isUCont* (λ(*x*::'*a*::*real_normed_vector*,*y*). *x*+*y*)›
proof (*rule isUCont_prod_metric*[*THEN iffD2*], *intro allI impI*, *simp*)
fix *e* :: *real* **assume** ‹*0* < *e*›
show ‹∃ *d*>*0*. ∀ *x y* :: '*a*. *dist x y* < *d* ⟶ (∀ *x' y'*. *dist x' y'* < *d* ⟶ *dist* (*x* + *x'*) (*y* + *y'*) < *e*)›
apply (*rule exI*[*of* _ ‹*e*/*2*›])
using ‹*0* < *e*› **apply** *auto*
by (*smt* (*verit*, *ccfv_SIG*) *dist_add_cancel dist_add_cancel2 dist_commute dist_triangle_lt*)
qed

1.3.4 Product is a Complete Metric Space

instance *prod* :: (*complete_space*, *complete_space*) *complete_space*
proof
fix *X* :: *nat* ⇒ '*a* × '*b* **assume** *Cauchy X*
have *1*: (λ*n*. *fst* (*X n*)) ⟶ *lim* (λ*n*. *fst* (*X n*))
using *Cauchy_fst* [*OF* ‹*Cauchy X*›]
by (*simp add*: *Cauchy_convergent_iff convergent_LIMSEQ_iff*)
have *2*: (λ*n*. *snd* (*X n*)) ⟶ *lim* (λ*n*. *snd* (*X n*))
using *Cauchy_snd* [*OF* ‹*Cauchy X*›]
by (*simp add*: *Cauchy_convergent_iff convergent_LIMSEQ_iff*)
have *X* ⟶ (*lim* (λ*n*. *fst* (*X n*)), *lim* (λ*n*. *snd* (*X n*)))
using *tendsto_Pair* [*OF 1 2*] **by** *simp*
then show *convergent X*
by (*rule convergentI*)
qed

1.3.5 Product is a Normed Vector Space

instantiation *prod* :: (*real_normed_vector*, *real_normed_vector*) *real_normed_vector*
begin

definition *norm_prod_def*[*code del*]:
norm x = *sqrt* ((*norm* (*fst x*))² + (*norm* (*snd x*))²)

definition *sgn_prod_def*:
sgn (*x*::'*a* × '*b*) = *scaleR* (*inverse* (*norm x*)) *x*

proposition *norm_Pair*: *norm* (*a*, *b*) = *sqrt* ((*norm a*)^2 + (*norm b*)^2)
unfolding *norm_prod_def* **by** *simp*

instance
proof
fix *r* :: *real* **and** *x y* :: *'a* × *'b*
show *norm x* = *0* ⟷ *x* = *0*
unfolding *norm_prod_def*
by (*simp add*: *prod_eq_iff*)
show *norm* (*x* + *y*) ≤ *norm x* + *norm y*
unfolding *norm_prod_def*
apply (*rule order_trans* [*OF* _ *real_sqrt_sum_squares_triangle_ineq*])
apply (*simp add*: *add_mono power_mono norm_triangle_ineq*)
done
show *norm* (*scaleR r x*) = |*r*| ∗ *norm x*
unfolding *norm_prod_def*
apply (*simp add*: *power_mult_distrib*)
apply (*simp add*: *distrib_left* [*symmetric*])
apply (*simp add*: *real_sqrt_mult*)
done
show *sgn x* = *scaleR* (*inverse* (*norm x*)) *x*
by (*rule sgn_prod_def*)
show *dist x y* = *norm* (*x* − *y*)
unfolding *dist_prod_def norm_prod_def*
by (*simp add*: *dist_norm*)
qed

end

declare [[*code abort*: *norm*::(*'a*::*real_normed_vector*∗*'b*::*real_normed_vector*) ⇒ *real*]]

instance *prod* :: (*banach*, *banach*) *banach* **..**

Pair operations are linear

lemma *bounded_linear_fst*: *bounded_linear fst*
using *fst_add fst_scaleR*
by (*rule bounded_linear_intro* [**where** *K*=*1*], *simp add*: *norm_prod_def*)

lemma *bounded_linear_snd*: *bounded_linear snd*
using *snd_add snd_scaleR*
by (*rule bounded_linear_intro* [**where** *K*=*1*], *simp add*: *norm_prod_def*)

lemmas *bounded_linear_fst_comp* = *bounded_linear_fst*[*THEN bounded_linear_compose*]

lemmas *bounded_linear_snd_comp* = *bounded_linear_snd*[*THEN bounded_linear_compose*]

lemma *bounded_linear_Pair*:

assumes f: *bounded_linear f*
assumes g: *bounded_linear g*
shows *bounded_linear* ($\lambda x.$ ($f\ x$, $g\ x$))
proof
interpret f: *bounded_linear f* **by** *fact*
interpret g: *bounded_linear g* **by** *fact*
fix $x\ y$ **and** r :: *real*
show (f ($x + y$), g ($x + y$)) = ($f\ x$, $g\ x$) + ($f\ y$, $g\ y$)
by (*simp add*: *f.add g.add*)
show (f ($r *_R x$), g ($r *_R x$)) = $r *_R$ ($f\ x$, $g\ x$)
by (*simp add*: *f.scale g.scale*)
obtain *Kf* **where** $0 < Kf$ **and** *norm_f*: $\bigwedge x.$ *norm* ($f\ x$) $\leq$ *norm* $x * Kf$
using *f.pos_bounded* **by** *fast*
obtain *Kg* **where** $0 < Kg$ **and** *norm_g*: $\bigwedge x.$ *norm* ($g\ x$) $\leq$ *norm* $x * Kg$
using *g.pos_bounded* **by** *fast*
have $\forall x.$ *norm* ($f\ x$, $g\ x$) $\leq$ *norm* $x * (Kf + Kg)$
apply (*rule allI*)
apply (*simp add*: *norm_Pair*)
apply (*rule order_trans* [*OF sqrt_add_le_add_sqrt*], *simp*, *simp*)
apply (*simp add*: *distrib_left*)
apply (*rule add_mono* [*OF norm_f norm_g*])
done
then show $\exists K.\ \forall x.$ *norm* ($f\ x$, $g\ x$) $\leq$ *norm* $x * K$ **..**
qed

Frechet derivatives involving pairs

proposition *has_derivative_Pair* [*derivative_intros*]:
assumes f: (f *has_derivative* f') (*at x within s*)
and g: (g *has_derivative* g') (*at x within s*)
shows (($\lambda x.$ ($f\ x$, $g\ x$)) *has_derivative* ($\lambda h.$ ($f'\ h$, $g'\ h$))) (*at x within s*)
proof (*rule has_derivativeI_sandwich*[*of 1*])
show *bounded_linear* ($\lambda h.$ ($f'\ h$, $g'\ h$))
using *f g* **by** (*intro bounded_linear_Pair has_derivative_bounded_linear*)
let *?Rf* = $\lambda y.$ $f\ y - f\ x - f'\ (y - x)$
let *?Rg* = $\lambda y.$ $g\ y - g\ x - g'\ (y - x)$
let *?R* = $\lambda y.$ (($f\ y$, $g\ y$) − ($f\ x$, $g\ x$) − ($f'\ (y - x)$, $g'\ (y - x)$))

show (($\lambda y.$ *norm* (*?Rf y*) / *norm* ($y - x$) + *norm* (*?Rg y*) / *norm* ($y - x$))
$\longrightarrow$ *0*) (*at x within s*)
using *f g* **by** (*intro tendsto_add_zero*) (*auto simp*: *has_derivative_iff_norm*)

fix y :: *'a* **assume** $y \neq x$
show *norm* (*?R y*) / *norm* ($y - x$) $\leq$ *norm* (*?Rf y*) / *norm* ($y - x$) + *norm*
(*?Rg y*) / *norm* ($y - x$)
unfolding *add_divide_distrib* [*symmetric*]
by (*simp add*: *norm_Pair divide_right_mono order_trans* [*OF sqrt_add_le_add_sqrt*])
qed *simp*

lemma *differentiable_Pair* [*simp*, *derivative_intros*]:
f differentiable at x within s ⟹ *g differentiable at x within s* ⟹
(λ*x*. (*f x*, *g x*)) *differentiable at x within s*
unfolding *differentiable_def* **by** (*blast intro*: *has_derivative_Pair*)

lemmas *has_derivative_fst* [*derivative_intros*] = *bounded_linear.has_derivative* [*OF bounded_linear_fst*]
lemmas *has_derivative_snd* [*derivative_intros*] = *bounded_linear.has_derivative* [*OF bounded_linear_snd*]

lemma *has_derivative_split* [*derivative_intros*]:
((λ*p*. *f* (*fst p*) (*snd p*)) *has_derivative f′*) *F* ⟹ ((λ(*a*, *b*). *f a b*) *has_derivative f′*) *F*
unfolding *split_beta′* **.**

Vector derivatives involving pairs

lemma *has_vector_derivative_Pair*[*derivative_intros*]:
assumes (*f has_vector_derivative f′*) (*at x within s*)
(*g has_vector_derivative g′*) (*at x within s*)
shows ((λ*x*. (*f x*, *g x*)) *has_vector_derivative* (*f′*, *g′*)) (*at x within s*)
using *assms*
by (*auto simp*: *has_vector_derivative_def intro*!: *derivative_eq_intros*)

lemma
fixes *x* :: *′a*::*real_normed_vector*
shows *norm_Pair1* [*simp*]: *norm* (*0*,*x*) = *norm x*
and *norm_Pair2* [*simp*]: *norm* (*x*,*0*) = *norm x*
by (*auto simp*: *norm_Pair*)

lemma *norm_commute*: *norm* (*x*,*y*) = *norm* (*y*,*x*)
by (*simp add*: *norm_Pair*)

lemma *norm_fst_le*: *norm x* ≤ *norm* (*x*,*y*)
by (*metis dist_fst_le fst_conv fst_zero norm_conv_dist*)

lemma *norm_snd_le*: *norm y* ≤ *norm* (*x*,*y*)
by (*metis dist_snd_le snd_conv snd_zero norm_conv_dist*)

lemma *norm_Pair_le*:
shows *norm* (*x*, *y*) ≤ *norm x* + *norm y*
unfolding *norm_Pair*
by (*metis norm_ge_zero sqrt_sum_squares_le_sum*)

lemma (**in** *vector_space_prod*) *span_Times_sing1*: *p.span* ({*0*} × *B*) = {*0*} × *vs2.span B*
apply (*rule p.span_unique*)
subgoal by (*auto intro*!: *vs1.span_base vs2.span_base*)
subgoal using *vs1.subspace_single_0 vs2.subspace_span* **by** (*rule subspace_Times*)

subgoal for T
proof *safe*
fix b
assume *subset_T*: $\{0\} \times B \subseteq T$ **and** *subspace*: *p.subspace* T **and** *b_span*: $b \in$ *vs2.span* B
then obtain t r **where** b: $b = (\sum a{\in}t.\ r\ a *b\ a)$ **and** t: *finite* t $t \subseteq B$
by (*auto simp*: *vs2.span_explicit*)
have $(0,\ b) = (\sum b{\in}t.\ scale\ (r\ b)\ (0,\ b))$
unfolding b *scale_prod sum_prod*
by *simp*
also have $\ldots \in T$
using ‹$t \subseteq B$› *subset_T*
by (*auto intro*!: *p.subspace_sum p.subspace_scale subspace*)
finally show $(0,\ b) \in T$ **.**
qed
done

lemma (**in** *vector_space_prod*) *span_Times_sing2*: *p.span* $(A \times \{0\}) =$ *vs1.span* $A \times \{0\}$
apply (*rule p.span_unique*)
subgoal by (*auto intro*!: *vs1.span_base vs2.span_base*)
subgoal using *vs1.subspace_span vs2.subspace_single_0* **by** (*rule subspace_Times*)
subgoal for T
proof *safe*
fix a
assume *subset_T*: $A \times \{0\} \subseteq T$ **and** *subspace*: *p.subspace* T **and** *a_span*: $a \in$ *vs1.span* A
then obtain t r **where** a: $a = (\sum a{\in}t.\ r\ a *a\ a)$ **and** t: *finite* t $t \subseteq A$
by (*auto simp*: *vs1.span_explicit*)
have $(a,\ 0) = (\sum a{\in}t.\ scale\ (r\ a)\ (a,\ 0))$
unfolding a *scale_prod sum_prod*
by *simp*
also have $\ldots \in T$
using ‹$t \subseteq A$› *subset_T*
by (*auto intro*!: *p.subspace_sum p.subspace_scale subspace*)
finally show $(a,\ 0) \in T$ **.**
qed
done

1.3.6 Product is Finite Dimensional

lemma (**in** *finite_dimensional_vector_space*) *zero_not_in_Basis*[*simp*]: $0 \notin$ *Basis*
using *dependent_zero local.independent_Basis* **by** *blast*

locale *finite_dimensional_vector_space_prod* = *vector_space_prod* + *finite_dimensional_vector_space*
begin

definition *Basis_pair* = $B1 \times \{0\} \cup \{0\} \times B2$

sublocale *p*: *finite_dimensional_vector_space scale Basis_pair*
proof *unfold_locales*
show *finite Basis_pair*
by (*auto intro*!: *finite_cartesian_product vs1.finite_Basis vs2.finite_Basis simp*: *Basis_pair_def*)
show *p.independent Basis_pair*
unfolding *p.dependent_def Basis_pair_def*
proof *safe*
fix *a*
assume *a*: $a \in B1$
assume $(a, 0) \in p.span\ (B1 \times \{0\} \cup \{0\} \times B2 - \{(a, 0)\})$
also have $B1 \times \{0\} \cup \{0\} \times B2 - \{(a, 0)\} = (B1 - \{a\}) \times \{0\} \cup \{0\} \times B2$
by *auto*
finally show *False*
using *a vs1.dependent_def vs1.independent_Basis*
by (*auto simp*: *p.span_Un span_Times_sing1 span_Times_sing2*)
next
fix *b*
assume *b*: $b \in B2$
assume $(0, b) \in p.span\ (B1 \times \{0\} \cup \{0\} \times B2 - \{(0, b)\})$
also have $(B1 \times \{0\} \cup \{0\} \times B2 - \{(0, b)\}) = B1 \times \{0\} \cup \{0\} \times (B2 - \{b\})$
by *auto*
finally show *False*
using *b vs2.dependent_def vs2.independent_Basis*
by (*auto simp*: *p.span_Un span_Times_sing1 span_Times_sing2*)
qed
show $p.span\ Basis\_pair = UNIV$
by (*auto simp*: *p.span_Un span_Times_sing2 span_Times_sing1 vs1.span_Basis vs2.span_Basis Basis_pair_def*)
qed

proposition *dim_Times*:
assumes *vs1.subspace S vs2.subspace T*
shows $p.dim(S \times T) = vs1.dim\ S + vs2.dim\ T$
proof −
interpret *p1*: *Vector_Spaces.linear s1 scale* $(\lambda x.\ (x, 0))$
by *unfold_locales* (*auto simp*: *scale_def*)
interpret *pair1*: *finite_dimensional_vector_space_pair* $(*a)$ *B1 scale Basis_pair*
by *unfold_locales*
interpret *p2*: *Vector_Spaces.linear s2 scale* $(\lambda x.\ (0, x))$
by *unfold_locales* (*auto simp*: *scale_def*)
interpret *pair2*: *finite_dimensional_vector_space_pair* $(*b)$ *B2 scale Basis_pair*
by *unfold_locales*
have *ss*: *p.subspace* $((\lambda x.\ (x, 0))\ `\ S)$ *p.subspace* $(Pair\ 0\ `\ T)$
by (*rule p1.subspace_image p2.subspace_image assms*)+

have $p.dim(S \times T) = p.dim(\{u + v \mid u\ v.\ u \in (\lambda x.\ (x,\ 0))\ `\ S \wedge v \in Pair\ 0\ `\ T\})$
by (*simp add*: *Times_eq_image_sum*)
moreover have $p.dim\ ((\lambda x.\ (x,\ 0::'c))\ `\ S) = vs1.dim\ S\ p.dim\ (Pair\ (0::'b)\ `\ T) = vs2.dim\ T$
by (*simp_all add*: *inj_on_def p1.linear_axioms pair1.dim_image_eq p2.linear_axioms pair2.dim_image_eq*)
moreover have $p.dim\ ((\lambda x.\ (x,\ 0))\ `\ S \cap Pair\ 0\ `\ T) = 0$
by (*subst p.dim_eq_0*) *auto*
ultimately show *?thesis*
using *p.dim_sums_Int* [*OF ss*] **by** *linarith*
qed

lemma *dimension_pair*: $p.dimension = vs1.dimension + vs2.dimension$
using *dim_Times*[*OF vs1.subspace_UNIV vs2.subspace_UNIV*]
by (*auto simp*: *p.dimension_def vs1.dimension_def vs2.dimension_def*)

end

end

1.4 Finite-Dimensional Inner Product Spaces

theory *Euclidean_Space*
imports
L2_Norm
Inner_Product
Product_Vector
begin

1.4.1 Interlude: Some properties of real sets

lemma *seq_mono_lemma*:
assumes $\forall (n::nat) \geq m.\ (d\ n :: real) < e\ n$
and $\forall n \geq m.\ e\ n \leq e\ m$
shows $\forall n \geq m.\ d\ n < e\ m$
using *assms* **by** *force*

1.4.2 Type class of Euclidean spaces

class *euclidean_space* = *real_inner* +
fixes *Basis* :: *'a set*
assumes *nonempty_Basis* [*simp*]: $Basis \neq \{\}$
assumes *finite_Basis* [*simp*]: *finite Basis*
assumes *inner_Basis*:
$[\![u \in Basis;\ v \in Basis]\!] \Longrightarrow inner\ u\ v = (if\ u = v\ then\ 1\ else\ 0)$
assumes *euclidean_all_zero_iff*:
$(\forall\, u{\in}Basis.\ inner\ x\ u = 0) \longleftrightarrow (x = 0)$

syntax *_type_dimension* :: *type* ⇒ *nat* ((*1DIM*/(*1* '(_')))
translations *DIM*('*a*) ⇀ *CONST card* (*CONST Basis* :: '*a set*)
typed_print_translation ‹
[(***const_syntax*** ‹*card*›,
fn ctxt => *fn* _ => *fn* [*Const* (***const_syntax*** ‹*Basis*›, *Type* (***type_name*** ‹*set*›,
[*T*]))] =>
Syntax.const ***syntax_const*** ‹*_type_dimension*› $ *Syntax_Phases.term_of_typ*
ctxt T)]
›

lemma (**in** *euclidean_space*) *norm_Basis*[*simp*]: *u* ∈ *Basis* ⟹ *norm u* = *1*
unfolding *norm_eq_sqrt_inner* **by** (*simp add*: *inner_Basis*)

lemma (**in** *euclidean_space*) *inner_same_Basis*[*simp*]: *u* ∈ *Basis* ⟹ *inner u u*
= *1*
by (*simp add*: *inner_Basis*)

lemma (**in** *euclidean_space*) *inner_not_same_Basis*: *u* ∈ *Basis* ⟹ *v* ∈ *Basis*
⟹ *u* ≠ *v* ⟹ *inner u v* = *0*
by (*simp add*: *inner_Basis*)

lemma (**in** *euclidean_space*) *sgn_Basis*: *u* ∈ *Basis* ⟹ *sgn u* = *u*
unfolding *sgn_div_norm* **by** (*simp add*: *scaleR_one*)

lemma *inner_sum_Basis*[*simp*]: *i* ∈ *Basis* ⟹ *inner* (∑ *Basis*) *i* = *1*
by (*simp add*: *inner_sum_left sum.If_cases inner_Basis*)

lemma (**in** *euclidean_space*) *Basis_zero* [*simp*]: *0* ∉ *Basis*
proof
assume *0* ∈ *Basis* **thus** *False*
using *inner_Basis* [*of 0 0*] **by** *simp*
qed

lemma (**in** *euclidean_space*) *nonzero_Basis*: *u* ∈ *Basis* ⟹ *u* ≠ *0*
by *clarsimp*

lemma (**in** *euclidean_space*) *SOME_Basis*: (*SOME i*. *i* ∈ *Basis*) ∈ *Basis*
by (*metis ex_in_conv nonempty_Basis someI_ex*)

lemma *norm_some_Basis* [*simp*]: *norm* (*SOME i*. *i* ∈ *Basis*) = *1*
by (*simp add*: *SOME_Basis*)

lemma (**in** *euclidean_space*) *inner_sum_left_Basis*[*simp*]:
b ∈ *Basis* ⟹ *inner* (∑ *i*∈*Basis*. *f i* ∗$_R$ *i*) *b* = *f b*
by (*simp add*: *inner_sum_left inner_Basis if_distrib comm_monoid_add_class.sum.If_cases*)

lemma (**in** *euclidean_space*) *euclidean_eqI*:
assumes *b*: ⋀*b*. *b* ∈ *Basis* ⟹ *inner x b* = *inner y b* **shows** *x* = *y*
proof −

from b **have** $\forall\, b{\in}Basis.\ inner\ (x - y)\ b = 0$
by (*simp add*: *inner_diff_left*)
then show $x = y$
by (*simp add*: *euclidean_all_zero_iff*)
qed

lemma (**in** *euclidean_space*) *euclidean_eq_iff*:
$x = y \longleftrightarrow (\forall\, b{\in}Basis.\ inner\ x\ b = inner\ y\ b)$
by (*auto intro*: *euclidean_eqI*)

lemma (**in** *euclidean_space*) *euclidean_representation_sum*:
$(\sum i{\in}Basis.\ f\ i *_R i) = b \longleftrightarrow (\forall\, i{\in}Basis.\ f\ i = inner\ b\ i)$
by (*subst euclidean_eq_iff*) *simp*

lemma (**in** *euclidean_space*) *euclidean_representation_sum'*:
$b = (\sum i{\in}Basis.\ f\ i *_R i) \longleftrightarrow (\forall\, i{\in}Basis.\ f\ i = inner\ b\ i)$
by (*auto simp add*: *euclidean_representation_sum*[*symmetric*])

lemma (**in** *euclidean_space*) *euclidean_representation*: $(\sum b{\in}Basis.\ inner\ x\ b *_R b) = x$
unfolding *euclidean_representation_sum* **by** *simp*

lemma (**in** *euclidean_space*) *euclidean_inner*: $inner\ x\ y = (\sum b{\in}Basis.\ (inner\ x\ b) * (inner\ y\ b))$
by (*subst* (*1 2*) *euclidean_representation* [*symmetric*])
(*simp add*: *inner_sum_right inner_Basis ac_simps*)

lemma (**in** *euclidean_space*) *choice_Basis_iff*:
fixes $P :: 'a \Rightarrow real \Rightarrow bool$
shows $(\forall\, i{\in}Basis.\ \exists\, x.\ P\ i\ x) \longleftrightarrow (\exists\, x.\ \forall\, i{\in}Basis.\ P\ i\ (inner\ x\ i))$
unfolding *bchoice_iff*
proof *safe*
fix f **assume** $\forall\, i{\in}Basis.\ P\ i\ (f\ i)$
then show $\exists\, x.\ \forall\, i{\in}Basis.\ P\ i\ (inner\ x\ i)$
by (*auto intro*!: *exI*[*of* _ $\sum i{\in}Basis.\ f\ i *_R i$])
qed *auto*

lemma (**in** *euclidean_space*) *bchoice_Basis_iff*:
fixes $P :: 'a \Rightarrow real \Rightarrow bool$
shows $(\forall\, i{\in}Basis.\ \exists\, x{\in}A.\ P\ i\ x) \longleftrightarrow (\exists\, x.\ \forall\, i{\in}Basis.\ inner\ x\ i \in A \wedge P\ i\ (inner\ x\ i))$
by (*simp add*: *choice_Basis_iff Bex_def*)

lemma (**in** *euclidean_space*) *euclidean_representation_sum_fun*:
$(\lambda x.\ \sum b{\in}Basis.\ inner\ (f\ x)\ b *_R b) = f$
by (*rule ext*) (*simp add*: *euclidean_representation_sum*)

lemma *euclidean_isCont*:
assumes $\bigwedge b.\ b \in Basis \Longrightarrow isCont\ (\lambda x.\ (inner\ (f\ x)\ b) *_R b)\ x$

shows *isCont f x*
apply (*subst euclidean_representation_sum_fun* [*symmetric*])
apply (*rule isCont_sum*)
apply (*blast intro*: *assms*)
done

lemma *DIM_positive* [*simp*]: *0 < DIM('a::euclidean_space)*
by (*simp add*: *card_gt_0_iff*)

lemma *DIM_ge_Suc0* [*simp*]: *Suc 0 ≤ card Basis*
by (*meson DIM_positive Suc_leI*)

lemma *sum_inner_Basis_scaleR* [*simp*]:
fixes *f* :: *'a::euclidean_space ⇒ 'b::real_vector*
assumes *b ∈ Basis* **shows** *(∑ i∈Basis. (inner i b) *R f i) = f b*
by (*simp add*: *comm_monoid_add_class.sum.remove* [*OF finite_Basis assms*]
assms inner_not_same_Basis comm_monoid_add_class.sum.neutral)

lemma *sum_inner_Basis_eq* [*simp*]:
assumes *b ∈ Basis* **shows** *(∑ i∈Basis. (inner i b) * f i) = f b*
by (*simp add*: *comm_monoid_add_class.sum.remove* [*OF finite_Basis assms*]
assms inner_not_same_Basis comm_monoid_add_class.sum.neutral)

lemma *sum_if_inner* [*simp*]:
assumes *i ∈ Basis j ∈ Basis*
shows *inner (∑ k∈Basis. if k = i then f i *R i else g k *R k) j = (if j=i then f j else g j)*
proof (*cases i=j*)
case *True*
with *assms* **show** *?thesis*
by (*auto simp*: *inner_sum_left if_distrib* [*of λx. inner x j*] *inner_Basis cong*: *if_cong*)
next
case *False*
have *(∑ k∈Basis. inner (if k = i then f i *R i else g k *R k) j) =*
(∑ k∈Basis. if k = j then g k else 0)
apply (*rule sum.cong*)
using *False assms* **by** (*auto simp*: *inner_Basis*)
also have *... = g j*
using *assms* **by** *auto*
finally show *?thesis*
using *False* **by** (*auto simp*: *inner_sum_left*)
qed

lemma *norm_le_componentwise*:
(⋀b. b ∈ Basis ⟹ abs(inner x b) ≤ abs(inner y b)) ⟹ norm x ≤ norm y
by (*auto simp*: *norm_le euclidean_inner* [*of x x*] *euclidean_inner* [*of y y*] *abs_le_square_iff power2_eq_square intro*!: *sum_mono*)

lemma *Basis_le_norm*: $b \in Basis \Longrightarrow |inner\ x\ b| \leq norm\ x$
by (*rule order_trans* [*OF Cauchy_Schwarz_ineq2*]) *simp*

lemma *norm_bound_Basis_le*: $b \in Basis \Longrightarrow norm\ x \leq e \Longrightarrow |inner\ x\ b| \leq e$
by (*metis Basis_le_norm order_trans*)

lemma *norm_bound_Basis_lt*: $b \in Basis \Longrightarrow norm\ x < e \Longrightarrow |inner\ x\ b| < e$
by (*metis Basis_le_norm le_less_trans*)

lemma *norm_le_l1*: $norm\ x \leq (\sum b{\in}Basis.\ |inner\ x\ b|)$
apply (*subst euclidean_representation*[*of x, symmetric*])
apply (*rule order_trans*[*OF norm_sum*])
apply (*auto intro*!: *sum_mono*)
done

lemma *sum_norm_allsubsets_bound*:
fixes $f :: 'a \Rightarrow 'n{::}euclidean\_space$
assumes *fP*: *finite P*
and *fPs*: $\bigwedge Q.\ Q \subseteq P \Longrightarrow norm\ (sum\ f\ Q) \leq e$
shows $(\sum x{\in}P.\ norm\ (f\ x)) \leq 2 * real\ DIM('n) * e$
proof −
have $(\sum x{\in}P.\ norm\ (f\ x)) \leq (\sum x{\in}P.\ \sum b{\in}Basis.\ |inner\ (f\ x)\ b|)$
by (*rule sum_mono*) (*rule norm_le_l1*)
also have $(\sum x{\in}P.\ \sum b{\in}Basis.\ |inner\ (f\ x)\ b|) = (\sum b{\in}Basis.\ \sum x{\in}P.\ |inner\ (f\ x)\ b|)$
by (*rule sum.swap*)
also have $\ldots \leq of\_nat\ (card\ (Basis :: 'n\ set)) * (2 * e)$
proof (*rule sum_bounded_above*)
fix $i :: 'n$
assume *i*: $i \in Basis$
have $norm\ (\sum x{\in}P.\ |inner\ (f\ x)\ i|) \leq$
$norm\ (inner\ (\sum x{\in}P \cap -\ \{x.\ inner\ (f\ x)\ i < 0\}.\ f\ x)\ i) + norm\ (inner\ (\sum x{\in}P \cap \{x.\ inner\ (f\ x)\ i < 0\}.\ f\ x)\ i)$
by (*simp add*: *abs_real_def sum.If_cases*[*OF fP*] *sum_negf norm_triangle_ineq4 inner_sum_left*
del: *real_norm_def*)
also have $\ldots \leq e + e$
unfolding *real_norm_def*
by (*intro add_mono norm_bound_Basis_le i fPs*) *auto*
finally show $(\sum x{\in}P.\ |inner\ (f\ x)\ i|) \leq 2{*}e$ **by** *simp*
qed
also have $\ldots = 2 * real\ DIM('n) * e$ **by** *simp*
finally show *?thesis* **.**
qed

1.4.3 Subclass relationships

instance *euclidean_space* ⊆ *perfect_space*

proof
fix $x :: 'a$ **show** $\neg$ *open* $\{x\}$
proof
assume *open* $\{x\}$
then obtain e **where** $0 < e$ **and** e: $\forall y.$ *dist* $y\ x < e \longrightarrow y = x$
unfolding *open_dist* **by** *fast*
define y **where** $y = x + scaleR\ (e/2)\ (SOME\ b.\ b \in Basis)$
have [*simp*]: $(SOME\ b.\ b \in Basis) \in Basis$
by (*rule someI_ex*) (*auto simp*: *ex_in_conv*)
from ‹$0 < e$› **have** $y \neq x$
unfolding *y_def* **by** (*auto intro*!: *nonzero_Basis*)
from ‹$0 < e$› **have** *dist* $y\ x < e$
unfolding *y_def* **by** (*simp add*: *dist_norm*)
from ‹$y \neq x$› **and** ‹*dist* $y\ x < e$› **show** *False*
using e **by** *simp*
qed
qed

1.4.4 Class instances

Type *real*

instantiation *real* :: *euclidean_space*
begin

definition
[*simp*]: *Basis* = $\{1::real\}$

instance
by *standard auto*

end

lemma *DIM_real*[*simp*]: *DIM*(*real*) = *1*
by *simp*

Type *complex*

instantiation *complex* :: *euclidean_space*
begin

definition *Basis_complex_def*: *Basis* = $\{1, \mathrm{i}\}$

instance
by *standard* (*auto simp add*: *Basis_complex_def intro*: *complex_eqI split*: *if_split_asm*)

end

lemma *DIM_complex*[*simp*]: *DIM*(*complex*) = *2*
unfolding *Basis_complex_def* **by** *simp*

lemma *complex_Basis_1* [*iff*]: (*1*::*complex*) ∈ *Basis*
by (*simp add*: *Basis_complex_def*)

lemma *complex_Basis_i* [*iff*]: i ∈ *Basis*
by (*simp add*: *Basis_complex_def*)

Type *'a* × *'b*

instantiation *prod* :: (*real_inner*, *real_inner*) *real_inner*
begin

definition *inner_prod_def*:
inner x y = *inner* (*fst x*) (*fst y*) + *inner* (*snd x*) (*snd y*)

lemma *inner_Pair* [*simp*]: *inner* (*a*, *b*) (*c*, *d*) = *inner a c* + *inner b d*
unfolding *inner_prod_def* **by** *simp*

instance
proof
fix *r* :: *real*
fix *x y z* :: *'a*::*real_inner* × *'b*::*real_inner*
show *inner x y* = *inner y x*
unfolding *inner_prod_def*
by (*simp add*: *inner_commute*)
show *inner* (*x* + *y*) *z* = *inner x z* + *inner y z*
unfolding *inner_prod_def*
by (*simp add*: *inner_add_left*)
show *inner* (*scaleR r x*) *y* = *r* ∗ *inner x y*
unfolding *inner_prod_def*
by (*simp add*: *distrib_left*)
show *0* ≤ *inner x x*
unfolding *inner_prod_def*
by (*intro add_nonneg_nonneg inner_ge_zero*)
show *inner x x* = *0* ⟷ *x* = *0*
unfolding *inner_prod_def prod_eq_iff*
by (*simp add*: *add_nonneg_eq_0_iff*)
show *norm x* = *sqrt* (*inner x x*)
unfolding *norm_prod_def inner_prod_def*
by (*simp add*: *power2_norm_eq_inner*)
qed

end

lemma *inner_Pair_0*: *inner x* (*0*, *b*) = *inner* (*snd x*) *b inner x* (*a*, *0*) = *inner* (*fst x*) *a*
by (*cases x*, *simp*)+

instantiation *prod* :: (*euclidean_space*, *euclidean_space*) *euclidean_space*

begin

definition
Basis = (λ*u*. (*u*, *0*)) ‘ *Basis* ∪ (λ*v*. (*0*, *v*)) ‘ *Basis*

lemma *sum_Basis_prod_eq*:
fixes *f*::('*a*∗'*b*)⇒('*a*∗'*b*)
shows *sum f Basis* = *sum* (λ*i*. *f* (*i*, *0*)) *Basis* + *sum* (λ*i*. *f* (*0*, *i*)) *Basis*
proof −
have *inj_on* (λ*u*. (*u*::'*a*, *0*::'*b*)) *Basis inj_on* (λ*u*. (*0*::'*a*, *u*::'*b*)) *Basis*
by (*auto intro*!: *inj_onI Pair_inject*)
thus *?thesis*
unfolding *Basis_prod_def*
by (*subst sum.union_disjoint*) (*auto simp*: *Basis_prod_def sum.reindex*)
qed

instance proof
show (*Basis* :: ('*a* × '*b*) *set*) ≠ {}
unfolding *Basis_prod_def* **by** *simp*
next
show *finite* (*Basis* :: ('*a* × '*b*) *set*)
unfolding *Basis_prod_def* **by** *simp*
next
fix *u v* :: '*a* × '*b*
assume *u* ∈ *Basis* **and** *v* ∈ *Basis*
thus *inner u v* = (*if u* = *v then 1 else 0*)
unfolding *Basis_prod_def inner_prod_def*
by (*auto simp add*: *inner_Basis split*: *if_split_asm*)
next
fix *x* :: '*a* × '*b*
show (∀ *u*∈*Basis*. *inner x u* = *0*) ⟷ *x* = *0*
unfolding *Basis_prod_def ball_Un ball_simps*
by (*simp add*: *inner_prod_def prod_eq_iff euclidean_all_zero_iff*)
qed

lemma *DIM_prod*[*simp*]: *DIM*('*a* × '*b*) = *DIM*('*a*) + *DIM*('*b*)
unfolding *Basis_prod_def*
by (*subst card_Un_disjoint*) (*auto intro*!: *card_image arg_cong2*[**where** *f*=(+)] *inj_onI*)

end

1.4.5 Locale instances

lemma *finite_dimensional_vector_space_euclidean*:
finite_dimensional_vector_space (∗*R*) *Basis*
proof *unfold_locales*
show *finite* (*Basis*::'*a set*) **by** (*metis finite_Basis*)
show *real_vector.independent* (*Basis*::'*a set*)

```
    unfolding dependent_def dependent_raw_def[symmetric]
    apply (subst span_finite)
    apply simp
    apply clarify
    apply (drule_tac f=inner a in arg_cong)
    apply (simp add: inner_Basis inner_sum_right eq_commute)
    done
  show module.span (*R) Basis = UNIV
    unfolding span_finite [OF finite_Basis] span_raw_def[symmetric]
    by (auto intro!: euclidean_representation[symmetric])
qed

interpretation eucl?: finite_dimensional_vector_space scaleR :: real => 'a =>
'a::euclidean_space Basis
  rewrites module.dependent (*R) = dependent
    and module.representation (*R) = representation
    and module.subspace (*R) = subspace
    and module.span (*R) = span
    and vector_space.extend_basis (*R) = extend_basis
    and vector_space.dim (*R) = dim
    and Vector_Spaces.linear (*R) (*R) = linear
    and Vector_Spaces.linear (*) (*R) = linear
    and finite_dimensional_vector_space.dimension Basis = DIM('a)
    and dimension = DIM('a)
  by (auto simp add: dependent_raw_def representation_raw_def
       subspace_raw_def span_raw_def extend_basis_raw_def dim_raw_def lin-
ear_def
      real_scaleR_def[abs_def]
      finite_dimensional_vector_space.dimension_def
      intro!: finite_dimensional_vector_space.dimension_def
      finite_dimensional_vector_space_euclidean)

interpretation eucl?: finite_dimensional_vector_space_pair_1
  scaleR::real⇒'a::euclidean_space⇒'a Basis
  scaleR::real⇒'b::real_vector ⇒ 'b
  by unfold_locales

interpretation eucl?: finite_dimensional_vector_space_prod scaleR scaleR Basis
Basis
  rewrites Basis_pair = Basis
    and module_prod.scale (*R) (*R) = (scaleR::_⇒_⇒('a × 'b))
proof −
  show finite_dimensional_vector_space_prod (*R) (*R) Basis Basis
    by unfold_locales
   interpret finite_dimensional_vector_space_prod (*R) (*R) Basis::'a set Ba-
sis::'b set
    by fact
  show Basis_pair = Basis
    unfolding Basis_pair_def Basis_prod_def by auto
```

show *module_prod.scale* $(*_R)$ $(*_R)$ = *scaleR*
by (*fact module_prod_scale_eq_scaleR*)
qed

end

1.5 Elementary Linear Algebra on Euclidean Spaces

theory *Linear_Algebra*
imports
Euclidean_Space
HOL−Library.Infinite_Set
begin

lemma *linear_simps*:
assumes *bounded_linear f*
shows
$f\ (a + b) = f\ a + f\ b$
$f\ (a - b) = f\ a - f\ b$
$f\ 0 = 0$
$f\ (-\ a) = -\ f\ a$
$f\ (s *_R v) = s *_R (f\ v)$
proof −
interpret *f*: *bounded_linear f* **by** *fact*
show $f\ (a + b) = f\ a + f\ b$ **by** (*rule f.add*)
show $f\ (a - b) = f\ a - f\ b$ **by** (*rule f.diff*)
show $f\ 0 = 0$ **by** (*rule f.zero*)
show $f\ (-\ a) = -\ f\ a$ **by** (*rule f.neg*)
show $f\ (s *_R v) = s *_R (f\ v)$ **by** (*rule f.scale*)
qed

lemma *finite_Atleast_Atmost_nat*[*simp*]: *finite* $\{f\ x\ |x.\ x \in (UNIV::'a::finite\ set)\}$
using *finite finite_image_set* **by** *blast*

lemma *substdbasis_expansion_unique*:
includes *inner_syntax*
assumes *d*: $d \subseteq Basis$
shows $(\sum i{\in}d.\ f\ i *_R i) = (x::'a::euclidean\_space) \longleftrightarrow$
$(\forall i{\in}Basis.\ (i \in d \longrightarrow f\ i = x \cdot i) \wedge (i \notin d \longrightarrow x \cdot i = 0))$
proof −
have ∗: $\bigwedge x\ a\ b\ P.\ x * (if\ P\ then\ a\ else\ b) = (if\ P\ then\ x * a\ else\ x * b)$
by *auto*
have ∗∗: *finite d*
by (*auto intro*: *finite_subset*[*OF assms*])
have ∗∗∗: $\bigwedge i.\ i \in Basis \Longrightarrow (\sum i{\in}d.\ f\ i *_R i) \cdot i = (\sum x{\in}d.\ if\ x = i\ then\ f\ x\ else\ 0)$
using *d*
by (*auto intro*!: *sum.cong simp*: *inner_Basis inner_sum_left*)
show *?thesis*

unfolding *euclidean_eq_iff*[**where** *'a='a*] **by** (*auto simp*: *sum.delta*[*OF* **] ***])
qed

lemma *independent_substdbasis*: $d \subseteq Basis \Longrightarrow independent\ d$
by (*rule independent_mono*[*OF independent_Basis*])

lemma *subset_translation_eq* [*simp*]:
fixes *a* :: *'a*::*real_vector* **shows** $(+)\ a\ `\ s \subseteq (+)\ a\ `\ t \longleftrightarrow s \subseteq t$
by *auto*

lemma *translate_inj_on*:
fixes *A* :: *'a*::*ab_group_add set*
shows *inj_on* $(\lambda x.\ a + x)\ A$
unfolding *inj_on_def* **by** *auto*

lemma *translation_assoc*:
fixes *a b* :: *'a*::*ab_group_add*
shows $(\lambda x.\ b + x)\ `\ ((\lambda x.\ a + x)\ `\ S) = (\lambda x.\ (a + b) + x)\ `\ S$
by *auto*

lemma *translation_invert*:
fixes *a* :: *'a*::*ab_group_add*
assumes $(\lambda x.\ a + x)\ `\ A = (\lambda x.\ a + x)\ `\ B$
shows $A = B$
using *assms translation_assoc* **by** *fastforce*

lemma *translation_galois*:
fixes *a* :: *'a*::*ab_group_add*
shows $T = ((\lambda x.\ a + x)\ `\ S) \longleftrightarrow S = ((\lambda x.\ (-\ a) + x)\ `\ T)$
by (*metis add.right_inverse group_cancel.rule0 translation_invert translation_assoc*)

lemma *translation_inverse_subset*:
assumes $((\lambda x.\ -\ a + x)\ `\ V) \leq (S :: 'n::ab\_group\_add\ set)$
shows $V \leq ((\lambda x.\ a + x)\ `\ S)$
by (*metis assms subset_image_iff translation_galois*)

1.5.1 More interesting properties of the norm

unbundle *inner_syntax*

Equality of vectors in terms of $(\cdot)$ products.

lemma *linear_componentwise*:
fixes *f*:: *'a*::*euclidean_space* $\Rightarrow$ *'b*::*real_inner*
assumes *lf*: *linear f*
shows $(f\ x) \cdot j = (\sum i{\in}Basis.\ (x{\cdot}i) * (f\ i{\cdot}j))$ (**is** *?lhs* = *?rhs*)
proof −
interpret *linear f* **by** *fact*
have *?rhs* = $(\sum i{\in}Basis.\ (x{\cdot}i) *_R (f\ i)){\cdot}j$

by (*simp add*: *inner_sum_left*)
then show *?thesis*
by (*simp add*: *euclidean_representation sum*[*symmetric*] *scale*[*symmetric*])
qed

lemma *vector_eq*: $x = y \longleftrightarrow x \cdot x = x \cdot y \wedge y \cdot y = x \cdot x$
by (*metis* (*no_types*, *opaque_lifting*) *inner_commute inner_diff_right inner_eq_zero_iff right_minus_eq*)

lemma *norm_triangle_half_r*:
$norm\ (y - x1) < e/2 \Longrightarrow norm\ (y - x2) < e/2 \Longrightarrow norm\ (x1 - x2) < e$
using *dist_triangle_half_r* **unfolding** *dist_norm*[*symmetric*] **by** *auto*

lemma *norm_triangle_half_l*:
assumes $norm\ (x - y) < e/2$ **and** $norm\ (x' - y) < e/2$
shows $norm\ (x - x') < e$
by (*metis assms dist_norm dist_triangle_half_l*)

lemma *abs_triangle_half_r*:
fixes $y :: 'a$::*linordered_field*
shows $abs\ (y - x1) < e/2 \Longrightarrow abs\ (y - x2) < e/2 \Longrightarrow abs\ (x1 - x2) < e$
by *linarith*

lemma *abs_triangle_half_l*:
fixes $y :: 'a$::*linordered_field*
assumes $abs\ (x - y) < e/2$ **and** $abs\ (x' - y) < e/2$
shows $abs\ (x - x') < e$
using *assms* **by** *linarith*

lemma *sum_clauses*:
shows $sum\ f\ \{\} = 0$
and *finite* $S \Longrightarrow sum\ f\ (insert\ x\ S) = (if\ x \in S\ then\ sum\ f\ S\ else\ f\ x + sum\ f\ S)$
by (*auto simp add*: *insert_absorb*)

lemma *vector_eq_ldot*: $(\forall x.\ x \cdot y = x \cdot z) \longleftrightarrow y = z$ **and** *vector_eq_rdot*: $(\forall z.\ x \cdot z = y \cdot z) \longleftrightarrow x = y$
by (*metis inner_commute vector_eq*)+

1.5.2 Substandard Basis

lemma *ex_card*:
assumes $n \leq card\ A$
shows $\exists S \subseteq A.\ card\ S = n$
by (*meson assms obtain_subset_with_card_n*)

lemma *subspace_substandard*: *subspace* $\{x::'a::euclidean\_space.\ (\forall i \in Basis.\ P\ i \longrightarrow x \cdot i = 0)\}$
by (*auto simp*: *subspace_def inner_add_left*)

lemma *dim_substandard*:
assumes *d*: $d \subseteq Basis$
shows *dim* $\{x::'a::euclidean\_space.\ \forall i{\in}Basis.\ i \notin d \longrightarrow x{\cdot}i = 0\} = card\ d$ (**is** *dim ?A = _*)
proof (*rule dim_unique*)
from *d* **show** $d \subseteq ?A$
by (*auto simp*: *inner_Basis*)
from *d* **show** *independent d*
by (*rule independent_mono* [*OF independent_Basis*])
have $x \in span\ d$ **if** $\forall i{\in}Basis.\ i \notin d \longrightarrow x \cdot i = 0$ **for** *x*
proof −
have *finite d*
by (*rule finite_subset* [*OF d finite_Basis*])
then have $(\sum i{\in}d.\ (x \cdot i) *_R i) \in span\ d$
by (*simp add*: *span_sum span_clauses*)
also have $(\sum i{\in}d.\ (x \cdot i) *_R i) = (\sum i{\in}Basis.\ (x \cdot i) *_R i)$
by (*rule sum.mono_neutral_cong_left* [*OF finite_Basis d*]) (*auto simp*: *that*)
finally show $x \in span\ d$
by (*simp only*: *euclidean_representation*)
qed
then show $?A \subseteq span\ d$ **by** *auto*
qed *simp*

1.5.3 Orthogonality

definition (**in** *real_inner*) *orthogonal x y* $\longleftrightarrow x \cdot y = 0$

context *real_inner*
begin

lemma *orthogonal_self*: *orthogonal x x* $\longleftrightarrow x = 0$
by (*simp add*: *orthogonal_def*)

lemma *orthogonal_clauses*:
orthogonal a 0
orthogonal a x $\Longrightarrow$ *orthogonal a* $(c *_R x)$
orthogonal a x $\Longrightarrow$ *orthogonal a* $(-x)$
orthogonal a x $\Longrightarrow$ *orthogonal a y* $\Longrightarrow$ *orthogonal a* $(x + y)$
orthogonal a x $\Longrightarrow$ *orthogonal a y* $\Longrightarrow$ *orthogonal a* $(x - y)$
orthogonal 0 a
orthogonal x a $\Longrightarrow$ *orthogonal* $(c *_R x)$ *a*
orthogonal x a $\Longrightarrow$ *orthogonal* $(-x)$ *a*
orthogonal x a $\Longrightarrow$ *orthogonal y a* $\Longrightarrow$ *orthogonal* $(x + y)$ *a*
orthogonal x a $\Longrightarrow$ *orthogonal y a* $\Longrightarrow$ *orthogonal* $(x - y)$ *a*
unfolding *orthogonal_def inner_add inner_diff* **by** *auto*

end

lemma *orthogonal_commute*: *orthogonal x y* $\longleftrightarrow$ *orthogonal y x*
by (*simp add*: *orthogonal_def inner_commute*)

lemma *orthogonal_scaleR* [*simp*]: $c \neq 0 \Longrightarrow$ *orthogonal* $(c *_R x)$ = *orthogonal x*
by (*rule ext*) (*simp add*: *orthogonal_def*)

lemma *pairwise_ortho_scaleR*:
pairwise $(\lambda i\ j.$ *orthogonal* $(f\ i)\ (g\ j))\ B$
$\Longrightarrow$ *pairwise* $(\lambda i\ j.$ *orthogonal* $(a\ i *_R f\ i)\ (a\ j *_R g\ j))\ B$
by (*auto simp*: *pairwise_def orthogonal_clauses*)

lemma *orthogonal_rvsum*:
$[\![$*finite s*; $\bigwedge y.\ y \in s \Longrightarrow$ *orthogonal x* $(f\ y)]\!] \Longrightarrow$ *orthogonal x* $(sum\ f\ s)$
by (*induction s rule*: *finite_induct*) (*auto simp*: *orthogonal_clauses*)

lemma *orthogonal_lvsum*:
$[\![$*finite s*; $\bigwedge x.\ x \in s \Longrightarrow$ *orthogonal* $(f\ x)\ y]\!] \Longrightarrow$ *orthogonal* $(sum\ f\ s)\ y$
by (*induction s rule*: *finite_induct*) (*auto simp*: *orthogonal_clauses*)

lemma *norm_add_Pythagorean*:
assumes *orthogonal a b*
shows $(norm\ (a + b))^2 = (norm\ a)^2 + (norm\ b)^2$
proof −
from *assms* **have** $(a - (0 - b)) \cdot (a - (0 - b)) = a \cdot a - (0 - b \cdot b)$
by (*simp add*: *algebra_simps orthogonal_def inner_commute*)
then show *?thesis*
by (*simp add*: *power2_norm_eq_inner*)
qed

lemma *norm_sum_Pythagorean*:
assumes *finite I pairwise* $(\lambda i\ j.$ *orthogonal* $(f\ i)\ (f\ j))\ I$
shows $(norm\ (sum\ f\ I))^2 = (\sum i \in I.\ (norm\ (f\ i))^2)$
using *assms*
proof (*induction I rule*: *finite_induct*)
case *empty* **then show** *?case* **by** *simp*
next
case (*insert x I*)
then have *orthogonal* $(f\ x)\ (sum\ f\ I)$
by (*metis pairwise_insert orthogonal_rvsum*)
with *insert* **show** *?case*
by (*simp add*: *pairwise_insert norm_add_Pythagorean*)
qed

1.5.4 Orthogonality of a transformation

definition *orthogonal_transformation f* $\longleftrightarrow$ *linear f* $\wedge$ $(\forall v\ w.\ f\ v \cdot f\ w = v \cdot w)$

lemma *orthogonal_transformation*:
orthogonal_transformation f $\longleftrightarrow$ *linear f* $\wedge$ $(\forall v.\ norm\ (f\ v) = norm\ v)$

by (*smt* (*verit*, *ccfv_threshold*) *dot_norm linear_add norm_eq_sqrt_inner orthogonal_transformation_def*)

lemma *orthogonal_transformation_id* [*simp*]: *orthogonal_transformation* (λ*x*. *x*)
by (*simp add*: *linear_iff orthogonal_transformation_def*)

lemma *orthogonal_orthogonal_transformation*:
orthogonal_transformation f ⟹ *orthogonal* (*f x*) (*f y*) ⟷ *orthogonal x y*
by (*simp add*: *orthogonal_def orthogonal_transformation_def*)

lemma *orthogonal_transformation_compose*:
⟦*orthogonal_transformation f*; *orthogonal_transformation g*⟧ ⟹ *orthogonal_transformation*(*f* ∘ *g*)
by (*auto simp*: *orthogonal_transformation_def linear_compose*)

lemma *orthogonal_transformation_neg*:
orthogonal_transformation(λ*x*. −(*f x*)) ⟷ *orthogonal_transformation f*
by (*auto simp*: *orthogonal_transformation_def dest*: *linear_compose_neg*)

lemma *orthogonal_transformation_scaleR*: *orthogonal_transformation f* ⟹ *f* (*c* *$_R$ *v*) = *c* *$_R$ *f v*
by (*simp add*: *linear_iff orthogonal_transformation_def*)

lemma *orthogonal_transformation_linear*:
orthogonal_transformation f ⟹ *linear f*
by (*simp add*: *orthogonal_transformation_def*)

lemma *orthogonal_transformation_inj*:
orthogonal_transformation f ⟹ *inj f*
unfolding *orthogonal_transformation_def inj_on_def*
by (*metis vector_eq*)

lemma *orthogonal_transformation_surj*:
orthogonal_transformation f ⟹ *surj f*
for *f* :: *'a*::*euclidean_space* ⇒ *'a*::*euclidean_space*
by (*simp add*: *linear_injective_imp_surjective orthogonal_transformation_inj orthogonal_transformation_linear*)

lemma *orthogonal_transformation_bij*:
orthogonal_transformation f ⟹ *bij f*
for *f* :: *'a*::*euclidean_space* ⇒ *'a*::*euclidean_space*
by (*simp add*: *bij_def orthogonal_transformation_inj orthogonal_transformation_surj*)

lemma *orthogonal_transformation_inv*:
orthogonal_transformation f ⟹ *orthogonal_transformation* (*inv f*)
for *f* :: *'a*::*euclidean_space* ⇒ *'a*::*euclidean_space*
by (*metis* (*no_types*, *opaque_lifting*) *bijection.inv_right bijection_def inj_linear_imp_inv_linear orthogonal_transformation orthogonal_transformation_bij orthogonal_transformation_inj*)

lemma *orthogonal_transformation_norm*:
orthogonal_transformation f $\Longrightarrow$ *norm* (*f x*) = *norm x*
by (*metis orthogonal_transformation*)

1.5.5 Bilinear functions

definition
bilinear :: ('*a*::*real_vector* $\Rightarrow$ '*b*::*real_vector* $\Rightarrow$ '*c*::*real_vector*) $\Rightarrow$ *bool* **where**
bilinear f $\longleftrightarrow$ ($\forall$ *x*. *linear* (λ*y*. *f x y*)) $\wedge$ ($\forall$ *y*. *linear* (λ*x*. *f x y*))

lemma *bilinear_ladd*: *bilinear h* $\Longrightarrow$ *h* (*x* + *y*) *z* = *h x z* + *h y z*
by (*simp add*: *bilinear_def linear_iff*)

lemma *bilinear_radd*: *bilinear h* $\Longrightarrow$ *h x* (*y* + *z*) = *h x y* + *h x z*
by (*simp add*: *bilinear_def linear_iff*)

lemma *bilinear_times*:
fixes *c*::'*a*::*real_algebra* **shows** *bilinear* (λ*x y*::'*a*. *x*∗*y*)
by (*auto simp*: *bilinear_def distrib_left distrib_right intro*!: *linearI*)

lemma *bilinear_lmul*: *bilinear h* $\Longrightarrow$ *h* (*c* $*_R$ *x*) *y* = *c* $*_R$ *h x y*
by (*simp add*: *bilinear_def linear_iff*)

lemma *bilinear_rmul*: *bilinear h* $\Longrightarrow$ *h x* (*c* $*_R$ *y*) = *c* $*_R$ *h x y*
by (*simp add*: *bilinear_def linear_iff*)

lemma *bilinear_lneg*: *bilinear h* $\Longrightarrow$ *h* (− *x*) *y* = − *h x y*
by (*drule bilinear_lmul* [*of* _ − *1*]) *simp*

lemma *bilinear_rneg*: *bilinear h* $\Longrightarrow$ *h x* (− *y*) = − *h x y*
by (*drule bilinear_rmul* [*of* _ _ − *1*]) *simp*

lemma (**in** *ab_group_add*) *eq_add_iff*: *x* = *x* + *y* $\longleftrightarrow$ *y* = *0*
using *add_left_imp_eq*[*of x y 0*] **by** *auto*

lemma *bilinear_lzero*:
assumes *bilinear h*
shows *h 0 x* = *0*
using *bilinear_ladd* [*OF assms*, *of 0 0 x*] **by** (*simp add*: *eq_add_iff field_simps*)

lemma *bilinear_rzero*:
assumes *bilinear h*
shows *h x 0* = *0*
using *bilinear_radd* [*OF assms*, *of x 0 0*] **by** (*simp add*: *eq_add_iff field_simps*)

lemma *bilinear_lsub*: *bilinear h* $\Longrightarrow$ *h* (*x* − *y*) *z* = *h x z* − *h y z*
using *bilinear_ladd* [*of h x* − *y*] **by** (*simp add*: *bilinear_lneg*)

lemma *bilinear_rsub*: *bilinear h* $\Longrightarrow$ *h z* (*x* − *y*) = *h z x* − *h z y*

using *bilinear_radd* [*of* h _ $x - y$] **by** (*simp add*: *bilinear_rneg*)

lemma *bilinear_sum*:
assumes *bilinear* h
shows h (*sum* f S) (*sum* g T) = *sum* ($\lambda(i,j)$. h (f i) (g j)) ($S \times T$)
proof −
interpret l: *linear* λx. h x y **for** y **using** *assms* **by** (*simp add*: *bilinear_def*)
interpret r: *linear* λy. h x y **for** x **using** *assms* **by** (*simp add*: *bilinear_def*)
have h (*sum* f S) (*sum* g T) = *sum* (λx. h (f x) (*sum* g T)) S
by (*simp add*: *l.sum*)
also have ... = *sum* (λx. *sum* (λy. h (f x) (g y)) T) S
by (*rule sum.cong*) (*simp_all add*: *r.sum*)
finally show *?thesis*
unfolding *sum.cartesian_product* .
qed

1.5.6 Adjoints

definition *adjoint* :: (('*a*::*real_inner*) ⇒ ('*b*::*real_inner*)) ⇒ '*b* ⇒ '*a* **where**
adjoint f = (*SOME* f'. $\forall x\ y$. $f\ x \cdot y = x \cdot f'\ y$)

lemma *adjoint_unique*:
assumes $\forall x\ y$. *inner* (f x) y = *inner* x (g y)
shows *adjoint* f = g
unfolding *adjoint_def*
proof (*rule some_equality*)
show $\forall x\ y$. *inner* (f x) y = *inner* x (g y)
by (*rule assms*)
next
fix h
assume $\forall x\ y$. *inner* (f x) y = *inner* x (h y)
then show h = g
by (*metis assms ext vector_eq_ldot*)
qed

TODO: The following lemmas about adjoints should hold for any Hilbert space (i.e. complete inner product space). (see https://en.wikipedia.org/wiki/Hermitian_adjoint)

lemma *adjoint_works*:
fixes f :: '*n*::*euclidean_space* ⇒ '*m*::*euclidean_space*
assumes *lf*: *linear* f
shows $x \cdot$ *adjoint* f y = f $x \cdot y$
proof −
interpret *linear* f **by** *fact*
have $\forall y$. $\exists w$. $\forall x$. $f\ x \cdot y = x \cdot w$
proof (*intro allI exI*)
fix y :: '*m* **and** x
let *?w* = ($\sum$ *i*∈*Basis*. (f $i \cdot y$) $*_R$ i) :: '*n*
have $f\ x \cdot y = f$ ($\sum$ *i*∈*Basis*. ($x \cdot i$) $*_R$ i) $\cdot\ y$

by (*simp add*: *euclidean_representation*)
also have ... = ($\sum i \in Basis$. $(x \cdot i) *_R f\ i) \cdot y$
by (*simp add*: *sum scale*)
finally show $f\ x \cdot y = x \cdot ?w$
by (*simp add*: *inner_sum_left inner_sum_right mult.commute*)
qed
then show *?thesis*
unfolding *adjoint_def choice_iff*
by (*intro someI2_ex*[**where** $Q=\lambda f'.\ x \cdot f'\ y = f\ x \cdot y$]) *auto*
qed

lemma *adjoint_clauses*:
fixes $f :: 'n::euclidean\_space \Rightarrow 'm::euclidean\_space$
assumes *lf*: *linear f*
shows $x \cdot adjoint\ f\ y = f\ x \cdot y$
and $adjoint\ f\ y \cdot x = y \cdot f\ x$
by (*simp_all add*: *adjoint_works*[*OF lf*] *inner_commute*)

lemma *adjoint_linear*:
fixes $f :: 'n::euclidean\_space \Rightarrow 'm::euclidean\_space$
assumes *lf*: *linear f*
shows *linear* (*adjoint f*)
by (*simp add*: *lf linear_iff euclidean_eq_iff*[**where** $'a='n$] *euclidean_eq_iff*[**where** $'a='m$]
adjoint_clauses[*OF lf*] *inner_distrib*)

lemma *adjoint_adjoint*:
fixes $f :: 'n::euclidean\_space \Rightarrow 'm::euclidean\_space$
assumes *lf*: *linear f*
shows *adjoint* (*adjoint f*) = *f*
by (*rule adjoint_unique*, *simp add*: *adjoint_clauses* [*OF lf*])

1.5.7 Euclidean Spaces as Typeclass

lemma *independent_Basis*: *independent Basis*
by (*rule independent_Basis*)

lemma *span_Basis* [*simp*]: *span Basis* = *UNIV*
by (*rule span_Basis*)

lemma *in_span_Basis*: $x \in span\ Basis$
unfolding *span_Basis* **..**

1.5.8 Linearity and Bilinearity continued

lemma *linear_bounded*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::real\_normed\_vector$
assumes *lf*: *linear f*
shows $\exists B.\ \forall x.\ norm\ (f\ x) \leq B * norm\ x$
proof

interpret *linear f* **by** *fact*
let *?B* = $\sum$ *b*∈*Basis*. *norm* (*f b*)
show ∀ *x*. *norm* (*f x*) ≤ *?B* ∗ *norm x*
proof
fix *x* :: *'a*
let *?g* = λ*b*. (*x* · *b*) ∗$_R$ *f b*
have *norm* (*f x*) = *norm* (*f* ($\sum$ *b*∈*Basis*. (*x* · *b*) ∗$_R$ *b*))
unfolding *euclidean_representation* **..**
also have ... = *norm* (*sum ?g Basis*)
by (*simp add*: *sum scale*)
finally have *th0*: *norm* (*f x*) = *norm* (*sum ?g Basis*) **.**
have *th*: *norm* (*?g i*) ≤ *norm* (*f i*) ∗ *norm x* **if** *i* ∈ *Basis* **for** *i*
proof −
from *Basis_le_norm*[*OF that*, *of x*]
show *norm* (*?g i*) ≤ *norm* (*f i*) ∗ *norm x*
unfolding *norm_scaleR* **by** (*metis mult.commute mult_left_mono norm_ge_zero*)
qed
from *sum_norm_le*[*of* _ *?g*, *OF th*]
show *norm* (*f x*) ≤ *?B* ∗ *norm x*
by (*simp add*: *sum_distrib_right th0*)
qed
qed

lemma *linear_conv_bounded_linear*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*real_normed_vector*
shows *linear f* ⟷ *bounded_linear f*
by (*metis mult.commute bounded_linear_axioms.intro bounded_linear_def linear_bounded*)

lemmas *linear_linear* = *linear_conv_bounded_linear*[*symmetric*]

lemma *inj_linear_imp_inv_bounded_linear*:
fixes *f*::*'a*::*euclidean_space* ⇒ *'a*
shows ⟦*bounded_linear f*; *inj f*⟧ ⟹ *bounded_linear* (*inv f*)
by (*simp add*: *inj_linear_imp_inv_linear linear_linear*)

lemma *linear_bounded_pos*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*real_normed_vector*
assumes *lf*: *linear f*
obtains *B* **where** *B* > *0* ⋀*x*. *norm* (*f x*) ≤ *B* ∗ *norm x*
by (*metis bounded_linear.pos_bounded lf linear_linear mult.commute*)

lemma *linear_invertible_bounded_below_pos*:
fixes *f* :: *'a*::*real_normed_vector* ⇒ *'b*::*euclidean_space*
assumes *linear f linear g* **and** *gf*: *g* ∘ *f* = *id*
obtains *B* **where** *B* > *0* ⋀*x*. *B* ∗ *norm x* ≤ *norm*(*f x*)
proof −
obtain *B* **where** *B* > *0* **and** *B*: ⋀*x*. *norm* (*g x*) ≤ *B* ∗ *norm x*
using *linear_bounded_pos* [*OF* ‹*linear g*›] **by** *blast*

```
  show thesis
  proof
    show 0 < 1/B
      by (simp add: ‹B > 0›)
    show 1/B * norm x ≤ norm (f x) for x
      by (smt (verit, ccfv_SIG) B ‹0 < B› gf comp_apply divide_inverse id_apply
inverse_eq_divide
            less_divide_eq mult.commute)
  qed
qed

lemma linear_inj_bounded_below_pos:
  fixes f :: 'a::real_normed_vector ⇒ 'b::euclidean_space
  assumes linear f inj f
  obtains B where B > 0 ⋀x. B * norm x ≤ norm(f x)
  using linear_injective_left_inverse [OF assms]
    linear_invertible_bounded_below_pos assms by blast

lemma bounded_linearI':
  fixes f ::'a::euclidean_space ⇒ 'b::real_normed_vector
  assumes ⋀x y. f (x + y) = f x + f y
    and ⋀c x. f (c *R x) = c *R f x
  shows bounded_linear f
  using assms linearI linear_conv_bounded_linear by blast

lemma bilinear_bounded:
  fixes h :: 'm::euclidean_space ⇒ 'n::euclidean_space ⇒ 'k::real_normed_vector
  assumes bh: bilinear h
  shows ∃ B. ∀ x y. norm (h x y) ≤ B * norm x * norm y
proof (clarify intro!: exI[of _ ∑ i∈Basis. ∑ j∈Basis. norm (h i j)])
  fix x :: 'm
  fix y :: 'n
  have norm (h x y) = norm (h (sum (λi. (x • i) *R i) Basis) (sum (λi. (y • i)
*R i) Basis))
    by (simp add: euclidean_representation)
  also have ... = norm (sum (λ (i,j). h ((x • i) *R i) ((y • j) *R j)) (Basis ×
Basis))
    unfolding bilinear_sum[OF bh] ..
  finally have th: norm (h x y) = ... .
  have ⋀i j. ⟦i ∈ Basis; j ∈ Basis⟧
        ⟹ |x • i| * (|y • j| * norm (h i j)) ≤ norm x * (norm y * norm (h i j))
    by (auto simp add: zero_le_mult_iff Basis_le_norm mult_mono)
  then show norm (h x y) ≤ (∑ i∈Basis. ∑ j∈Basis. norm (h i j)) * norm x *
norm y
    unfolding sum_distrib_right th sum.cartesian_product
    by (clarsimp simp add: bilinear_rmul[OF bh] bilinear_lmul[OF bh]
      field_simps simp del: scaleR_scaleR intro!: sum_norm_le)
qed
```

lemma *bilinear_conv_bounded_bilinear*:
fixes *h* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space* ⇒ *'c*::*real_normed_vector*
shows *bilinear h* ⟷ *bounded_bilinear h*
proof
assume *bilinear h*
show *bounded_bilinear h*
proof
fix *x y z*
show *h* (*x* + *y*) *z* = *h x z* + *h y z*
using ‹*bilinear h*› **unfolding** *bilinear_def linear_iff* **by** *simp*
next
fix *x y z*
show *h x* (*y* + *z*) = *h x y* + *h x z*
using ‹*bilinear h*› **unfolding** *bilinear_def linear_iff* **by** *simp*
next
show *h* (*scaleR r x*) *y* = *scaleR r* (*h x y*) *h x* (*scaleR r y*) = *scaleR r* (*h x y*)
for *r x y*
using ‹*bilinear h*› **unfolding** *bilinear_def linear_iff*
by *simp_all*
next
have ∃ *B*. ∀ *x y*. *norm* (*h x y*) ≤ *B* ∗ *norm x* ∗ *norm y*
using ‹*bilinear h*› **by** (*rule bilinear_bounded*)
then show ∃ *K*. ∀ *x y*. *norm* (*h x y*) ≤ *norm x* ∗ *norm y* ∗ *K*
by (*simp add*: *ac_simps*)
qed
next
assume *bounded_bilinear h*
then interpret *h*: *bounded_bilinear h* **.**
show *bilinear h*
unfolding *bilinear_def linear_conv_bounded_linear*
using *h.bounded_linear_left h.bounded_linear_right* **by** *simp*
qed

lemma *bilinear_bounded_pos*:
fixes *h* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space* ⇒ *'c*::*real_normed_vector*
assumes *bh*: *bilinear h*
shows ∃ *B* > *0*. ∀ *x y*. *norm* (*h x y*) ≤ *B* ∗ *norm x* ∗ *norm y*
by (*metis mult.assoc bh bilinear_conv_bounded_bilinear bounded_bilinear.pos_bounded mult.commute*)

lemma *bounded_linear_imp_has_derivative*:
bounded_linear f ⟹ (*f has_derivative f*) *net*
by (*auto simp add*: *has_derivative_def linear_diff linear_linear linear_def dest*: *bounded_linear.linear*)

lemma *linear_imp_has_derivative*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*real_normed_vector*
shows *linear f* ⟹ (*f has_derivative f*) *net*
by (*simp add*: *bounded_linear_imp_has_derivative linear_conv_bounded_linear*)

lemma *bounded_linear_imp_differentiable*: *bounded_linear f* $\Longrightarrow$ *f differentiable net*
using *bounded_linear_imp_has_derivative differentiable_def* **by** *blast*

lemma *linear_imp_differentiable*:
fixes *f* :: *'a*::*euclidean_space* $\Rightarrow$ *'b*::*real_normed_vector*
shows *linear f* $\Longrightarrow$ *f differentiable net*
by (*metis linear_imp_has_derivative differentiable_def*)

lemma *of_real_differentiable* [*simp*,*derivative_intros*]: *of_real differentiable F*
by (*simp add*: *bounded_linear_imp_differentiable bounded_linear_of_real*)

1.5.9 We continue

lemma *independent_bound*:
fixes *S* :: *'a*::*euclidean_space set*
shows *independent S* $\Longrightarrow$ *finite S* $\wedge$ *card S* $\leq$ *DIM*(*'a*)
by (*metis dim_subset_UNIV finiteI_independent dim_span_eq_card_independent*)

lemmas *independent_imp_finite* = *finiteI_independent*

corollary *independent_card_le*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *independent S*
shows *card S* $\leq$ *DIM*(*'a*)
using *assms independent_bound* **by** *auto*

lemma *dependent_biggerset*:
fixes *S* :: *'a*::*euclidean_space set*
shows (*finite S* $\Longrightarrow$ *card S* $>$ *DIM*(*'a*)) $\Longrightarrow$ *dependent S*
by (*metis independent_bound not_less*)

Picking an orthogonal replacement for a spanning set.

lemma *vector_sub_project_orthogonal*:
fixes *b x* :: *'a*::*euclidean_space*
shows *b* $\cdot$ (*x* $-$ ((*b* $\cdot$ *x*) / (*b* $\cdot$ *b*)) $*_R$ *b*) = *0*
unfolding *inner_simps* **by** *auto*

lemma *pairwise_orthogonal_insert*:
assumes *pairwise orthogonal S*
and $\bigwedge y$. *y* $\in$ *S* $\Longrightarrow$ *orthogonal x y*
shows *pairwise orthogonal* (*insert x S*)
using *assms* **by** (*auto simp*: *pairwise_def orthogonal_commute*)

lemma *basis_orthogonal*:
fixes *B* :: *'a*::*real_inner set*
assumes *fB*: *finite B*
shows $\exists$ *C*. *finite C* $\wedge$ *card C* $\leq$ *card B* $\wedge$ *span C* = *span B* $\wedge$ *pairwise orthogonal*

```
C
  (is  ∃ C. ?P B C)
  using fB
proof (induct rule: finite_induct)
  case empty
  then show ?case
    using pairwise_empty by blast
next
  case (insert a B)
  note fB = ‹finite B› and aB = ‹a ∉ B›
  from ‹∃ C. finite C ∧ card C ≤ card B ∧ span C = span B ∧ pairwise orthogonal
C›
  obtain C where C: finite C card C ≤ card B
    span C = span B pairwise orthogonal C by blast
  let ?a = a − sum (λx. (x · a / (x · x)) *R x) C
  let ?C = insert ?a C
  from C(1) have fC: finite ?C
    by simp
  have cC: card ?C ≤ card (insert a B)
    using C aB card_insert_if local.insert(1) by fastforce
  {
    fix x k
    have th0: ⋀(a::'a) b c. a − (b − c) = c + (a − b)
      by (simp add: field_simps)
    have x − k *R (a − (∑ x∈C. (x · a / (x · x)) *R x)) ∈ span C ⟷ x − k *R
a ∈ span C
      unfolding scaleR_right_diff_distrib th0
      by (intro span_add_eq span_scale span_sum span_base)
  }
  then have SC: span ?C = span (insert a B)
    unfolding set_eq_iff span_breakdown_eq C(3)[symmetric] by auto
  {
    fix y
    assume yC: y ∈ C
    then have Cy: C = insert y (C − {y})
      by blast
    have fth: finite (C − {y})
      using C by simp
    have y ≠ 0 ⟹ ∀ x∈C − {y}. x · a ∗ (x · y) / (x · x) = 0
      using ‹pairwise orthogonal C›
        by (metis Cy DiffE div_0 insertCI mult_zero_right orthogonal_def pair-
wise_insert)
    then have orthogonal ?a y
      unfolding orthogonal_def
      unfolding inner_diff inner_sum_left right_minus_eq
      unfolding sum.remove [OF ‹finite C› ‹y ∈ C›]
      by (auto simp add: sum.neutral inner_commute[of y a])
  }
  with ‹pairwise orthogonal C› have CPO: pairwise orthogonal ?C
```

```
    by (rule pairwise_orthogonal_insert)
  from fC cC SC CPO have ?P (insert a B) ?C
    by blast
  then show ?case by blast
qed

lemma orthogonal_basis_exists:
  fixes V :: ('a::euclidean_space) set
  shows ∃ B. independent B ∧ B ⊆ span V ∧ V ⊆ span B ∧ (card B = dim V)
∧ pairwise orthogonal B
proof −
  from basis_exists[of V] obtain B where
    B: B ⊆ V independent B V ⊆ span B card B = dim V
    by force
  from B have fB: finite B card B = dim V
    using independent_bound by auto
  from basis_orthogonal[OF fB(1)] obtain C where
    C: finite C card C ≤ card B span C = span B pairwise orthogonal C
    by blast
  from C B have CSV: C ⊆ span V
    by (metis span_superset span_mono subset_trans)
  from span_mono[OF B(3)] C have SVC: span V ⊆ span C
    by (simp add: span_span)
  from C fB have card C ≤ dim V
    by simp
  moreover have dim V ≤ card C
    using span_card_ge_dim[OF CSV SVC C(1)]
    by simp
  ultimately have card C = dim V
    using C(1) by simp
  with C B CSV show ?thesis
    by (metis SVC card_eq_dim dim_span)
qed
```

Low-dimensional subset is in a hyperplane (weak orthogonal complement).

```
lemma span_not_univ_orthogonal:
  fixes S :: 'a::euclidean_space set
  assumes sU: span S ≠ UNIV
  shows ∃ a::'a. a ≠ 0 ∧ (∀ x ∈ span S. a · x = 0)
proof −
  from sU obtain a where a: a ∉ span S
    by blast
  from orthogonal_basis_exists obtain B where
    B: independent B B ⊆ span S S ⊆ span B card B = dim S pairwise orthogonal
B
    by blast
  from B have fB: finite B card B = dim S
    using independent_bound by auto
  have sSB: span S = span B
```

```
    by (simp add: B span_eq)
  let ?a = a − sum (λb. (a · b / (b · b)) *R b) B
  have sum (λb. (a · b / (b · b)) *R b) B ∈ span S
    by (simp add: sSB span_base span_mul span_sum)
  with a have a0:?a ≠ 0
    by auto
  have ?a · x = 0 if x∈span B for x
  proof (rule span_induct [OF that])
    show subspace {x. ?a · x = 0}
      by (auto simp add: subspace_def inner_add)
  next
    {
      fix x
      assume x: x ∈ B
      from x have B': B = insert x (B − {x})
        by blast
      have fth: finite (B − {x})
        using fB by simp
      have (∑ b∈B − {x}. a · b ∗ (b · x) / (b · b)) = 0 if x ≠ 0
      by (smt (verit) B' B(5) DiffD2 divide_eq_0_iff inner_real_def inner_zero_right
insertCI orthogonal_def pairwise_insert sum.neutral)
      then have ?a · x = 0
        apply (subst B')
        using fB fth
        unfolding sum_clauses(2)[OF fth]
        by (auto simp add: inner_add_left inner_diff_left inner_sum_left)
    }
    then show ?a · x = 0 if x ∈ B for x
      using that by blast
    qed
  with a0 sSB show ?thesis
    by blast
qed

lemma span_not_univ_subset_hyperplane:
  fixes S :: 'a::euclidean_space set
  assumes SU: span S ≠ UNIV
  shows ∃ a. a ≠0 ∧ span S ⊆ {x. a · x = 0}
  using span_not_univ_orthogonal[OF SU] by auto

lemma lowdim_subset_hyperplane:
  fixes S :: 'a::euclidean_space set
  assumes d: dim S < DIM('a)
  shows ∃ a::'a. a ≠ 0 ∧ span S ⊆ {x. a · x = 0}
  using d dim_eq_full nless_le span_not_univ_subset_hyperplane by blast

lemma linear_eq_stdbasis:
  fixes f :: 'a::euclidean_space ⇒ _
  assumes lf: linear f
```

and *lg*: *linear g*
and *fg*: $\bigwedge b.\ b \in Basis \Longrightarrow f\ b = g\ b$
shows $f = g$
using *linear_eq_on_span*[*OF lf lg*, *of Basis*] *fg* **by** *auto*

Similar results for bilinear functions.

lemma *bilinear_eq*:
assumes *bf*: *bilinear f*
and *bg*: *bilinear g*
and *SB*: $S \subseteq span\ B$
and *TC*: $T \subseteq span\ C$
and $x{\in}S\ y{\in}T$
and *fg*: $\bigwedge x\ y.\ [\![x \in B; y{\in}\ C]\!] \Longrightarrow f\ x\ y = g\ x\ y$
shows $f\ x\ y = g\ x\ y$
proof −
let *?P* = $\{x.\ \forall y{\in}\ span\ C.\ f\ x\ y = g\ x\ y\}$
from *bf bg* **have** *sp*: *subspace ?P*
unfolding *bilinear_def linear_iff subspace_def bf bg*
by (*auto simp add*: *span_zero bilinear_lzero*[*OF bf*] *bilinear_lzero*[*OF bg*]
span_add Ball_def
intro: *bilinear_ladd*[*OF bf*])
have *sfg*: $\bigwedge x.\ x \in B \Longrightarrow subspace\ \{a.\ f\ x\ a = g\ x\ a\}$
by (*auto simp*: *subspace_def bf bg bilinear_rzero bilinear_radd bilinear_rmul*)
have $\forall y{\in}\ span\ C.\ f\ x\ y = g\ x\ y$ **if** $x \in span\ B$ **for** *x*
using *span_induct* [*OF that sp*] *fg sfg span_induct* **by** *blast*
then show *?thesis*
using *SB TC assms* **by** *auto*
qed

lemma *bilinear_eq_stdbasis*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space* ⇒ _
assumes *bf*: *bilinear f*
and *bg*: *bilinear g*
and *fg*: $\bigwedge i\ j.\ i \in Basis \Longrightarrow j \in Basis \Longrightarrow f\ i\ j = g\ i\ j$
shows $f = g$
using *bilinear_eq*[*OF bf bg equalityD2*[*OF span_Basis*] *equalityD2*[*OF span_Basis*]]
fg **by** *blast*

1.5.10 Infinity norm

definition *infnorm* (*x*::*'a*::*euclidean_space*) = *Sup* $\{|x \cdot b|\ |b.\ b \in Basis\}$

lemma *infnorm_set_image*:
fixes *x* :: *'a*::*euclidean_space*
shows $\{|x \cdot i|\ |i.\ i \in Basis\} = (\lambda i.\ |x \cdot i|)\ `\ Basis$
by *blast*

lemma *infnorm_Max*:
fixes *x* :: *'a*::*euclidean_space*

shows *infnorm x = Max ((λi. |x · i|) ' Basis)*
by (*simp add*: *infnorm_def infnorm_set_image cSup_eq_Max*)

lemma *infnorm_set_lemma*:
fixes *x* :: *'a*::*euclidean_space*
shows *finite* {|*x* · *i*| |*i*. *i* ∈ *Basis*}
and {|*x* · *i*| |*i*. *i* ∈ *Basis*} ≠ {}
unfolding *infnorm_set_image* **by** *auto*

lemma *infnorm_pos_le*:
fixes *x* :: *'a*::*euclidean_space*
shows *0* ≤ *infnorm x*
by (*simp add*: *infnorm_Max Max_ge_iff ex_in_conv*)

lemma *infnorm_triangle*:
fixes *x* :: *'a*::*euclidean_space*
shows *infnorm* (*x* + *y*) ≤ *infnorm x* + *infnorm y*
proof −
have ∗: ⋀*a b c d* :: *real*. |*a*| ≤ *c* ⟹ |*b*| ≤ *d* ⟹ |*a* + *b*| ≤ *c* + *d*
by *simp*
show *?thesis*
by (*auto simp*: *infnorm_Max inner_add_left intro*!: ∗)
qed

lemma *infnorm_eq_0*:
fixes *x* :: *'a*::*euclidean_space*
shows *infnorm x = 0* ⟷ *x = 0*
proof −
have *infnorm x* ≤ *0* ⟷ *x = 0*
unfolding *infnorm_Max* **by** (*simp add*: *euclidean_all_zero_iff*)
then show *?thesis*
using *infnorm_pos_le*[*of x*] **by** *simp*
qed

lemma *infnorm_0*: *infnorm 0 = 0*
by (*simp add*: *infnorm_eq_0*)

lemma *infnorm_neg*: *infnorm* (− *x*) = *infnorm x*
unfolding *infnorm_def* **by** *simp*

lemma *infnorm_sub*: *infnorm* (*x* − *y*) = *infnorm* (*y* − *x*)
by (*metis infnorm_neg minus_diff_eq*)

lemma *absdiff_infnorm*: |*infnorm x* − *infnorm y*| ≤ *infnorm* (*x* − *y*)
by (*smt* (*verit*, *del_insts*) *diff_add_cancel infnorm_sub infnorm_triangle*)

lemma *real_abs_infnorm*: |*infnorm x*| = *infnorm x*
using *infnorm_pos_le*[*of x*] **by** *arith*

lemma *Basis_le_infnorm*:
fixes $x :: 'a::euclidean\_space$
shows $b \in Basis \Longrightarrow |x \cdot b| \leq infnorm\ x$
by (*simp add*: *infnorm_Max*)

lemma *infnorm_mul*: $infnorm\ (a *_R x) = |a| * infnorm\ x$
unfolding *infnorm_Max*
proof (*safe intro*!: *Max_eqI*)
let $?B = (\lambda i.\ |x \cdot i|)\ `\ Basis$
{ fix $b :: 'a$
assume $b \in Basis$
then show $|a *_R x \cdot b| \leq |a| * Max\ ?B$
by (*simp add*: *abs_mult mult_left_mono*)
next
from *Max_in*[*of* *?B*] **obtain** b **where** $b \in Basis\ Max\ ?B = |x \cdot b|$
by (*auto simp del*: *Max_in*)
then show $|a| * Max\ ((\lambda i.\ |x \cdot i|)\ `\ Basis) \in (\lambda i.\ |a *_R x \cdot i|)\ `\ Basis$
by (*intro image_eqI*[**where** $x{=}b$]) (*auto simp*: *abs_mult*)
}
qed *simp*

lemma *infnorm_mul_lemma*: $infnorm\ (a *_R x) \leq |a| * infnorm\ x$
unfolding *infnorm_mul* **..**

lemma *infnorm_pos_lt*: $infnorm\ x > 0 \longleftrightarrow x \neq 0$
using *infnorm_pos_le*[*of* *x*] *infnorm_eq_0*[*of* *x*] **by** *arith*

Prove that it differs only up to a bound from Euclidean norm.

lemma *infnorm_le_norm*: $infnorm\ x \leq norm\ x$
by (*simp add*: *Basis_le_norm infnorm_Max*)

lemma *norm_le_infnorm*:
fixes $x :: 'a::euclidean\_space$
shows $norm\ x \leq sqrt\ DIM('a) * infnorm\ x$
unfolding *norm_eq_sqrt_inner id_def*
proof (*rule real_le_lsqrt*)
show $sqrt\ DIM('a) * infnorm\ x \geq 0$
by (*simp add*: *zero_le_mult_iff infnorm_pos_le*)
have $x \cdot x \leq (\sum b{\in}Basis.\ x \cdot b * (x \cdot b))$
by (*metis euclidean_inner order_refl*)
also have $\ldots \leq DIM('a) * |infnorm\ x|^2$
by (*rule sum_bounded_above*) (*metis Basis_le_infnorm abs_le_square_iff power2_eq_square real_abs_infnorm*)
also have $\ldots \leq (sqrt\ DIM('a) * infnorm\ x)^2$
by (*simp add*: *power_mult_distrib*)
finally show $x \cdot x \leq (sqrt\ DIM('a) * infnorm\ x)^2$ **.**
qed

lemma *tendsto_infnorm* [*tendsto_intros*]:

assumes $(f \longrightarrow a)$ *F*
shows $((\lambda x.\ infnorm\ (f\ x)) \longrightarrow infnorm\ a)$ *F*
proof (*rule tendsto_compose* [*OF LIM_I assms*])
fix *r* :: *real*
assume $r > 0$
then show $\exists\, s{>}0.\ \forall x.\ x \neq a \wedge norm\ (x - a) < s \longrightarrow norm\ (infnorm\ x - infnorm\ a) < r$
by (*metis real_norm_def le_less_trans absdiff_infnorm infnorm_le_norm*)
qed

Equality in Cauchy-Schwarz and triangle inequalities.

lemma *norm_cauchy_schwarz_eq*: $x \cdot y = norm\ x * norm\ y \longleftrightarrow norm\ x *_R y = norm\ y *_R x$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof (*cases x=0*)
case *True*
then show *?thesis*
by *auto*
next
case *False*
from *inner_eq_zero_iff*[*of norm y* $*_R$ *x* $-$ *norm x* $*_R$ *y*]
have *?rhs* $\longleftrightarrow$
(*norm y* $*$ (*norm y* $*$ *norm x* $*$ *norm x* $-$ *norm x* $*$ ($x \cdot y$)) $-$
norm x $*$ (*norm y* $*$ ($y \cdot x$) $-$ *norm x* $*$ *norm y* $*$ *norm y*) $= 0$)
using *False* **unfolding** *inner_simps*
by (*auto simp add*: *power2_norm_eq_inner*[*symmetric*] *power2_eq_square inner_commute field_simps*)
also have ... $\longleftrightarrow$ ($2 *$ *norm x* $*$ *norm y* $*$ (*norm x* $*$ *norm y* $- x \cdot y$) $= 0$)
using *False* **by** (*simp add*: *field_simps inner_commute*)
also have ... $\longleftrightarrow$ *?lhs*
using *False* **by** *auto*
finally show *?thesis* **by** *metis*
qed

lemma *norm_cauchy_schwarz_abs_eq*:
$|x \cdot y| = norm\ x * norm\ y \longleftrightarrow$
norm x $*_R$ *y* $=$ *norm y* $*_R$ *x* $\vee$ *norm x* $*_R$ *y* $= -$ *norm y* $*_R$ *x*
using *norm_cauchy_schwarz_eq* [*symmetric*, *of x y*]
using *norm_cauchy_schwarz_eq* [*symmetric*, *of* $-x$ *y*] *Cauchy_Schwarz_ineq2* [*of x y*]
by *auto*

lemma *norm_triangle_eq*:
fixes *x y* :: *'a*::*real_inner*
shows *norm* $(x + y) =$ *norm x* $+$ *norm y* $\longleftrightarrow$ *norm x* $*_R$ *y* $=$ *norm y* $*_R$ *x*
proof (*cases* $x = 0 \vee y = 0$)
case *True*
then show *?thesis*
by *force*

next
case *False*
then have *n*: *norm x > 0 norm y > 0*
by *auto*
have *norm (x + y) = norm x + norm y ⟷ (norm (x + y))² = (norm x + norm y)²*
by *simp*
also have *... ⟷ norm x *R y = norm y *R x*
by (*smt* (*verit*, *best*) *dot_norm inner_real_def inner_simps norm_cauchy_schwarz_eq power2_eq_square*)
finally show *?thesis* **.**
qed

lemma *dist_triangle_eq*:
fixes *x y z* :: *'a*::*real_inner*
shows *dist x z = dist x y + dist y z ⟷*
*norm (x − y) *R (y − z) = norm (y − z) *R (x − y)*
by (*metis* (*no_types*, *lifting*) *add_diff_eq diff_add_cancel dist_norm norm_triangle_eq*)

1.5.11 Collinearity

definition *collinear* :: *'a*::*real_vector set ⇒ bool*
where *collinear S ⟷ (∃ u. ∀ x ∈ S. ∀ y ∈ S. ∃ c. x − y = c *R u)*

lemma *collinear_alt*:
*collinear S ⟷ (∃ u v. ∀ x ∈ S. ∃ c. x = u + c *R v)* (**is** *?lhs = ?rhs*)
proof
assume *?lhs*
then show *?rhs*
unfolding *collinear_def* **by** (*metis add.commute diff_add_cancel*)
next
assume *?rhs*
then obtain *u v* **where** *: ⋀x. x ∈ S ⟹ ∃ c. x = u + c *R v*
by *auto*
have *∃ c. x − y = c *R v* **if** *x ∈ S y ∈ S* **for** *x y*
by (*metis *[OF ‹x ∈ S›] *[OF ‹y ∈ S›] scaleR_left.diff add_diff_cancel_left*)
then show *?lhs*
using *collinear_def* **by** *blast*
qed

lemma *collinear*:
fixes *S* :: *'a*::{*perfect_space*,*real_vector*} *set*
shows *collinear S ⟷ (∃ u. u ≠ 0 ∧ (∀ x ∈ S. ∀ y ∈ S. ∃ c. x − y = c *R u))*
proof −
have *∃ v. v ≠ 0 ∧ (∀ x∈S. ∀ y∈S. ∃ c. x − y = c *R v)*
if *∀ x∈S. ∀ y∈S. ∃ c. x − y = c *R u u=0* **for** *u*
proof −
have *∀ x∈S. ∀ y∈S. x = y*
using *that* **by** *auto*

```
    moreover
    obtain v::'a where v ≠ 0
      using UNIV_not_singleton [of 0] by auto
    ultimately have ∀ x∈S. ∀ y∈S. ∃ c. x − y = c *R v
      by auto
    then show ?thesis
      using ‹v ≠ 0› by blast
  qed
  then show ?thesis
    by (metis collinear_def)
qed

lemma collinear_subset: ⟦collinear T; S ⊆ T⟧ ⟹ collinear S
  by (meson collinear_def subsetCE)

lemma collinear_empty [iff]: collinear {}
  by (simp add: collinear_def)

lemma collinear_sing [iff]: collinear {x}
  by (simp add: collinear_def)

lemma collinear_2 [iff]: collinear {x, y}
 by (simp add: collinear_def) (metis minus_diff_eq scaleR_left.minus scaleR_one)

lemma collinear_lemma: collinear {0, x, y} ⟷ x = 0 ∨ y = 0 ∨ (∃ c. y = c
*R x)
  (is ?lhs ⟷ ?rhs)
proof (cases x = 0 ∨ y = 0)
  case True
  then show ?thesis
    by (auto simp: insert_commute)
next
  case False
  show ?thesis
  proof
    assume h: ?lhs
    then obtain u where u: ∀ x∈ {0,x,y}. ∀ y∈ {0,x,y}. ∃ c. x − y = c *R u
      unfolding collinear_def by blast
    from u[rule_format, of x 0] u[rule_format, of y 0]
    obtain cx and cy where
      cx: x = cx *R u and cy: y = cy *R u
      by auto
    from cx cy False have cx0: cx ≠ 0 and cy0: cy ≠ 0 by auto
    let ?d = cy / cx
    from cx cy cx0 have y = ?d *R x
      by simp
    then show ?rhs using False by blast
  next
    assume h: ?rhs
```

```
    then obtain c where c: y = c *R x
      using False by blast
    show ?lhs
      apply (simp add: collinear_def c)
       by (metis (mono_tags, lifting) scaleR_left.minus scaleR_left_diff_distrib
scaleR_one)
  qed
qed

lemma collinear_iff_Reals: collinear {0::complex,w,z} ⟷ z/w ∈ ℝ
proof
  show z/w ∈ ℝ ⟹ collinear {0,w,z}
   by (metis Reals_cases collinear_lemma nonzero_divide_eq_eq scaleR_conv_of_real)
qed (auto simp: collinear_lemma scaleR_conv_of_real)

lemma collinear_scaleR_iff: collinear {0, α *R w, β *R z} ⟷ collinear {0,w,z}
∨ α=0 ∨ β=0
  (is ?lhs = ?rhs)
proof (cases α=0 ∨ β=0)
  case False
  then have (∃ c. β *R z = (c * α) *R w) = (∃ c. z = c *R w)
    by (metis mult.commute scaleR_scaleR vector_fraction_eq_iff)
  then show ?thesis
    by (auto simp add: collinear_lemma)
qed (auto simp: collinear_lemma)

lemma norm_cauchy_schwarz_equal: |x · y| = norm x * norm y ⟷ collinear
{0, x, y}
proof (cases x=0)
  case True
  then show ?thesis
    by (auto simp: insert_commute)
next
  case False
  then have nnz: norm x ≠ 0
    by auto
  show ?thesis
  proof
    assume |x · y| = norm x * norm y
    then show collinear {0, x, y}
      unfolding norm_cauchy_schwarz_abs_eq collinear_lemma
      by (meson eq_vector_fraction_iff nnz)
  next
    assume collinear {0, x, y}
    with False show |x · y| = norm x * norm y
      unfolding norm_cauchy_schwarz_abs_eq collinear_lemma  by (auto simp:
abs_if)
  qed
qed
```

lemma *norm_triangle_eq_imp_collinear*:
fixes *x y* :: *'a*::*real_inner*
assumes *norm* (*x* + *y*) = *norm x* + *norm y*
shows *collinear*{*0*,*x*,*y*}
using *assms norm_cauchy_schwarz_abs_eq norm_cauchy_schwarz_equal norm_triangle_eq*

by *blast*

1.5.12 Properties of special hyperplanes

lemma *subspace_hyperplane*: *subspace* {*x*. *a* · *x* = *0*}
by (*simp add*: *subspace_def inner_right_distrib*)

lemma *subspace_hyperplane2*: *subspace* {*x*. *x* · *a* = *0*}
by (*simp add*: *inner_commute inner_right_distrib subspace_def*)

lemma *special_hyperplane_span*:
fixes *S* :: *'n*::*euclidean_space set*
assumes *k* ∈ *Basis*
shows {*x*. *k* · *x* = *0*} = *span* (*Basis* − {*k*})
proof −
have ∗: *x* ∈ *span* (*Basis* − {*k*}) **if** *k* · *x* = *0* **for** *x*
proof −
have *x* = (∑ *b*∈*Basis*. (*x* · *b*) ∗*R b*)
by (*simp add*: *euclidean_representation*)
also have ... = (∑ *b* ∈ *Basis* − {*k*}. (*x* · *b*) ∗*R b*)
by (*auto simp*: *sum.remove* [*of* _ *k*] *inner_commute assms that*)
finally have *x* = (∑ *b*∈*Basis* − {*k*}. (*x* · *b*) ∗*R b*) .
then show *?thesis*
by (*simp add*: *span_finite*)
qed
show *?thesis*
apply (*rule span_subspace* [*symmetric*])
using *assms*
apply (*auto simp*: *inner_not_same_Basis intro*: ∗ *subspace_hyperplane*)
done
qed

lemma *dim_special_hyperplane*:
fixes *k* :: *'n*::*euclidean_space*
shows *k* ∈ *Basis* ⟹ *dim* {*x*. *k* · *x* = *0*} = *DIM*(*'n*) − *1*
by (*metis Diff_subset card_Diff_singleton indep_card_eq_dim_span independent_substdbasis special_hyperplane_span*)

proposition *dim_hyperplane*:
fixes *a* :: *'a*::*euclidean_space*
assumes *a* ≠ *0*
shows *dim* {*x*. *a* · *x* = *0*} = *DIM*(*'a*) − *1*

proof −
have *span0*: *span* {*x*. *a* · *x* = *0*} = {*x*. *a* · *x* = *0*}
by (*rule span_unique*) (*auto simp*: *subspace_hyperplane*)
then obtain *B* **where** *independent B*
and *Bsub*: *B* ⊆ {*x*. *a* · *x* = *0*}
and *subspB*: {*x*. *a* · *x* = *0*} ⊆ *span B*
and *card0*: (*card B* = *dim* {*x*. *a* · *x* = *0*})
and *ortho*: *pairwise orthogonal B*
using *orthogonal_basis_exists* **by** *metis*
with *assms* **have** *a* ∉ *span B*
by (*metis* (*mono_tags*, *lifting*) *span_eq inner_eq_zero_iff mem_Collect_eq span0*)
then have *ind*: *independent* (*insert a B*)
by (*simp add*: ‹*independent B*› *independent_insert*)
have *finite B*
using ‹*independent B*› *independent_bound* **by** *blast*
have *UNIV* ⊆ *span* (*insert a B*)
proof fix *y*::*'a*
obtain *r z* **where** *y* = *r* ∗$_R$ *a* + *z a* · *z* = *0*
by (*metis add.commute diff_add_cancel vector_sub_project_orthogonal*)
then show *y* ∈ *span* (*insert a B*)
by (*metis* (*mono_tags*, *lifting*) *Bsub add_diff_cancel_left'*
mem_Collect_eq span0 span_breakdown_eq span_eq subspB)
qed
then have *DIM*(*'a*) = *dim*(*insert a B*)
by (*metis independent_Basis span_Basis dim_eq_card top.extremum_uniqueI*)
then show *?thesis*
by (*metis One_nat_def* ‹*a* ∉ *span B*› ‹*finite B*› *card0 card_insert_disjoint*
diff_Suc_Suc diff_zero dim_eq_card_independent ind span_base)
qed

lemma *lowdim_eq_hyperplane*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *dim S* = *DIM*(*'a*) − *1*
obtains *a* **where** *a* ≠ *0* **and** *span S* = {*x*. *a* · *x* = *0*}
proof −
obtain *b* **where** *b*: *b* ≠ *0 span S* ⊆ {*a*. *b* · *a* = *0*}
by (*metis DIM_positive assms diff_less zero_less_one lowdim_subset_hyperplane*)
then show *?thesis*
by (*metis assms dim_hyperplane dim_span dim_subset subspace_dim_equal subspace_hyperplane subspace_span that*)
qed

lemma *dim_eq_hyperplane*:
fixes *S* :: *'n*::*euclidean_space set*
shows *dim S* = *DIM*(*'n*) − *1* ⟷ (∃ *a*. *a* ≠ *0* ∧ *span S* = {*x*. *a* · *x* = *0*})
by (*metis One_nat_def dim_hyperplane dim_span lowdim_eq_hyperplane*)

1.5.13 Orthogonal bases and Gram-Schmidt process

lemma *pairwise_orthogonal_independent*:
assumes *pairwise orthogonal S* **and** *0* $\notin$ *S*
shows *independent S*
proof −
have *0*: $\bigwedge x\ y.\ [\![x \neq y;\ x \in S;\ y \in S]\!] \Longrightarrow x \cdot y = 0$
using *assms* **by** (*simp add*: *pairwise_def orthogonal_def*)
have *False* **if** $a \in S$ **and** *a*: $a \in$ *span* $(S - \{a\})$ **for** *a*
proof −
obtain *T U* **where** $T \subseteq S - \{a\}$ $a = (\sum v{\in}T.\ U\ v *_R v)$
using *a* **by** (*force simp*: *span_explicit*)
then have $a \cdot a = a \cdot (\sum v{\in}T.\ U\ v *_R v)$
by *simp*
also have $\ldots = 0$
apply (*simp add*: *inner_sum_right*)
by (*smt* (*verit*) *0 DiffE* ‹$T \subseteq S - \{a\}$› *in_mono insertCI mult_not_zero sum.neutral that*(*1*))
finally show *?thesis*
using ‹*0* $\notin$ *S*› ‹$a \in S$› **by** *auto*
qed
then show *?thesis*
by (*force simp*: *dependent_def*)
qed

lemma *pairwise_orthogonal_imp_finite*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *pairwise orthogonal S*
shows *finite S*
by (*metis Set.set_insert assms finite_insert independent_bound pairwise_insert*
pairwise_orthogonal_independent)

lemma *subspace_orthogonal_to_vector*: *subspace* $\{y.\ orthogonal\ x\ y\}$
by (*simp add*: *subspace_def orthogonal_clauses*)

lemma *subspace_orthogonal_to_vectors*: *subspace* $\{y.\ \forall x \in S.\ orthogonal\ x\ y\}$
by (*simp add*: *subspace_def orthogonal_clauses*)

lemma *orthogonal_to_span*:
assumes *a*: $a \in$ *span S* **and** *x*: $\bigwedge y.\ y \in S \Longrightarrow$ *orthogonal x y*
shows *orthogonal x a*
by (*metis a orthogonal_clauses*(*1*,*2*,*4*)
span_induct_alt x)

proposition *Gram_Schmidt_step*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *S*: *pairwise orthogonal S* **and** *x*: $x \in$ *span S*
shows *orthogonal x* $(a - (\sum b{\in}S.\ (b \cdot a\ /\ (b \cdot b)) *_R b))$
proof −

```
  have finite S
    by (simp add: S pairwise_orthogonal_imp_finite)
  have orthogonal (a − (∑ b∈S. (b · a / (b · b)) *R b)) x
      if x ∈ S for x
  proof −
    have a · x = (∑ y∈S. if y = x then y · a else 0)
      by (simp add: ‹finite S› inner_commute that)
    also have ... = (∑ b∈S. b · a * (b · x) / (b · b))
      apply (rule sum.cong [OF refl], simp)
      by (meson S orthogonal_def pairwise_def that)
   finally show ?thesis
     by (simp add: orthogonal_def algebra_simps inner_sum_left)
  qed
  then show ?thesis
    using orthogonal_to_span orthogonal_commute x by blast
qed


lemma orthogonal_extension_aux:
  fixes S :: 'a::euclidean_space set
  assumes finite T finite S pairwise orthogonal S
    shows ∃ U. pairwise orthogonal (S ∪ U) ∧ span (S ∪ U) = span (S ∪ T)
using assms
proof (induction arbitrary: S)
  case empty then show ?case
    by simp (metis sup_bot_right)
next
  case (insert a T)
  have 0: ⋀x y. ⟦x ≠ y; x ∈ S; y ∈ S⟧ ⟹ x · y = 0
    using insert by (simp add: pairwise_def orthogonal_def)
  define a' where a' = a − (∑ b∈S. (b · a / (b · b)) *R b)
  obtain U where orthU: pairwise orthogonal (S ∪ insert a' U)
            and spanU: span (insert a' S ∪ U) = span (insert a' S ∪ T)
    by (rule exE [OF insert.IH [of insert a' S]])
      (auto simp: Gram_Schmidt_step a'_def insert.prems orthogonal_commute
        pairwise_orthogonal_insert span_clauses)
  have orthS: ⋀x. x ∈ S ⟹ a' · x = 0
    using Gram_Schmidt_step a'_def insert.prems orthogonal_commute orthogo-
nal_def span_base by blast
  have span (S ∪ insert a' U) = span (insert a' (S ∪ T))
    using spanU by simp
  also have ... = span (insert a (S ∪ T))
   by (simp add: a'_def span_neg span_sum span_base span_mul eq_span_insert_eq)
  also have ... = span (S ∪ insert a T)
    by simp
  finally show ?case
    using orthU by blast
qed
```

proposition *orthogonal_extension*:
fixes S :: *'a*::*euclidean_space set*
assumes S: *pairwise orthogonal S*
obtains U **where** *pairwise orthogonal* $(S \cup U)$ *span* $(S \cup U) = span\ (S \cup T)$
proof −
obtain B **where** *finite B span B* = *span T*
using *basis_subspace_exists* [*of span T*] *subspace_span* **by** *metis*
with *orthogonal_extension_aux* [*of B S*]
obtain U **where** *pairwise orthogonal* $(S \cup U)$ *span* $(S \cup U) = span\ (S \cup B)$
using *assms pairwise_orthogonal_imp_finite* **by** *auto*
with ‹*span B* = *span T*› **show** *?thesis*
by (*rule_tac U*=*U* **in** *that*) (*auto simp*: *span_Un*)
qed

corollary *orthogonal_extension_strong*:
fixes S :: *'a*::*euclidean_space set*
assumes S: *pairwise orthogonal S*
obtains U **where** $U \cap (insert\ 0\ S) = \{\}$ *pairwise orthogonal* $(S \cup U)$
span $(S \cup U) = span\ (S \cup T)$
proof −
obtain U **where** U: *pairwise orthogonal* $(S \cup U)$ *span* $(S \cup U) = span\ (S \cup T)$
using *orthogonal_extension assms* **by** *blast*
moreover have *pairwise orthogonal* $(S \cup (U - insert\ 0\ S))$
by (*smt* (*verit*, *best*) *Un_Diff_Int Un_iff U pairwise_def*)
ultimately show *?thesis*
by (*metis Diff_disjoint Un_Diff_cancel Un_insert_left inf_commute span_insert_0 that*)
qed

1.5.14 Decomposing a vector into parts in orthogonal subspaces

existence of orthonormal basis for a subspace.

lemma *orthogonal_spanningset_subspace*:
fixes S :: *'a* :: *euclidean_space set*
assumes *subspace S*
obtains B **where** $B \subseteq S$ *pairwise orthogonal B span B* = *S*
by (*metis assms basis_orthogonal basis_subspace_exists span_eq*)

lemma *orthogonal_basis_subspace*:
fixes S :: *'a* :: *euclidean_space set*
assumes *subspace S*
obtains B **where** $0 \notin B$ $B \subseteq S$ *pairwise orthogonal B independent B*
card B = *dim S span B* = *S*
by (*metis assms dependent_zero orthogonal_basis_exists span_eq span_eq_iff*)

proposition *orthonormal_basis_subspace*:

```
  fixes S :: 'a :: euclidean_space set
  assumes subspace S
  obtains B where B ⊆ S pairwise orthogonal B
             and ⋀x. x ∈ B ⟹ norm x = 1
             and independent B card B = dim S span B = S
proof −
  obtain B where 0 ∉ B B ⊆ S
            and orth: pairwise orthogonal B
            and independent B card B = dim S span B = S
    by (blast intro: orthogonal_basis_subspace [OF assms])
  have 1: (λx. x /R norm x) ‘ B ⊆ S
    using ‹span B = S› span_superset span_mul by fastforce
  have 2: pairwise orthogonal ((λx. x /R norm x) ‘ B)
    using orth by (force simp: pairwise_def orthogonal_clauses)
  have 3: ⋀x. x ∈ (λx. x /R norm x) ‘ B ⟹ norm x = 1
    by (metis (no_types, lifting) ‹0 ∉ B› image_iff norm_sgn sgn_div_norm)
  have 4: independent ((λx. x /R norm x) ‘ B)
    by (metis 2 3 norm_zero pairwise_orthogonal_independent zero_neq_one)
  have inj_on (λx. x /R norm x) B
  proof
    fix x y
    assume x ∈ B y ∈ B x /R norm x = y /R norm y
    moreover have ⋀i. i ∈ B ⟹ norm (i /R norm i) = 1
      using 3 by blast
    ultimately show x = y
      by (metis norm_eq_1 orth orthogonal_clauses(7) orthogonal_commute or-
thogonal_def pairwise_def zero_neq_one)
  qed
  then have 5: card ((λx. x /R norm x) ‘ B) = dim S
    by (metis ‹card B = dim S› card_image)
  have 6: span ((λx. x /R norm x) ‘ B) = S
    by (metis 1 4 5 assms card_eq_dim independent_imp_finite span_subspace)
  show ?thesis
    by (rule that [OF 1 2 3 4 5 6])
qed


proposition orthogonal_to_subspace_exists_gen:
  fixes S :: 'a :: euclidean_space set
  assumes span S ⊂ span T
  obtains x where x ≠ 0 x ∈ span T ⋀y. y ∈ span S ⟹ orthogonal x y
proof −
  obtain B where B ⊆ span S and orthB: pairwise orthogonal B
            and ⋀x. x ∈ B ⟹ norm x = 1
            and independent B card B = dim S span B = span S
    by (metis dim_span orthonormal_basis_subspace subspace_span)
  with assms obtain u where spanBT: span B ⊆ span T and u ∉ span B u ∈
span T
    by auto
```

obtain C **where** *orthBC*: *pairwise orthogonal* $(B \cup C)$ **and** *spanBC*: *span* $(B \cup C) = span\ (B \cup \{u\})$
by (*blast intro*: *orthogonal_extension* [*OF orthB*])
show *thesis*
proof (*cases* $C \subseteq insert\ 0\ B$)
case *True*
then have $C \subseteq span\ B$
using *span_eq*
by (*metis span_insert_0 subset_trans*)
moreover have $u \in span\ (B \cup C)$
using ‹*span* $(B \cup C) = span\ (B \cup \{u\})$› *span_superset* **by** *force*
ultimately show *?thesis*
using *True* ‹$u \notin span\ B$›
by (*metis Un_insert_left span_insert_0 sup.orderE*)
next
case *False*
then obtain x **where** $x \in C\ x \neq 0\ x \notin B$
by *blast*
then have $x \in span\ T$
by (*smt* (*verit*, *ccfv_SIG*) *Set.set_insert* ‹$u \in span\ T$› *empty_subsetI insert_subset*
le_sup_iff spanBC spanBT span_mono span_span span_superset subset_trans)
moreover have *orthogonal x y* **if** $y \in span\ B$ **for** y
using *that*
proof (*rule span_induct*)
show *subspace* $\{a.\ orthogonal\ x\ a\}$
by (*simp add*: *subspace_orthogonal_to_vector*)
show $\bigwedge b.\ b \in B \Longrightarrow orthogonal\ x\ b$
by (*metis Un_iff* ‹$x \in C$› ‹$x \notin B$› *orthBC pairwise_def*)
qed
ultimately show *?thesis*
using ‹$x \neq 0$› *that* ‹*span* $B = span\ S$› **by** *auto*
qed
qed

corollary *orthogonal_to_subspace_exists*:
fixes $S :: 'a :: euclidean\_space\ set$
assumes $dim\ S < DIM('a)$
obtains x **where** $x \neq 0\ \bigwedge y.\ y \in span\ S \Longrightarrow orthogonal\ x\ y$
proof −
have $span\ S \subset UNIV$
by (*metis assms dim_eq_full order_less_imp_not_less top.not_eq_extremum*)
with *orthogonal_to_subspace_exists_gen* [*of S UNIV*] *that* **show** *?thesis*
by (*auto*)
qed

corollary *orthogonal_to_vector_exists*:
fixes $x :: 'a :: euclidean\_space$

assumes $2 \leq DIM('a)$
obtains y **where** $y \neq 0$ *orthogonal* x y
proof −
have *dim* $\{x\} < DIM('a)$
using *assms* **by** *auto*
then show *thesis*
by (*rule orthogonal_to_subspace_exists*) (*simp add*: *orthogonal_commute span_base that*)
qed

proposition *orthogonal_subspace_decomp_exists*:
fixes $S :: 'a :: euclidean\_space\ set$
obtains y z
where $y \in span\ S$
and $\bigwedge w.\ w \in span\ S \Longrightarrow orthogonal\ z\ w$
and $x = y + z$
proof −
obtain T **where** $0 \notin T$ $T \subseteq span\ S$ *pairwise orthogonal* T *independent* T
card $T = dim\ (span\ S)$ *span* $T = span\ S$
using *orthogonal_basis_subspace subspace_span* **by** *blast*
let $?a = \sum b{\in}T.\ (b \cdot x\ /\ (b \cdot b)) *_R b$
have *orth*: *orthogonal* $(x - ?a)\ w$ **if** $w \in span\ S$ **for** w
by (*simp add*: *Gram_Schmidt_step* ‹*pairwise orthogonal T*› ‹*span T = span S*›
orthogonal_commute that)
with *that*[*of* $?a\ x{-}?a$] ‹$T \subseteq span\ S$› **show** *?thesis*
by (*simp add*: *span_mul span_sum subsetD*)
qed

lemma *orthogonal_subspace_decomp_unique*:
fixes $S :: 'a :: euclidean\_space\ set$
assumes $x + y = x' + y'$
and *ST*: $x \in span\ S\ x' \in span\ S\ y \in span\ T\ y' \in span\ T$
and *orth*: $\bigwedge a\ b.\ [\![a \in S;\ b \in T]\!] \Longrightarrow orthogonal\ a\ b$
shows $x = x' \wedge y = y'$
proof −
have $x + y - y' = x'$
by (*simp add*: *assms*)
moreover have $\bigwedge a\ b.\ [\![a \in span\ S;\ b \in span\ T]\!] \Longrightarrow orthogonal\ a\ b$
by (*meson orth orthogonal_commute orthogonal_to_span*)
ultimately have $0 = x' - x$
using *assms*
by (*metis add.commute add_diff_cancel_right' diff_right_commute orthogonal_self span_diff*)
with *assms* **show** *?thesis* **by** *auto*
qed

lemma *vector_in_orthogonal_spanningset*:
fixes $a :: 'a{::}euclidean\_space$

obtains S **where** $a \in S$ *pairwise orthogonal* S *span* $S = UNIV$
by (*metis UnI1 Un_UNIV_right insertI1 orthogonal_extension pairwise_singleton span_UNIV*)

lemma *vector_in_orthogonal_basis*:
fixes $a :: 'a$::*euclidean_space*
assumes $a \neq 0$
obtains S **where** $a \in S$ $0 \notin S$ *pairwise orthogonal* S *independent* S *finite* S
span $S = UNIV$ *card* $S = DIM('a)$
proof −
obtain S **where** S: $a \in S$ *pairwise orthogonal* S *span* $S = UNIV$
using *vector_in_orthogonal_spanningset* **.**
show *thesis*
proof
show *pairwise orthogonal* $(S - \{0\})$
using *pairwise_mono* $S(2)$ **by** *blast*
show *independent* $(S - \{0\})$
by (*simp add*: ‹*pairwise orthogonal* $(S - \{0\})$› *pairwise_orthogonal_independent*)
show *finite* $(S - \{0\})$
using ‹*independent* $(S - \{0\})$› *independent_imp_finite* **by** *blast*
show *card* $(S - \{0\}) = DIM('a)$
using *span_delete_0* [*of* S] S
by (*simp add*: ‹*independent* $(S - \{0\})$› *indep_card_eq_dim_span*)
qed (*use* S ‹$a \neq 0$› **in** *auto*)
qed

lemma *vector_in_orthonormal_basis*:
fixes $a :: 'a$::*euclidean_space*
assumes *norm* $a = 1$
obtains S **where** $a \in S$ *pairwise orthogonal* S $\bigwedge x.\ x \in S \Longrightarrow$ *norm* $x = 1$
independent S *card* $S = DIM('a)$ *span* $S = UNIV$
proof −
have $a \neq 0$
using *assms* **by** *auto*
then obtain S **where** $a \in S$ $0 \notin S$ *finite* S
and S: *pairwise orthogonal* S *independent* S *span* $S = UNIV$ *card* $S = DIM('a)$
by (*metis vector_in_orthogonal_basis*)
let *?S* $= (\lambda x.\ x\ /_R$ *norm* $x)$ ‘ S
show *thesis*
proof
show $a \in$ *?S*
using ‹$a \in S$› *assms image_iff* **by** *fastforce*
next
show *pairwise orthogonal ?S*
using ‹*pairwise orthogonal* S› **by** (*auto simp*: *pairwise_def orthogonal_def*)
show $\bigwedge x.\ x \in (\lambda x.\ x\ /_R$ *norm* $x)$ ‘ $S \Longrightarrow$ *norm* $x = 1$
using ‹$0 \notin S$› **by** (*auto simp*: *field_split_simps*)
then show *ind*: *independent ?S*

by (*metis* ‹*pairwise orthogonal* ((λ*x*. *x* /$_R$ *norm x*) ‘ *S*)› *norm_zero pairwise_orthogonal_independent zero_neq_one*)
have *inj_on* (λ*x*. *x* /$_R$ *norm x*) *S*
unfolding *inj_on_def*
by (*metis* (*full_types*) *S*(*1*) ‹*0* ∉ *S*› *inverse_nonzero_iff_nonzero norm_eq_zero orthogonal_scaleR orthogonal_self pairwise_def*)
then show *card ?S* = *DIM*('*a*)
by (*simp add*: *card_image S*)
then show *span ?S* = *UNIV*
by (*metis ind dim_eq_card dim_eq_full*)
qed
qed

proposition *dim_orthogonal_sum*:
fixes *A* :: '*a*::*euclidean_space set*
assumes ⋀*x y*. ⟦*x* ∈ *A*; *y* ∈ *B*⟧ ⟹ *x* ∙ *y* = *0*
shows *dim*(*A* ∪ *B*) = *dim A* + *dim B*
proof −
have *1*: ⋀*x y*. ⟦*x* ∈ *span A*; *y* ∈ *B*⟧ ⟹ *x* ∙ *y* = *0*
by (*erule span_induct* [*OF* _ *subspace_hyperplane2*]; *simp add*: *assms*)
have ⋀*x y*. ⟦*x* ∈ *span A*; *y* ∈ *span B*⟧ ⟹ *x* ∙ *y* = *0*
using *1* **by** (*simp add*: *span_induct* [*OF* _ *subspace_hyperplane*])
then have *0*: ⋀*x y*. ⟦*x* ∈ *span A*; *y* ∈ *span B*⟧ ⟹ *x* ∙ *y* = *0*
by *simp*
have *dim*(*A* ∪ *B*) = *dim* (*span* (*A* ∪ *B*))
by (*simp*)
also have *span* (*A* ∪ *B*) = ((λ(*a*, *b*). *a* + *b*) ‘ (*span A* × *span B*))
by (*auto simp add*: *span_Un image_def*)
also have *dim* ... = *dim* {*x* + *y* |*x y*. *x* ∈ *span A* ∧ *y* ∈ *span B*}
by (*auto intro*!: *arg_cong* [**where** *f*=*dim*])
also have ... = *dim* {*x* + *y* |*x y*. *x* ∈ *span A* ∧ *y* ∈ *span B*} + *dim*(*span A* ∩ *span B*)
by (*auto dest*: *0*)
also have ... = *dim A* + *dim B*
using *dim_sums_Int* **by** *fastforce*
finally show *?thesis* **.**
qed

lemma *dim_subspace_orthogonal_to_vectors*:
fixes *A* :: '*a*::*euclidean_space set*
assumes *subspace A subspace B A* ⊆ *B*
shows *dim* {*y* ∈ *B*. ∀ *x* ∈ *A*. *orthogonal x y*} + *dim A* = *dim B*
proof −
have *dim* (*span* ({*y* ∈ *B*. ∀ *x*∈*A*. *orthogonal x y*} ∪ *A*)) = *dim* (*span B*)
proof (*rule arg_cong* [**where** *f*=*dim*, *OF subset_antisym*])
show *span* ({*y* ∈ *B*. ∀ *x*∈*A*. *orthogonal x y*} ∪ *A*) ⊆ *span B*
by (*simp add*: ‹*A* ⊆ *B*› *Collect_restrict span_mono*)
next
have ∗: *x* ∈ *span* ({*y* ∈ *B*. ∀ *x*∈*A*. *orthogonal x y*} ∪ *A*)

if $x \in B$ **for** x
proof −
obtain y z **where** $x = y + z$ $y \in span\ A$ **and** $orth$: $\bigwedge w.\ w \in span\ A \Longrightarrow orthogonal\ z\ w$
using *orthogonal_subspace_decomp_exists* [*of A x*] *that* **by** *auto*
moreover
have $y \in span\ B$
using ‹$y \in span\ A$› *assms*(*3*) *span_mono* **by** *blast*
ultimately have $z \in B \wedge (\forall x.\ x \in A \longrightarrow orthogonal\ x\ z)$
using *assms* **by** (*metis orthogonal_commute span_add_eq span_eq_iff that*)
then have *z*: $z \in span\ \{y \in B.\ \forall x{\in}A.\ orthogonal\ x\ y\}$
by (*simp add*: *span_base*)
then show *?thesis*
by (*smt* (*verit*, *best*) ‹$x = y + z$› ‹$y \in span\ A$› *le_sup_iff span_add_eq span_subspace_induct*
span_superset subset_iff subspace_span)
qed
show $span\ B \subseteq span\ (\{y \in B.\ \forall x{\in}A.\ orthogonal\ x\ y\} \cup A)$
by (*rule span_minimal*) (*auto intro*: ∗ *span_minimal*)
qed
then show *?thesis*
by (*metis* (*no_types*, *lifting*) *dim_orthogonal_sum dim_span mem_Collect_eq*
orthogonal_commute orthogonal_def)
qed

1.5.15 Linear functions are (uniformly) continuous on any set

1.5.16 Topological properties of linear functions

lemma *linear_lim_0*:
assumes *bounded_linear f*
shows $(f \longrightarrow 0)\ (at\ (0))$
proof −
interpret *f*: *bounded_linear f* **by** *fact*
have $(f \longrightarrow f\ 0)\ (at\ 0)$
using *tendsto_ident_at* **by** (*rule f.tendsto*)
then show *?thesis* **unfolding** *f.zero* **.**
qed

lemma *linear_continuous_at*:
bounded_linear f ⟹*continuous* (*at a*) *f*
by (*simp add*: *bounded_linear.isUCont isUCont_isCont*)

lemma *linear_continuous_within*:
bounded_linear f ⟹ *continuous* (*at x within s*) *f*
using *continuous_at_imp_continuous_at_within linear_continuous_at* **by** *blast*

lemma *linear_continuous_on*:
bounded_linear f ⟹ *continuous_on s f*

using *continuous_at_imp_continuous_on*[*of s f*] **using** *linear_continuous_at*[*of f*] **by** *auto*

lemma *Lim_linear*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space* **and** *h* :: *'b* ⇒ *'c*::*real_normed_vector*
assumes (*f* ⟶ *l*) *F* *linear h*
shows ((λ*x*. *h*(*f x*)) ⟶ *h l*) *F*
proof −
obtain *B* **where** *B*: *B* > *0* ⋀*x*. *norm* (*h x*) ≤ *B* ∗ *norm x*
using *linear_bounded_pos* [*OF* ‹*linear h*›] **by** *blast*
show *?thesis*
unfolding *tendsto_iff*
by (*simp add*: *assms bounded_linear.tendsto linear_linear tendstoD*)
qed

lemma *linear_continuous_compose*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space* **and** *g* :: *'b* ⇒ *'c*::*real_normed_vector*
assumes *continuous F f linear g*
shows *continuous F* (λ*x*. *g*(*f x*))
using *assms* **unfolding** *continuous_def* **by** (*rule Lim_linear*)

lemma *linear_continuous_on_compose*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space* **and** *g* :: *'b* ⇒ *'c*::*real_normed_vector*
assumes *continuous_on S f linear g*
shows *continuous_on S* (λ*x*. *g*(*f x*))
using *assms* **by** (*simp add*: *continuous_on_eq_continuous_within linear_continuous_compose*)

Also bilinear functions, in composition form

lemma *bilinear_continuous_compose*:
fixes *h* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space* ⇒ *'c*::*real_normed_vector*
assumes *continuous F f continuous F g bilinear h*
shows *continuous F* (λ*x*. *h* (*f x*) (*g x*))
using *assms bilinear_conv_bounded_bilinear bounded_bilinear.continuous* **by** *blast*

lemma *bilinear_continuous_on_compose*:
fixes *h* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space* ⇒ *'c*::*real_normed_vector*
and *f* :: *'d*::*t2_space* ⇒ *'a*
assumes *continuous_on S f continuous_on S g bilinear h*
shows *continuous_on S* (λ*x*. *h* (*f x*) (*g x*))
using *assms* **by** (*simp add*: *continuous_on_eq_continuous_within bilinear_continuous_compose*)

end

1.6 Affine Sets

theory *Affine*
imports *Linear_Algebra*
begin

lemma *if_smult*: (*if P then x else* (*y*::*real*)) $*_R$ *v* = (*if P then x* $*_R$ *v else y* $*_R$ *v*)
by (*fact if_distrib*)

lemma *sum_delta_notmem*:
assumes $x \notin s$
shows *sum* (λy. *if* ($y = x$) *then P x else Q y*) *s* = *sum Q s*
and *sum* (λy. *if* ($x = y$) *then P x else Q y*) *s* = *sum Q s*
and *sum* (λy. *if* ($y = x$) *then P y else Q y*) *s* = *sum Q s*
and *sum* (λy. *if* ($x = y$) *then P y else Q y*) *s* = *sum Q s*
apply (*rule_tac* [!] *sum.cong*)
using *assms*
apply *auto*
done

lemmas *independent_finite* = *independent_imp_finite*

lemma *span_substd_basis*:
assumes *d*: $d \subseteq$ *Basis*
shows *span d* = $\{x.\ \forall i{\in}$*Basis*. $i \notin d \longrightarrow x{\cdot}i = 0\}$
(**is** _ = *?B*)
proof −
have $d \subseteq$ *?B*
using *d* **by** (*auto simp*: *inner_Basis*)
moreover have *s*: *subspace ?B*
using *subspace_substandard*[*of* $\lambda i.\ i \notin d$] .
ultimately have *span d* $\subseteq$ *?B*
using *span_mono*[*of d ?B*] *span_eq_iff*[*of ?B*] **by** *blast*
moreover have $*$: *card d* $\leq$ *dim* (*span d*)
using *independent_card_le_dim*[*of d span d*] *independent_substdbasis*[*OF assms*]
span_superset[*of d*]
by *auto*
moreover from $*$ **have** *dim ?B* $\leq$ *dim* (*span d*)
using *dim_substandard*[*OF assms*] **by** *auto*
ultimately show *?thesis*
using *s subspace_dim_equal*[*of span d ?B*] *subspace_span*[*of d*] **by** *auto*
qed

lemma *basis_to_substdbasis_subspace_isomorphism*:
fixes *B* :: *'a*::*euclidean_space set*
assumes *independent B*
shows $\exists f\ d$::*'a set*. *card d* = *card B* $\wedge$ *linear f* $\wedge$ *f* ‘ *B* = *d* $\wedge$
f ‘ *span B* = $\{x.\ \forall i{\in}$*Basis*. $i \notin d \longrightarrow x \cdot i = 0\}$ $\wedge$ *inj_on f* (*span B*) $\wedge$ $d \subseteq$ *Basis*
proof −
have *B*: *card B* = *dim B*
using *dim_unique*[*of B B card B*] *assms span_superset*[*of B*] **by** *auto*
have *dim B* $\leq$ *card* (*Basis* :: *'a set*)
using *dim_subset_UNIV*[*of B*] **by** *simp*

from *obtain_subset_with_card_n*[*OF this*] **obtain** $d :: 'a\ set$ **where** d: $d \subseteq Basis$ **and** t: $card\ d = dim\ B$
by *auto*
let $?t = \{x::'a::euclidean\_space.\ \forall i{\in}Basis.\ i \notin d \longrightarrow x{\cdot}i = 0\}$
have $\exists f.\ linear\ f \wedge f\ `\ B = d \wedge f\ `\ span\ B = ?t \wedge inj\_on\ f\ (span\ B)$
proof (*intro basis_to_basis_subspace_isomorphism subspace_span subspace_substandard span_superset*)
show $d \subseteq \{x.\ \forall i{\in}Basis.\ i \notin d \longrightarrow x \cdot i = 0\}$
using *d inner_not_same_Basis* **by** *blast*
qed (*auto simp*: *span_substd_basis independent_substdbasis dim_substandard d t B assms*)
with t ‹$card\ B = dim\ B$› d **show** *?thesis* **by** *auto*
qed

1.6.1 Affine set and affine hull

definition *affine* :: $'a::real\_vector\ set \Rightarrow bool$
where $affine\ s \longleftrightarrow (\forall x{\in}s.\ \forall y{\in}s.\ \forall u\ v.\ u + v = 1 \longrightarrow u *_R x + v *_R y \in s)$

lemma *affine_alt*: $affine\ s \longleftrightarrow (\forall x{\in}s.\ \forall y{\in}s.\ \forall u::real.\ (1 - u) *_R x + u *_R y \in s)$
unfolding *affine_def* **by** (*metis eq_diff_eq'*)

lemma *affine_empty* [*iff*]: $affine\ \{\}$
unfolding *affine_def* **by** *auto*

lemma *affine_sing* [*iff*]: $affine\ \{x\}$
unfolding *affine_alt* **by** (*auto simp*: *scaleR_left_distrib* [*symmetric*])

lemma *affine_UNIV* [*iff*]: *affine UNIV*
unfolding *affine_def* **by** *auto*

lemma *affine_Inter* [*intro*]: $(\bigwedge s.\ s{\in}f \Longrightarrow affine\ s) \Longrightarrow affine\ (\bigcap f)$
unfolding *affine_def* **by** *auto*

lemma *affine_Int*[*intro*]: $affine\ s \Longrightarrow affine\ t \Longrightarrow affine\ (s \cap t)$
unfolding *affine_def* **by** *auto*

lemma *affine_scaling*: $affine\ s \Longrightarrow affine\ (image\ (\lambda x.\ c *_R x)\ s)$
apply (*clarsimp simp add*: *affine_def*)
apply (*rule_tac x*=$u *_R x + v *_R y$ **in** *image_eqI*)
apply (*auto simp*: *algebra_simps*)
done

lemma *affine_affine_hull* [*simp*]: *affine*(*affine hull s*)
unfolding *hull_def*
using *affine_Inter*[*of* $\{t.\ affine\ t \wedge s \subseteq t\}$] **by** *auto*

lemma *affine_hull_eq*[*simp*]: $(affine\ hull\ s = s) \longleftrightarrow affine\ s$

by (*metis affine_affine_hull hull_same*)

lemma *affine_hyperplane*: *affine* {*x*. *a* ⋅ *x* = *b*}
by (*simp add*: *affine_def algebra_simps*) (*metis distrib_right mult.left_neutral*)

Some explicit formulations

Formalized by Lars Schewe.

lemma *affine*:
fixes *V*::*'a*::*real_vector set*
shows *affine V* ⟷
(∀ *S u*. *finite S* ∧ *S* ≠ {} ∧ *S* ⊆ *V* ∧ *sum u S* = *1* ⟶ (∑ *x*∈*S*. *u x* ∗*R* *x*) ∈ *V*)
proof −
have *u* ∗*R* *x* + *v* ∗*R* *y* ∈ *V* **if** *x* ∈ *V y* ∈ *V u* + *v* = *1*::*real*
and ∗: ⋀*S u*. ⟦*finite S*; *S* ≠ {}; *S* ⊆ *V*; *sum u S* = *1*⟧ ⟹ (∑ *x*∈*S*. *u x* ∗*R* *x*) ∈ *V* **for** *x y u v*
proof (*cases x* = *y*)
case *True*
then show *?thesis*
using *that* **by** (*metis scaleR_add_left scaleR_one*)
next
case *False*
then show *?thesis*
using *that* ∗[*of* {*x*,*y*} λ*w*. *if w* = *x then u else v*] **by** *auto*
qed
moreover have (∑ *x*∈*S*. *u x* ∗*R* *x*) ∈ *V*
if ∗: ⋀*x y u v*. ⟦*x*∈*V*; *y*∈*V*; *u* + *v* = *1*⟧ ⟹ *u* ∗*R* *x* + *v* ∗*R* *y* ∈ *V*
and *finite S S* ≠ {} *S* ⊆ *V sum u S* = *1* **for** *S u*
proof −
define *n* **where** *n* = *card S*
consider *card S* = *0* | *card S* = *1* | *card S* = *2* | *card S* > *2* **by** *linarith*
then show (∑ *x*∈*S*. *u x* ∗*R* *x*) ∈ *V*
proof *cases*
assume *card S* = *1*
then obtain *a* **where** *S*={*a*}
by (*auto simp*: *card_Suc_eq*)
then show *?thesis*
using *that* **by** *simp*
next
assume *card S* = *2*
then obtain *a b* **where** *S* = {*a*, *b*}
by (*metis Suc_1 card_1_singletonE card_Suc_eq*)
then show *?thesis*
using ∗[*of a b*] *that*
by (*auto simp*: *sum_clauses*(*2*))
next
assume *card S* > *2*
then show *?thesis* **using** *that n_def*

```
    proof (induct n arbitrary: u S)
      case 0
      then show ?case by auto
    next
      case (Suc n u S)
      have sum u S = card S if ¬ (∃x∈S. u x ≠ 1)
        using that unfolding card_eq_sum by auto
      with Suc.prems obtain x where x ∈ S and x: u x ≠ 1 by force
      have c: card (S − {x}) = card S − 1
        by (simp add: Suc.prems(3) ‹x ∈ S›)
      have sum u (S − {x}) = 1 − u x
        by (simp add: Suc.prems sum_diff1 ‹x ∈ S›)
      with x have eq1: inverse (1 − u x) * sum u (S − {x}) = 1
        by auto
      have inV: (∑ y∈S − {x}. (inverse (1 − u x) * u y) *R y) ∈ V
      proof (cases card (S − {x}) > 2)
        case True
        then have S: S − {x} ≠ {} card (S − {x}) = n
          using Suc.prems c by force+
        show ?thesis
        proof (rule Suc.hyps)
          show (∑ a∈S − {x}. inverse (1 − u x) * u a) = 1
            by (auto simp: eq1 sum_distrib_left[symmetric])
        qed (use S Suc.prems True in auto)
      next
        case False
        then have card (S − {x}) = Suc (Suc 0)
          using Suc.prems c by auto
        then obtain a b where ab: (S − {x}) = {a, b} a≠b
          unfolding card_Suc_eq by auto
        then show ?thesis
          using eq1 ‹S ⊆ V›
         by (auto simp: sum_distrib_left distrib_left intro!: Suc.prems(2)[of a b])
      qed
      have u x + (1 − u x) = 1 ⟹
        u x *R x + (1 − u x) *R ((∑ y∈S − {x}. u y *R y) /R (1 − u x)) ∈ V
            by (rule Suc.prems) (use ‹x ∈ S› Suc.prems inV in ‹auto simp:
scaleR_right.sum›)
      moreover have (∑ a∈S. u a *R a) = u x *R x + (∑ a∈S − {x}. u a *R
a)
        by (meson Suc.prems(3) sum.remove ‹x ∈ S›)
      ultimately show (∑ x∈S. u x *R x) ∈ V
        by (simp add: x)
    qed
  qed (use ‹S≠{}› ‹finite S› in auto)
 qed
 ultimately show ?thesis
   unfolding affine_def by meson
qed
```

lemma *affine_hull_explicit*:
affine hull $p = \{y.\ \exists S\ u.\ \mathit{finite}\ S \wedge S \neq \{\} \wedge S \subseteq p \wedge \mathit{sum}\ u\ S = 1 \wedge \mathit{sum}\ (\lambda v.\ u\ v *_R v)\ S = y\}$
(**is** _ = *?rhs*)
proof (*rule hull_unique*)
show $p \subseteq$ *?rhs*
proof (*intro subsetI CollectI exI conjI*)
show $\bigwedge x.\ \mathit{sum}\ (\lambda z.\ 1)\ \{x\} = 1$
by *auto*
qed *auto*
show *?rhs* $\subseteq T$ **if** $p \subseteq T$ *affine* T **for** T
using *that* **unfolding** *affine* **by** *blast*
show *affine ?rhs*
unfolding *affine_def*
proof *clarify*
fix $u\ v$ :: *real* **and** *sx ux sy uy*
assume *uv*: $u + v = 1$
and *x*: *finite sx* $sx \neq \{\}$ $sx \subseteq p$ *sum ux sx* = (*1*::*real*)
and *y*: *finite sy* $sy \neq \{\}$ $sy \subseteq p$ *sum uy sy* = (*1*::*real*)
have **: $(sx \cup sy) \cap sx = sx$ $(sx \cup sy) \cap sy = sy$
by *auto*
show $\exists S\ w.\ \mathit{finite}\ S \wedge S \neq \{\} \wedge S \subseteq p \wedge$
$\mathit{sum}\ w\ S = 1 \wedge (\sum v{\in}S.\ w\ v *_R v) = u *_R (\sum v{\in}sx.\ ux\ v *_R v) + v *_R (\sum v{\in}sy.\ uy\ v *_R v)$
proof (*intro exI conjI*)
show *finite* $(sx \cup sy)$
using *x y* **by** *auto*
show *sum* $(\lambda i.\ (\mathit{if}\ i{\in}sx\ \mathit{then}\ u * ux\ i\ \mathit{else}\ 0) + (\mathit{if}\ i{\in}sy\ \mathit{then}\ v * uy\ i\ \mathit{else}\ 0))$ $(sx \cup sy) = 1$
using *x y uv*
by (*simp add*: *sum_Un sum.distrib sum.inter_restrict*[*symmetric*] *sum_distrib_left*[*symmetric*] **)
have $(\sum i{\in}sx \cup sy.\ ((\mathit{if}\ i \in sx\ \mathit{then}\ u * ux\ i\ \mathit{else}\ 0) + (\mathit{if}\ i \in sy\ \mathit{then}\ v * uy\ i\ \mathit{else}\ 0)) *_R i)$
$= (\sum i{\in}sx.\ (u * ux\ i) *_R i) + (\sum i{\in}sy.\ (v * uy\ i) *_R i)$
using *x y*
unfolding *scaleR_left_distrib scaleR_zero_left if_smult*
by (*simp add*: *sum_Un sum.distrib sum.inter_restrict*[*symmetric*] **)
also have ... $= u *_R (\sum v{\in}sx.\ ux\ v *_R v) + v *_R (\sum v{\in}sy.\ uy\ v *_R v)$
unfolding *scaleR_scaleR*[*symmetric*] *scaleR_right.sum* [*symmetric*] **by** *blast*
finally show $(\sum i{\in}sx \cup sy.\ ((\mathit{if}\ i \in sx\ \mathit{then}\ u * ux\ i\ \mathit{else}\ 0) + (\mathit{if}\ i \in sy\ \mathit{then}\ v * uy\ i\ \mathit{else}\ 0)) *_R i)$
$= u *_R (\sum v{\in}sx.\ ux\ v *_R v) + v *_R (\sum v{\in}sy.\ uy\ v *_R v)$.
qed (*use x y* **in** *auto*)
qed
qed

lemma *affine_hull_finite*:
assumes *finite S*
shows *affine hull S* = {*y*. ∃ *u*. *sum u S* = *1* ∧ *sum* (λ*v*. *u v* ∗*R v*) *S* = *y*}
proof −
have ∗: ∃ *h*. *sum h S* = *1* ∧ (∑ *v*∈*S*. *h v* ∗*R v*) = *x*
if *F* ⊆ *S finite F F* ≠ {} **and** *sum*: *sum u F* = *1* **and** *x*: (∑ *v*∈*F*. *u v* ∗*R v*) = *x* **for** *x F u*
proof −
have *S* ∩ *F* = *F*
using *that* **by** *auto*
show *?thesis*
proof (*intro exI conjI*)
show (∑ *x*∈*S*. *if x* ∈ *F then u x else 0*) = *1*
by (*metis* (*mono_tags*, *lifting*) ‹*S* ∩ *F* = *F*› *assms sum.inter_restrict sum*)
show (∑ *v*∈*S*. (*if v* ∈ *F then u v else 0*) ∗*R v*) = *x*
by (*simp add*: *if_smult cong*: *if_cong*) (*metis* (*no_types*) ‹*S* ∩ *F* = *F*› *assms sum.inter_restrict x*)
qed
qed
show *?thesis*
unfolding *affine_hull_explicit* **using** *assms*
by (*fastforce dest*: ∗)
qed

Stepping theorems and hence small special cases

lemma *affine_hull_empty*[*simp*]: *affine hull* {} = {}
by *simp*

lemma *affine_hull_finite_step*:
fixes *y* :: *'a*::*real_vector*
shows *finite S* ⟹
(∃ *u*. *sum u* (*insert a S*) = *w* ∧ *sum* (λ*x*. *u x* ∗*R x*) (*insert a S*) = *y*) ⟷
(∃ *v u*. *sum u S* = *w* − *v* ∧ *sum* (λ*x*. *u x* ∗*R x*) *S* = *y* − *v* ∗*R a*) (**is** *_* ⟹ *?lhs* = *?rhs*)
proof −
assume *fin*: *finite S*
show *?lhs* = *?rhs*
proof
assume *?lhs*
then obtain *u* **where** *u*: *sum u* (*insert a S*) = *w* ∧ (∑ *x*∈*insert a S*. *u x* ∗*R x*) = *y*
by *auto*
show *?rhs*
proof (*cases a* ∈ *S*)
case *True*
then show *?thesis*
using *u* **by** (*simp add*: *insert_absorb*) (*metis diff_zero real_vector.scale_zero_left*)
next

```
      case False
      show ?thesis
        by (rule exI [where x=u a]) (use u fin False in auto)
    qed
  next
    assume ?rhs
    then obtain v u where vu: sum u S = w - v  (∑x∈S. u x *R x) = y - v
*R a
      by auto
    have *: ⋀x M. (if x = a then v else M) *R x = (if x = a then v *R x else M
*R x)
      by auto
    show ?lhs
    proof (cases a ∈ S)
      case True
      show ?thesis
        by (rule exI [where x=λx. (if x=a then v else 0) + u x])
           (simp add: True scaleR_left_distrib sum.distrib sum_clauses fin vu *
cong: if_cong)
    next
      case False
      then show ?thesis
        apply (rule_tac x=λx. if x=a then v else u x in exI)
        apply (simp add: vu sum_clauses(2)[OF fin] *)
        by (simp add: sum_delta_notmem(3) vu)
    qed
  qed
qed

lemma affine_hull_2:
  fixes a b :: 'a::real_vector
  shows affine hull {a,b} = {u *R a + v *R b| u v. (u + v = 1)}
  (is ?lhs = ?rhs)
proof -
  have *:
    ⋀x y z. z = x - y ⟷ y + z = (x::real)
    ⋀x y z. z = x - y ⟷ y + z = (x::'a) by auto
  have ?lhs = {y. ∃u. sum u {a, b} = 1 ∧ (∑v∈{a, b}. u v *R v) = y}
    using affine_hull_finite[of {a,b}] by auto
  also have ... = {y. ∃v u. u b = 1 - v ∧ u b *R b = y - v *R a}
    by (simp add: affine_hull_finite_step[of {b} a])
  also have ... = ?rhs unfolding * by auto
  finally show ?thesis by auto
qed

lemma affine_hull_3:
  fixes a b c :: 'a::real_vector
  shows affine hull {a,b,c} = { u *R a + v *R b + w *R c| u v w. u + v + w =
1}
```

proof −
have ∗:
⋀*x y z*. *z* = *x* − *y* ⟷ *y* + *z* = (*x*::*real*)
⋀*x y z*. *z* = *x* − *y* ⟷ *y* + *z* = (*x*::′*a*) **by** *auto*
show *?thesis*
apply (*simp add*: *affine_hull_finite affine_hull_finite_step*)
unfolding ∗
apply *safe*
apply (*metis add.assoc*)
apply (*rule_tac x*=*u* **in** *exI*, *force*)
done
qed

lemma *mem_affine*:
assumes *affine S x* ∈ *S y* ∈ *S u* + *v* = *1*
shows *u* ∗*R x* + *v* ∗*R y* ∈ *S*
using *assms affine_def*[*of S*] **by** *auto*

lemma *mem_affine_3*:
assumes *affine S x* ∈ *S y* ∈ *S z* ∈ *S u* + *v* + *w* = *1*
shows *u* ∗*R x* + *v* ∗*R y* + *w* ∗*R z* ∈ *S*
proof −
have *u* ∗*R x* + *v* ∗*R y* + *w* ∗*R z* ∈ *affine hull* {*x*, *y*, *z*}
using *affine_hull_3*[*of x y z*] *assms* **by** *auto*
moreover
have *affine hull* {*x*, *y*, *z*} ⊆ *affine hull S*
using *hull_mono*[*of* {*x*, *y*, *z*} *S*] *assms* **by** *auto*
moreover
have *affine hull S* = *S*
using *assms affine_hull_eq*[*of S*] **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *mem_affine_3_minus*:
assumes *affine S x* ∈ *S y* ∈ *S z* ∈ *S*
shows *x* + *v* ∗*R* (*y*−*z*) ∈ *S*
using *mem_affine_3*[*of S x y z 1 v* −*v*] *assms*
by (*simp add*: *algebra_simps*)

corollary *mem_affine_3_minus2*:
⟦*affine S*; *x* ∈ *S*; *y* ∈ *S*; *z* ∈ *S*⟧ ⟹ *x* − *v* ∗*R* (*y*−*z*) ∈ *S*
by (*metis add_uminus_conv_diff mem_affine_3_minus real_vector.scale_minus_left*)

Some relations between affine hull and subspaces

lemma *affine_hull_insert_subset_span*:
affine hull (*insert a S*) ⊆ {*a* + *v*| *v* . *v* ∈ *span* {*x* − *a* | *x* . *x* ∈ *S*}}
proof −
have ∃*v T u*. *x* = *a* + *v* ∧ (*finite T* ∧ *T* ⊆ {*x* − *a* |*x*. *x* ∈ *S*} ∧ (∑ *v*∈*T*. *u v*

```
*R v) = v)
    if finite F F ≠ {} F ⊆ insert a S sum u F = 1 (∑ v∈F. u v *R v) = x
    for x F u
  proof −
    have *: (λx. x − a) ‘ (F − {a}) ⊆ {x − a |x. x ∈ S}
      using that by auto
    show ?thesis
    proof (intro exI conjI)
      show finite ((λx. x − a) ‘ (F − {a}))
        by (simp add: that(1))
      show (∑ v∈(λx. x − a) ‘ (F − {a}). u(v+a) *R v) = x−a
        by (simp add: sum.reindex[unfolded inj_on_def] algebra_simps
            sum_subtractf scaleR_left.sum[symmetric] sum_diff1 that)
    qed (use ‹F ⊆ insert a S› in auto)
  qed
  then show ?thesis
    unfolding affine_hull_explicit span_explicit by fast
qed

lemma affine_hull_insert_span:
  assumes a ∉ S
  shows affine hull (insert a S) = {a + v | v . v ∈ span {x − a | x.  x ∈ S}}
proof −
  have *: ∃ G u. finite G ∧ G ≠ {} ∧ G ⊆ insert a S ∧ sum u G = 1 ∧ (∑ v∈G.
u v *R v) = y
    if v ∈ span {x − a |x. x ∈ S} y = a + v for y v
  proof −
    from that
    obtain T u where u: finite T T ⊆ {x − a |x. x ∈ S} a + (∑ v∈T. u v *R v)
= y
      unfolding span_explicit by auto
    define F where F = (λx. x + a) ‘ T
    have F: finite F F ⊆ S (∑ v∈F. u (v − a) *R (v − a)) = y − a
     unfolding F_def using u by (auto simp: sum.reindex[unfolded inj_on_def])
    have *: F ∩ {a} = {} F ∩ − {a} = F
      using F assms by auto
    show ∃ G u. finite G ∧ G ≠ {} ∧ G ⊆ insert a S ∧ sum u G = 1 ∧ (∑ v∈G.
u v *R v) = y
      apply (rule_tac x = insert a F in exI)
      apply (rule_tac x = λx. if x=a then 1 − sum (λx. u (x − a)) F else u (x −
a) in exI)
      using assms F
     apply (auto simp:  sum_clauses sum.If_cases if_smult sum_subtractf scaleR_left.sum
algebra_simps *)
      done
  qed
  show ?thesis
   by (intro subset_antisym affine_hull_insert_subset_span) (auto simp: affine_hull_explicit
dest!: *)
```

qed

lemma *affine_hull_span*:
assumes $a \in S$
shows *affine hull* $S = \{a + v \mid v.\ v \in span\ \{x - a \mid x.\ x \in S - \{a\}\}\}$
using *affine_hull_insert_span*[*of* a $S - \{a\}$, *unfolded insert_Diff*[*OF assms*]]
by *auto*

Parallel affine sets

definition *affine_parallel* :: *'a*::*real_vector set* ⇒ *'a*::*real_vector set* ⇒ *bool*
where *affine_parallel* $S\ T \longleftrightarrow (\exists a.\ T = (\lambda x.\ a + x)\ `\ S)$

lemma *affine_parallel_expl_aux*:
fixes $S\ T$ :: *'a*::*real_vector set*
assumes $\bigwedge x.\ x \in S \longleftrightarrow a + x \in T$
shows $T = (\lambda x.\ a + x)\ `\ S$
proof −
have $x \in ((\lambda x.\ a + x)\ `\ S)$ **if** $x \in T$ **for** x
using *that*
by (*simp add*: *image_iff*) (*metis add.commute diff_add_cancel assms*)
moreover have $T \geq (\lambda x.\ a + x)\ `\ S$
using *assms* **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *affine_parallel_expl*: *affine_parallel* $S\ T \longleftrightarrow (\exists a.\ \forall x.\ x \in S \longleftrightarrow a + x \in T)$
by (*auto simp add*: *affine_parallel_def*)
(*use affine_parallel_expl_aux* [*of* S _ T] **in** *blast*)

lemma *affine_parallel_reflex*: *affine_parallel* $S\ S$
unfolding *affine_parallel_def*
using *image_add_0* **by** *blast*

lemma *affine_parallel_commut*:
assumes *affine_parallel* $A\ B$
shows *affine_parallel* $B\ A$
proof −
from *assms* **obtain** a **where** B: $B = (\lambda x.\ a + x)\ `\ A$
unfolding *affine_parallel_def* **by** *auto*
have [*simp*]: $(\lambda x.\ x - a) = plus\ (-\ a)$ **by** (*simp add*: *fun_eq_iff*)
from B **show** *?thesis*
using *translation_galois* [*of* B a A]
unfolding *affine_parallel_def* **by** *blast*
qed

lemma *affine_parallel_assoc*:
assumes *affine_parallel* $A\ B$

```
    and affine_parallel B C
  shows affine_parallel A C
proof −
  from assms obtain ab where B = (λx. ab + x) ‘ A
    unfolding affine_parallel_def by auto
  moreover
  from assms obtain bc where C = (λx. bc + x) ‘ B
    unfolding affine_parallel_def by auto
  ultimately show ?thesis
    using translation_assoc[of bc ab A] unfolding affine_parallel_def by auto
qed

lemma affine_translation_aux:
  fixes a :: 'a::real_vector
  assumes affine ((λx. a + x) ‘ S)
  shows affine S
proof −
  {
    fix x y u v
    assume xy: x ∈ S y ∈ S (u :: real) + v = 1
    then have (a + x) ∈ ((λx. a + x) ‘ S) (a + y) ∈ ((λx. a + x) ‘ S)
      by auto
    then have h1: u *R  (a + x) + v *R (a + y) ∈ (λx. a + x) ‘ S
      using xy assms unfolding affine_def by auto
    have u *R (a + x) + v *R (a + y) = (u + v) *R a + (u *R x + v *R y)
      by (simp add: algebra_simps)
    also have ... = a + (u *R x + v *R y)
      using ‹u + v = 1› by auto
    ultimately have a + (u *R x + v *R y) ∈ (λx. a + x) ‘ S
      using h1 by auto
    then have u *R x + v *R y ∈ S by auto
  }
  then show ?thesis unfolding affine_def by auto
qed

lemma affine_translation:
  affine S ⟷ affine ((+) a ‘ S) for a :: 'a::real_vector
proof
  show affine ((+) a ‘ S) if affine S
    using that translation_assoc [of − a a S]
    by (auto intro: affine_translation_aux [of − a ((+) a ‘ S)])
  show affine S if affine ((+) a ‘ S)
    using that by (rule affine_translation_aux)
qed

lemma parallel_is_affine:
  fixes S T :: 'a::real_vector set
  assumes affine S affine_parallel S T
  shows affine T
```

proof −
from *assms* **obtain** *a* **where** $T = (\lambda x.\ a + x)\ `\ S$
unfolding *affine_parallel_def* **by** *auto*
then show *?thesis*
using *affine_translation assms* **by** *auto*
qed

lemma *subspace_imp_affine*: *subspace s* $\Longrightarrow$ *affine s*
unfolding *subspace_def affine_def* **by** *auto*

lemma *affine_hull_subset_span*: (*affine hull s*) $\subseteq$ (*span s*)
by (*metis hull_minimal span_superset subspace_imp_affine subspace_span*)

Subspace parallel to an affine set

lemma *subspace_affine*: *subspace S* $\longleftrightarrow$ *affine S* $\wedge$ *0* $\in$ *S*
proof −
have *h0*: *subspace S* $\Longrightarrow$ *affine S* $\wedge$ *0* $\in$ *S*
using *subspace_imp_affine*[*of S*] *subspace_0* **by** *auto*
{
assume *assm*: *affine S* $\wedge$ *0* $\in$ *S*
{
fix *c* :: *real*
fix *x*
assume *x*: *x* $\in$ *S*
have *c* $*_R$ *x* = (*1*−*c*) $*_R$ *0* + *c* $*_R$ *x* **by** *auto*
moreover
have (*1* − *c*) $*_R$ *0* + *c* $*_R$ *x* $\in$ *S*
using *affine_alt*[*of S*] *assm x* **by** *auto*
ultimately have *c* $*_R$ *x* $\in$ *S* **by** *auto*
}
then have *h1*: $\forall\, c.\ \forall\, x \in S.\ c *_R x \in S$ **by** *auto*

{
fix *x y*
assume *xy*: *x* $\in$ *S y* $\in$ *S*
define *u* **where** *u* = (*1* :: *real*)/*2*
have (*1*/*2*) $*_R$ (*x*+*y*) = (*1*/*2*) $*_R$ (*x*+*y*)
by *auto*
moreover
have (*1*/*2*) $*_R$ (*x*+*y*)=(*1*/*2*) $*_R$ *x* + (*1*−(*1*/*2*)) $*_R$ *y*
by (*simp add*: *algebra_simps*)
moreover
have (*1* − *u*) $*_R$ *x* + *u* $*_R$ *y* $\in$ *S*
using *affine_alt*[*of S*] *assm xy* **by** *auto*
ultimately
have (*1*/*2*) $*_R$ (*x*+*y*) $\in$ *S*
using *u_def* **by** *auto*
moreover

```
    have x + y = 2 *R ((1/2) *R (x+y))
      by auto
    ultimately
    have x + y ∈ S
      using h1[rule_format, of (1/2) *R (x+y) 2] by auto
   }
   then have ∀ x ∈ S. ∀ y ∈ S. x + y ∈ S
    by auto
   then have subspace S
    using h1 assm unfolding subspace_def by auto
  }
  then show ?thesis using h0 by metis
qed

lemma affine_diffs_subspace:
  assumes affine S a ∈ S
  shows subspace ((λx. (−a)+x) ‘ S)
proof −
  have [simp]: (λx. x − a) = plus (− a) by (simp add: fun_eq_iff)
  have affine ((λx. (−a)+x) ‘ S)
    using affine_translation assms by blast
  moreover have 0 ∈ ((λx. (−a)+x) ‘ S)
    using assms exI[of (λx. x∈S ∧ −a+x = 0) a] by auto
  ultimately show ?thesis using subspace_affine by auto
qed

lemma affine_diffs_subspace_subtract:
  subspace ((λx. x − a) ‘ S) if affine S a ∈ S
  using that affine_diffs_subspace [of _ a] by simp

lemma parallel_subspace_explicit:
  assumes affine S
    and a ∈ S
  assumes L ≡ {y. ∃ x ∈ S. (−a) + x = y}
  shows subspace L ∧ affine_parallel S L
proof −
  from assms have L = plus (− a) ‘ S by auto
  then have par: affine_parallel S L
    unfolding affine_parallel_def ..
  then have affine L using assms parallel_is_affine by auto
  moreover have 0 ∈ L
    using assms by auto
  ultimately show ?thesis
    using subspace_affine par by auto
qed

lemma parallel_subspace_aux:
  assumes subspace A
    and subspace B
```

```
    and affine_parallel A B
  shows A ⊇ B
proof −
  from assms obtain a where a: ∀ x. x ∈ A ⟷ a + x ∈ B
    using affine_parallel_expl[of A B] by auto
  then have −a ∈ A
    using assms subspace_0[of B] by auto
  then have a ∈ A
    using assms subspace_neg[of A −a] by auto
  then show ?thesis
    using assms a unfolding subspace_def by auto
qed

lemma parallel_subspace:
  assumes subspace A
    and subspace B
    and affine_parallel A B
  shows A = B
proof
  show A ⊇ B
    using assms parallel_subspace_aux by auto
  show A ⊆ B
    using assms parallel_subspace_aux[of B A] affine_parallel_commut by auto
qed

lemma affine_parallel_subspace:
  assumes affine S S ≠ {}
  shows ∃!L. subspace L ∧ affine_parallel S L
proof −
  have ex: ∃ L. subspace L ∧ affine_parallel S L
    using assms parallel_subspace_explicit by auto
  {
    fix L1 L2
    assume ass: subspace L1 ∧ affine_parallel S L1 subspace L2 ∧ affine_parallel
S L2
    then have affine_parallel L1 L2
      using affine_parallel_commut[of S L1] affine_parallel_assoc[of L1 S L2] by
auto
    then have L1 = L2
      using ass parallel_subspace by auto
  }
  then show ?thesis using ex by auto
qed
```

1.6.2 Affine Dependence

Formalized by Lars Schewe.

```
definition affine_dependent :: 'a::real_vector set ⇒ bool
  where affine_dependent s ⟷ (∃ x∈s. x ∈ affine hull (s − {x}))
```

lemma *affine_dependent_imp_dependent*: *affine_dependent s* $\Longrightarrow$ *dependent s*
unfolding *affine_dependent_def dependent_def*
using *affine_hull_subset_span* **by** *auto*

lemma *affine_dependent_subset*:
$[\![$*affine_dependent s*; *s* $\subseteq$ *t*$]\!] \Longrightarrow$ *affine_dependent t*
apply (*simp add*: *affine_dependent_def Bex_def*)
apply (*blast dest*: *hull_mono* [*OF Diff_mono* [*OF _ subset_refl*]])
done

lemma *affine_independent_subset*:
shows $[\![\neg$ *affine_dependent t*; *s* $\subseteq$ *t*$]\!] \Longrightarrow \neg$ *affine_dependent s*
by (*metis affine_dependent_subset*)

lemma *affine_independent_Diff*:
$\neg$ *affine_dependent s* $\Longrightarrow \neg$ *affine_dependent*(*s* $-$ *t*)
by (*meson Diff_subset affine_dependent_subset*)

proposition *affine_dependent_explicit*:
affine_dependent p $\longleftrightarrow$
(∃ *S u*. *finite S* ∧ *S* ⊆ *p* ∧ *sum u S* = *0* ∧ (∃ *v*∈*S*. *u v* ≠ *0*) ∧ *sum* (λ*v*. *u v* $*_R$ *v*) *S* = *0*)
proof −
have ∃ *S u*. *finite S* ∧ *S* ⊆ *p* ∧ *sum u S* = *0* ∧ (∃ *v*∈*S*. *u v* ≠ *0*) ∧ (∑ *w*∈*S*. *u w* $*_R$ *w*) = *0*
if (∑ *w*∈*S*. *u w* $*_R$ *w*) = *x x* ∈ *p finite S S* ≠ {} *S* ⊆ *p* − {*x*} *sum u S* = *1* **for** *x S u*
proof (*intro exI conjI*)
have *x* ∉ *S*
using *that* **by** *auto*
then show (∑ *v* ∈ *insert x S*. *if v* = *x then* − *1 else u v*) = *0*
using *that* **by** (*simp add*: *sum_delta_notmem*)
show (∑ *w* ∈ *insert x S*. (*if w* = *x then* − *1 else u w*) $*_R$ *w*) = *0*
using *that* ‹*x* ∉ *S*› **by** (*simp add*: *if_smult sum_delta_notmem cong*: *if_cong*)
qed (*use that* **in** *auto*)
moreover have ∃ *x*∈*p*. ∃ *S u*. *finite S* ∧ *S* ≠ {} ∧ *S* ⊆ *p* − {*x*} ∧ *sum u S* = *1* ∧ (∑ *v*∈*S*. *u v* $*_R$ *v*) = *x*
if (∑ *v*∈*S*. *u v* $*_R$ *v*) = *0 finite S S* ⊆ *p sum u S* = *0 v* ∈ *S u v* ≠ *0* **for** *S u v*
proof (*intro bexI exI conjI*)
have *S* ≠ {*v*}
using *that* **by** *auto*
then show *S* − {*v*} ≠ {}
using *that* **by** *auto*
show (∑ *x* ∈ *S* − {*v*}. − (*1* / *u v*) ∗ *u x*) = *1*
unfolding *sum_distrib_left*[*symmetric*] *sum_diff1*[*OF* ‹*finite S*›] **by** (*simp add*: *that*)
show (∑ *x*∈*S* − {*v*}. (− (*1* / *u v*) ∗ *u x*) $*_R$ *x*) = *v*
unfolding *sum_distrib_left* [*symmetric*] *scaleR_scaleR*[*symmetric*]

```
        scaleR_right.sum [symmetric] sum_diff1[OF ‹finite S›]
    using that by auto
  show S - {v} ⊆ p - {v}
    using that by auto
 qed (use that in auto)
 ultimately show ?thesis
  unfolding affine_dependent_def affine_hull_explicit by auto
qed

lemma affine_dependent_explicit_finite:
 fixes S :: 'a::real_vector set
 assumes finite S
 shows affine_dependent S ⟷
  (∃ u. sum u S = 0 ∧ (∃ v∈S. u v ≠ 0) ∧ sum (λv. u v *R v) S = 0)
 (is ?lhs = ?rhs)
proof
 have *: ⋀vt u v. (if vt then u v else 0) *R v = (if vt then (u v) *R v else 0::'a)
  by auto
 assume ?lhs
 then obtain t u v where
  finite t t ⊆ S sum u t = 0 v∈t u v ≠ 0 (∑ v∈t. u v *R v) = 0
  unfolding affine_dependent_explicit by auto
 then show ?rhs
  apply (rule_tac x=λx. if x∈t then u x else 0 in exI)
  apply (auto simp: * sum.inter_restrict[OF assms, symmetric] Int_absorb1[OF
‹t⊆S›])
  done
next
 assume ?rhs
 then obtain u v where sum u S = 0 v∈S u v ≠ 0 (∑ v∈S. u v *R v) = 0
  by auto
 then show ?lhs unfolding affine_dependent_explicit
  using assms by auto
qed

lemma dependent_imp_affine_dependent:
 assumes dependent {x - a| x . x ∈ s}
  and a ∉ s
 shows affine_dependent (insert a s)
proof -
 from assms(1)[unfolded dependent_explicit] obtain S u v
  where obt: finite S S ⊆ {x - a |x. x ∈ s} v∈S u v ≠ 0 (∑ v∈S. u v *R v) =
0
  by auto
 define t where t = (λx. x + a) ‘ S

 have inj: inj_on (λx. x + a) S
  unfolding inj_on_def by auto
 have 0 ∉ S
```

```
    using obt(2) assms(2) unfolding subset_eq by auto
  have fin: finite t and t ⊆ s
    unfolding t_def using obt(1,2) by auto
  then have finite (insert a t) and insert a t ⊆ insert a s
    by auto
  moreover have ∗: ⋀P Q. (∑ x∈t. (if x = a then P x else Q x)) = (∑ x∈t. Q
x)
    apply (rule sum.cong)
    using ‹a∉s› ‹t⊆s›
    apply auto
    done
  have (∑ x∈insert a t. if x = a then − (∑ x∈t. u (x − a)) else u (x − a)) = 0
    unfolding sum_clauses(2)[OF fin] ∗ using ‹a∉s› ‹t⊆s› by auto
  moreover have ∃ v∈insert a t. (if v = a then − (∑ x∈t. u (x − a)) else u (v −
a)) ≠ 0
    using obt(3,4) ‹0∉S›
    by (rule_tac x=v + a in bexI) (auto simp: t_def)
  moreover have ∗: ⋀P Q. (∑ x∈t. (if x = a then P x else Q x) ∗R x) = (∑ x∈t.
Q x ∗R x)
    using ‹a∉s› ‹t⊆s› by (auto intro!: sum.cong)
  have (∑ x∈t. u (x − a)) ∗R a = (∑ v∈t. u (v − a) ∗R v)
    unfolding scaleR_left.sum
    unfolding t_def and sum.reindex[OF inj] and o_def
    using obt(5)
    by (auto simp: sum.distrib scaleR_right_distrib)
  then have (∑ v∈insert a t. (if v = a then − (∑ x∈t. u (x − a)) else u (v − a))
∗R v) = 0
    unfolding sum_clauses(2)[OF fin]
    using ‹a∉s› ‹t⊆s›
    by (auto simp: ∗)
  ultimately show ?thesis
    unfolding affine_dependent_explicit
    apply (rule_tac x=insert a t in exI, auto)
    done
qed

lemma affine_dependent_biggerset:
  fixes s :: 'a::euclidean_space set
  assumes finite s card s ≥ DIM('a) + 2
  shows affine_dependent s
proof −
  have s ≠ {} using assms by auto
  then obtain a where a∈s by auto
  have ∗: {x − a |x. x ∈ s − {a}} = (λx. x − a) ‘ (s − {a})
    by auto
  have card {x − a |x. x ∈ s − {a}} = card (s − {a})
    unfolding ∗ by (simp add: card_image inj_on_def)
  also have ... > DIM('a) using assms(2)
    unfolding card_Diff_singleton[OF ‹a∈s›] by auto
```

```
  finally show ?thesis
    apply (subst insert_Diff[OF ‹a∈s›, symmetric])
    apply (rule dependent_imp_affine_dependent)
    apply (rule dependent_biggerset, auto)
    done
qed

lemma affine_dependent_biggerset_general:
  assumes finite (S :: 'a::euclidean_space set)
    and card S ≥ dim S + 2
  shows affine_dependent S
proof −
  from assms(2) have S ≠ {} by auto
  then obtain a where a∈S by auto
  have ∗: {x − a |x. x ∈ S − {a}} = (λx. x − a) ‘ (S − {a})
    by auto
  have ∗∗: card {x − a |x. x ∈ S − {a}} = card (S − {a})
    by (metis (no_types, lifting) ∗ card_image diff_add_cancel inj_on_def)
  have dim {x − a |x. x ∈ S − {a}} ≤ dim S
    using ‹a∈S› by (auto simp: span_base span_diff intro: subset_le_dim)
  also have ... < dim S + 1 by auto
  also have ... ≤ card (S − {a})
    using assms card_Diff_singleton[OF ‹a∈S›] by auto
  finally show ?thesis
    apply (subst insert_Diff[OF ‹a∈S›, symmetric])
    apply (rule dependent_imp_affine_dependent)
    apply (rule dependent_biggerset_general)
    unfolding ∗∗
    apply auto
    done
qed
```

1.6.3 Some Properties of Affine Dependent Sets

```
lemma affine_independent_0 [simp]: ¬ affine_dependent {}
  by (simp add: affine_dependent_def)

lemma affine_independent_1 [simp]: ¬ affine_dependent {a}
  by (simp add: affine_dependent_def)

lemma affine_independent_2 [simp]: ¬ affine_dependent {a,b}
  by (simp add: affine_dependent_def insert_Diff_if hull_same)

lemma affine_hull_translation: affine hull ((λx. a + x) ‘ S) = (λx. a + x) ‘
(affine hull S)
proof −
  have affine ((λx. a + x) ‘ (affine hull S))
    using affine_translation affine_affine_hull by blast
  moreover have (λx. a + x) ‘ S ⊆ (λx. a + x) ‘ (affine hull S)
```

```
    using hull_subset[of S] by auto
  ultimately have h1: affine hull ((λx. a + x) ‘ S) ⊆ (λx. a + x) ‘ (affine hull
S)
    by (metis hull_minimal)
  have affine((λx. −a + x) ‘ (affine hull ((λx. a + x) ‘ S)))
    using affine_translation affine_affine_hull by blast
  moreover have (λx. −a + x) ‘ (λx. a + x) ‘ S ⊆ (λx. −a + x) ‘ (affine hull
((λx. a + x) ‘ S))
    using hull_subset[of (λx. a + x) ‘ S] by auto
  moreover have S = (λx. −a + x) ‘ (λx. a + x) ‘ S
    using translation_assoc[of −a a] by auto
  ultimately have (λx. −a + x) ‘ (affine hull ((λx. a + x) ‘ S)) >= (affine hull
S)
    by (metis hull_minimal)
  then have affine hull ((λx. a + x) ‘ S) >= (λx. a + x) ‘ (affine hull S)
    by auto
  then show ?thesis using h1 by auto
qed

lemma affine_dependent_translation:
  assumes affine_dependent S
  shows affine_dependent ((λx. a + x) ‘ S)
proof −
  obtain x where x: x ∈ S ∧ x ∈ affine hull (S − {x})
    using assms affine_dependent_def by auto
  have (+) a ‘ (S − {x}) = (+) a ‘ S − {a + x}
    by auto
  then have a + x ∈ affine hull ((λx. a + x) ‘ S − {a + x})
    using affine_hull_translation[of a S − {x}] x by auto
  moreover have a + x ∈ (λx. a + x) ‘ S
    using x by auto
  ultimately show ?thesis
    unfolding affine_dependent_def by auto
qed

lemma affine_dependent_translation_eq:
  affine_dependent S ⟷ affine_dependent ((λx. a + x) ‘ S)
proof −
  {
    assume affine_dependent ((λx. a + x) ‘ S)
    then have affine_dependent S
    using affine_dependent_translation[of ((λx. a + x) ‘ S) −a] translation_assoc[of
−a a]
      by auto
  }
  then show ?thesis
    using affine_dependent_translation by auto
qed
```

lemma *affine_hull_0_dependent*:
assumes $0 \in$ *affine hull* S
shows *dependent* S
proof −
obtain s u **where** s_u: *finite* $s \wedge s \neq \{\} \wedge s \subseteq S \wedge$ *sum* u $s = 1 \wedge (\sum v{\in}s.\ u\ v *_R v) = 0$
using *assms affine_hull_explicit*[*of* S] **by** *auto*
then have $\exists v{\in}s.\ u\ v \neq 0$ **by** *auto*
then have *finite* $s \wedge s \subseteq S \wedge (\exists v{\in}s.\ u\ v \neq 0 \wedge (\sum v{\in}s.\ u\ v *_R v) = 0)$
using s_u **by** *auto*
then show *?thesis*
unfolding *dependent_explicit*[*of* S] **by** *auto*
qed

lemma *affine_dependent_imp_dependent2*:
assumes *affine_dependent* (*insert* 0 S)
shows *dependent* S
proof −
obtain x **where** x: $x \in$ *insert* 0 $S \wedge x \in$ *affine hull* (*insert* 0 $S - \{x\}$)
using *affine_dependent_def*[*of* (*insert* 0 S)] *assms* **by** *blast*
then have $x \in$ *span* (*insert* 0 $S - \{x\}$)
using *affine_hull_subset_span* **by** *auto*
moreover have *span* (*insert* 0 $S - \{x\}$) = *span* ($S - \{x\}$)
using *insert_Diff_if*[*of* 0 S $\{x\}$] *span_insert_0*[*of* $S-\{x\}$] **by** *auto*
ultimately have $x \in$ *span* ($S - \{x\}$) **by** *auto*
then have $x \neq 0 \Longrightarrow$ *dependent* S
using x *dependent_def* **by** *auto*
moreover
{
assume $x = 0$
then have $0 \in$ *affine hull* S
using x *hull_mono*[*of* $S - \{0\}$ S] **by** *auto*
then have *dependent* S
using *affine_hull_0_dependent* **by** *auto*
}
ultimately show *?thesis* **by** *auto*
qed

lemma *affine_dependent_iff_dependent*:
assumes $a \notin S$
shows *affine_dependent* (*insert* a S) $\longleftrightarrow$ *dependent* ($(\lambda x.\ -a + x)$ ` S)
proof −
have $((+)\ (-\ a)$ ` $S) = \{x - a |\ x\ .\ x \in S\}$ **by** *auto*
then show *?thesis*
using *affine_dependent_translation_eq*[*of* (*insert* a S) $-a$]
affine_dependent_imp_dependent2 assms
dependent_imp_affine_dependent[*of* a S]
by (*auto simp del*: *uminus_add_conv_diff*)
qed

lemma *affine_dependent_iff_dependent2*:
assumes $a \in S$
shows *affine_dependent* $S \longleftrightarrow$ *dependent* $((\lambda x.\ -a + x)\ `\ (S-\{a\}))$
proof −
have *insert* $a\ (S - \{a\}) = S$
using *assms* **by** *auto*
then show *?thesis*
using *assms affine_dependent_iff_dependent*[*of* $a\ S-\{a\}$] **by** *auto*
qed

lemma *affine_hull_insert_span_gen*:
affine hull $(insert\ a\ s) = (\lambda x.\ a + x)\ `\ span\ ((\lambda x.\ -\ a + x)\ `\ s)$
proof −
have *h1*: $\{x - a\ |x.\ x \in s\} = ((\lambda x.\ -a+x)\ `\ s)$
by *auto*
{
assume $a \notin s$
then have *?thesis*
using *affine_hull_insert_span*[*of* $a\ s$] *h1* **by** *auto*
}
moreover
{
assume *a1*: $a \in s$
have $\exists x.\ x \in s \wedge -a+x=0$
apply (*rule exI*[*of* _ a])
using *a1*
apply *auto*
done
then have *insert 0* $((\lambda x.\ -a+x)\ `\ (s - \{a\})) = (\lambda x.\ -a+x)\ `\ s$
by *auto*
then have *span* $((\lambda x.\ -a+x)\ `\ (s - \{a\}))$=*span* $((\lambda x.\ -a+x)\ `\ s)$
using *span_insert_0*[*of* $(+)\ (-\ a)\ `\ (s - \{a\})$] **by** (*auto simp del*: *uminus_add_conv_diff*)
moreover have $\{x - a\ |x.\ x \in (s - \{a\})\} = ((\lambda x.\ -a+x)\ `\ (s - \{a\}))$
by *auto*
moreover have $insert\ a\ (s - \{a\}) = insert\ a\ s$
by *auto*
ultimately have *?thesis*
using *affine_hull_insert_span*[*of* $a\ s-\{a\}$] **by** *auto*
}
ultimately show *?thesis* **by** *auto*
qed

lemma *affine_hull_span2*:
assumes $a \in s$
shows *affine hull* $s = (\lambda x.\ a+x)\ `\ span\ ((\lambda x.\ -a+x)\ `\ (s-\{a\}))$
using *affine_hull_insert_span_gen*[*of* $a\ s - \{a\}$, *unfolded insert_Diff*[*OF assms*]]
by *auto*

lemma *affine_hull_span_gen*:
assumes $a \in$ *affine hull s*
shows *affine hull s* $=$ $(\lambda x.\ a+x)$ ‘ *span* $((\lambda x.\ -a+x)$ ‘ $s)$
proof −
have *affine hull* (*insert a s*) $=$ *affine hull s*
using *hull_redundant*[*of a affine s*] *assms* **by** *auto*
then show *?thesis*
using *affine_hull_insert_span_gen*[*of a s*] **by** *auto*
qed

lemma *affine_hull_span_0*:
assumes $0 \in$ *affine hull S*
shows *affine hull S* $=$ *span S*
using *affine_hull_span_gen*[*of 0 S*] *assms* **by** *auto*

lemma *extend_to_affine_basis_nonempty*:
fixes $S\ V :: 'n$::*real_vector set*
assumes $\neg$ *affine_dependent S* $S \subseteq V\ S \neq \{\}$
shows $\exists T.\ \neg$ *affine_dependent* $T \wedge S \subseteq T \wedge T \subseteq V \wedge$ *affine hull* $T =$ *affine hull* V
proof −
obtain a **where** a: $a \in S$
using *assms* **by** *auto*
then have *h0*: *independent* $((\lambda x.\ -a + x)$ ‘ $(S-\{a\}))$
using *affine_dependent_iff_dependent2 assms* **by** *auto*
obtain B **where** B:
$(\lambda x.\ -a+x)$ ‘ $(S - \{a\}) \subseteq B \wedge B \subseteq (\lambda x.\ -a+x)$ ‘ $V \wedge$ *independent* $B \wedge (\lambda x.\ -a+x)$ ‘ $V \subseteq$ *span* B
using *assms*
by (*blast intro*: *maximal_independent_subset_extend*[*OF _ h0*, *of* $(\lambda x.\ -a + x)$ ‘ V])
define T **where** $T = (\lambda x.\ a+x)$ ‘ *insert 0 B*
then have $T =$ *insert a* $((\lambda x.\ a+x)$ ‘ $B)$
by *auto*
then have *affine hull* $T = (\lambda x.\ a+x)$ ‘ *span B*
using *affine_hull_insert_span_gen*[*of a* $((\lambda x.\ a+x)$ ‘ $B)$] *translation_assoc*[*of* $-a$ *a B*]
by *auto*
then have $V \subseteq$ *affine hull T*
using *B assms translation_inverse_subset*[*of a V span B*]
by *auto*
moreover have $T \subseteq V$
using *T_def B a assms* **by** *auto*
ultimately have *affine hull* $T =$ *affine hull* V
by (*metis Int_absorb1 Int_absorb2 hull_hull hull_mono*)
moreover have $S \subseteq T$
using *T_def B translation_inverse_subset*[*of a* $S-\{a\}$ *B*]
by *auto*

moreover have ¬ *affine_dependent T*
using *T_def affine_dependent_translation_eq*[*of insert 0 B*]
affine_dependent_imp_dependent2 B
by *auto*
ultimately show *?thesis* **using** ‹*T* ⊆ *V*› **by** *auto*
qed

lemma *affine_basis_exists*:
fixes *V* :: *'n*::*real_vector set*
shows ∃ *B*. *B* ⊆ *V* ∧ ¬ *affine_dependent B* ∧ *affine hull V* = *affine hull B*
proof (*cases V* = {})
case *True*
then show *?thesis*
using *affine_independent_0* **by** *auto*
next
case *False*
then obtain *x* **where** *x* ∈ *V* **by** *auto*
then show *?thesis*
using *affine_dependent_def*[*of* {*x*}] *extend_to_affine_basis_nonempty*[*of* {*x*} *V*]
by *auto*
qed

proposition *extend_to_affine_basis*:
fixes *S V* :: *'n*::*real_vector set*
assumes ¬ *affine_dependent S S* ⊆ *V*
obtains *T* **where** ¬ *affine_dependent T S* ⊆ *T T* ⊆ *V affine hull T* = *affine hull V*
proof (*cases S* = {})
case *True* **then show** *?thesis*
using *affine_basis_exists* **by** (*metis empty_subsetI that*)
next
case *False*
then show *?thesis* **by** (*metis assms extend_to_affine_basis_nonempty that*)
qed

1.6.4 Affine Dimension of a Set

definition *aff_dim* :: (*'a*::*euclidean_space*) *set* ⇒ *int*
where *aff_dim V* =
(*SOME d* :: *int*.
∃ *B*. *affine hull B* = *affine hull V* ∧ ¬ *affine_dependent B* ∧ *of_nat* (*card B*) = *d* + *1*)

lemma *aff_dim_basis_exists*:
fixes *V* :: (*'n*::*euclidean_space*) *set*
shows ∃ *B*. *affine hull B* = *affine hull V* ∧ ¬ *affine_dependent B* ∧ *of_nat* (*card B*) = *aff_dim V* + *1*
proof −

```
  obtain B where ¬ affine_dependent B ∧ affine hull B = affine hull V
    using affine_basis_exists[of V] by auto
  then show ?thesis
    unfolding aff_dim_def
      some_eq_ex[of λd. ∃ B. affine hull B = affine hull V ∧ ¬ affine_dependent
B ∧ of_nat (card B) = d + 1]
    apply auto
    apply (rule exI[of _ int (card B) − (1 :: int)])
    apply (rule exI[of _ B], auto)
    done
qed

lemma affine_hull_eq_empty [simp]: affine hull S = {} ⟷ S = {}
by (metis affine_empty subset_empty subset_hull)

lemma empty_eq_affine_hull[simp]: {} = affine hull S ⟷ S = {}
by (metis affine_hull_eq_empty)

lemma aff_dim_parallel_subspace_aux:
  fixes B :: 'n::euclidean_space set
  assumes ¬ affine_dependent B a ∈ B
  shows finite B ∧ ((card B) − 1 = dim (span ((λx. −a+x) ‘ (B−{a}))))
proof −
  have independent ((λx. −a + x) ‘ (B−{a}))
    using affine_dependent_iff_dependent2 assms by auto
  then have fin: dim (span ((λx. −a+x) ‘ (B−{a}))) = card ((λx. −a + x) ‘
(B−{a}))
    finite ((λx. −a + x) ‘ (B − {a}))
    using indep_card_eq_dim_span[of (λx. −a+x) ‘ (B−{a})] by auto
  show ?thesis
  proof (cases (λx. −a + x) ‘ (B − {a}) = {})
    case True
    have B = insert a ((λx. a + x) ‘ (λx. −a + x) ‘ (B − {a}))
      using translation_assoc[of a −a (B − {a})] assms by auto
    then have B = {a} using True by auto
    then show ?thesis using assms fin by auto
  next
    case False
    then have card ((λx. −a + x) ‘ (B − {a})) > 0
      using fin by auto
    moreover have h1: card ((λx. −a + x) ‘ (B−{a})) = card (B−{a})
      by (rule card_image) (use translate_inj_on in blast)
    ultimately have card (B−{a}) > 0 by auto
    then have ∗: finite (B − {a})
      using card_gt_0_iff[of (B − {a})] by auto
    then have card (B − {a}) = card B − 1
      using card_Diff_singleton assms by auto
    with ∗ show ?thesis using fin h1 by auto
  qed
```

qed

lemma *aff_dim_parallel_subspace*:
fixes *V L* :: *'n*::*euclidean_space set*
assumes *V* ≠ {}
and *subspace L*
and *affine_parallel* (*affine hull V*) *L*
shows *aff_dim V* = *int* (*dim L*)
proof −
obtain *B* **where**
B: *affine hull B* = *affine hull V* ∧ ¬ *affine_dependent B* ∧ *int* (*card B*) = *aff_dim V* + *1*
using *aff_dim_basis_exists* **by** *auto*
then have *B* ≠ {}
using *assms B*
by *auto*
then obtain *a* **where** *a*: *a* ∈ *B* **by** *auto*
define *Lb* **where** *Lb* = *span* ((λ*x*. −*a*+*x*) ‘ (*B*−{*a*}))
moreover have *affine_parallel* (*affine hull B*) *Lb*
using *Lb_def B assms affine_hull_span2*[*of a B*] *a*
affine_parallel_commut[*of Lb* (*affine hull B*)]
unfolding *affine_parallel_def*
by *auto*
moreover have *subspace Lb*
using *Lb_def subspace_span* **by** *auto*
moreover have *affine hull B* ≠ {}
using *assms B* **by** *auto*
ultimately have *L* = *Lb*
using *assms affine_parallel_subspace*[*of affine hull B*] *affine_affine_hull*[*of B*] *B*
by *auto*
then have *dim L* = *dim Lb*
by *auto*
moreover have *card B* − *1* = *dim Lb* **and** *finite B*
using *Lb_def aff_dim_parallel_subspace_aux a B* **by** *auto*
ultimately show *?thesis*
using *B* ‹*B* ≠ {}› *card_gt_0_iff*[*of B*] **by** *auto*
qed

lemma *aff_independent_finite*:
fixes *B* :: *'n*::*euclidean_space set*
assumes ¬ *affine_dependent B*
shows *finite B*
proof −
{
assume *B* ≠ {}
then obtain *a* **where** *a* ∈ *B* **by** *auto*
then have *?thesis*
using *aff_dim_parallel_subspace_aux assms* **by** *auto*

```
  }
  then show ?thesis by auto
qed


lemma aff_dim_empty:
  fixes S :: 'n::euclidean_space set
  shows S = {} ⟷ aff_dim S = -1
proof -
  obtain B where *: affine hull B = affine hull S
    and ¬ affine_dependent B
    and int (card B) = aff_dim S + 1
    using aff_dim_basis_exists by auto
  moreover
  from * have S = {} ⟷ B = {}
    by auto
  ultimately show ?thesis
    using aff_independent_finite[of B] card_gt_0_iff[of B] by auto
qed

lemma aff_dim_empty_eq [simp]: aff_dim ({}::'a::euclidean_space set) = -1
  by (simp add: aff_dim_empty [symmetric])

lemma aff_dim_affine_hull [simp]: aff_dim (affine hull S) = aff_dim S
  unfolding aff_dim_def using hull_hull[of _ S] by auto

lemma aff_dim_affine_hull2:
  assumes affine hull S = affine hull T
  shows aff_dim S = aff_dim T
  unfolding aff_dim_def using assms by auto

lemma aff_dim_unique:
  fixes B V :: 'n::euclidean_space set
  assumes affine hull B = affine hull V ∧ ¬ affine_dependent B
  shows of_nat (card B) = aff_dim V + 1
proof (cases B = {})
  case True
  then have V = {}
    using assms
    by auto
  then have aff_dim V = (-1::int)
    using aff_dim_empty by auto
  then show ?thesis
    using ‹B = {}› by auto
next
  case False
  then obtain a where a: a ∈ B by auto
  define Lb where Lb = span ((λx. -a+x) ` (B-{a}))
  have affine_parallel (affine hull B) Lb
```

```
    using Lb_def affine_hull_span2[of a B] a
      affine_parallel_commut[of Lb (affine hull B)]
    unfolding affine_parallel_def by auto
  moreover have subspace Lb
    using Lb_def subspace_span by auto
  ultimately have aff_dim B = int(dim Lb)
    using aff_dim_parallel_subspace[of B Lb] ‹B ≠ {}› by auto
  moreover have (card B) − 1 = dim Lb finite B
    using Lb_def aff_dim_parallel_subspace_aux a assms by auto
  ultimately have of_nat (card B) = aff_dim B + 1
    using ‹B ≠ {}› card_gt_0_iff[of B] by auto
  then show ?thesis
    using aff_dim_affine_hull2 assms by auto
qed

lemma aff_dim_affine_independent:
  fixes B :: 'n::euclidean_space set
  assumes ¬ affine_dependent B
  shows of_nat (card B) = aff_dim B + 1
  using aff_dim_unique[of B B] assms by auto

lemma affine_independent_iff_card:
    fixes s :: 'a::euclidean_space set
    shows ¬ affine_dependent s ⟷ finite s ∧ aff_dim s = int(card s) − 1
  apply (rule iffI)
  apply (simp add: aff_dim_affine_independent aff_independent_finite)
  by (metis affine_basis_exists [of s] aff_dim_unique card_subset_eq diff_add_cancel
of_nat_eq_iff)

lemma aff_dim_sing [simp]:
  fixes a :: 'n::euclidean_space
  shows aff_dim {a} = 0
  using aff_dim_affine_independent[of {a}] affine_independent_1 by auto

lemma aff_dim_2 [simp]:
  fixes a :: 'n::euclidean_space
  shows aff_dim {a,b} = (if a = b then 0 else 1)
proof (clarsimp)
  assume a ≠ b
  then have aff_dim{a,b} = card{a,b} − 1
    using affine_independent_2 [of a b] aff_dim_affine_independent by fastforce
  also have ... = 1
    using ‹a ≠ b› by simp
  finally show aff_dim {a, b} = 1 .
qed

lemma aff_dim_inner_basis_exists:
  fixes V :: ('n::euclidean_space) set
  shows ∃ B. B ⊆ V ∧ affine hull B = affine hull V ∧
```

```
    ¬ affine_dependent B ∧ of_nat (card B) = aff_dim V + 1
proof −
  obtain B where B: ¬ affine_dependent B B ⊆ V affine hull B = affine hull V
    using affine_basis_exists[of V] by auto
  then have of_nat(card B) = aff_dim V+1 using aff_dim_unique by auto
  with B show ?thesis by auto
qed

lemma aff_dim_le_card:
  fixes V :: 'n::euclidean_space set
  assumes finite V
  shows aff_dim V ≤ of_nat (card V) − 1
proof −
  obtain B where B: B ⊆ V of_nat (card B) = aff_dim V + 1
    using aff_dim_inner_basis_exists[of V] by auto
  then have card B ≤ card V
    using assms card_mono by auto
  with B show ?thesis by auto
qed

lemma aff_dim_parallel_eq:
  fixes S T :: 'n::euclidean_space set
  assumes affine_parallel (affine hull S) (affine hull T)
  shows aff_dim S = aff_dim T
proof −
  {
    assume T ≠ {} S ≠ {}
    then obtain L where L: subspace L ∧ affine_parallel (affine hull T) L
      using affine_parallel_subspace[of affine hull T]
        affine_affine_hull[of T]
      by auto
    then have aff_dim T = int (dim L)
      using aff_dim_parallel_subspace ‹T ≠ {}› by auto
    moreover have ∗: subspace L ∧ affine_parallel (affine hull S) L
      using L affine_parallel_assoc[of affine hull S affine hull T L] assms by auto
    moreover from ∗ have aff_dim S = int (dim L)
      using aff_dim_parallel_subspace ‹S ≠ {}› by auto
    ultimately have ?thesis by auto
  }
  moreover
  {
    assume S = {}
    then have S = {} and T = {}
      using assms
      unfolding affine_parallel_def
      by auto
    then have ?thesis using aff_dim_empty by auto
  }
  moreover
```

```
  {
    assume T = {}
    then have S = {} and T = {}
      using assms
      unfolding affine_parallel_def
      by auto
    then have ?thesis
      using aff_dim_empty by auto
  }
  ultimately show ?thesis by blast
qed

lemma aff_dim_translation_eq:
  aff_dim ((+) a ` S) = aff_dim S for a :: 'n::euclidean_space
proof −
  have affine_parallel (affine hull S) (affine hull ((λx. a + x) ` S))
    unfolding affine_parallel_def
    apply (rule exI[of _ a])
    using affine_hull_translation[of a S]
    apply auto
    done
  then show ?thesis
    using aff_dim_parallel_eq[of S (λx. a + x) ` S] by auto
qed

lemma aff_dim_translation_eq_subtract:
  aff_dim ((λx. x − a) ` S) = aff_dim S for a :: 'n::euclidean_space
  using aff_dim_translation_eq [of − a] by (simp cong: image_cong_simp)

lemma aff_dim_affine:
  fixes S L :: 'n::euclidean_space set
  assumes S ≠ {}
    and affine S
    and subspace L
    and affine_parallel S L
  shows aff_dim S = int (dim L)
proof −
  have ∗: affine hull S = S
    using assms affine_hull_eq[of S] by auto
  then have affine_parallel (affine hull S) L
    using assms by (simp add: ∗)
  then show ?thesis
    using assms aff_dim_parallel_subspace[of S L] by blast
qed

lemma dim_affine_hull:
  fixes S :: 'n::euclidean_space set
  shows dim (affine hull S) = dim S
proof −
```

```
  have dim (affine hull S) ≥ dim S
    using dim_subset by auto
  moreover have dim (span S) ≥ dim (affine hull S)
    using dim_subset affine_hull_subset_span by blast
  moreover have dim (span S) = dim S
    using dim_span by auto
  ultimately show ?thesis by auto
qed

lemma aff_dim_subspace:
  fixes S :: 'n::euclidean_space set
  assumes subspace S
  shows aff_dim S = int (dim S)
proof (cases S={})
  case True with assms show ?thesis
    by (simp add: subspace_affine)
next
  case False
 with aff_dim_affine[of S S] assms subspace_imp_affine[of S] affine_parallel_reflex[of
S] subspace_affine
  show ?thesis by auto
qed

lemma aff_dim_zero:
  fixes S :: 'n::euclidean_space set
  assumes 0 ∈ affine hull S
  shows aff_dim S = int (dim S)
proof −
  have subspace (affine hull S)
    using subspace_affine[of affine hull S] affine_affine_hull assms
    by auto
  then have aff_dim (affine hull S) = int (dim (affine hull S))
    using assms aff_dim_subspace[of affine hull S] by auto
  then show ?thesis
    using aff_dim_affine_hull[of S] dim_affine_hull[of S]
    by auto
qed

lemma aff_dim_eq_dim:
  aff_dim S = int (dim ((+) (− a) ' S)) if a ∈ affine hull S
    for S :: 'n::euclidean_space set
proof −
  have 0 ∈ affine hull (+) (− a) ' S
    unfolding affine_hull_translation
    using that by (simp add: ac_simps)
  with aff_dim_zero show ?thesis
    by (metis aff_dim_translation_eq)
qed
```

lemma *aff_dim_eq_dim_subtract*:
  aff_dim S = int (dim ((λx. x − a) ‘ S)) **if** *a ∈ affine hull S*
    for *S :: 'n::euclidean_space set*
  using *aff_dim_eq_dim [of a] that* **by** (*simp cong*: *image_cong_simp*)

lemma *aff_dim_UNIV [simp]*: *aff_dim (UNIV :: 'n::euclidean_space set) = int(DIM('n))*
  using *aff_dim_subspace[of (UNIV :: 'n::euclidean_space set)]*
    dim_UNIV[**where** *'a='n::euclidean_space*]
  by *auto*

lemma *aff_dim_geq*:
  fixes *V :: 'n::euclidean_space set*
  shows *aff_dim V ≥ −1*
proof −
  obtain *B* **where** *affine hull B = affine hull V*
    and ¬ *affine_dependent B*
    and *int (card B) = aff_dim V + 1*
    using *aff_dim_basis_exists* **by** *auto*
  then show *?thesis* **by** *auto*
qed

lemma *aff_dim_negative_iff [simp]*:
  fixes *S :: 'n::euclidean_space set*
  shows *aff_dim S < 0 ⟷ S = {}*
by (*metis aff_dim_empty aff_dim_geq diff_0 eq_iff zle_diff1_eq*)

lemma *aff_lowdim_subset_hyperplane*:
  fixes *S :: 'a::euclidean_space set*
  assumes *aff_dim S < DIM('a)*
  obtains *a b* **where** *a ≠ 0 S ⊆ {x. a · x = b}*
proof (*cases S={}*)
  case *True*
  moreover
  have (*SOME b. b ∈ Basis*) *≠ 0*
    by (*metis norm_some_Basis norm_zero zero_neq_one*)
  ultimately show *?thesis*
    using *that* **by** *blast*
next
  case *False*
  then obtain *c S'* **where** *c ∉ S' S = insert c S'*
    by (*meson equals0I mk_disjoint_insert*)
  have *dim ((+) (−c) ‘ S) < DIM('a)*
   by (*metis ‹S = insert c S'› aff_dim_eq_dim assms hull_inc insertI1 of_nat_less_imp_less*)
  then obtain *a* **where** *a ≠ 0 span ((+) (−c) ‘ S) ⊆ {x. a · x = 0}*
    using *lowdim_subset_hyperplane* **by** *blast*
  moreover
  have *a · w = a · c* **if** *span ((+) (− c) ‘ S) ⊆ {x. a · x = 0} w ∈ S* **for** *w*
  proof −
    have *w−c ∈ span ((+) (− c) ‘ S)*

```
    by (simp add: span_base ‹w ∈ S›)
   with that have w−c ∈ {x. a · x = 0}
    by blast
   then show ?thesis
    by (auto simp: algebra_simps)
  qed
  ultimately have S ⊆ {x. a · x = a · c}
   by blast
  then show ?thesis
   by (rule that[OF ‹a ≠ 0›])
qed

lemma affine_independent_card_dim_diffs:
  fixes S :: 'a :: euclidean_space set
  assumes ¬ affine_dependent S a ∈ S
   shows card S = dim ((λx. x − a) ‘ S) + 1
proof −
  have non: ¬ affine_dependent (insert a S)
   by (simp add: assms insert_absorb)
  have finite S
   by (meson assms aff_independent_finite)
  with ‹a ∈ S› have card S ≠ 0 by auto
  moreover have dim ((λx. x − a) ‘ S) = card S − 1
   using aff_dim_eq_dim_subtract aff_dim_unique ‹a ∈ S› hull_inc insert_absorb
non by fastforce
  ultimately show ?thesis
   by auto
qed

lemma independent_card_le_aff_dim:
  fixes B :: 'n::euclidean_space set
  assumes B ⊆ V
  assumes ¬ affine_dependent B
  shows int (card B) ≤ aff_dim V + 1
proof −
  obtain T where T: ¬ affine_dependent T ∧ B ⊆ T ∧ T ⊆ V ∧ affine hull T
= affine hull V
   by (metis assms extend_to_affine_basis[of B V])
  then have of_nat (card T) = aff_dim V + 1
   using aff_dim_unique by auto
  then show ?thesis
   using T card_mono[of T B] aff_independent_finite[of T] by auto
qed

lemma aff_dim_subset:
  fixes S T :: 'n::euclidean_space set
  assumes S ⊆ T
  shows aff_dim S ≤ aff_dim T
proof −
```

```
  obtain B where B: ¬ affine_dependent B B ⊆ S affine hull B = affine hull S
    of_nat (card B) = aff_dim S + 1
    using aff_dim_inner_basis_exists[of S] by auto
  then have int (card B) ≤ aff_dim T + 1
    using assms independent_card_le_aff_dim[of B T] by auto
  with B show ?thesis by auto
qed

lemma aff_dim_le_DIM:
  fixes S :: 'n::euclidean_space set
  shows aff_dim S ≤ int (DIM('n))
proof −
  have aff_dim (UNIV :: 'n::euclidean_space set) = int(DIM('n))
    using aff_dim_UNIV by auto
  then show aff_dim (S:: 'n::euclidean_space set) ≤ int(DIM('n))
    using aff_dim_subset[of S (UNIV :: ('n::euclidean_space) set)] subset_UNIV
by auto
qed

lemma affine_dim_equal:
  fixes S :: 'n::euclidean_space set
  assumes affine S affine T S ≠ {} S ⊆ T aff_dim S = aff_dim T
  shows S = T
proof −
  obtain a where a ∈ S using assms by auto
  then have a ∈ T using assms by auto
  define LS where LS = {y. ∃ x ∈ S. (−a) + x = y}
  then have ls: subspace LS affine_parallel S LS
    using assms parallel_subspace_explicit[of S a LS] ‹a ∈ S› by auto
  then have h1: int(dim LS) = aff_dim S
    using assms aff_dim_affine[of S LS] by auto
  have T ≠ {} using assms by auto
  define LT where LT = {y. ∃ x ∈ T. (−a) + x = y}
  then have lt: subspace LT ∧ affine_parallel T LT
    using assms parallel_subspace_explicit[of T a LT] ‹a ∈ T› by auto
  then have int(dim LT) = aff_dim T
    using assms aff_dim_affine[of T LT] ‹T ≠ {}› by auto
  then have dim LS = dim LT
    using h1 assms by auto
  moreover have LS ≤ LT
    using LS_def LT_def assms by auto
  ultimately have LS = LT
    using subspace_dim_equal[of LS LT] ls lt by auto
  moreover have S = {x. ∃ y ∈ LS. a+y=x}
    using LS_def by auto
  moreover have T = {x. ∃ y ∈ LT. a+y=x}
    using LT_def by auto
  ultimately show ?thesis by auto
qed
```

lemma *aff_dim_eq_0*:
fixes S :: *'a::euclidean_space set*
shows *aff_dim* $S = 0 \longleftrightarrow (\exists a.\ S = \{a\})$
proof (*cases* $S = \{\}$)
case *True*
then show *?thesis*
by *auto*
next
case *False*
then obtain a **where** $a \in S$ **by** *auto*
show *?thesis*
proof *safe*
assume *0*: *aff_dim* $S = 0$
have $\neg\ \{a,b\} \subseteq S$ **if** $b \neq a$ **for** b
by (*metis 0 aff_dim_2 aff_dim_subset not_one_le_zero that*)
then show $\exists a.\ S = \{a\}$
using ‹$a \in S$› **by** *blast*
qed *auto*
qed

lemma *affine_hull_UNIV*:
fixes S :: *'n::euclidean_space set*
assumes *aff_dim* S = *int*(*DIM*(*'n*))
shows *affine hull* S = (*UNIV* :: (*'n::euclidean_space*) *set*)
proof −
have $S \neq \{\}$
using *assms aff_dim_empty*[*of S*] **by** *auto*
have *h0*: $S \subseteq$ *affine hull S*
using *hull_subset*[*of S _*] **by** *auto*
have *h1*: *aff_dim* (*UNIV* :: (*'n::euclidean_space*) *set*) = *aff_dim S*
using *aff_dim_UNIV assms* **by** *auto*
then have *h2*: *aff_dim* (*affine hull S*) ≤ *aff_dim* (*UNIV* :: (*'n::euclidean_space*) *set*)
using *aff_dim_le_DIM*[*of affine hull S*] *assms h0* **by** *auto*
have *h3*: *aff_dim S* ≤ *aff_dim* (*affine hull S*)
using *h0 aff_dim_subset*[*of S affine hull S*] *assms* **by** *auto*
then have *h4*: *aff_dim* (*affine hull S*) = *aff_dim* (*UNIV* :: (*'n::euclidean_space*) *set*)
using *h0 h1 h2* **by** *auto*
then show *?thesis*
using *affine_dim_equal*[*of affine hull S* (*UNIV* :: (*'n::euclidean_space*) *set*)]
affine_affine_hull[*of S*] *affine_UNIV assms h4 h0* ‹$S \neq \{\}$›
by *auto*
qed

lemma *disjoint_affine_hull*:
fixes s :: *'n::euclidean_space set*
assumes ¬ *affine_dependent s* $t \subseteq s$ $u \subseteq s$ $t \cap u = \{\}$

shows (*affine hull t*) ∩ (*affine hull u*) = {}
proof −
from *assms*(*1*) **have** *finite s*
by (*simp add*: *aff_independent_finite*)
with *assms*(*2*,*3*) **have** *finite t finite u*
by (*blast intro*: *finite_subset*)+
have *False* **if** *y* ∈ *affine hull t* **and** *y* ∈ *affine hull u* **for** *y*
proof −
from *that* **obtain** *a b*
where *a1* [*simp*]: *sum a t* = *1*
and [*simp*]: *sum* (λ*v*. *a v* ∗*R v*) *t* = *y*
and [*simp*]: *sum b u* = *1 sum* (λ*v*. *b v* ∗*R v*) *u* = *y*
by (*auto simp*: *affine_hull_finite* ‹*finite t*› ‹*finite u*›)
define *c* **where** *c x* = (*if x* ∈ *t then a x else if x* ∈ *u then* −(*b x*) *else 0*) **for** *x*
from *assms*(*2*,*3*,*4*) **have** [*simp*]: *s* ∩ *t* = *t s* ∩ − *t* ∩ *u* = *u*
by *auto*
have *sum c s* = *0*
by (*simp add*: *c_def comm_monoid_add_class.sum.If_cases* ‹*finite s*› *sum_negf*)
moreover have ¬ (∀ *v*∈*s*. *c v* = *0*)
by (*metis* (*no_types*) *IntD1* ‹*s* ∩ *t* = *t*› *a1 c_def sum.neutral zero_neq_one*)
moreover have (∑ *v*∈*s*. *c v* ∗*R v*) = *0*
by (*simp add*: *c_def if_smult sum_negf comm_monoid_add_class.sum.If_cases* ‹*finite s*›)
ultimately show *?thesis*
using *assms*(*1*) ‹*finite s*› **by** (*auto simp*: *affine_dependent_explicit*)
qed
then show *?thesis* **by** *blast*
qed

end

1.7 Convex Sets and Functions

theory *Convex*
imports
Affine
HOL−Library.Set_Algebras
begin

1.7.1 Convex Sets

definition *convex* :: *'a*::*real_vector set* ⇒ *bool*
where *convex s* ⟷ (∀ *x*∈*s*. ∀ *y*∈*s*. ∀ *u*≥*0*. ∀ *v*≥*0*. *u* + *v* = *1* ⟶ *u* ∗*R x* + *v* ∗*R y* ∈ *s*)

lemma *convexI*:
assumes ⋀*x y u v*. *x* ∈ *s* ⟹ *y* ∈ *s* ⟹ *0* ≤ *u* ⟹ *0* ≤ *v* ⟹ *u* + *v* = *1* ⟹ *u* ∗*R x* + *v* ∗*R y* ∈ *s*
shows *convex s*

by (*simp add*: *assms convex_def*)

lemma *convexD*:
assumes *convex s* **and** $x \in s$ **and** $y \in s$ **and** $0 \leq u$ **and** $0 \leq v$ **and** $u + v = 1$
shows $u *_R x + v *_R y \in s$
using *assms* **unfolding** *convex_def* **by** *fast*

lemma *convex_alt*: *convex s* $\longleftrightarrow$ $(\forall x{\in}s.\ \forall y{\in}s.\ \forall u.\ 0 \leq u \wedge u \leq 1 \longrightarrow ((1 - u) *_R x + u *_R y) \in s)$
(**is** _ $\longleftrightarrow$ *?alt*)
by (*smt* (*verit*) *convexD convexI*)

lemma *convexD_alt*:
assumes *convex s* $a \in s$ $b \in s$ $0 \leq u$ $u \leq 1$
shows $((1 - u) *_R a + u *_R b) \in s$
using *assms* **unfolding** *convex_alt* **by** *auto*

lemma *mem_convex_alt*:
assumes *convex S* $x \in S$ $y \in S$ $u \geq 0$ $v \geq 0$ $u + v > 0$
shows $((u/(u+v)) *_R x + (v/(u+v)) *_R y) \in S$
using *assms*
by (*simp add*: *convex_def zero_le_divide_iff add_divide_distrib* [*symmetric*])

lemma *convex_empty*[*intro*,*simp*]: *convex* {}
unfolding *convex_def* **by** *simp*

lemma *convex_singleton*[*intro*,*simp*]: *convex* {*a*}
unfolding *convex_def* **by** (*auto simp*: *scaleR_left_distrib*[*symmetric*])

lemma *convex_UNIV*[*intro*,*simp*]: *convex UNIV*
unfolding *convex_def* **by** *auto*

lemma *convex_Inter*: $(\bigwedge s.\ s{\in}f \Longrightarrow \mathit{convex}\ s) \Longrightarrow \mathit{convex}(\bigcap f)$
unfolding *convex_def* **by** *auto*

lemma *convex_Int*: *convex s* $\Longrightarrow$ *convex t* $\Longrightarrow$ *convex* $(s \cap t)$
unfolding *convex_def* **by** *auto*

lemma *convex_INT*: $(\bigwedge i.\ i \in A \Longrightarrow \mathit{convex}\ (B\ i)) \Longrightarrow \mathit{convex}\ (\bigcap i{\in}A.\ B\ i)$
unfolding *convex_def* **by** *auto*

lemma *convex_Times*: *convex s* $\Longrightarrow$ *convex t* $\Longrightarrow$ *convex* $(s \times t)$
unfolding *convex_def* **by** *auto*

lemma *convex_halfspace_le*: *convex* $\{x.\ \mathit{inner}\ a\ x \leq b\}$
unfolding *convex_def*
by (*auto simp*: *inner_add intro*!: *convex_bound_le*)

lemma *convex_halfspace_ge*: *convex* $\{x.\ \mathit{inner}\ a\ x \geq b\}$

proof −
have ∗: $\{x.\ inner\ a\ x \geq b\} = \{x.\ inner\ (-a)\ x \leq -b\}$
by *auto*
show *?thesis*
unfolding ∗ **using** *convex_halfspace_le*[*of* $-a$ $-b$] **by** *auto*
qed

lemma *convex_halfspace_abs_le*: *convex* $\{x.\ |inner\ a\ x| \leq b\}$
proof −
have ∗: $\{x.\ |inner\ a\ x| \leq b\} = \{x.\ inner\ a\ x \leq b\} \cap \{x.\ -b \leq inner\ a\ x\}$
by *auto*
show *?thesis*
unfolding ∗ **by** (*simp add*: *convex_Int convex_halfspace_ge convex_halfspace_le*)
qed

lemma *convex_hyperplane*: *convex* $\{x.\ inner\ a\ x = b\}$
proof −
have ∗: $\{x.\ inner\ a\ x = b\} = \{x.\ inner\ a\ x \leq b\} \cap \{x.\ inner\ a\ x \geq b\}$
by *auto*
show *?thesis* **using** *convex_halfspace_le convex_halfspace_ge*
by (*auto intro*!: *convex_Int simp*: ∗)
qed

lemma *convex_halfspace_lt*: *convex* $\{x.\ inner\ a\ x < b\}$
unfolding *convex_def*
by (*auto simp*: *convex_bound_lt inner_add*)

lemma *convex_halfspace_gt*: *convex* $\{x.\ inner\ a\ x > b\}$
using *convex_halfspace_lt*[*of* $-a$ $-b$] **by** *auto*

lemma *convex_halfspace_Re_ge*: *convex* $\{x.\ Re\ x \geq b\}$
using *convex_halfspace_ge*[*of b 1*::*complex*] **by** *simp*

lemma *convex_halfspace_Re_le*: *convex* $\{x.\ Re\ x \leq b\}$
using *convex_halfspace_le*[*of 1*::*complex b*] **by** *simp*

lemma *convex_halfspace_Im_ge*: *convex* $\{x.\ Im\ x \geq b\}$
using *convex_halfspace_ge*[*of b* i] **by** *simp*

lemma *convex_halfspace_Im_le*: *convex* $\{x.\ Im\ x \leq b\}$
using *convex_halfspace_le*[*of* i *b*] **by** *simp*

lemma *convex_halfspace_Re_gt*: *convex* $\{x.\ Re\ x > b\}$
using *convex_halfspace_gt*[*of b 1*::*complex*] **by** *simp*

lemma *convex_halfspace_Re_lt*: *convex* $\{x.\ Re\ x < b\}$
using *convex_halfspace_lt*[*of 1*::*complex b*] **by** *simp*

lemma *convex_halfspace_Im_gt*: *convex* $\{x.\ Im\ x > b\}$

using *convex_halfspace_gt*[*of b* i] **by** *simp*

lemma *convex_halfspace_Im_lt*: *convex* {*x*. *Im x* < *b*}
using *convex_halfspace_lt*[*of* i *b*] **by** *simp*

lemma *convex_real_interval* [*iff*]:
fixes *a b* :: *real*
shows *convex* {*a*..} **and** *convex* {..*b*}
and *convex* {*a*<..} **and** *convex* {..<*b*}
and *convex* {*a*..*b*} **and** *convex* {*a*<..*b*}
and *convex* {*a*..<*b*} **and** *convex* {*a*<..<*b*}
proof −
have {*a*..} = {*x*. *a* ≤ *inner 1 x*}
by *auto*
then show *1*: *convex* {*a*..}
by (*simp only*: *convex_halfspace_ge*)
have {..*b*} = {*x*. *inner 1 x* ≤ *b*}
by *auto*
then show *2*: *convex* {..*b*}
by (*simp only*: *convex_halfspace_le*)
have {*a*<..} = {*x*. *a* < *inner 1 x*}
by *auto*
then show *3*: *convex* {*a*<..}
by (*simp only*: *convex_halfspace_gt*)
have {..<*b*} = {*x*. *inner 1 x* < *b*}
by *auto*
then show *4*: *convex* {..<*b*}
by (*simp only*: *convex_halfspace_lt*)
have {*a*..*b*} = {*a*..} ∩ {..*b*}
by *auto*
then show *convex* {*a*..*b*}
by (*simp only*: *convex_Int 1 2*)
have {*a*<..*b*} = {*a*<..} ∩ {..*b*}
by *auto*
then show *convex* {*a*<..*b*}
by (*simp only*: *convex_Int 3 2*)
have {*a*..<*b*} = {*a*..} ∩ {..<*b*}
by *auto*
then show *convex* {*a*..<*b*}
by (*simp only*: *convex_Int 1 4*)
have {*a*<..<*b*} = {*a*<..} ∩ {..<*b*}
by *auto*
then show *convex* {*a*<..<*b*}
by (*simp only*: *convex_Int 3 4*)
qed

lemma *convex_Reals*: *convex* ℝ
by (*simp add*: *convex_def scaleR_conv_of_real*)

1.7.2 Explicit expressions for convexity in terms of arbitrary sums

lemma *convex_sum*:
fixes $C :: 'a::real\_vector\ set$
assumes *finite* S
and *convex* C
and a: $(\sum\ i \in S.\ a\ i) = 1$ $\bigwedge i.\ i \in S \Longrightarrow a\ i \geq 0$
and C: $\bigwedge i.\ i \in S \Longrightarrow y\ i \in C$
shows $(\sum\ j \in S.\ a\ j *_R y\ j) \in C$
using ‹*finite* S› a C
proof (*induction arbitrary*: a *set*: *finite*)
case *empty*
then show *?case* **by** *simp*
next
case (*insert* i S)
then have $0 \leq sum\ a\ S$
by (*simp add*: *sum_nonneg*)
have $a\ i *_R y\ i + (\sum j{\in}S.\ a\ j *_R y\ j) \in C$
proof (*cases* $sum\ a\ S = 0$)
case *True* **with** *insert* **show** *?thesis*
by (*simp add*: *sum_nonneg_eq_0_iff*)
next
case *False*
with ‹$0 \leq sum\ a\ S$› **have** $0 < sum\ a\ S$
by *simp*
then have $(\sum j{\in}S.\ (a\ j\ /\ sum\ a\ S) *_R y\ j) \in C$
using *insert*
by (*simp add*: *insert.IH flip*: *sum_divide_distrib*)
with ‹*convex* C› *insert* ‹$0 \leq sum\ a\ S$›
have $a\ i *_R y\ i + sum\ a\ S *_R (\sum j{\in}S.\ (a\ j\ /\ sum\ a\ S) *_R y\ j) \in C$
by (*simp add*: *convex_def*)
then show *?thesis*
by (*simp add*: *scaleR_sum_right False*)
qed
then show *?case* **using** ‹*finite* S› **and** ‹$i \notin S$›
by *simp*
qed

lemma *convex*:
convex $S \longleftrightarrow$
$(\forall (k::nat)\ u\ x.\ (\forall i.\ 1{\leq}i \wedge i{\leq}k \longrightarrow 0 \leq u\ i \wedge x\ i \in S) \wedge (sum\ u\ \{1..k\} = 1)$
$\longrightarrow sum\ (\lambda i.\ u\ i *_R x\ i)\ \{1..k\} \in S)$
(**is** *?lhs* = *?rhs*)
proof
show *?lhs* $\Longrightarrow$ *?rhs*
by (*metis* (*full_types*) *atLeastAtMost_iff convex_sum finite_atLeastAtMost*)
assume $*$: $\forall k\ u\ x.\ (\forall\ i :: nat.\ 1 \leq i \wedge i \leq k \longrightarrow 0 \leq u\ i \wedge x\ i \in S) \wedge sum\ u\ \{1..k\} = 1$
$\longrightarrow (\sum i = 1..k.\ u\ i *_R (x\ i :: 'a)) \in S$

```
  {
    fix μ :: real
    fix x y :: 'a
    assume xy: x ∈ S y ∈ S
    assume mu: μ ≥ 0 μ ≤ 1
    let ?u = λi. if (i :: nat) = 1 then μ else 1 − μ
    let ?x = λi. if (i :: nat) = 1 then x else y
    have {1 :: nat .. 2} ∩ − {x. x = 1} = {2}
      by auto
    then have S: (∑ j ∈ {1..2}. ?u j *R ?x j) ∈ S
      using sum.If_cases[of {(1 :: nat) .. 2} λx. x = 1 λx. μ λx. 1 − μ]
      using mu xy ∗ by auto
    have grarr: (∑ j ∈ {Suc (Suc 0)..2}. ?u j *R ?x j) = (1 − μ) *R y
     using sum.atLeast_Suc_atMost[of Suc (Suc 0) 2 λ j. (1 − μ) *R y] by auto
    with sum.atLeast_Suc_atMost
    have (∑ j ∈ {1..2}. ?u j *R ?x j) = μ *R x + (1 − μ) *R y
      by (smt (verit, best) Suc_1 Suc_eq_plus1 add_0 le_add1)
    then have (1 − μ) *R y + μ *R x ∈ S
      using S by (auto simp: add.commute)
  }
  then show convex S
    unfolding convex_alt by auto
qed


lemma convex_explicit:
  fixes S :: 'a::real_vector set
  shows convex S ⟷
    (∀ t u. finite t ∧ t ⊆ S ∧ (∀ x∈t. 0 ≤ u x) ∧ sum u t = 1 ⟶ sum (λx. u x *R
x) t ∈ S)
proof safe
  fix t
  fix u :: 'a ⇒ real
  assume convex S
    and finite t
    and t ⊆ S ∀ x∈t. 0 ≤ u x sum u t = 1
  then show (∑ x∈t. u x *R x) ∈ S
    by (simp add: convex_sum subsetD)
next
  assume ∗: ∀ t. ∀ u. finite t ∧ t ⊆ S ∧ (∀ x∈t. 0 ≤ u x) ∧
    sum u t = 1 ⟶ (∑ x∈t. u x *R x) ∈ S
  show convex S
    unfolding convex_alt
  proof safe
    fix x y
    fix μ :: real
    assume ∗∗: x ∈ S y ∈ S 0 ≤ μ μ ≤ 1
    show (1 − μ) *R x + μ *R y ∈ S
    proof (cases x = y)
```

```
    case False
    then show ?thesis
      using *[rule_format, of {x, y} λ z. if z = x then 1 − μ else μ] **
      by auto
  next
    case True
    then show ?thesis
      using *[rule_format, of {x, y} λ z. 1] **
      by (auto simp: field_simps real_vector.scale_left_diff_distrib)
  qed
 qed
qed

lemma convex_finite:
 assumes finite S
 shows convex S ⟷ (∀ u. (∀ x∈S. 0 ≤ u x) ∧ sum u S = 1 ⟶ sum (λx. u x
*R x) S ∈ S)
     (is ?lhs = ?rhs)
proof
 { have if_distrib_arg: ⋀P f g x. (if P then f else g) x = (if P then f x else g x)
    by simp
   fix T :: 'a set and u :: 'a ⇒ real
   assume sum: ∀ u. (∀ x∈S. 0 ≤ u x) ∧ sum u S = 1 ⟶ (∑ x∈S. u x *R x) ∈
S
   assume *: ∀ x∈T. 0 ≤ u x sum u T = 1
   assume T ⊆ S
   then have S ∩ T = T by auto
   with sum[THEN spec[where x=λx. if x∈T then u x else 0]] *
   have (∑ x∈T. u x *R x) ∈ S
    by (auto simp: assms sum.If_cases if_distrib if_distrib_arg) }
 moreover assume ?rhs
 ultimately show ?lhs
   unfolding convex_explicit by auto
qed (auto simp: convex_explicit assms)
```

1.7.3 Convex Functions on a Set

```
definition convex_on :: 'a::real_vector set ⇒ ('a ⇒ real) ⇒ bool
 where convex_on S f ⟷
   (∀ x∈S. ∀ y∈S. ∀ u≥0. ∀ v≥0. u + v = 1 ⟶ f (u *R x + v *R y) ≤ u * f x
+ v * f y)

definition concave_on :: 'a::real_vector set ⇒ ('a ⇒ real) ⇒ bool
 where concave_on S f ≡ convex_on S (λx. − f x)

lemma concave_on_iff:
 concave_on S f ⟷
   (∀ x∈S. ∀ y∈S. ∀ u≥0. ∀ v≥0. u + v = 1 ⟶ f (u *R x + v *R y) ≥ u * f x
+ v * f y)
```

by (*auto simp*: *concave_on_def convex_on_def algebra_simps*)

lemma *convex_onI* [*intro?*]:
assumes $\bigwedge t\ x\ y.\ t > 0 \Longrightarrow t < 1 \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow$
$f\ ((1 - t) *_R x + t *_R y) \leq (1 - t) * f\ x + t * f\ y$
shows *convex_on A f*
unfolding *convex_on_def*
by (*smt* (*verit*, *del_insts*) *assms mult_cancel_right1 mult_eq_0_iff scaleR_collapse scaleR_eq_0_iff*)

lemma *convex_on_linorderI* [*intro?*]:
fixes $A :: ('a::\{linorder,real\_vector\})\ set$
assumes $\bigwedge t\ x\ y.\ t > 0 \Longrightarrow t < 1 \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow x < y \Longrightarrow$
$f\ ((1 - t) *_R x + t *_R y) \leq (1 - t) * f\ x + t * f\ y$
shows *convex_on A f*
by (*smt* (*verit*, *best*) *add.commute assms convex_onI distrib_left linorder_cases mult.commute mult_cancel_left2 scaleR_collapse*)

lemma *convex_onD*:
assumes *convex_on A f*
shows $\bigwedge t\ x\ y.\ t \geq 0 \Longrightarrow t \leq 1 \Longrightarrow x \in A \Longrightarrow y \in A \Longrightarrow$
$f\ ((1 - t) *_R x + t *_R y) \leq (1 - t) * f\ x + t * f\ y$
using *assms* **by** (*auto simp*: *convex_on_def*)

lemma *convex_onD_Icc*:
assumes $convex\_on\ \{x..y\}\ f\ x \leq (y :: \_ :: \{real\_vector,preorder\})$
shows $\bigwedge t.\ t \geq 0 \Longrightarrow t \leq 1 \Longrightarrow$
$f\ ((1 - t) *_R x + t *_R y) \leq (1 - t) * f\ x + t * f\ y$
using *assms*(*2*) **by** (*intro convex_onD* [*OF assms*(*1*)]) *simp_all*

lemma *convex_on_subset*: *convex_on t f* $\Longrightarrow S \subseteq t \Longrightarrow$ *convex_on S f*
unfolding *convex_on_def* **by** *auto*

lemma *convex_on_add* [*intro*]:
assumes *convex_on S f*
and *convex_on S g*
shows *convex_on S* $(\lambda x.\ f\ x + g\ x)$
proof −
{
fix *x y*
assume $x \in S\ y \in S$
moreover
fix *u v* :: *real*
assume $0 \leq u\ 0 \leq v\ u + v = 1$
ultimately
have $f\ (u *_R x + v *_R y) + g\ (u *_R x + v *_R y) \leq (u * f\ x + v * f\ y) + (u * g\ x + v * g\ y)$
using *assms* **unfolding** *convex_on_def* **by** (*auto simp*: *add_mono*)
then have $f\ (u *_R x + v *_R y) + g\ (u *_R x + v *_R y) \leq u * (f\ x + g\ x) + v$

```
* (f y + g y)
      by (simp add: field_simps)
  }
  then show ?thesis
    unfolding convex_on_def by auto
qed

lemma convex_on_cmul [intro]:
  fixes c :: real
  assumes 0 ≤ c
    and convex_on S f
  shows convex_on S (λx. c * f x)
proof −
  have *: u * (c * fx) + v * (c * fy) = c * (u * fx + v * fy)
    for u c fx v fy :: real
    by (simp add: field_simps)
  show ?thesis using assms(2) and mult_left_mono [OF _ assms(1)]
    unfolding convex_on_def and * by auto
qed

lemma convex_lower:
  assumes convex_on S f
    and x ∈ S
    and y ∈ S
    and 0 ≤ u
    and 0 ≤ v
    and u + v = 1
  shows f (u *R x + v *R y) ≤ max (f x) (f y)
proof −
  let ?m = max (f x) (f y)
  have u * f x + v * f y ≤ u * max (f x) (f y) + v * max (f x) (f y)
    using assms(4,5) by (auto simp: mult_left_mono add_mono)
  also have ... = max (f x) (f y)
    using assms(6) by (simp add: distrib_right [symmetric])
  finally show ?thesis
    using assms unfolding convex_on_def by fastforce
qed

lemma convex_on_dist [intro]:
  fixes S :: 'a::real_normed_vector set
  shows convex_on S (λx. dist a x)
proof (clarsimp simp: convex_on_def dist_norm)
  fix x y
  assume x ∈ S y ∈ S
  fix u v :: real
  assume 0 ≤ u
  assume 0 ≤ v
  assume u + v = 1
  have a = u *R a + v *R a
```

by (*metis* ‹$u + v = 1$› *scaleR_left.add scaleR_one*)
then have $a - (u *_R x + v *_R y) = (u *_R (a - x)) + (v *_R (a - y))$
by (*auto simp*: *algebra_simps*)
then show *norm* $(a - (u *_R x + v *_R y)) \leq u * norm\ (a - x) + v * norm\ (a - y)$
by (*smt* (*verit*, *best*) ‹$0 \leq u$› ‹$0 \leq v$› *norm_scaleR norm_triangle_ineq*)
qed

1.7.4 Arithmetic operations on sets preserve convexity

lemma *convex_linear_image*:
assumes *linear f* **and** *convex S*
shows *convex* $(f\ `\ S)$
proof −
interpret *f*: *linear f* **by** *fact*
from ‹*convex S*› **show** *convex* $(f\ `\ S)$
by (*simp add*: *convex_def f.scaleR* [*symmetric*] *f.add* [*symmetric*])
qed

lemma *convex_linear_vimage*:
assumes *linear f* **and** *convex S*
shows *convex* $(f -` S)$
proof −
interpret *f*: *linear f* **by** *fact*
from ‹*convex S*› **show** *convex* $(f -` S)$
by (*simp add*: *convex_def f.add f.scaleR*)
qed

lemma *convex_scaling*:
assumes *convex S*
shows *convex* $((\lambda x.\ c *_R x)\ `\ S)$
by (*simp add*: *assms convex_linear_image*)

lemma *convex_scaled*:
assumes *convex S*
shows *convex* $((\lambda x.\ x *_R c)\ `\ S)$
by (*simp add*: *assms convex_linear_image*)

lemma *convex_negations*:
assumes *convex S*
shows *convex* $((\lambda x.\ -\ x)\ `\ S)$
by (*simp add*: *assms convex_linear_image module_hom_uminus*)

lemma *convex_sums*:
assumes *convex S*
and *convex T*
shows *convex* $(\bigcup x \in S.\ \bigcup y \in T.\ \{x + y\})$
proof −
have *linear* $(\lambda(x, y).\ x + y)$

```
    by (auto intro: linearI simp: scaleR_add_right)
  with assms have convex ((λ(x, y). x + y) ‘ (S × T))
    by (intro convex_linear_image convex_Times)
  also have ((λ(x, y). x + y) ‘ (S × T)) = (⋃x∈ S. ⋃y ∈ T. {x + y})
    by auto
  finally show ?thesis .
qed

lemma convex_differences:
  assumes convex S convex T
  shows convex (⋃x∈ S. ⋃y ∈ T. {x − y})
proof −
  have {x − y| x y. x ∈ S ∧ y ∈ T} = {x + y |x y. x ∈ S ∧ y ∈ uminus ‘ T}
    by (auto simp: diff_conv_add_uminus simp del: add_uminus_conv_diff)
  then show ?thesis
    using convex_sums[OF assms(1) convex_negations[OF assms(2)]] by auto
qed

lemma convex_translation:
  convex ((+) a ‘ S) if convex S
proof −
  have (⋃ x∈ {a}. ⋃y ∈ S. {x + y}) = (+) a ‘ S
    by auto
  then show ?thesis
    using convex_sums [OF convex_singleton [of a] that] by auto
qed

lemma convex_translation_subtract:
  convex ((λb. b − a) ‘ S) if convex S
  using convex_translation [of S − a] that by (simp cong: image_cong_simp)

lemma convex_affinity:
  assumes convex S
  shows convex ((λx. a + c *R x) ‘ S)
proof −
  have (λx. a + c *R x) ‘ S = (+) a ‘ (*R) c ‘ S
    by auto
  then show ?thesis
    using convex_translation[OF convex_scaling[OF assms], of a c] by auto
qed

lemma convex_on_sum:
  fixes a :: 'a ⇒ real
    and y :: 'a ⇒ 'b::real_vector
    and f :: 'b ⇒ real
  assumes finite S S ≠ {}
    and convex_on C f
    and convex C
    and (∑ i ∈ S. a i) = 1
```

```
    and ⋀i. i ∈ S ⟹ a i ≥ 0
    and ⋀i. i ∈ S ⟹ y i ∈ C
  shows f (∑ i ∈ S. a i *R y i) ≤ (∑ i ∈ S. a i * f (y i))
  using assms
proof (induct S arbitrary: a rule: finite_ne_induct)
  case (singleton i)
  then show ?case
    by auto
next
  case (insert i S)
  then have yai: y i ∈ C a i ≥ 0
    by auto
  with insert have conv: ⋀x y μ. x ∈ C ⟹ y ∈ C ⟹ 0 ≤ μ ⟹ μ ≤ 1 ⟹
      f (μ *R x + (1 - μ) *R y) ≤ μ * f x + (1 - μ) * f y
    by (simp add: convex_on_def)
  show ?case
  proof (cases a i = 1)
    case True
    with insert have (∑ j ∈ S. a j) = 0
      by auto
    with insert show ?thesis
      by (simp add: sum_nonneg_eq_0_iff)
  next
    case False
    then have ai1: a i < 1
      using sum_nonneg_leq_bound[of insert i S a] insert by force
    then have i0: 1 - a i > 0
      by auto
    let ?a = λj. a j / (1 - a i)
    have a_nonneg: ?a j ≥ 0 if j ∈ S for j
      using i0 insert that by fastforce
    have (∑ j ∈ insert i S. a j) = 1
      using insert by auto
    then have (∑ j ∈ S. a j) = 1 - a i
      using sum.insert insert by fastforce
    then have (∑ j ∈ S. a j) / (1 - a i) = 1
      using i0 by auto
    then have a1: (∑ j ∈ S. ?a j) = 1
      unfolding sum_divide_distrib by simp
    have asum: (∑ j ∈ S. ?a j *R y j) ∈ C
      using insert convex_sum [OF ‹finite S› ‹convex C› a1 a_nonneg] by auto
    have asum_le: f (∑ j ∈ S. ?a j *R y j) ≤ (∑ j ∈ S. ?a j * f (y j))
      using a_nonneg a1 insert by blast
    have f (∑ j ∈ insert i S. a j *R y j) = f ((∑ j ∈ S. a j *R y j) + a i *R y i)
      by (simp add: add.commute insert.hyps)
    also have ... = f (((1 - a i) * inverse (1 - a i)) *R (∑ j ∈ S. a j *R y j)
+ a i *R y i)
      using i0 by auto
    also have ... = f ((1 - a i) *R (∑ j ∈ S. (a j * inverse (1 - a i)) *R y j)
```

```
+ a i *R y i)
      using scaleR_right.sum[of inverse (1 − a i) λ j. a j *R y j S, symmetric]
      by (auto simp: algebra_simps)
    also have ... = f ((1 − a i) *R (∑ j ∈ S. ?a j *R y j) + a i *R y i)
      by (auto simp: divide_inverse)
    also have ... ≤ (1 − a i) *R f ((∑ j ∈ S. ?a j *R y j)) + a i * f (y i)
      using ai1 by (smt (verit) asum conv real_scaleR_def yai)
    also have ... ≤ (1 − a i) * (∑ j ∈ S. ?a j * f (y j)) + a i * f (y i)
      using asum_le i0 by fastforce
    also have ... = (∑ j ∈ S. a j * f (y j)) + a i * f (y i)
      using i0 by (auto simp: sum_distrib_left)
    finally show ?thesis
      using insert by auto
  qed
qed

lemma convex_on_alt:
  fixes C :: 'a::real_vector set
  shows convex_on C f ⟷
        (∀ x ∈ C. ∀ y ∈ C. ∀ μ :: real. μ ≥ 0 ∧ μ ≤ 1 ⟶
         f (μ *R x + (1 − μ) *R y) ≤ μ * f x + (1 − μ) * f y)
  by (smt (verit) convex_on_def)

lemma convex_on_diff:
  fixes f :: real ⇒ real
  assumes f: convex_on I f
    and I: x ∈ I y ∈ I
    and t: x < t t < y
  shows (f x − f t) / (x − t) ≤ (f x − f y) / (x − y)
    and (f x − f y) / (x − y) ≤ (f t − f y) / (t − y)
proof −
  define a where a ≡ (t − y) / (x − y)
  with t have 0 ≤ a 0 ≤ 1 − a
    by (auto simp: field_simps)
  with f ‹x ∈ I› ‹y ∈ I› have cvx: f (a * x + (1 − a) * y) ≤ a * f x + (1 − a)
* f y
    by (auto simp: convex_on_def)
  have a * x + (1 − a) * y = a * (x − y) + y
    by (simp add: field_simps)
  also have ... = t
    unfolding a_def using ‹x < t› ‹t < y› by simp
  finally have f t ≤ a * f x + (1 − a) * f y
    using cvx by simp
  also have ... = a * (f x − f y) + f y
    by (simp add: field_simps)
  finally have f t − f y ≤ a * (f x − f y)
    by simp
  with t show (f x − f t) / (x − t) ≤ (f x − f y) / (x − y)
    by (simp add: le_divide_eq divide_le_eq field_simps a_def)
```

```
  with t show (f x − f y) / (x − y) ≤ (f t − f y) / (t − y)
    by (simp add: le_divide_eq divide_le_eq field_simps)
qed

lemma pos_convex_function:
  fixes f :: real ⇒ real
  assumes convex C
    and leq: ⋀x y. x ∈ C ⟹ y ∈ C ⟹ f' x * (y − x) ≤ f y − f x
  shows convex_on C f
  unfolding convex_on_alt
  using assms
proof safe
  fix x y μ :: real
  let ?x = μ *R x + (1 − μ) *R y
  assume *: convex C x ∈ C y ∈ C μ ≥ 0 μ ≤ 1
  then have 1 − μ ≥ 0 by auto
  then have xpos: ?x ∈ C
    using * unfolding convex_alt by fastforce
  have geq: μ * (f x − f ?x) + (1 − μ) * (f y − f ?x) ≥
      μ * f' ?x * (x − ?x) + (1 − μ) * f' ?x * (y − ?x)
    using add_mono [OF mult_left_mono [OF leq [OF xpos *(2)] ‹μ ≥ 0›]
        mult_left_mono [OF leq [OF xpos *(3)] ‹1 − μ ≥ 0›]]
    by auto
  then have μ * f x + (1 − μ) * f y − f ?x ≥ 0
    by (auto simp: field_simps)
  then show f (μ *R x + (1 − μ) *R y) ≤ μ * f x + (1 − μ) * f y
    by auto
qed

lemma atMostAtLeast_subset_convex:
  fixes C :: real set
  assumes convex C
    and x ∈ C y ∈ C x < y
  shows {x .. y} ⊆ C
proof safe
  fix z assume z: z ∈ {x .. y}
  have less: z ∈ C if *: x < z z < y
  proof −
    let ?μ = (y − z) / (y − x)
    have 0 ≤ ?μ ?μ ≤ 1
      using assms * by (auto simp: field_simps)
    then have comb: ?μ * x + (1 − ?μ) * y ∈ C
      using assms iffD1[OF convex_alt, rule_format, of C y x ?μ]
      by (simp add: algebra_simps)
    have ?μ * x + (1 − ?μ) * y = (y − z) * x / (y − x) + (1 − (y − z) / (y −
x)) * y
      by (auto simp: field_simps)
    also have ... = ((y − z) * x + (y − x − (y − z)) * y) / (y − x)
      using assms by (simp only: add_divide_distrib) (auto simp: field_simps)
```

```
    also have ... = z
      using assms by (auto simp: field_simps)
    finally show ?thesis
      using comb by auto
  qed
  show z ∈ C
    using z less assms by (auto simp: le_less)
qed

lemma f''_imp_f':
  fixes f :: real ⇒ real
  assumes convex C
    and f': ⋀x. x ∈ C ⟹ DERIV f x :> (f' x)
    and f'': ⋀x. x ∈ C ⟹ DERIV f' x :> (f'' x)
    and pos: ⋀x. x ∈ C ⟹ f'' x ≥ 0
    and x: x ∈ C
    and y: y ∈ C
  shows f' x * (y - x) ≤ f y - f x
  using assms
proof -
  have f y - f x ≥ f' x * (y - x) f' y * (x - y) ≤ f x - f y
    if *: x ∈ C y ∈ C y > x for x y :: real
  proof -
    from * have ge: y - x > 0 y - x ≥ 0 and le: x - y < 0 x - y ≤ 0
      by auto
    then obtain z1 where z1: z1 > x z1 < y f y - f x = (y - x) * f' z1
      using subsetD[OF atMostAtLeast_subset_convex[OF ‹convex C› ‹x ∈ C› ‹y
∈ C› ‹x < y›],
          THEN f', THEN MVT2[OF ‹x < y›, rule_format, unfolded atLeastAt-
Most_iff[symmetric]]]
      by auto
    then have z1 ∈ C
      using atMostAtLeast_subset_convex ‹convex C› ‹x ∈ C› ‹y ∈ C› ‹x < y›
      by fastforce
    obtain z2 where z2: z2 > x z2 < z1 f' z1 - f' x = (z1 - x) * f'' z2
     using subsetD[OF atMostAtLeast_subset_convex[OF ‹convex C› ‹x ∈ C› ‹z1
∈ C› ‹x < z1›],
        THEN f'', THEN MVT2[OF ‹x < z1›, rule_format, unfolded atLeastAt-
Most_iff[symmetric]]] z1
      by auto
    obtain z3 where z3: z3 > z1 z3 < y f' y - f' z1 = (y - z1) * f'' z3
      using subsetD[OF atMostAtLeast_subset_convex[OF ‹convex C› ‹z1 ∈ C›
‹y ∈ C› ‹z1 < y›],
        THEN f'', THEN MVT2[OF ‹z1 < y›, rule_format, unfolded atLeastAt-
Most_iff[symmetric]]] z1
      by auto
    from z1 have f x - f y = (x - y) * f' z1
      by (simp add: field_simps)
    then have cool': f' y - (f x - f y) / (x - y) = (y - z1) * f'' z3
```

```
    using le(1) z3(3) by auto
  have z3 ∈ C
    using z3 * atMostAtLeast_subset_convex ‹convex C› ‹x ∈ C› ‹z1 ∈ C› ‹x
< z1›
    by fastforce
  then have B': f'' z3 ≥ 0
    using assms by auto
  with cool' have f' y − (f x − f y) / (x − y) ≥ 0
    using z1 by auto
  then have res: f' y * (x − y) ≤ f x − f y
    by (meson diff_ge_0_iff_ge le(1) neg_divide_le_eq)
  have cool: (f y − f x) / (y − x) − f' x = (z1 − x) * f'' z2
    using le(1) z1(3) z2(3) by auto
  have z2 ∈ C
    using z2 z1 * atMostAtLeast_subset_convex ‹convex C› ‹z1 ∈ C› ‹y ∈ C›
‹z1 < y›
    by fastforce
  with z1 assms have (z1 − x) * f'' z2 ≥ 0
    by auto
  then show f y − f x ≥ f' x * (y − x) f' y * (x − y) ≤ f x − f y
    using that(3) z1(3) res cool by auto
 qed
 then show ?thesis
  using x y by fastforce
qed

lemma f''_ge0_imp_convex:
 fixes f :: real ⇒ real
 assumes convex C
  and ⋀x. x ∈ C ⟹ DERIV f x :> (f' x)
  and ⋀x. x ∈ C ⟹ DERIV f' x :> (f'' x)
  and ⋀x. x ∈ C ⟹ f'' x ≥ 0
 shows convex_on C f
 by (metis assms f''_imp_f' pos_convex_function)

lemma f''_le0_imp_concave:
 fixes f :: real ⇒ real
 assumes convex C
  and ⋀x. x ∈ C ⟹ DERIV f x :> (f' x)
  and ⋀x. x ∈ C ⟹ DERIV f' x :> (f'' x)
  and ⋀x. x ∈ C ⟹ f'' x ≤ 0
 shows concave_on C f
 unfolding concave_on_def
 by (rule assms f''_ge0_imp_convex derivative_eq_intros | simp)+

lemma log_concave:
 fixes b :: real
 assumes b > 1
 shows concave_on {0<..} (λ x. log b x)
```

using *assms*
by (*intro f''_le0_imp_concave derivative_eq_intros* | *simp*)+

lemma *ln_concave*: *concave_on* {*0*<..} *ln*
unfolding *log_ln* **by** (*simp add*: *log_concave*)

lemma *minus_log_convex*:
fixes *b* :: *real*
assumes *b* > *1*
shows *convex_on* {*0* <..} (λ *x*. − *log b x*)
using *assms concave_on_def log_concave* **by** *blast*

lemma *powr_convex*:
assumes *p* ≥ *1* **shows** *convex_on* {*0*<..} (λ*x*. *x powr p*)
using *assms*
by (*intro f''_ge0_imp_convex derivative_eq_intros* | *simp*)+

lemma *exp_convex*: *convex_on UNIV exp*
by (*intro f''_ge0_imp_convex derivative_eq_intros* | *simp*)+

1.7.5 Convexity of real functions

lemma *convex_on_realI*:
assumes *connected A*
and ⋀*x*. *x* ∈ *A* ⟹ (*f has_real_derivative f' x*) (*at x*)
and ⋀*x y*. *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ *x* ≤ *y* ⟹ *f' x* ≤ *f' y*
shows *convex_on A f*
proof (*rule convex_on_linorderI*)
fix *t x y* :: *real*
assume *t*: *t* > *0 t* < *1*
assume *xy*: *x* ∈ *A y* ∈ *A x* < *y*
define *z* **where** *z* = (*1* − *t*) ∗ *x* + *t* ∗ *y*
with ‹*connected A*› **and** *xy* **have** *ivl*: {*x*..*y*} ⊆ *A*
using *connected_contains_Icc* **by** *blast*

from *xy t* **have** *xz*: *z* > *x*
by (*simp add*: *z_def algebra_simps*)
have *y* − *z* = (*1* − *t*) ∗ (*y* − *x*)
by (*simp add*: *z_def algebra_simps*)
also from *xy t* **have** ... > *0*
by (*intro mult_pos_pos*) *simp_all*
finally have *yz*: *z* < *y*
by *simp*

from *assms xz yz ivl t* **have** ∃ξ. ξ > *x* ∧ ξ < *z* ∧ *f z* − *f x* = (*z* − *x*) ∗ *f'* ξ
by (*intro MVT2*) (*auto intro*!: *assms*(*2*))
then obtain ξ **where** ξ: ξ > *x* ξ < *z f'* ξ = (*f z* − *f x*) / (*z* − *x*)
by *auto*
from *assms xz yz ivl t* **have** ∃η. η > *z* ∧ η < *y* ∧ *f y* − *f z* = (*y* − *z*) ∗ *f'* η

```
    by (intro MVT2) (auto intro!: assms(2))
  then obtain η where η: η > z η < y f' η = (f y - f z) / (y - z)
    by auto

  from η(3) have (f y - f z) / (y - z) = f' η ..
  also from ξ η ivl have ξ ∈ A η ∈ A
    by auto
  with ξ η have f' η ≥ f' ξ
    by (intro assms(3)) auto
  also from ξ(3) have f' ξ = (f z - f x) / (z - x) .
  finally have (f y - f z) * (z - x) ≥ (f z - f x) * (y - z)
    using xz yz by (simp add: field_simps)
  also have z - x = t * (y - x)
    by (simp add: z_def algebra_simps)
  also have y - z = (1 - t) * (y - x)
    by (simp add: z_def algebra_simps)
  finally have (f y - f z) * t ≥ (f z - f x) * (1 - t)
    using xy by simp
  then show (1 - t) * f x + t * f y ≥ f ((1 - t) *R x + t *R y)
    by (simp add: z_def algebra_simps)
qed

lemma convex_on_inverse:
  fixes A :: real set
  assumes A ⊆ {0<..}
  shows convex_on A inverse
proof -
  have convex_on {0::real<..} inverse
  proof (intro convex_on_realI)
    fix u v :: real
    assume u ∈ {0<..} v ∈ {0<..} u ≤ v
    with assms show -inverse (u^2) ≤ -inverse (v^2)
      by simp
  next
    show ⋀x. x ∈ {0<..} ⟹ (inverse has_real_derivative - inverse (x^2)) (at x)
      by (rule derivative_eq_intros | simp add: power2_eq_square)+
  qed auto
  then show ?thesis
    using assms convex_on_subset by blast
qed

lemma convex_onD_Icc':
  assumes convex_on {x..y} f c ∈ {x..y}
  defines d ≡ y - x
  shows f c ≤ (f y - f x) / d * (c - x) + f x
proof (cases x y rule: linorder_cases)
  case less
  then have d: d > 0
    by (simp add: d_def)
```

from *assms*(*2*) *less* **have** *A*: *0* ≤ (*c* − *x*) / *d* (*c* − *x*) / *d* ≤ *1*
by (*simp_all add*: *d_def field_split_simps*)
have *f c* = *f* (*x* + (*c* − *x*) ∗ *1*)
by *simp*
also from *less* **have** *1* = ((*y* − *x*) / *d*)
by (*simp add*: *d_def*)
also from *d* **have** *x* + (*c* − *x*) ∗ ... = (*1* − (*c* − *x*) / *d*) ∗*R* *x* + ((*c* − *x*) / *d*) ∗*R* *y*
by (*simp add*: *field_simps*)
also have *f* ... ≤ (*1* − (*c* − *x*) / *d*) ∗ *f x* + (*c* − *x*) / *d* ∗ *f y*
using *assms less* **by** (*intro convex_onD_Icc*) *simp_all*
also from *d* **have** ... = (*f y* − *f x*) / *d* ∗ (*c* − *x*) + *f x*
by (*simp add*: *field_simps*)
finally show *?thesis* **.**
qed (*use assms* **in** *auto*)

lemma *convex_onD_Icc′*:
assumes *convex_on* {*x*..*y*} *f c* ∈ {*x*..*y*}
defines *d* ≡ *y* − *x*
shows *f c* ≤ (*f x* − *f y*) / *d* ∗ (*y* − *c*) + *f y*
proof (*cases x y rule*: *linorder_cases*)
case *less*
then have *d*: *d* > *0*
by (*simp add*: *d_def*)
from *assms*(*2*) *less* **have** *A*: *0* ≤ (*y* − *c*) / *d* (*y* − *c*) / *d* ≤ *1*
by (*simp_all add*: *d_def field_split_simps*)
have *f c* = *f* (*y* − (*y* − *c*) ∗ *1*)
by *simp*
also from *less* **have** *1* = ((*y* − *x*) / *d*)
by (*simp add*: *d_def*)
also from *d* **have** *y* − (*y* − *c*) ∗ ... = (*1* − (*1* − (*y* − *c*) / *d*)) ∗*R* *x* + (*1* − (*y* − *c*) / *d*) ∗*R* *y*
by (*simp add*: *field_simps*)
also have *f* ... ≤ (*1* − (*1* − (*y* − *c*) / *d*)) ∗ *f x* + (*1* − (*y* − *c*) / *d*) ∗ *f y*
using *assms less* **by** (*intro convex_onD_Icc*) (*simp_all add*: *field_simps*)
also from *d* **have** ... = (*f x* − *f y*) / *d* ∗ (*y* − *c*) + *f y*
by (*simp add*: *field_simps*)
finally show *?thesis* **.**
qed (*use assms* **in** *auto*)

1.7.6 Some inequalities

lemma *Youngs_inequality_0*:
fixes *a*::*real*
assumes *0* ≤ *α* *0* ≤ *β* *α*+*β* = *1* *a*>*0* *b*>*0*
shows *a powr α* ∗ *b powr β* ≤ *α*∗*a* + *β*∗*b*
proof −
have *α* ∗ *ln a* + *β* ∗ *ln b* ≤ *ln* (*α* ∗ *a* + *β* ∗ *b*)
using *assms ln_concave* **by** (*simp add*: *concave_on_iff*)

moreover have $0 < \alpha * a + \beta * b$
using *assms* **by** (*smt* (*verit*) *mult_pos_pos split_mult_pos_le*)
ultimately show *?thesis*
using *assms* **by** (*simp add*: *powr_def mult_exp_exp flip*: *ln_ge_iff*)
qed

lemma *Youngs_inequality*:
fixes *p*::*real*
assumes $p{>}1$ $q{>}1$ $1/p + 1/q = 1$ $a{\geq}0$ $b{\geq}0$
shows $a * b \leq a \text{ powr } p / p + b \text{ powr } q / q$
proof (*cases* $a{=}0 \vee b{=}0$)
case *False*
then show *?thesis*
using *Youngs_inequality_0* [*of* $1/p$ $1/q$ *a powr p b powr q*] *assms*
by (*simp add*: *powr_powr*)
qed (*use assms* **in** *auto*)

lemma *Cauchy_Schwarz_ineq_sum*:
fixes *a* :: *'a* ⇒ *'b*::*linordered_field*
shows $(\sum i{\in}I.\ a\ i * b\ i)^2 \leq (\sum i{\in}I.\ (a\ i)^2) * (\sum i{\in}I.\ (b\ i)^2)$
proof (*cases* $(\sum i{\in}I.\ (b\ i)^2) > 0$)
case *False*
then consider $\bigwedge i.\ i{\in}I \Longrightarrow b\ i = 0 \mid$ *infinite I*
by (*metis* (*mono_tags*, *lifting*) *sum_pos2 zero_le_power2 zero_less_power2*)
thus *?thesis*
by *fastforce*
next
case *True*
define *r* **where** $r \equiv (\sum i{\in}I.\ a\ i * b\ i) / (\sum i{\in}I.\ (b\ i)^2)$
with *True* **have** $*$: $(\sum i{\in}I.\ a\ i * b\ i) = r * (\sum i{\in}I.\ (b\ i)^2)$
by *simp*
have $0 \leq (\sum i{\in}I.\ (a\ i - r * b\ i)^2)$
by (*meson sum_nonneg zero_le_power2*)
also have ... $= (\sum i{\in}I.\ (a\ i)^2) - 2 * r * (\sum i{\in}I.\ a\ i * b\ i) + r^2 * (\sum i{\in}I.\ (b\ i)^2)$
by (*simp add*: *algebra_simps power2_eq_square sum_distrib_left flip*: *sum.distrib*)
also have $\ldots = (\sum i{\in}I.\ (a\ i)^2) - (\sum i{\in}I.\ a\ i * b\ i) * r$
by (*simp add*: $*$ *power2_eq_square*)
also have $\ldots = (\sum i{\in}I.\ (a\ i)^2) - ((\sum i{\in}I.\ a\ i * b\ i))^2 / (\sum i{\in}I.\ (b\ i)^2)$
by (*simp add*: *r_def power2_eq_square*)
finally have $0 \leq (\sum i{\in}I.\ (a\ i)^2) - ((\sum i{\in}I.\ a\ i * b\ i))^2 / (\sum i{\in}I.\ (b\ i)^2)$.
hence $((\sum i{\in}I.\ a\ i * b\ i))^2 / (\sum i{\in}I.\ (b\ i)^2) \leq (\sum i{\in}I.\ (a\ i)^2)$
by (*simp add*: *le_diff_eq*)
thus $((\sum i{\in}I.\ a\ i * b\ i))^2 \leq (\sum i{\in}I.\ (a\ i)^2) * (\sum i{\in}I.\ (b\ i)^2)$
by (*simp add*: *pos_divide_le_eq True*)
qed

1.7.7 Misc related lemmas

lemma *convex_translation_eq* [*simp*]:
convex ((+) *a* ‘ *s*) ⟷ *convex s*
by (*metis convex_translation translation_galois*)

lemma *convex_translation_subtract_eq* [*simp*]:
convex ((λ*b*. *b* − *a*) ‘ *s*) ⟷ *convex s*
using *convex_translation_eq* [*of* − *a*] **by** (*simp cong*: *image_cong_simp*)

lemma *convex_linear_image_eq* [*simp*]:
fixes *f* :: ′*a*::*real_vector* ⇒ ′*b*::*real_vector*
shows ⟦*linear f*; *inj f*⟧ ⟹ *convex* (*f* ‘ *s*) ⟷ *convex s*
by (*metis* (*no_types*) *convex_linear_image convex_linear_vimage inj_vimage_image_eq*)

lemma *vector_choose_size*:
assumes 0 ≤ *c*
obtains *x* :: ′*a*::{*real_normed_vector*, *perfect_space*} **where** *norm x* = *c*
proof −
obtain *a*::′*a* **where** *a* ≠ 0
using *UNIV_not_singleton UNIV_eq_I set_zero singletonI* **by** *fastforce*
then show *?thesis*
by (*rule_tac x*=*scaleR* (*c* / *norm a*) *a* **in** *that*) (*simp add*: *assms*)
qed

lemma *vector_choose_dist*:
assumes 0 ≤ *c*
obtains *y* :: ′*a*::{*real_normed_vector*, *perfect_space*} **where** *dist x y* = *c*
by (*metis add_diff_cancel_left′ assms dist_commute dist_norm vector_choose_size*)

lemma *sum_delta′′*:
fixes *s*::′*a*::*real_vector set*
assumes *finite s*
shows (∑ *x*∈*s*. (*if y* = *x then f x else* 0) ∗*R x*) = (*if y*∈*s then* (*f y*) ∗*R y else* 0)
proof −
have ∗: ⋀*x y*. (*if y* = *x then f x else* (0::*real*)) ∗*R x* = (*if x*=*y then* (*f x*) ∗*R x*
else 0)
by *auto*
show *?thesis*
unfolding ∗ **using** *sum.delta*[*OF assms*, *of y* λ*x*. *f x* ∗*R x*] **by** *auto*
qed

1.7.8 Cones

definition *cone* :: ′*a*::*real_vector set* ⇒ *bool*
where *cone s* ⟷ (∀ *x*∈*s*. ∀ *c*≥0. *c* ∗*R x* ∈ *s*)

lemma *cone_empty*[*intro*, *simp*]: *cone* {}
unfolding *cone_def* **by** *auto*

lemma *cone_univ*[*intro*, *simp*]: *cone UNIV*
unfolding *cone_def* **by** *auto*

lemma *cone_Inter*[*intro*]: $\forall s{\in}f.\ cone\ s \Longrightarrow cone\ (\bigcap f)$
unfolding *cone_def* **by** *auto*

lemma *subspace_imp_cone*: *subspace S* $\Longrightarrow$ *cone S*
by (*simp add*: *cone_def subspace_scale*)

Conic hull

lemma *cone_cone_hull*: *cone* (*cone hull S*)
unfolding *hull_def* **by** *auto*

lemma *cone_hull_eq*: *cone hull S* $= S \longleftrightarrow$ *cone S*
by (*metis cone_cone_hull hull_same*)

lemma *mem_cone*:
assumes *cone S x* $\in S$ $c \geq 0$
shows $c *_R x \in S$
using *assms cone_def*[*of S*] **by** *auto*

lemma *cone_contains_0*:
assumes *cone S*
shows $S \neq \{\} \longleftrightarrow 0 \in S$
using *assms mem_cone* **by** *fastforce*

lemma *cone_0*: *cone* $\{0\}$
unfolding *cone_def* **by** *auto*

lemma *cone_Union*[*intro*]: $(\forall s{\in}f.\ cone\ s) \longrightarrow cone\ (\bigcup f)$
unfolding *cone_def* **by** *blast*

lemma *cone_iff*:
assumes $S \neq \{\}$
shows *cone S* $\longleftrightarrow 0 \in S \wedge (\forall c.\ c > 0 \longrightarrow ((*_R)\ c)\ `\ S = S)$ (**is** _ = *?rhs*)
proof
assume *cone S*
{
fix *c* :: *real*
assume $c > 0$
have $x \in ((*_R)\ c)\ `\ S$ **if** $x \in S$ **for** x
using ‹*cone S*› ‹$c{>}0$› *mem_cone*[*of S x* $1/c$] *that*
exI[*of* $(\lambda t.\ t \in S \wedge x = c *_R t)$ $(1\ /\ c) *_R x$]
by *auto*
then have $((*_R)\ c)\ `\ S = S$
using ‹$0 < c$› ‹*cone S*› *mem_cone* **by** *fastforce*
}
then show $0 \in S \wedge (\forall c.\ c > 0 \longrightarrow ((*_R)\ c)\ `\ S = S)$

```
    using ‹cone S› cone_contains_0[of S] assms by auto
next
  show ?rhs ⟹ cone S
    by (metis Convex.cone_def imageI order_neq_le_trans scaleR_zero_left)
qed

lemma cone_hull_empty: cone hull {} = {}
  by (metis cone_empty cone_hull_eq)

lemma cone_hull_empty_iff: S = {} ⟷ cone hull S = {}
  by (metis cone_hull_empty hull_subset subset_empty)

lemma cone_hull_contains_0: S ≠ {} ⟷ 0 ∈ cone hull S
  by (metis cone_cone_hull cone_contains_0 cone_hull_empty_iff)

lemma mem_cone_hull:
  assumes x ∈ S c ≥ 0
  shows c *R x ∈ cone hull S
  by (metis assms cone_cone_hull hull_inc mem_cone)

proposition cone_hull_expl: cone hull S = {c *R x | c x. c ≥ 0 ∧ x ∈ S}
  (is ?lhs = ?rhs)
proof
  have ?rhs ∈ Collect cone
    using Convex.cone_def by fastforce
  moreover have S ⊆ ?rhs
    by (smt (verit) mem_Collect_eq scaleR_one subsetI)
  ultimately show ?lhs ⊆ ?rhs
    using hull_minimal by blast
qed (use mem_cone_hull in auto)

lemma convex_cone:
  convex S ∧ cone S ⟷ (∀ x∈S. ∀ y∈S. (x + y) ∈ S) ∧ (∀ x∈S. ∀ c≥0. (c *R x)
∈ S)
  (is ?lhs = ?rhs)
proof −
  {
    fix x y
    assume x∈S y∈S and ?lhs
    then have 2 *R x ∈S 2 *R y ∈ S convex S
      unfolding cone_def by auto
    then have x + y ∈ S
      using convexD [OF ‹convex S›, of 2*R x 2*R y]
    by (smt (verit, ccfv_threshold) field_sum_of_halves scaleR_2 scaleR_half_double)
  }
  then show ?thesis
    unfolding convex_def cone_def by blast
qed
```

1.7.9 Connectedness of convex sets

lemma *convex_connected*:
fixes $S :: 'a::real\_normed\_vector\ set$
assumes *convex S*
shows *connected S*
proof (*rule connectedI*)
fix *A B*
assume *open A open B A ∩ B ∩ S = {} S ⊆ A ∪ B*
moreover
assume *A ∩ S ≠ {} B ∩ S ≠ {}*
then obtain *a b* **where** *a*: *a ∈ A a ∈ S* **and** *b*: *b ∈ B b ∈ S* **by** *auto*
define *f* **where** [*abs_def*]: *f u = u *$_R$ *a + (1 − u) *$_R$ *b* **for** *u*
then have *continuous_on {0 .. 1} f*
by (*auto intro*!: *continuous_intros*)
then have *connected (f ‘ {0 .. 1})*
by (*auto intro*!: *connected_continuous_image*)
note *connectedD*[*OF this*, *of A B*]
moreover have *a ∈ A ∩ f ‘ {0 .. 1}*
using *a* **by** (*auto intro*!: *image_eqI*[*of _ _ 1*] *simp*: *f_def*)
moreover have *b ∈ B ∩ f ‘ {0 .. 1}*
using *b* **by** (*auto intro*!: *image_eqI*[*of _ _ 0*] *simp*: *f_def*)
moreover have *f ‘ {0 .. 1} ⊆ S*
using ‹*convex S*› *a b* **unfolding** *convex_def f_def* **by** *auto*
ultimately show *False* **by** *auto*
qed

corollary *connected_UNIV*[*intro*]: *connected* (*UNIV* :: $'a::real\_normed\_vector\ set$)
by (*simp add*: *convex_connected*)

lemma *convex_prod*:
assumes $\bigwedge i.\ i \in Basis \Longrightarrow convex\ \{x.\ P\ i\ x\}$
shows *convex* $\{x.\ \forall i{\in}Basis.\ P\ i\ (x{\cdot}i)\}$
using *assms* **by** (*auto simp*: *inner_add_left convex_def*)

lemma *convex_positive_orthant*: *convex* $\{x::'a::euclidean\_space.\ (\forall i{\in}Basis.\ 0 \le x{\cdot}i)\}$
by (*rule convex_prod*) (*simp flip*: *atLeast_def*)

1.7.10 Convex hull

lemma *convex_convex_hull* [*iff*]: *convex* (*convex hull s*)
by (*metis* (*mono_tags*) *convex_Inter hull_def mem_Collect_eq*)

lemma *convex_hull_subset*:
s ⊆ convex hull t ⟹ convex hull s ⊆ convex hull t
by (*simp add*: *subset_hull*)

lemma *convex_hull_eq*: *convex hull s = s ⟷ convex s*
by (*metis convex_convex_hull hull_same*)

Convex hull is "preserved" by a linear function

lemma *convex_hull_linear_image*:
assumes *f*: *linear f*
shows *f ‘ (convex hull S) = convex hull (f ‘ S)*
proof
show *convex hull (f ‘ S) ⊆ f ‘ (convex hull S)*
by (*intro hull_minimal image_mono hull_subset convex_linear_image assms convex_convex_hull*)
show *f ‘ (convex hull S) ⊆ convex hull (f ‘ S)*
by (*meson convex_convex_hull convex_linear_vimage f hull_minimal hull_subset image_subset_iff_subset_vimage*)
qed

lemma *in_convex_hull_linear_image*:
assumes *linear f x ∈ convex hull S*
shows *f x ∈ convex hull (f ‘ S)*
using *assms convex_hull_linear_image image_eqI* **by** *blast*

lemma *convex_hull_Times*:
convex hull (S × T) = (convex hull S) × (convex hull T)
proof
show *convex hull (S × T) ⊆ (convex hull S) × (convex hull T)*
by (*intro hull_minimal Sigma_mono hull_subset convex_Times convex_convex_hull*)
have *(x, y) ∈ convex hull (S × T)* **if** *x*: *x ∈ convex hull S* **and** *y*: *y ∈ convex hull T* **for** *x y*
proof (*rule hull_induct* [*OF x*], *rule hull_induct* [*OF y*])
fix *x y* **assume** *x ∈ S* **and** *y ∈ T*
then show *(x, y) ∈ convex hull (S × T)*
by (*simp add*: *hull_inc*)
next
fix *x* **let** *?S = ((λy. (0, y)) −‘ (λp. (− x, 0) + p) ‘ (convex hull S × T))*
have *convex ?S*
by (*intro convex_linear_vimage convex_translation convex_convex_hull*, *simp add*: *linear_iff*)
also have *?S = {y. (x, y) ∈ convex hull (S × T)}*
by (*auto simp*: *image_def Bex_def*)
finally show *convex {y. (x, y) ∈ convex hull (S × T)}* **.**
next
show *convex {x. (x, y) ∈ convex hull S × T}*
proof −
fix *y* **let** *?S = ((λx. (x, 0)) −‘ (λp. (0, − y) + p) ‘ (convex hull S × T))*
have *convex ?S*
by (*intro convex_linear_vimage convex_translation convex_convex_hull*, *simp add*: *linear_iff*)
also have *?S = {x. (x, y) ∈ convex hull (S × T)}*
by (*auto simp*: *image_def Bex_def*)
finally show *convex {x. (x, y) ∈ convex hull (S × T)}* **.**
qed
qed

then show (*convex hull S*) × (*convex hull T*) ⊆ *convex hull* (*S* × *T*)
unfolding *subset_eq split_paired_Ball_Sigma* **by** *blast*
qed

Stepping theorems for convex hulls of finite sets

lemma *convex_hull_empty*[*simp*]: *convex hull* {} = {}
by (*simp add*: *hull_same*)

lemma *convex_hull_singleton*[*simp*]: *convex hull* {*a*} = {*a*}
by (*simp add*: *hull_same*)

lemma *convex_hull_insert*:
fixes *S* :: *'a*::*real_vector set*
assumes *S* ≠ {}
shows *convex hull* (*insert a S*) =
{*x*. ∃ *u*≥*0*. ∃ *v*≥*0*. ∃ *b*. (*u* + *v* = *1*) ∧ *b* ∈ (*convex hull S*) ∧ (*x* = *u* ∗*R* *a* + *v* ∗*R* *b*)}
(**is** _ = *?hull*)
proof (*intro equalityI hull_minimal subsetI*)
fix *x*
assume *x* ∈ *insert a S*
then show *x* ∈ *?hull*
unfolding *insert_iff*
proof
assume *x* = *a*
then show *?thesis*
by (*smt* (*verit*, *del_insts*) *add.right_neutral assms ex_in_conv hull_inc mem_Collect_eq scaleR_one scaleR_zero_left*)
next
assume *x* ∈ *S*
with *hull_subset* **show** *?thesis*
by *force*
qed
next
fix *x*
assume *x* ∈ *?hull*
then obtain *u v b* **where** *obt*: *u*≥*0 v*≥*0 u* + *v* = *1 b* ∈ *convex hull S x* = *u* ∗*R* *a* + *v* ∗*R* *b*
by *auto*
have *a* ∈ *convex hull insert a S b* ∈ *convex hull insert a S*
using *hull_mono*[*of S insert a S convex*] *hull_mono*[*of* {*a*} *insert a S convex*] **and** *obt*(*4*)
by *auto*
then show *x* ∈ *convex hull insert a S*
unfolding *obt*(*5*) **using** *obt*(*1*−*3*)
by (*rule convexD* [*OF convex_convex_hull*])
next
show *convex ?hull*

```
  proof (rule convexI)
    fix x y u v
    assume as: (0::real) ≤ u 0 ≤ v u + v = 1 and x: x ∈ ?hull and y: y ∈ ?hull
    from x obtain u1 v1 b1 where
      obt1: u1≥0 v1≥0 u1 + v1 = 1 b1 ∈ convex hull S and xeq: x = u1 *R a +
v1 *R b1
      by auto
    from y obtain u2 v2 b2 where
      obt2: u2≥0 v2≥0 u2 + v2 = 1 b2 ∈ convex hull S and yeq: y = u2 *R a +
v2 *R b2
      by auto
    have *: ⋀(x::'a) s1 s2. x - s1 *R x - s2 *R x = ((1::real) - (s1 + s2)) *R x
      by (auto simp: algebra_simps)
    have ∃ b ∈ convex hull S. u *R x + v *R y =
      (u * u1) *R a + (v * u2) *R a + (b - (u * u1) *R b - (v * u2) *R b)
    proof (cases u * v1 + v * v2 = 0)
      case True
      have *: ⋀(x::'a) s1 s2. x - s1 *R x - s2 *R x = ((1::real) - (s1 + s2)) *R
x
        by (auto simp: algebra_simps)
      have eq0: u * v1 = 0 v * v2 = 0
      using True mult_nonneg_nonneg[OF ‹u≥0› ‹v1≥0›] mult_nonneg_nonneg[OF
‹v≥0› ‹v2≥0›]
        by arith+
      then have u * u1 + v * u2 = 1
        using as(3) obt1(3) obt2(3) by auto
      then show ?thesis
        using * eq0 as obt1(4) xeq yeq by auto
    next
      case False
      have 1 - (u * u1 + v * u2) = (u + v) - (u * u1 + v * u2)
        by (simp add: as(3))
      also have ... = u * v1 + v * v2
        by (smt (verit, ccfv_SIG) distrib_left mult_cancel_left1 obt1(3) obt2(3))
      finally have **:1 - (u * u1 + v * u2) = u * v1 + v * v2 .
      let ?b = ((u * v1) / (u * v1 + v * v2)) *R b1 + ((v * v2) / (u * v1 + v *
v2)) *R b2
      have zeroes: 0 ≤ u * v1 + v * v2 0 ≤ u * v1 0 ≤ u * v1 + v * v2 0 ≤ v *
v2
        using as obt1 obt2 by auto
      show ?thesis
      proof
        show u *R x + v *R y = (u * u1) *R a + (v * u2) *R a + (?b - (u * u1)
*R ?b - (v * u2) *R ?b)
          unfolding xeq yeq * **
          using False by (auto simp: scaleR_left_distrib scaleR_right_distrib)
        show ?b ∈ convex hull S
            using False mem_convex_alt obt1(4) obt2(4) zeroes(2) zeroes(4) by
fastforce
```

```
      qed
    qed
    then obtain b where b: b ∈ convex hull S
      u *R x + v *R y = (u * u1) *R a + (v * u2) *R a + (b − (u * u1) *R b −
(v * u2) *R b) ..
    obtain u1: u1 ≤ 1 and u2: u2 ≤ 1
      using obt1 obt2 by auto
    have u1 * u + u2 * v ≤ max u1 u2 * u + max u1 u2 * v
      by (smt (verit, ccfv_SIG) as mult_right_mono)
    also have ... ≤ 1
      unfolding distrib_left[symmetric] and as(3) using u1 u2 by auto
    finally have le1: u1 * u + u2 * v ≤ 1 .
    show u *R x + v *R y ∈ ?hull
    proof (intro CollectI exI conjI)
      show 0 ≤ u * u1 + v * u2
        by (simp add: as obt1(1) obt2(1))
      show 0 ≤ 1 − u * u1 − v * u2
        by (simp add: le1 diff_diff_add mult.commute)
    qed (use b in ‹auto simp: algebra_simps›)
  qed
qed

lemma convex_hull_insert_alt:
   convex hull (insert a S) =
     (if S = {} then {a}
      else {(1 − u) *R a + u *R x |x u. 0 ≤ u ∧ u ≤ 1 ∧ x ∈ convex hull S})
  apply (simp add: convex_hull_insert)
  using diff_add_cancel diff_ge_0_iff_ge
  by (smt (verit, del_insts) Collect_cong)
```

Explicit expression for convex hull

```
proposition convex_hull_indexed:
  fixes S :: 'a::real_vector set
  shows convex hull S =
    {y. ∃k u x. (∀ i∈{1::nat .. k}. 0 ≤ u i ∧ x i ∈ S) ∧
              (sum u {1..k} = 1) ∧ (∑ i = 1..k. u i *R x i) = y}
    (is ?xyz = ?hull)
proof (rule hull_unique [OF _ convexI])
  show S ⊆ ?hull
    by (clarsimp, rule_tac x=1 in exI, rule_tac x=λx. 1 in exI, auto)
next
  fix T
  assume S ⊆ T convex T
  then show ?hull ⊆ T
    by (blast intro: convex_sum)
next
  fix x y u v
  assume uv: 0 ≤ u 0 ≤ v u + v = (1::real)
```

assume *xy*: $x \in$ *?hull* $y \in$ *?hull*
from *xy* **obtain** *k1 u1 x1* **where**
x [*rule_format*]: $\forall i{\in}\{1{::}nat..k1\}.\ 0{\le}u1\ i \wedge x1\ i \in S$
$sum\ u1\ \{Suc\ 0..k1\} = 1\ (\sum i = Suc\ 0..k1.\ u1\ i *_R x1\ i) = x$
by *auto*
from *xy* **obtain** *k2 u2 x2* **where**
y [*rule_format*]: $\forall i{\in}\{1{::}nat..k2\}.\ 0{\le}u2\ i \wedge x2\ i \in S$
$sum\ u2\ \{Suc\ 0..k2\} = 1\ (\sum i = Suc\ 0..k2.\ u2\ i *_R x2\ i) = y$
by *auto*
have $*$: $\bigwedge P\ (x{::}'a)\ y\ s\ t\ i.\ (if\ P\ i\ then\ s\ else\ t) *_R (if\ P\ i\ then\ x\ else\ y) = (if\ P\ i\ then\ s *_R x\ else\ t *_R y)$
$\{1..k1 + k2\} \cap \{1..k1\} = \{1..k1\}\ \{1..k1 + k2\} \cap - \{1..k1\} = (\lambda i.\ i + k1)\ `\ \{1..k2\}$
by *auto*
have *inj*: *inj_on* $(\lambda i.\ i + k1)\ \{1..k2\}$
unfolding *inj_on_def* **by** *auto*
let *?uu* $= \lambda i.\ if\ i \in \{1..k1\}\ then\ u * u1\ i\ else\ v * u2\ (i - k1)$
let *?xx* $= \lambda i.\ if\ i \in \{1..k1\}\ then\ x1\ i\ else\ x2\ (i - k1)$
show $u *_R x + v *_R y \in$ *?hull*
proof (*intro CollectI exI conjI ballI*)
show $0 \le$ *?uu i ?xx* $i \in S$ **if** $i \in \{1..k1{+}k2\}$ **for** i
using *that* **by** (*auto simp add*: *le_diff_conv uv(1) x(1) uv(2) y(1)*)
show $(\sum i = 1..k1 + k2.$ *?uu i*$) = 1\ (\sum i = 1..k1 + k2.$ *?uu i* $*_R$ *?xx i*$) = u *_R x + v *_R y$
unfolding $*$ *sum.If_cases*[*OF finite_atLeastAtMost*[*of 1 k1 + k2*]]
sum.reindex[*OF inj*] *Collect_mem_eq o_def*
unfolding *scaleR_scaleR*[*symmetric*] *scaleR_right.sum* [*symmetric*] *sum_distrib_left*[*symmetric*]
by (*simp_all add*: *sum_distrib_left*[*symmetric*] *x(2,3) y(2,3) uv(3)*)
qed
qed

lemma *convex_hull_finite*:
fixes $S :: 'a{::}real\_vector\ set$
assumes *finite S*
shows *convex hull* $S = \{y.\ \exists u.\ (\forall x{\in}S.\ 0 \le u\ x) \wedge sum\ u\ S = 1 \wedge sum\ (\lambda x.\ u\ x *_R x)\ S = y\}$
(**is** *?HULL* $= \_$)
proof (*rule hull_unique* [*OF* $\_$ *convexI*]; *clarify*)
fix x
assume $x \in S$
then show $\exists u.\ (\forall x{\in}S.\ 0 \le u\ x) \wedge sum\ u\ S = 1 \wedge (\sum x{\in}S.\ u\ x *_R x) = x$
by (*rule_tac* $x{=}\lambda y.\ if\ x{=}y\ then\ 1\ else\ 0$ **in** *exI*) (*auto simp*: *sum.delta'*[*OF assms*] *sum_delta''*[*OF assms*])
next
fix $u\ v :: real$
assume uv: $0 \le u\ 0 \le v\ u + v = 1$
fix ux **assume** ux [*rule_format*]: $\forall x{\in}S.\ 0 \le ux\ x\ sum\ ux\ S = (1{::}real)$
fix uy **assume** uy [*rule_format*]: $\forall x{\in}S.\ 0 \le uy\ x\ sum\ uy\ S = (1{::}real)$
have $0 \le u * ux\ x + v * uy\ x$ **if** $x{\in}S$ **for** x

by (*simp add*: *that uv ux*(*1*) *uy*(*1*))
moreover
have $(\sum x{\in}S.\ u * ux\ x + v * uy\ x) = 1$
unfolding *sum.distrib* **and** *sum_distrib_left*[*symmetric*] *ux*(*2*) *uy*(*2*)
using *uv*(*3*) **by** *auto*
moreover
have $(\sum x{\in}S.\ (u * ux\ x + v * uy\ x) *_R x) = u *_R (\sum x{\in}S.\ ux\ x *_R x) + v *_R (\sum x{\in}S.\ uy\ x *_R x)$
unfolding *scaleR_left_distrib sum.distrib scaleR_scaleR*[*symmetric*] *scaleR_right.sum*[*symmetric*]
by *auto*
ultimately
show $\exists uc.\ (\forall x{\in}S.\ 0 \le uc\ x) \wedge sum\ uc\ S = 1 \wedge (\sum x{\in}S.\ uc\ x *_R x) = u *_R (\sum x{\in}S.\ ux\ x *_R x) + v *_R (\sum x{\in}S.\ uy\ x *_R x)$
by (*rule_tac x*=$\lambda x.\ u * ux\ x + v * uy\ x$ **in** *exI*, *auto*)
qed (*use assms* **in** ‹*auto simp*: *convex_explicit*›)

Another formulation

Formalized by Lars Schewe.

lemma *convex_hull_explicit*:
fixes *p* :: *'a*::*real_vector set*
shows *convex hull p* =
$\{y.\ \exists S\ u.\ finite\ S \wedge S \subseteq p \wedge (\forall x{\in}S.\ 0 \le u\ x) \wedge sum\ u\ S = 1 \wedge sum\ (\lambda v.\ u\ v *_R v)\ S = y\}$
(**is** *?lhs* = *?rhs*)
proof (*intro subset_antisym subsetI*)
fix *x*
assume $x \in convex\ hull\ p$
then obtain *k u y* **where**
obt: $\forall i{\in}\{1{::}nat..k\}.\ 0 \le u\ i \wedge y\ i \in p\ sum\ u\ \{1..k\} = 1\ (\sum i = 1..k.\ u\ i *_R y\ i) = x$
unfolding *convex_hull_indexed* **by** *auto*
have *fin*: *finite* $\{1..k\}$ **by** *auto*
{
fix *j*
assume $j{\in}\{1..k\}$
then have $y\ j \in p \wedge 0 \le sum\ u\ \{i.\ Suc\ 0 \le i \wedge i \le k \wedge y\ i = y\ j\}$
by (*metis* (*mono_tags*, *lifting*) *One_nat_def atLeastAtMost_iff mem_Collect_eq obt*(*1*) *sum_nonneg*)
}
moreover have $(\sum v{\in}y\ `\ \{1..k\}.\ sum\ u\ \{i \in \{1..k\}.\ y\ i = v\}) = 1$
unfolding *sum.image_gen*[*OF fin*, *symmetric*] **using** *obt*(*2*) **by** *auto*
moreover have $(\sum v{\in}y\ `\ \{1..k\}.\ sum\ u\ \{i \in \{1..k\}.\ y\ i = v\} *_R v) = x$
using *sum.image_gen*[*OF fin*, *of* $\lambda i.\ u\ i *_R y\ i\ y$, *symmetric*]
unfolding *scaleR_left.sum* **using** *obt*(*3*) **by** *auto*
ultimately
have $\exists S\ u.\ finite\ S \wedge S \subseteq p \wedge (\forall x{\in}S.\ 0 \le u\ x) \wedge sum\ u\ S = 1 \wedge (\sum v{\in}S.\ u$

$v *_R v) = x$
by (*smt* (*verit*, *ccfv_SIG*) *imageE mem_Collect_eq obt*(*1*) *subsetI sum.cong sum.infinite sum_nonneg*)
then show $x \in$ *?rhs* **by** *auto*
next
fix y
assume $y \in$ *?rhs*
then obtain S u **where**
S: *finite* S $S \subseteq p$ $\forall x{\in}S.\ 0 \le u\ x$ *sum* u $S = 1$ $(\sum v{\in}S.\ u\ v *_R v) = y$
by *auto*
obtain f **where** f: *inj_on* f $\{1..card\ S\}$ f ' $\{1..card\ S\} = S$
using *ex_bij_betw_nat_finite_1*[*OF* $S(1)$] **unfolding** *bij_betw_def* **by** *auto*
then have $0 \le u\ (f\ i)$ $f\ i \in p$ **if** $i \in \{1..card\ S\}$ **for** i
using S ‹$i \in \{1..card\ S\}$› **by** *blast*+
moreover
{
fix y
assume $y{\in}S$
then obtain i **where** $i{\in}\{1..card\ S\}$ $f\ i = y$
by (*metis* $f(2)$ *image_iff*)
then have $\{x.\ Suc\ 0 \le x \wedge x \le card\ S \wedge f\ x = y\} = \{i\}$
using $f(1)$ *inj_onD* **by** *fastforce*
then have $(\sum x{\in}\{x \in \{1..card\ S\}.\ f\ x = y\}.\ u\ (f\ x)) = u\ y$
$(\sum x{\in}\{x \in \{1..card\ S\}.\ f\ x = y\}.\ u\ (f\ x) *_R f\ x) = u\ y *_R y$
by (*simp_all add*: *sum_constant_scaleR* ‹$f\ i = y$›)
}
then have $(\sum x = 1..card\ S.\ u\ (f\ x)) = 1$ $(\sum i = 1..card\ S.\ u\ (f\ i) *_R f\ i) = y$
by (*metis* (*mono_tags*, *lifting*) $S(4,5)$ *f sum.reindex_cong*)+
ultimately
show $y \in$ *convex hull p*
unfolding *convex_hull_indexed*
by (*smt* (*verit*, *del_insts*) *mem_Collect_eq sum.cong*)
qed

A stepping theorem for that expansion

lemma *convex_hull_finite_step*:
fixes S :: *'a*::*real_vector set*
assumes *finite* S
shows
$(\exists u.\ (\forall x{\in}insert\ a\ S.\ 0 \le u\ x) \wedge sum\ u\ (insert\ a\ S) = w \wedge sum\ (\lambda x.\ u\ x *_R x)\ (insert\ a\ S) = y)$
$\longleftrightarrow (\exists v{\ge}0.\ \exists u.\ (\forall x{\in}S.\ 0 \le u\ x) \wedge sum\ u\ S = w - v \wedge sum\ (\lambda x.\ u\ x *_R x)\ S = y - v *_R a)$
(**is** *?lhs* = *?rhs*)
proof (*cases* $a \in S$)
case *True*
then have $*$: $insert\ a\ S = S$ **by** *auto*
show *?thesis*

proof
assume *?lhs*
then show *?rhs*
unfolding $*$ **by** *force*
next
have *fin*: *finite* (*insert a S*) **using** *assms* **by** *auto*
assume *?rhs*
then obtain *v u* **where** *uv*: $v{\geq}0$ $\forall x{\in}S.\ 0 \leq u\ x$ *sum u S* $= w - v$ $(\sum x{\in}S.\ u\ x *_R x) = y - v *_R a$
by *auto*
then show *?lhs*
using *uv True assms*
apply (*rule_tac* $x = \lambda x.$ (*if* $a = x$ *then* v *else* 0) $+$ $u\ x$ **in** *exI*)
apply (*auto simp*: *sum_clauses scaleR_left_distrib sum.distrib sum_delta''*[*OF fin*])
done
qed
next
case *False*
show *?thesis*
proof
assume *?lhs*
then obtain *u* **where** *u*: $\forall x{\in}$*insert a S*. $0 \leq u\ x$ *sum u* (*insert a S*) $= w$ $(\sum x{\in}$*insert a S*. $u\ x *_R x) = y$
by *auto*
then show *?rhs*
using *u* ‹$a{\notin}S$› **by** (*rule_tac* $x{=}u\ a$ **in** *exI*) (*auto simp*: *sum_clauses assms*)
next
assume *?rhs*
then obtain *v u* **where** *uv*: $v{\geq}0$ $\forall x{\in}S.\ 0 \leq u\ x$ *sum u S* $= w - v$ $(\sum x{\in}S.\ u\ x *_R x) = y - v *_R a$
by *auto*
moreover
have $(\sum x{\in}S.$ *if* $a = x$ *then* v *else* $u\ x) =$ *sum u S* $(\sum x{\in}S.$ (*if* $a = x$ *then* v *else* $u\ x) *_R x) = (\sum x{\in}S.\ u\ x *_R x)$
using *False* **by** (*auto intro*!: *sum.cong*)
ultimately show *?lhs*
using *False* **by** (*rule_tac* $x{=}\lambda x.$ *if* $a = x$ *then* v *else* $u\ x$ **in** *exI*) (*auto simp*: *sum_clauses*(*2*)[*OF assms*])
qed
qed

Hence some special cases

lemma *convex_hull_2*: *convex hull* $\{a,b\} = \{u *_R a + v *_R b \mid u\ v.\ 0 \leq u \wedge 0 \leq v \wedge u + v = 1\}$
(**is** *?lhs* $=$ *?rhs*)
proof −
have $**$: *finite* $\{b\}$ **by** *auto*

have $\bigwedge x\ v\ u.\ [\![0 \le v;\ v \le 1;\ (1 - v) *_R b = x - v *_R a]\!]$
$\Longrightarrow \exists u\ v.\ x = u *_R a + v *_R b \wedge 0 \le u \wedge 0 \le v \wedge u + v = 1$
by (*metis add.commute diff_add_cancel diff_ge_0_iff_ge*)
moreover
have $\bigwedge u\ v.\ [\![0 \le u;\ 0 \le v;\ u + v = 1]\!]$
$\Longrightarrow \exists p{\ge}0.\ \exists q.\ 0 \le q\ b \wedge q\ b = 1 - p \wedge q\ b *_R b = u *_R a + v *_R b - p *_R a$
apply (*rule_tac x=u* **in** *exI*, *simp*)
apply (*rule_tac x=λx. v* **in** *exI*, *simp*)
done
ultimately show *?thesis*
using *convex_hull_finite_step*[*OF* **, *of a 1*]
by (*auto simp add*: *convex_hull_finite*)
qed

lemma *convex_hull_2_alt*: *convex hull* {*a*,*b*} = {$a + u *_R (b - a) \mid u.\ 0 \le u \wedge u \le 1$}
unfolding *convex_hull_2*
proof (*rule Collect_cong*)
have *: $\bigwedge x\ y$ ::*real*. $x + y = 1 \longleftrightarrow x = 1 - y$
by *auto*
fix *x*
show $(\exists v\ u.\ x = v *_R a + u *_R b \wedge 0 \le v \wedge 0 \le u \wedge v + u = 1) \longleftrightarrow$
$(\exists u.\ x = a + u *_R (b - a) \wedge 0 \le u \wedge u \le 1)$
apply (*simp add*: *)
by (*rule ex_cong1*) (*auto simp*: *algebra_simps*)
qed

lemma *convex_hull_3*:
convex hull {*a*,*b*,*c*} = {$u *_R a + v *_R b + w *_R c \mid u\ v\ w.\ 0 \le u \wedge 0 \le v \wedge 0 \le w \wedge u + v + w = 1$}
proof −
have *fin*: *finite* {*a*,*b*,*c*} *finite* {*b*,*c*} *finite* {*c*}
by *auto*
have *: $\bigwedge x\ y\ z$ ::*real*. $x + y + z = 1 \longleftrightarrow x = 1 - y - z$
by (*auto simp*: *field_simps*)
show *?thesis*
unfolding *convex_hull_finite*[*OF fin*(*1*)] **and** *convex_hull_finite_step*[*OF fin*(*2*)] **and** *
unfolding *convex_hull_finite_step*[*OF fin*(*3*)]
apply (*rule Collect_cong*, *simp*)
apply *auto*
apply (*rule_tac x=va* **in** *exI*)
apply (*rule_tac x=u c* **in** *exI*, *simp*)
apply (*rule_tac x=1 − v − w* **in** *exI*, *simp*)
apply (*rule_tac x=v* **in** *exI*, *simp*)
apply (*rule_tac x=λx. w* **in** *exI*, *simp*)
done
qed

lemma *convex_hull_3_alt*:
convex hull {*a*,*b*,*c*} = {*a* + *u* ∗*R* (*b* − *a*) + *v* ∗*R* (*c* − *a*) | *u v*. 0 ≤ *u* ∧ 0 ≤ *v* ∧ *u* + *v* ≤ 1}
proof −
have ∗: ⋀*x y z* ::*real*. *x* + *y* + *z* = 1 ⟷ *x* = 1 − *y* − *z*
by *auto*
show *?thesis*
unfolding *convex_hull_3*
apply (*auto simp*: ∗)
apply (*rule_tac x*=*v* **in** *exI*)
apply (*rule_tac x*=*w* **in** *exI*)
apply (*simp add*: *algebra_simps*)
apply (*rule_tac x*=*u* **in** *exI*)
apply (*rule_tac x*=*v* **in** *exI*)
apply (*simp add*: *algebra_simps*)
done
qed

1.7.11 Relations among closure notions and corresponding hulls

lemma *affine_imp_convex*: *affine s* ⟹ *convex s*
unfolding *affine_def convex_def* **by** *auto*

lemma *convex_affine_hull* [*simp*]: *convex* (*affine hull S*)
by (*simp add*: *affine_imp_convex*)

lemma *subspace_imp_convex*: *subspace s* ⟹ *convex s*
using *subspace_imp_affine affine_imp_convex* **by** *auto*

lemma *convex_hull_subset_span*: (*convex hull s*) ⊆ (*span s*)
by (*metis hull_minimal span_superset subspace_imp_convex subspace_span*)

lemma *convex_hull_subset_affine_hull*: (*convex hull s*) ⊆ (*affine hull s*)
by (*metis affine_affine_hull affine_imp_convex hull_minimal hull_subset*)

lemma *aff_dim_convex_hull*:
fixes *S* :: *'n*::*euclidean_space set*
shows *aff_dim* (*convex hull S*) = *aff_dim S*
by (*smt* (*verit*) *aff_dim_affine_hull aff_dim_subset convex_hull_subset_affine_hull hull_subset*)

1.7.12 Caratheodory's theorem

lemma *convex_hull_caratheodory_aff_dim*:
fixes *p* :: (*'a*::*euclidean_space*) *set*
shows *convex hull p* =
{*y*. ∃ *S u*. *finite S* ∧ *S* ⊆ *p* ∧ *card S* ≤ *aff_dim p* + 1 ∧

```
      (∀x∈S. 0 ≤ u x) ∧ sum u S = 1 ∧ sum (λv. u v *R v) S = y}
  unfolding convex_hull_explicit set_eq_iff mem_Collect_eq
proof (intro allI iffI)
  fix y
  let ?P = λn. ∃S u. finite S ∧ card S = n ∧ S ⊆ p ∧ (∀x∈S. 0 ≤ u x) ∧
    sum u S = 1 ∧ (∑v∈S. u v *R v) = y
  assume ∃S u. finite S ∧ S ⊆ p ∧ (∀x∈S. 0 ≤ u x) ∧ sum u S = 1 ∧ (∑v∈S.
u v *R v) = y
  then obtain N where ?P N by auto
  then have ∃n≤N. (∀k<n. ¬ ?P k) ∧ ?P n
    by (rule_tac ex_least_nat_le, auto)
  then obtain n where ?P n and smallest: ∀k<n. ¬ ?P k
    by blast
  then obtain S u where S: finite S card S = n S⊆p
    and u: ∀x∈S. 0 ≤ u x sum u S = 1  (∑v∈S. u v *R v) = y by auto

  have card S ≤ aff_dim p + 1
  proof (rule ccontr, simp only: not_le)
    assume aff_dim p + 1 < card S
    then have affine_dependent S
      by (smt (verit) independent_card_le_aff_dim S(3))
    then obtain w v where wv: sum w S = 0 v∈S w v ≠ 0 (∑v∈S. w v *R v)
= 0
      using affine_dependent_explicit_finite[OF S(1)] by auto
    define i where i = (λv. (u v) / (− w v)) ‘ {v∈S. w v < 0}
    define t where t = Min i
    have ∃x∈S. w x < 0
      by (smt (verit, best) S(1) sum_pos2 wv)
    then have i ≠ {} unfolding i_def by auto
    then have t ≥ 0
      using Min_ge_iff[of i 0] and S(1) u[unfolded le_less]
      unfolding t_def i_def
      by (auto simp: divide_le_0_iff)
    have t: ∀v∈S. u v + t * w v ≥ 0
    proof
      fix v
      assume v ∈ S
      then have v: 0 ≤ u v
        using u(1) by blast
      show 0 ≤ u v + t * w v
      proof (cases w v < 0)
        case False
        thus ?thesis using v ‹t≥0› by auto
      next
        case True
        then have t ≤ u v / (− w v)
          using ‹v∈S› S unfolding t_def i_def by (auto intro: Min_le)
        then show ?thesis
          unfolding real_0_le_add_iff
```

```
        using True neg_le_minus_divide_eq by auto
      qed
    qed
    obtain a where a ∈ S and t = (λv. (u v) / (− w v)) a and w a < 0
      using Min_in[OF _ ‹i≠{}›] and S(1) unfolding i_def t_def by auto
    then have a: a ∈ S u a + t * w a = 0 by auto
    have *: ⋀f. sum f (S − {a}) = sum f S − ((f a)::'b::ab_group_add)
      unfolding sum.remove[OF S(1) ‹a∈S›] by auto
    have (∑ v∈S. u v + t * w v) = 1
       by (metis add.right_neutral mult_zero_right sum.distrib sum_distrib_left
u(2) wv(1))
    moreover have (∑ v∈S. u v *R v + (t * w v) *R v) − (u a *R a + (t * w a)
*R a) = y
       unfolding sum.distrib u(3) scaleR_scaleR[symmetric] scaleR_right.sum
[symmetric] wv(4)
      using a(2) [THEN eq_neg_iff_add_eq_0 [THEN iffD2]] by simp
    ultimately have ?P (n − 1)
      apply (rule_tac x=(S − {a}) in exI)
      apply (rule_tac x=λv. u v + t * w v in exI)
      using S t a
      apply (auto simp: * scaleR_left_distrib)
      done
    then show False
      using smallest[THEN spec[where x=n − 1]] by auto
  qed
  then show ∃S u. finite S ∧ S ⊆ p ∧ card S ≤ aff_dim p + 1 ∧
      (∀x∈S. 0 ≤ u x) ∧ sum u S = 1 ∧ (∑ v∈S. u v *R v) = y
    using S u by auto
qed auto

lemma caratheodory_aff_dim:
  fixes p :: ('a::euclidean_space) set
  shows convex hull p = {x. ∃S. finite S ∧ S ⊆ p ∧ card S ≤ aff_dim p + 1 ∧ x
∈ convex hull S}
       (is ?lhs = ?rhs)
proof
  have ⋀x S u. ⟦finite S; S ⊆ p; int (card S) ≤ aff_dim p + 1; ∀x∈S. 0 ≤ u x;
sum u S = 1⟧
             ⟹ (∑ v∈S. u v *R v) ∈ convex hull S
   by (metis (mono_tags, lifting) convex_convex_hull convex_explicit hull_subset)
  then show ?lhs ⊆ ?rhs
    by (subst convex_hull_caratheodory_aff_dim, auto)
qed (use hull_mono in auto)

lemma convex_hull_caratheodory:
  fixes p :: ('a::euclidean_space) set
  shows convex hull p =
          {y. ∃S u. finite S ∧ S ⊆ p ∧ card S ≤ DIM('a) + 1 ∧
            (∀x∈S. 0 ≤ u x) ∧ sum u S = 1 ∧ sum (λv. u v *R v) S = y}
```

(**is** *?lhs = ?rhs*)
proof (*intro set_eqI iffI*)
fix *x*
assume *x ∈ ?lhs* **then show** *x ∈ ?rhs*
unfolding *convex_hull_caratheodory_aff_dim*
using *aff_dim_le_DIM* [*of p*] **by** *fastforce*
qed (*auto simp*: *convex_hull_explicit*)

theorem *caratheodory*:
convex hull p =
{*x*::*'a*::*euclidean_space*. ∃ *S*. *finite S* ∧ *S* ⊆ *p* ∧ *card S* ≤ *DIM*(*'a*) + *1* ∧ *x* ∈ *convex hull S*}
proof *safe*
fix *x*
assume *x ∈ convex hull p*
then obtain *S u* **where** *finite S S* ⊆ *p card S* ≤ *DIM*(*'a*) + *1*
∀ *x*∈*S*. *0* ≤ *u x sum u S* = *1* (∑ *v*∈*S*. *u v* ∗*R v*) = *x*
unfolding *convex_hull_caratheodory* **by** *auto*
then show ∃ *S*. *finite S* ∧ *S* ⊆ *p* ∧ *card S* ≤ *DIM*(*'a*) + *1* ∧ *x ∈ convex hull S*
using *convex_hull_finite* **by** *fastforce*
qed (*use hull_mono* **in** *force*)

1.7.13 Some Properties of subset of standard basis

lemma *affine_hull_substd_basis*:
assumes *d* ⊆ *Basis*
shows *affine hull* (*insert 0 d*) = {*x*::*'a*::*euclidean_space*. ∀ *i*∈*Basis*. *i* ∉ *d* ⟶ *x*•*i* = *0*}
(**is** *affine hull* (*insert 0 ?A*) = *?B*)
proof −
have ∗: ⋀*A*. (+) (*0*::*'a*) ‘ *A* = *A* ⋀*A*. (+) (− (*0*::*'a*)) ‘ *A* = *A*
by *auto*
show *?thesis*
unfolding *affine_hull_insert_span_gen span_substd_basis*[*OF assms*,*symmetric*]
∗ **..**
qed

lemma *affine_hull_convex_hull* [*simp*]: *affine hull* (*convex hull S*) = *affine hull S*
by (*metis Int_absorb1 Int_absorb2 convex_hull_subset_affine_hull hull_hull hull_mono hull_subset*)

1.7.14 Moving and scaling convex hulls

lemma *convex_hull_set_plus*:
convex hull (*S* + *T*) = *convex hull S* + *convex hull T*
by (*simp add*: *set_plus_image linear_iff scaleR_right_distrib convex_hull_Times*

flip: *convex_hull_linear_image*)

lemma *translation_eq_singleton_plus*: $(\lambda x.\ a + x)\ `\ T = \{a\} + T$
unfolding *set_plus_def* **by** *auto*

lemma *convex_hull_translation*:
convex hull $((\lambda x.\ a + x)\ `\ S) = (\lambda x.\ a + x)\ `\ (\textit{convex hull}\ S)$
by (*simp add*: *convex_hull_set_plus translation_eq_singleton_plus*)

lemma *convex_hull_scaling*:
convex hull $((\lambda x.\ c *_R x)\ `\ S) = (\lambda x.\ c *_R x)\ `\ (\textit{convex hull}\ S)$
by (*simp add*: *convex_hull_linear_image*)

lemma *convex_hull_affinity*:
convex hull $((\lambda x.\ a + c *_R x)\ `\ S) = (\lambda x.\ a + c *_R x)\ `\ (\textit{convex hull}\ S)$
by (*metis convex_hull_scaling convex_hull_translation image_image*)

1.7.15 Convexity of cone hulls

lemma *convex_cone_hull*:
assumes *convex S*
shows *convex* (*cone hull S*)
proof (*rule convexI*)
fix *x y*
assume *xy*: $x \in \textit{cone hull}\ S\ y \in \textit{cone hull}\ S$
then have $S \neq \{\}$
using *cone_hull_empty_iff*[*of S*] **by** *auto*
fix *u v* :: *real*
assume *uv*: $u \geq 0\ v \geq 0\ u + v = 1$
then have $*$: $u *_R x \in \textit{cone hull}\ S\ v *_R y \in \textit{cone hull}\ S$
by (*simp_all add*: *cone_cone_hull mem_cone uv xy*)
then obtain *cx* :: *real* **and** *xx*
and *cy* :: *real* **and** *yy* **where** *x*: $u *_R x = cx *_R xx\ cx \geq 0\ xx \in S$
and *y*: $v *_R y = cy *_R yy\ cy \geq 0\ yy \in S$
using *cone_hull_expl*[*of S*] **by** *auto*

have $u *_R x + v *_R y \in \textit{cone hull}\ S$ **if** $cx + cy \leq 0$
using $*(1)$ *nless_le that x*(2) *y* **by** *fastforce*
moreover
have $u *_R x + v *_R y \in \textit{cone hull}\ S$ **if** $cx + cy > 0$
proof −
have $(cx\ /\ (cx + cy)) *_R xx + (cy\ /\ (cx + cy)) *_R yy \in S$
using *assms mem_convex_alt*[*of S xx yy cx cy*] *x y that* **by** *auto*
then have $cx *_R xx + cy *_R yy \in \textit{cone hull}\ S$
using *mem_cone_hull*[*of* $(cx/(cx+cy)) *_R xx + (cy/(cx+cy)) *_R yy$ *S cx+cy*]
‹*cx+cy>0*›
by (*auto simp*: *scaleR_right_distrib*)
then show *?thesis*
using *x y* **by** *auto*
qed
moreover have $cx + cy \leq 0 \vee cx + cy > 0$ **by** *auto*

ultimately show $u *_R x + v *_R y \in$ *cone hull* S **by** *blast*
qed

lemma *cone_convex_hull*:
assumes *cone* S
shows *cone* (*convex hull* S)
by (*metis* (*no_types*, *lifting*) *affine_hull_convex_hull affine_hull_eq_empty assms cone_iff convex_hull_scaling hull_inc*)

1.7.16 Radon's theorem

Formalized by Lars Schewe.

lemma *Radon_ex_lemma*:
assumes *finite c affine_dependent c*
shows $\exists u.$ *sum* u $c = 0 \wedge (\exists v{\in}c.\ u\ v \neq 0) \wedge$ *sum* $(\lambda v.\ u\ v *_R v)$ $c = 0$
using *affine_dependent_explicit_finite assms* **by** *blast*

lemma *Radon_s_lemma*:
assumes *finite* S
and *sum* f $S = (0{::}real)$
shows *sum* f $\{x{\in}S.\ 0 < f\ x\} = -$ *sum* f $\{x{\in}S.\ f\ x < 0\}$
proof −
have $\bigwedge x.$ (*if* $f\ x < 0$ *then* $f\ x$ *else* 0) + (*if* $0 < f\ x$ *then* $f\ x$ *else* 0) $= f\ x$
by *auto*
then show *?thesis*
using *assms* **by** (*simp add*: *sum.inter_filter flip*: *sum.distrib add_eq_0_iff*)
qed

lemma *Radon_v_lemma*:
assumes *finite* S
and *sum* f $S = 0$
and $\forall x.$ $g\ x = (0{::}real) \longrightarrow f\ x = (0{::}'a{::}euclidean\_space)$
shows (*sum* f $\{x{\in}S.\ 0 < g\ x\}) = -$ *sum* f $\{x{\in}S.\ g\ x < 0\}$
proof −
have $\bigwedge x.$ (*if* $0 < g\ x$ *then* $f\ x$ *else* 0) + (*if* $g\ x < 0$ *then* $f\ x$ *else* 0) $= f\ x$
using *assms* **by** *auto*
then show *?thesis*
using *assms* **by** (*simp add*: *sum.inter_filter eq_neg_iff_add_eq_0 flip*: *sum.distrib add_eq_0_iff*)
qed

lemma *Radon_partition*:
assumes *finite C affine_dependent C*
shows $\exists M\ P.\ M \cap P = \{\} \wedge M \cup P = C \wedge$ (*convex hull* M) $\cap$ (*convex hull* P) $\neq \{\}$
proof −
obtain $u\ v$ **where** *uv*: *sum* u $C = 0$ $v{\in}C$ $u\ v \neq 0$ $(\sum v{\in}C.\ u\ v *_R v) = 0$
using *Radon_ex_lemma*[*OF assms*] **by** *auto*
have *fin*: *finite* $\{x \in C.\ 0 < u\ x\}$ *finite* $\{x \in C.\ 0 > u\ x\}$

using *assms*(*1*) **by** *auto*
define *z* **where** *z* = *inverse* (*sum u* {*x*∈*C*. *u x* > *0*}) ∗$_R$ *sum* (λ*x*. *u x* ∗$_R$ *x*) {*x*∈*C*. *u x* > *0*}
have *sum u* {*x* ∈ *C*. *0* < *u x*} ≠ *0*
proof (*cases u v* ≥ *0*)
case *False*
then have *u v* < *0* **by** *auto*
then show *?thesis*
by (*smt* (*verit*) *assms*(*1*) *fin*(*1*) *mem_Collect_eq sum.neutral_const sum_mono_inv uv*)
next
case *True*
with *fin uv* **show** *sum u* {*x* ∈ *C*. *0* < *u x*} ≠ *0*
by (*smt* (*verit*) *fin*(*1*) *mem_Collect_eq sum_nonneg_eq_0_iff uv*)
qed
then have ∗: *sum u* {*x*∈*C*. *u x* > *0*} > *0*
unfolding *less_le* **by** (*metis* (*no_types*, *lifting*) *mem_Collect_eq sum_nonneg*)
moreover have *sum u* ({*x* ∈ *C*. *0* < *u x*} ∪ {*x* ∈ *C*. *u x* < *0*}) = *sum u C*
(∑ *x*∈{*x* ∈ *C*. *0* < *u x*} ∪ {*x* ∈ *C*. *u x* < *0*}. *u x* ∗$_R$ *x*) = (∑ *x*∈*C*. *u x* ∗$_R$ *x*)
using *assms*(*1*)
by (*rule_tac*[!] *sum.mono_neutral_left*, *auto*)
then have *sum u* {*x* ∈ *C*. *0* < *u x*} = − *sum u* {*x* ∈ *C*. *0* > *u x*}
(∑ *x*∈{*x* ∈ *C*. *0* < *u x*}. *u x* ∗$_R$ *x*) = − (∑ *x*∈{*x* ∈ *C*. *0* > *u x*}. *u x* ∗$_R$ *x*)
unfolding *eq_neg_iff_add_eq_0*
using *uv*(*1*,*4*)
by (*auto simp*: *sum.union_inter_neutral*[*OF fin*, *symmetric*])
moreover have ∀ *x*∈{*v* ∈ *C*. *u v* < *0*}. *0* ≤ *inverse* (*sum u* {*x* ∈ *C*. *0* < *u x*}) ∗ − *u x*
using ∗ **by** (*fastforce intro*: *mult_nonneg_nonneg*)
ultimately have *z* ∈ *convex hull* {*v* ∈ *C*. *u v* ≤ *0*}
unfolding *convex_hull_explicit mem_Collect_eq*
apply (*rule_tac x*={*v* ∈ *C*. *u v* < *0*} **in** *exI*)
apply (*rule_tac x*=λ*y*. *inverse* (*sum u* {*x*∈*C*. *u x* > *0*}) ∗ − *u y* **in** *exI*)
using *assms*(*1*) **unfolding** *scaleR_scaleR*[*symmetric*] *scaleR_right.sum* [*symmetric*]

by (*auto simp*: *z_def sum_negf sum_distrib_left*[*symmetric*])
moreover have ∀ *x*∈{*v* ∈ *C*. *0* < *u v*}. *0* ≤ *inverse* (*sum u* {*x* ∈ *C*. *0* < *u x*}) ∗ *u x*
using ∗ **by** (*fastforce intro*: *mult_nonneg_nonneg*)
then have *z* ∈ *convex hull* {*v* ∈ *C*. *u v* > *0*}
unfolding *convex_hull_explicit mem_Collect_eq*
apply (*rule_tac x*={*v* ∈ *C*. *0* < *u v*} **in** *exI*)
apply (*rule_tac x*=λ*y*. *inverse* (*sum u* {*x*∈*C*. *u x* > *0*}) ∗ *u y* **in** *exI*)
using *assms*(*1*)
unfolding *scaleR_scaleR*[*symmetric*] *scaleR_right.sum* [*symmetric*]
using ∗ **by** (*auto simp*: *z_def sum_negf sum_distrib_left*[*symmetric*])
ultimately show *?thesis*
apply (*rule_tac x*={*v*∈*C*. *u v* ≤ *0*} **in** *exI*)
apply (*rule_tac x*={*v*∈*C*. *u v* > *0*} **in** *exI*, *auto*)

done
qed

theorem *Radon*:
assumes *affine_dependent c*
obtains *M P* **where** $M \subseteq c$ $P \subseteq c$ $M \cap P = \{\}$ (*convex hull M*) $\cap$ (*convex hull P*) $\neq \{\}$
by (*smt* (*verit*) *Radon_partition affine_dependent_explicit affine_dependent_explicit_finite assms le_sup_iff*)

1.7.17 Helly's theorem

lemma *Helly_induct*:
fixes $\mathcal{F}$ :: *'a*::*euclidean_space set set*
assumes *card* $\mathcal{F} = n$
and $n \geq DIM('a) + 1$
and $\forall S{\in}\mathcal{F}.$ *convex* S $\forall T{\subseteq}\mathcal{F}.$ *card* $T = DIM('a) + 1 \longrightarrow \bigcap T \neq \{\}$
shows $\bigcap \mathcal{F} \neq \{\}$
using *assms*
proof (*induction n arbitrary*: $\mathcal{F}$)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
have *finite* $\mathcal{F}$
using ‹*card* $\mathcal{F}$ = *Suc n*› **by** (*auto intro*: *card_ge_0_finite*)
show $\bigcap \mathcal{F} \neq \{\}$
proof (*cases* $n = DIM('a)$)
case *True*
then show *?thesis*
by (*simp add*: *Suc.prems*)
next
case *False*
have $\bigcap(\mathcal{F} - \{S\}) \neq \{\}$ **if** $S \in \mathcal{F}$ **for** S
proof (*rule Suc.IH*[*rule_format*])
show *card* $(\mathcal{F} - \{S\}) = n$
by (*simp add*: *Suc.prems*(*1*) ‹*finite* $\mathcal{F}$› *that*)
show $DIM('a) + 1 \leq n$
using *False Suc.prems*(*2*) **by** *linarith*
show $\bigwedge t.$ $[\![t \subseteq \mathcal{F} - \{S\}; \textit{card } t = DIM('a) + 1]\!] \Longrightarrow \bigcap t \neq \{\}$
by (*simp add*: *Suc.prems*(*4*) *subset_Diff_insert*)
qed (*use Suc* **in** *auto*)
then have $\forall S{\in}\mathcal{F}.\ \exists x.\ x \in \bigcap(\mathcal{F} - \{S\})$
by *blast*
then obtain *X* **where** *X*: $\bigwedge S.\ S{\in}\mathcal{F} \Longrightarrow X\ S \in \bigcap(\mathcal{F} - \{S\})$
by *metis*
show *?thesis*
proof (*cases inj_on X* $\mathcal{F}$)
case *False*

then obtain S T **where** $S{\neq}T$ **and** st: $S{\in}\mathcal{F}$ $T{\in}\mathcal{F}$ $X\ S = X\ T$
unfolding *inj_on_def* **by** *auto*
then have $*$: $\bigcap\mathcal{F} = \bigcap(\mathcal{F} - \{S\}) \cap \bigcap(\mathcal{F} - \{T\})$ **by** *auto*
show *?thesis*
by (*metis* $*$ *X disjoint_iff_not_equal st*)
next
case *True*
then obtain M P **where** *mp*: $M \cap P = \{\}$ $M \cup P = X$ ‘ $\mathcal{F}$ *convex hull M* ∩ *convex hull P* $\neq \{\}$
using *Radon_partition*[*of X* ‘ $\mathcal{F}$] **and** *affine_dependent_biggerset*[*of X* ‘ $\mathcal{F}$]
unfolding *card_image*[*OF True*] **and** ‹*card* $\mathcal{F}$ = *Suc n*›
using *Suc*(*3*) ‹*finite* $\mathcal{F}$› **and** *False*
by *auto*
have $M \subseteq X$ ‘ $\mathcal{F}$ $P \subseteq X$ ‘ $\mathcal{F}$
using *mp*(*2*) **by** *auto*
then obtain $\mathcal{G}$ $\mathcal{H}$ **where** *gh*:$M = X$ ‘ $\mathcal{G}$ $P = X$ ‘ $\mathcal{H}$ $\mathcal{G} \subseteq \mathcal{F}$ $\mathcal{H} \subseteq \mathcal{F}$
unfolding *subset_image_iff* **by** *auto*
then have $\mathcal{F} \cup (\mathcal{G} \cup \mathcal{H}) = \mathcal{F}$ **by** *auto*
then have $\mathcal{F}$: $\mathcal{F} = \mathcal{G} \cup \mathcal{H}$
using *inj_on_Un_image_eq_iff*[*of X* $\mathcal{F}$ $\mathcal{G} \cup \mathcal{H}$] **and** *True*
unfolding *mp*(*2*)[*unfolded image_Un*[*symmetric*] *gh*]
by *auto*
have $*$: $\mathcal{G} \cap \mathcal{H} = \{\}$
using *gh local.mp*(*1*) **by** *blast*
have *convex hull* (*X* ‘ $\mathcal{H}$) $\subseteq \bigcap\mathcal{G}$ *convex hull* (*X* ‘ $\mathcal{G}$) $\subseteq \bigcap\mathcal{H}$
by (*rule hull_minimal*; *use X* $*$ $\mathcal{F}$ **in** ‹*auto simp*: *Suc.prems*(*3*) *convex_Inter*›)+
then show *?thesis*
unfolding $\mathcal{F}$ **using** *mp*(*3*)[*unfolded gh*] **by** *blast*
qed
qed
qed

theorem *Helly*:
fixes $\mathcal{F}$ :: *'a*::*euclidean_space set set*
assumes *card* $\mathcal{F} \geq$ *DIM*(*'a*) + *1* $\forall s{\in}\mathcal{F}$. *convex s*
and $\bigwedge t$. $[\![t{\subseteq}\mathcal{F}$; *card t* = *DIM*(*'a*) + *1*$]\!] \Longrightarrow \bigcap t \neq \{\}$
shows $\bigcap\mathcal{F} \neq \{\}$
using *Helly_induct assms* **by** *blast*

1.7.18 Epigraphs of convex functions

definition *epigraph S* (*f* :: _ ⇒ *real*) = {*xy*. *fst xy* ∈ *S* ∧ *f* (*fst xy*) ≤ *snd xy*}

lemma *mem_epigraph*: (*x*, *y*) ∈ *epigraph S f* ⟷ *x* ∈ *S* ∧ *f x* ≤ *y*
unfolding *epigraph_def* **by** *auto*

lemma *convex_epigraph*: *convex* (*epigraph S f*) ⟷ *convex_on S f* ∧ *convex S*
proof *safe*

assume *L*: *convex* (*epigraph S f*)
then show *convex_on S f*
by (*auto simp*: *convex_def convex_on_def epigraph_def*)
show *convex S*
using *L* **by** (*fastforce simp*: *convex_def convex_on_def epigraph_def*)
next
assume *convex_on S f convex S*
then show *convex* (*epigraph S f*)
unfolding *convex_def convex_on_def epigraph_def*
apply *safe*
apply (*rule_tac* [*2*] *y*=*u* ∗ *f a* + *v* ∗ *f aa* **in** *order_trans*)
apply (*auto intro*!:*mult_left_mono add_mono*)
done
qed

lemma *convex_epigraphI*: *convex_on S f* ⟹ *convex S* ⟹ *convex* (*epigraph S f*)
unfolding *convex_epigraph* **by** *auto*

lemma *convex_epigraph_convex*: *convex S* ⟹ *convex_on S f* ⟷ *convex*(*epigraph S f*)
by (*simp add*: *convex_epigraph*)

Use this to derive general bound property of convex function

lemma *convex_on*:
assumes *convex S*
shows *convex_on S f* ⟷
(∀ *k u x*. (∀ *i*∈{*1*..*k*::*nat*}. *0* ≤ *u i* ∧ *x i* ∈ *S*) ∧ *sum u* {*1*..*k*} = *1* ⟶
f (*sum* (λ*i*. *u i* ∗$_R$ *x i*) {*1*..*k*}) ≤ *sum* (λ*i*. *u i* ∗ *f*(*x i*)) {*1*..*k*})
(**is** *?lhs* = (∀ *k u x*. *?rhs k u x*))
proof
assume *?lhs*
then have §: *convex* {*xy*. *fst xy* ∈ *S* ∧ *f* (*fst xy*) ≤ *snd xy*}
by (*metis assms convex_epigraph epigraph_def*)
show ∀ *k u x*. *?rhs k u x*
proof (*intro allI*)
fix *k u x*
show *?rhs k u x*
using §
unfolding *convex mem_Collect_eq fst_sum snd_sum*
apply *safe*
apply (*drule_tac x*=*k* **in** *spec*)
apply (*drule_tac x*=*u* **in** *spec*)
apply (*drule_tac x*=λ*i*. (*x i*, *f* (*x i*)) **in** *spec*)
apply *simp*
done
qed
next

assume $\forall k\ u\ x.$ *?rhs k u x*
then show *?lhs*
unfolding *convex_epigraph_convex*[*OF assms*] *convex epigraph_def Ball_def mem_Collect_eq fst_sum snd_sum*
using *assms*[*unfolded convex*] **apply** *clarsimp*
apply (*rule_tac* $y=\sum i = 1..k.\ u\ i * f\ (fst\ (x\ i))$ **in** *order_trans*)
by (*auto simp add*: *mult_left_mono intro*: *sum_mono*)
qed

1.7.19 A bound within a convex hull

lemma *convex_on_convex_hull_bound*:
assumes *convex_on* (*convex hull S*) *f*
and $\forall x{\in}S.\ f\ x \leq b$
shows $\forall x{\in}$ *convex hull S*. $f\ x \leq b$
proof
fix *x*
assume $x \in$ *convex hull S*
then obtain *k u v* **where**
u: $\forall i{\in}\{1..k{::}nat\}.\ 0 \leq u\ i \wedge v\ i \in S$ *sum u* $\{1..k\} = 1$ $(\sum i = 1..k.\ u\ i *_R v\ i) = x$
unfolding *convex_hull_indexed mem_Collect_eq* **by** *auto*
have $(\sum i = 1..k.\ u\ i * f\ (v\ i)) \leq b$
using *sum_mono*[*of* $\{1..k\}$ $\lambda i.\ u\ i * f\ (v\ i)$ $\lambda i.\ u\ i * b$]
unfolding *sum_distrib_right*[*symmetric*] *u*(*2*) *mult_1*
using *assms*(*2*) *mult_left_mono u*(*1*) **by** *blast*
then show $f\ x \leq b$
using *assms*(*1*)[*unfolded convex_on*[*OF convex_convex_hull*], *rule_format*, *of k u v*]
using *hull_inc u* **by** *fastforce*
qed

lemma *convex_set_plus*:
assumes *convex S* **and** *convex T* **shows** *convex* $(S + T)$
by (*metis assms convex_hull_eq convex_hull_set_plus*)

lemma *convex_set_sum*:
assumes $\bigwedge i.\ i \in A \Longrightarrow$ *convex* $(B\ i)$
shows *convex* $(\sum i{\in}A.\ B\ i)$
using *assms*
by (*induction A rule*: *infinite_finite_induct*) (*auto simp*: *convex_set_plus*)

lemma *finite_set_sum*:
assumes $\forall i{\in}A.$ *finite* $(B\ i)$ **shows** *finite* $(\sum i{\in}A.\ B\ i)$
using *assms*
by (*induction A rule*: *infinite_finite_induct*) (*auto simp*: *finite_set_plus*)

lemma *box_eq_set_sum_Basis*:
$\{x.\ \forall i{\in}Basis.\ x{\cdot}i \in B\ i\} = (\sum i{\in}Basis.\ (\lambda x.\ x *_R i)\ `\ (B\ i))$ (**is** *?lhs = ?rhs*)

proof −
have $\bigwedge x.\ \forall i{\in}Basis.\ x \cdot i \in B\ i \Longrightarrow$
$\exists s.\ x = sum\ s\ Basis \wedge (\forall i{\in}Basis.\ s\ i \in (\lambda x.\ x *_R i)\ `\ B\ i)$
by (*metis* (*mono_tags*, *lifting*) *euclidean_representation image_iff*)
moreover
have *sum f Basis* · $i \in B\ i$ **if** $i \in Basis$ **and** f: $\forall i{\in}Basis.\ f\ i \in (\lambda x.\ x *_R i)\ `\ B\ i$ **for** *i f*
proof −
have $(\sum x{\in}Basis - \{i\}.\ f\ x \cdot i) = 0$
proof (*intro strip sum.neutral*)
show $f\ x \cdot i = 0$ **if** $x \in Basis - \{i\}$ **for** *x*
using *that f* ‹$i \in Basis$› *inner_Basis that* **by** *fastforce*
qed
then have $(\sum x{\in}Basis.\ f\ x \cdot i) = f\ i \cdot i$
by (*metis* (*no_types*) ‹$i \in Basis$› *add.right_neutral sum.remove* [*OF finite_Basis*])
then have $(\sum x{\in}Basis.\ f\ x \cdot i) \in B\ i$
using *f that*(*1*) **by** *auto*
then show *?thesis*
by (*simp add*: *inner_sum_left*)
qed
ultimately show *?thesis*
by (*subst set_sum_alt* [*OF finite_Basis*]) *auto*
qed

lemma *convex_hull_set_sum*:
convex hull $(\sum i{\in}A.\ B\ i) = (\sum i{\in}A.\ convex\ hull\ (B\ i))$
by (*induction A rule*: *infinite_finite_induct*) (*auto simp*: *convex_hull_set_plus*)

end

1.8 Definition of Finite Cartesian Product Type

theory *Finite_Cartesian_Product*
imports
Euclidean_Space
L2_Norm
HOL−Library.Numeral_Type
HOL−Library.Countable_Set
HOL−Library.FuncSet
begin

1.8.1 Finite Cartesian products, with indexing and lambdas

typedef $('a, 'b)\ vec = UNIV :: ('b{::}finite \Rightarrow 'a)\ set$
morphisms *vec_nth vec_lambda* **..**

declare *vec_lambda_inject* [*simplified*, *simp*]

bundle *vec_syntax* **begin**
notation
vec_nth (**infixl** $ *90*) **and**
vec_lambda (**binder** χ *10*)
end

bundle *no_vec_syntax* **begin**
no_notation
vec_nth (**infixl** $ *90*) **and**
vec_lambda (**binder** χ *10*)
end

unbundle *vec_syntax*

Concrete syntax for ('*a*, '*b*) *vec*:

- '*a*^'*b* becomes ('*a*, '*b*::*finite*) *vec*
- '*a*^'*b*::_ becomes ('*a*, '*b*) *vec* without extra sort-constraint

syntax *_vec_type* :: *type* ⇒ *type* ⇒ *type* (**infixl** ^ *15*)
parse_translation ‹
let
fun vec t u = *Syntax.const* ***type_syntax*** ‹*vec*› $ *t* $ *u*;
fun finite_vec_tr [*t*, *u*] =
(*case Term_Position.strip_positions u of*
v as Free (*x*, _) =>
if Lexicon.is_tid x then
vec t (*Syntax.const* ***syntax_const*** ‹*_ofsort*› $ *v* $
Syntax.const ***class_syntax*** ‹*finite*›)
else vec t u
| _ => *vec t u*)
in
[(***syntax_const*** ‹*_vec_type*›, *K finite_vec_tr*)]
end
›

lemma *vec_eq_iff*: (*x* = *y*) ⟷ (∀ *i*. *x*$*i* = *y*$*i*)
by (*simp add*: *vec_nth_inject* [*symmetric*] *fun_eq_iff*)

lemma *vec_lambda_beta* [*simp*]: *vec_lambda g* $ *i* = *g i*
by (*simp add*: *vec_lambda_inverse*)

lemma *vec_lambda_unique*: (∀ *i*. *f*$*i* = *g i*) ⟷ *vec_lambda g* = *f*
by (*auto simp add*: *vec_eq_iff*)

lemma *vec_lambda_eta* [*simp*]: (χ *i*. (*g*$*i*)) = *g*
by (*simp add*: *vec_eq_iff*)

1.8.2 Cardinality of vectors

instance *vec* :: (*finite*, *finite*) *finite*
proof
  show *finite* (*UNIV* :: ('*a*, '*b*) *vec set*)
  proof (*subst bij_betw_finite*)
    show *bij_betw vec_nth UNIV* (*Pi* (*UNIV* :: '*b set*) (λ_. *UNIV* :: '*a set*))
      by (*intro bij_betwI*[*of* _ _ _ *vec_lambda*]) (*auto simp*: *vec_eq_iff*)
    have *finite* (*PiE* (*UNIV* :: '*b set*) (λ_. *UNIV* :: '*a set*))
      by (*intro finite_PiE*) *auto*
   also have (*PiE* (*UNIV* :: '*b set*) (λ_. *UNIV* :: '*a set*)) = *Pi UNIV* (λ_. *UNIV*)
      by *auto*
    finally show *finite* ... **.**
  qed
qed

lemma *countable_PiE*:
  finite I ⟹ (⋀*i*. *i* ∈ *I* ⟹ *countable* (*F i*)) ⟹ *countable* (*Pi*$_E$ *I F*)
  by (*induct I arbitrary*: *F rule*: *finite_induct*) (*auto simp*: *PiE_insert_eq*)

instance *vec* :: (*countable*, *finite*) *countable*
proof
  have *countable* (*UNIV* :: ('*a*, '*b*) *vec set*)
  proof (*rule countableI_bij2*)
    show *bij_betw vec_nth UNIV* (*Pi* (*UNIV* :: '*b set*) (λ_. *UNIV* :: '*a set*))
      by (*intro bij_betwI*[*of* _ _ _ *vec_lambda*]) (*auto simp*: *vec_eq_iff*)
    have *countable* (*PiE* (*UNIV* :: '*b set*) (λ_. *UNIV* :: '*a set*))
      by (*intro countable_PiE*) *auto*
   also have (*PiE* (*UNIV* :: '*b set*) (λ_. *UNIV* :: '*a set*)) = *Pi UNIV* (λ_. *UNIV*)
      by *auto*
    finally show *countable* ... **.**
  qed
  thus ∃ *t*::('*a*, '*b*) *vec* ⇒ *nat*. *inj t*
    by (*auto elim*!: *countableE*)
qed

lemma *infinite_UNIV_vec*:
  assumes *infinite* (*UNIV* :: '*a set*)
  shows *infinite* (*UNIV* :: ('*a*^~'*b*) *set*)
proof (*subst bij_betw_finite*)
  show *bij_betw vec_nth UNIV* (*Pi* (*UNIV* :: '*b set*) (λ_. *UNIV* :: '*a set*))
    by (*intro bij_betwI*[*of* _ _ _ *vec_lambda*]) (*auto simp*: *vec_eq_iff*)
  have *infinite* (*PiE* (*UNIV* :: '*b set*) (λ_. *UNIV* :: '*a set*)) (**is** *infinite ?A*)
  proof
    assume *finite ?A*
    hence *finite* ((λ*f*. *f undefined*) ` *?A*)
      by (*rule finite_imageI*)
    also have (λ*f*. *f undefined*) ` *?A* = *UNIV*
      by *auto*
    finally show *False*

```
    using ‹infinite (UNIV :: 'a set)› by contradiction
  qed
  also have ?A = Pi UNIV (λ_. UNIV)
    by auto
  finally show infinite (Pi (UNIV :: 'b set) (λ_. UNIV :: 'a set)) .
qed

proposition CARD_vec [simp]:
  CARD('a^'b) = CARD('a) ^ CARD('b)
proof (cases finite (UNIV :: 'a set))
  case True
  show ?thesis
  proof (subst bij_betw_same_card)
    show bij_betw vec_nth UNIV (Pi (UNIV :: 'b set) (λ_. UNIV :: 'a set))
      by (intro bij_betwI[of _ _ _ vec_lambda]) (auto simp: vec_eq_iff)
    have CARD('a) ^ CARD('b) = card (PiE (UNIV :: 'b set) (λ_. UNIV :: 'a
set))
      (is _ = card ?A)
      by (subst card_PiE) (auto)
    also have ?A = Pi UNIV (λ_. UNIV)
      by auto
    finally show card ... = CARD('a) ^ CARD('b) ..
  qed
qed (simp_all add: infinite_UNIV_vec)

lemma countable_vector:
  fixes B:: 'n::finite ⇒ 'a set
  assumes ⋀i. countable (B i)
  shows countable {V. ∀ i::'n::finite. V $ i ∈ B i}
proof −
  have f ∈ ($) ` {V. ∀ i. V $ i ∈ B i} if f ∈ Pi_E UNIV B for f
  proof −
    have ∃ W. (∀ i. W $ i ∈ B i) ∧ ($) W = f
      by (metis that PiE_iff UNIV_I vec_lambda_inverse)
    then show f ∈ ($) ` {v. ∀ i. v $ i ∈ B i}
      by blast
  qed
  then have Pi_E UNIV B = vec_nth ` {V. ∀ i::'n. V $ i ∈ B i}
    by blast
  then have countable (vec_nth ` {V. ∀ i. V $ i ∈ B i})
    by (metis finite_class.finite_UNIV countable_PiE assms)
  then have countable (vec_lambda ` vec_nth ` {V. ∀ i. V $ i ∈ B i})
    by auto
  then show ?thesis
    by (simp add: image_comp o_def vec_nth_inverse)
qed
```

1.8.3 Group operations and class instances

instantiation *vec* :: (*zero*, *finite*) *zero*
begin
  definition *0* ≡ (χ *i*. *0*)
  instance ..
end

instantiation *vec* :: (*plus*, *finite*) *plus*
begin
  definition (+) ≡ (λ *x y*. (χ *i*. *x*$*i* + *y*$*i*))
  instance ..
end

instantiation *vec* :: (*minus*, *finite*) *minus*
begin
  definition (−) ≡ (λ *x y*. (χ *i*. *x*$*i* − *y*$*i*))
  instance ..
end

instantiation *vec* :: (*uminus*, *finite*) *uminus*
begin
  definition *uminus* ≡ (λ *x*. (χ *i*. − (*x*$*i*)))
  instance ..
end

lemma *zero_index* [*simp*]: *0* $ *i* = *0*
  unfolding *zero_vec_def* **by** *simp*

lemma *vector_add_component* [*simp*]: (*x* + *y*)$*i* = *x*$*i* + *y*$*i*
  unfolding *plus_vec_def* **by** *simp*

lemma *vector_minus_component* [*simp*]: (*x* − *y*)$*i* = *x*$*i* − *y*$*i*
  unfolding *minus_vec_def* **by** *simp*

lemma *vector_uminus_component* [*simp*]: (− *x*)$*i* = − (*x*$*i*)
  unfolding *uminus_vec_def* **by** *simp*

instance *vec* :: (*semigroup_add*, *finite*) *semigroup_add*
  by *standard* (*simp add*: *vec_eq_iff add.assoc*)

instance *vec* :: (*ab_semigroup_add*, *finite*) *ab_semigroup_add*
  by *standard* (*simp add*: *vec_eq_iff add.commute*)

instance *vec* :: (*monoid_add*, *finite*) *monoid_add*
  by *standard* (*simp_all add*: *vec_eq_iff*)

instance *vec* :: (*comm_monoid_add*, *finite*) *comm_monoid_add*
  by *standard* (*simp add*: *vec_eq_iff*)

instance *vec* :: (*cancel_semigroup_add*, *finite*) *cancel_semigroup_add*
by *standard* (*simp_all add*: *vec_eq_iff*)

instance *vec* :: (*cancel_ab_semigroup_add*, *finite*) *cancel_ab_semigroup_add*
by *standard* (*simp_all add*: *vec_eq_iff diff_diff_eq*)

instance *vec* :: (*cancel_comm_monoid_add*, *finite*) *cancel_comm_monoid_add*
..

instance *vec* :: (*group_add*, *finite*) *group_add*
by *standard* (*simp_all add*: *vec_eq_iff*)

instance *vec* :: (*ab_group_add*, *finite*) *ab_group_add*
by *standard* (*simp_all add*: *vec_eq_iff*)

1.8.4 Basic componentwise operations on vectors

instantiation *vec* :: (*times*, *finite*) *times*
begin

definition $(*) \equiv (\lambda\ x\ y.\ \ (\chi\ i.\ (x\$i) * (y\$i)))$
instance ..

end

instantiation *vec* :: (*one*, *finite*) *one*
begin

definition $1 \equiv (\chi\ i.\ 1)$
instance ..

end

instantiation *vec* :: (*ord*, *finite*) *ord*
begin

definition $x \leq y \longleftrightarrow (\forall\, i.\ x\$i \leq y\$i)$
definition $x < (y::'a\,\hat{}\,'b) \longleftrightarrow x \leq y \wedge \neg\ y \leq x$
instance ..

end

The ordering on one-dimensional vectors is linear.

instance *vec*:: (*order*, *finite*) *order*
by *standard* (*auto simp*: *less_eq_vec_def less_vec_def vec_eq_iff*
intro: *order.trans order.antisym order.strict_implies_order*)

instance *vec* :: (*linorder*, *CARD_1*) *linorder*
proof

obtain $a :: 'b$ **where** *all*: $\bigwedge P.\ (\forall\, i.\ P\ i) \longleftrightarrow P\ a$
proof −
have $CARD\ ('b) = 1$ **by** (*rule CARD_1*)
then obtain $b :: 'b$ **where** $UNIV = \{b\}$ **by** (*auto iff*: *card_Suc_eq*)
then have $\bigwedge P.\ (\forall\, i{\in}UNIV.\ P\ i) \longleftrightarrow P\ b$ **by** *auto*
then show *thesis* **by** (*auto intro*: *that*)
qed
fix $x\ y :: 'a\hat{}'b{::}CARD\_1$
note [*simp*] = *less_eq_vec_def less_vec_def all vec_eq_iff field_simps*
show $x \le y \vee y \le x$ **by** *auto*
qed

Constant Vectors

definition $vec\ x = (\chi\ i.\ x)$

Also the scalar-vector multiplication.

definition $vector\_scalar\_mult{::}\ 'a{::}times \Rightarrow 'a\ \hat{}\ 'n \Rightarrow 'a\ \hat{}\ 'n$ (**infixl** $*s\ 70$)
where $c *s\ x = (\chi\ i.\ c * (x\$i))$

scalar product

definition $scalar\_product :: 'a :: semiring\_1\ \hat{}\ 'n \Rightarrow 'a\ \hat{}\ 'n \Rightarrow 'a$ **where**
$scalar\_product\ v\ w = (\sum\ i \in UNIV.\ v\ \$\ i * w\ \$\ i)$

1.8.5 Real vector space

instantiation *vec* :: (*real_vector*, *finite*) *real_vector*
begin

definition $scaleR \equiv (\lambda\ r\ x.\ (\chi\ i.\ scaleR\ r\ (x\$i)))$

lemma *vector_scaleR_component* [*simp*]: $(scaleR\ r\ x)\$i = scaleR\ r\ (x\$i)$
unfolding *scaleR_vec_def* **by** *simp*

instance
by *standard* (*simp_all add*: *vec_eq_iff scaleR_left_distrib scaleR_right_distrib*)

end

1.8.6 Topological space

instantiation *vec* :: (*topological_space*, *finite*) *topological_space*
begin

definition [*code del*]:
$open\ (S :: ('a\ \hat{}\ 'b)\ set) \longleftrightarrow$
$(\forall\, x{\in}S.\ \exists\, A.\ (\forall\, i.\ open\ (A\ i) \wedge x\$i \in A\ i) \wedge$
$(\forall\, y.\ (\forall\, i.\ y\$i \in A\ i) \longrightarrow y \in S))$

instance proof

```
  show open (UNIV :: ('a ^ 'b) set)
    unfolding open_vec_def by auto
next
  fix S T :: ('a ^ 'b) set
  assume open S open T thus open (S ∩ T)
    unfolding open_vec_def
    apply clarify
    apply (drule (1) bspec)+
    apply (clarify, rename_tac Sa Ta)
    apply (rule_tac x=λi. Sa i ∩ Ta i in exI)
    apply (simp add: open_Int)
    done
next
  fix K :: ('a ^ 'b) set set
  assume ∀S∈K. open S thus open (⋃K)
    unfolding open_vec_def
    by (metis Union_iff)
qed

end

lemma open_vector_box: ∀i. open (S i) ⟹ open {x. ∀i. x $ i ∈ S i}
  unfolding open_vec_def by auto

lemma open_vimage_vec_nth: open S ⟹ open ((λx. x $ i) -` S)
  unfolding open_vec_def
  apply clarify
  apply (rule_tac x=λk. if k = i then S else UNIV in exI, simp)
  done

lemma closed_vimage_vec_nth: closed S ⟹ closed ((λx. x $ i) -` S)
  unfolding closed_open vimage_Compl [symmetric]
  by (rule open_vimage_vec_nth)

lemma closed_vector_box: ∀i. closed (S i) ⟹ closed {x. ∀i. x $ i ∈ S i}
proof -
  have {x. ∀i. x $ i ∈ S i} = (⋂i. (λx. x $ i) -` S i) by auto
  thus ∀i. closed (S i) ⟹ closed {x. ∀i. x $ i ∈ S i}
    by (simp add: closed_INT closed_vimage_vec_nth)
qed

lemma tendsto_vec_nth [tendsto_intros]:
  assumes ((λx. f x) ⟶ a) net
  shows ((λx. f x $ i) ⟶ a $ i) net
proof (rule topological_tendstoI)
  fix S assume open S a $ i ∈ S
  then have open ((λy. y $ i) -` S) a ∈ ((λy. y $ i) -` S)
    by (simp_all add: open_vimage_vec_nth)
  with assms have eventually (λx. f x ∈ (λy. y $ i) -` S) net
```

```
    by (rule topological_tendstoD)
  then show eventually (λx. f x $ i ∈ S) net
    by simp
qed

lemma isCont_vec_nth [simp]: isCont f a ⟹ isCont (λx. f x $ i) a
  unfolding isCont_def by (rule tendsto_vec_nth)

lemma vec_tendstoI:
  assumes ⋀i. ((λx. f x $ i) ⟶ a $ i) net
  shows ((λx. f x) ⟶ a) net
proof (rule topological_tendstoI)
  fix S assume open S and a ∈ S
  then obtain A where A: ⋀i. open (A i) ⋀i. a $ i ∈ A i
    and S: ⋀y. ∀i. y $ i ∈ A i ⟹ y ∈ S
    unfolding open_vec_def by metis
  have ⋀i. eventually (λx. f x $ i ∈ A i) net
    using assms A by (rule topological_tendstoD)
  hence eventually (λx. ∀i. f x $ i ∈ A i) net
    by (rule eventually_all_finite)
  thus eventually (λx. f x ∈ S) net
    by (rule eventually_mono, simp add: S)
qed

lemma tendsto_vec_lambda [tendsto_intros]:
  assumes ⋀i. ((λx. f x i) ⟶ a i) net
  shows ((λx. χ i. f x i) ⟶ (χ i. a i)) net
  using assms by (simp add: vec_tendstoI)

lemma open_image_vec_nth: assumes open S shows open ((λx. x $ i) ` S)
proof (rule openI)
  fix a assume a ∈ (λx. x $ i) ` S
  then obtain z where a = z $ i and z ∈ S ..
  then obtain A where A: ∀i. open (A i) ∧ z $ i ∈ A i
    and S: ∀y. (∀i. y $ i ∈ A i) ⟶ y ∈ S
    using ‹open S› unfolding open_vec_def by auto
  hence A i ⊆ (λx. x $ i) ` S
    by (clarsimp, rule_tac x=χ j. if j = i then x else z $ j in image_eqI,
      simp_all)
  hence open (A i) ∧ a ∈ A i ∧ A i ⊆ (λx. x $ i) ` S
    using A ‹a = z $ i› by simp
  then show ∃T. open T ∧ a ∈ T ∧ T ⊆ (λx. x $ i) ` S by − (rule exI)
qed

instance vec :: (perfect_space, finite) perfect_space
proof
  fix x :: 'a ^ 'b show ¬ open {x}
  proof
    assume open {x}
```

hence $\forall i.$ *open* $((\lambda x.\ x\ \$\ i)\ `\ \{x\})$ **by** (*fast intro*: *open_image_vec_nth*)
hence $\forall i.$ *open* $\{x\ \$\ i\}$ **by** *simp*
thus *False* **by** (*simp add*: *not_open_singleton*)
qed
qed

1.8.7 Metric space

instantiation *vec* :: (*metric_space*, *finite*) *dist*
begin

definition
dist x y = *L2_set* $(\lambda i.$ *dist* $(x\$i)$ $(y\$i))$ *UNIV*

instance ..
end

instantiation *vec* :: (*metric_space*, *finite*) *uniformity_dist*
begin

definition [*code del*]:
(*uniformity* :: $(('a\hat{}'b::\_) \times ('a\hat{}'b::\_))$ *filter*) =
(*INF* $e\in\{0 <..\}.$ *principal* $\{(x, y).$ *dist x y* $< e\}$)

instance
by *standard* (*rule uniformity_vec_def*)
end

declare *uniformity_Abort*[**where** $'a='a$ :: *metric_space* $\hat{}$ $'b$ :: *finite*, *code*]

instantiation *vec* :: (*metric_space*, *finite*) *metric_space*
begin

proposition *dist_vec_nth_le*: *dist* $(x\ \$\ i)$ $(y\ \$\ i) \leq$ *dist x y*
unfolding *dist_vec_def* **by** (*rule member_le_L2_set*) *simp_all*

instance proof
fix *x y* :: $'a\ \hat{}\ 'b$
show *dist x y* $= 0 \longleftrightarrow x = y$
unfolding *dist_vec_def*
by (*simp add*: *L2_set_eq_0_iff vec_eq_iff*)
next
fix *x y z* :: $'a\ \hat{}\ 'b$
show *dist x y* $\leq$ *dist x z* + *dist y z*
unfolding *dist_vec_def*
apply (*rule order_trans* [*OF* _ *L2_set_triangle_ineq*])
apply (*simp add*: *L2_set_mono dist_triangle2*)
done
next

```
  fix S :: ('a ^ 'b) set
  have *: open S ⟷ (∀ x∈S. ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ S)
  proof
    assume open S show ∀ x∈S. ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ S
    proof
      fix x assume x ∈ S
      obtain A where A: ∀ i. open (A i) ∀ i. x $ i ∈ A i
        and S: ∀ y. (∀ i. y $ i ∈ A i) ⟶ y ∈ S
        using ‹open S› and ‹x ∈ S› unfolding open_vec_def by metis
      have ∀ i∈UNIV. ∃ r>0. ∀ y. dist y (x $ i) < r ⟶ y ∈ A i
        using A unfolding open_dist by simp
      hence ∃ r. ∀ i∈UNIV. 0 < r i ∧ (∀ y. dist y (x $ i) < r i ⟶ y ∈ A i)
        by (rule finite_set_choice [OF finite])
      then obtain r where r1: ∀ i. 0 < r i
        and r2: ∀ i y. dist y (x $ i) < r i ⟶ y ∈ A i by fast
      have 0 < Min (range r) ∧ (∀ y. dist y x < Min (range r) ⟶ y ∈ S)
        by (simp add: r1 r2 S le_less_trans [OF dist_vec_nth_le])
      thus ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ S ..
    qed
  next
    assume *: ∀ x∈S. ∃ e>0. ∀ y. dist y x < e ⟶ y ∈ S show open S
    proof (unfold open_vec_def, rule)
      fix x assume x ∈ S
      then obtain e where 0 < e and S: ∀ y. dist y x < e ⟶ y ∈ S
        using * by fast
      define r where [abs_def]: r i = e / sqrt (of_nat CARD('b)) for i :: 'b
      from ‹0 < e› have r: ∀ i. 0 < r i
        unfolding r_def by simp_all
      from ‹0 < e› have e: e = L2_set r UNIV
        unfolding r_def by (simp add: L2_set_constant)
      define A where A i = {y. dist (x $ i) y < r i} for i
      have ∀ i. open (A i) ∧ x $ i ∈ A i
        unfolding A_def by (simp add: open_ball r)
      moreover have ∀ y. (∀ i. y $ i ∈ A i) ⟶ y ∈ S
       by (simp add: A_def S dist_vec_def e L2_set_strict_mono dist_commute)
      ultimately show ∃ A. (∀ i. open (A i) ∧ x $ i ∈ A i) ∧
        (∀ y. (∀ i. y $ i ∈ A i) ⟶ y ∈ S) by metis
    qed
  qed
  show open S = (∀ x∈S. ∀ F (x', y) in uniformity. x' = x ⟶ y ∈ S)
    unfolding * eventually_uniformity_metric
    by (simp del: split_paired_All add: dist_vec_def dist_commute)
qed

end

lemma Cauchy_vec_nth:
  Cauchy (λn. X n) ⟹ Cauchy (λn. X n $ i)
  unfolding Cauchy_def by (fast intro: le_less_trans [OF dist_vec_nth_le])
```

```
lemma vec_CauchyI:
  fixes X :: nat ⇒ 'a::metric_space ^ 'n
  assumes X: ⋀i. Cauchy (λn. X n $ i)
  shows Cauchy (λn. X n)
proof (rule metric_CauchyI)
  fix r :: real assume 0 < r
  hence 0 < r / of_nat CARD('n) (is 0 < ?s) by simp
  define N where N i = (LEAST N. ∀m≥N. ∀n≥N. dist (X m $ i) (X n $ i)
< ?s) for i
  define M where M = Max (range N)
  have ⋀i. ∃N. ∀m≥N. ∀n≥N. dist (X m $ i) (X n $ i) < ?s
    using X ‹0 < ?s› by (rule metric_CauchyD)
  hence ⋀i. ∀m≥N i. ∀n≥N i. dist (X m $ i) (X n $ i) < ?s
    unfolding N_def by (rule LeastI_ex)
  hence M: ⋀i. ∀m≥M. ∀n≥M. dist (X m $ i) (X n $ i) < ?s
    unfolding M_def by simp
  {
    fix m n :: nat
    assume M ≤ m M ≤ n
    have dist (X m) (X n) = L2_set (λi. dist (X m $ i) (X n $ i)) UNIV
      unfolding dist_vec_def ..
    also have ... ≤ sum (λi. dist (X m $ i) (X n $ i)) UNIV
      by (rule L2_set_le_sum [OF zero_le_dist])
    also have ... < sum (λi::'n. ?s) UNIV
      by (rule sum_strict_mono, simp_all add: M ‹M ≤ m› ‹M ≤ n›)
    also have ... = r
      by simp
    finally have dist (X m) (X n) < r .
  }
  hence ∀m≥M. ∀n≥M. dist (X m) (X n) < r
    by simp
  then show ∃M. ∀m≥M. ∀n≥M. dist (X m) (X n) < r ..
qed

instance vec :: (complete_space, finite) complete_space
proof
  fix X :: nat ⇒ 'a ^ 'b assume Cauchy X
  have ⋀i. (λn. X n $ i) ⟶ lim (λn. X n $ i)
    using Cauchy_vec_nth [OF ‹Cauchy X›]
    by (simp add: Cauchy_convergent_iff convergent_LIMSEQ_iff)
  hence X ⟶ vec_lambda (λi. lim (λn. X n $ i))
    by (simp add: vec_tendstoI)
  then show convergent X
    by (rule convergentI)
qed
```

1.8.8 Normed vector space

instantiation *vec* :: (*real_normed_vector*, *finite*) *real_normed_vector*
begin

definition *norm x* = *L2_set* (λ*i*. *norm* (*x*$*i*)) *UNIV*

definition *sgn* (*x*::′*a*^′*b*) = *scaleR* (*inverse* (*norm x*)) *x*

instance proof
fix *a* :: *real* **and** *x y* :: ′*a* ^ ′*b*
show *norm x* = *0* ⟷ *x* = *0*
unfolding *norm_vec_def*
by (*simp add*: *L2_set_eq_0_iff vec_eq_iff*)
show *norm* (*x* + *y*) ≤ *norm x* + *norm y*
unfolding *norm_vec_def*
apply (*rule order_trans* [*OF* _ *L2_set_triangle_ineq*])
apply (*simp add*: *L2_set_mono norm_triangle_ineq*)
done
show *norm* (*scaleR a x*) = |*a*| ∗ *norm x*
unfolding *norm_vec_def*
by (*simp add*: *L2_set_right_distrib*)
show *sgn x* = *scaleR* (*inverse* (*norm x*)) *x*
by (*rule sgn_vec_def*)
show *dist x y* = *norm* (*x* − *y*)
unfolding *dist_vec_def norm_vec_def*
by (*simp add*: *dist_norm*)
qed

end

lemma *norm_nth_le*: *norm* (*x* $ *i*) ≤ *norm x*
unfolding *norm_vec_def*
by (*rule member_le_L2_set*) *simp_all*

lemma *norm_le_componentwise_cart*:
fixes *x* :: ′*a*::*real_normed_vector*^′*n*
assumes ⋀*i*. *norm*(*x*$*i*) ≤ *norm*(*y*$*i*)
shows *norm x* ≤ *norm y*
unfolding *norm_vec_def*
by (*rule L2_set_mono*) (*auto simp*: *assms*)

lemma *component_le_norm_cart*: |*x*$*i*| ≤ *norm x*
by (*metis norm_nth_le real_norm_def*)

lemma *norm_bound_component_le_cart*: *norm x* ≤ *e* ==> |*x*$*i*| ≤ *e*
by (*metis component_le_norm_cart order_trans*)

lemma *norm_bound_component_lt_cart*: *norm x* < *e* ==> |*x*$*i*| < *e*
by (*metis component_le_norm_cart le_less_trans*)

lemma *norm_le_l1_cart*: *norm x* ≤ *sum*(λ*i*. |*x*$*i*|) *UNIV*
by (*simp add*: *norm_vec_def L2_set_le_sum*)

lemma *bounded_linear_vec_nth*[*intro*]: *bounded_linear* (λ*x*. *x* $ *i*)
proof
show ∃ *K*. ∀ *x*. *norm* (*x* $ *i*) ≤ *norm x* ∗ *K*
by (*metis mult.commute mult.left_neutral norm_nth_le*)
qed *auto*

instance *vec* :: (*banach*, *finite*) *banach* **..**

1.8.9 Inner product space

instantiation *vec* :: (*real_inner*, *finite*) *real_inner*
begin

definition *inner x y* = *sum* (λ*i*. *inner* (*x*$*i*) (*y*$*i*)) *UNIV*

instance proof
fix *r* :: *real* **and** *x y z* :: *'a* ^ *'b*
show *inner x y* = *inner y x*
unfolding *inner_vec_def*
by (*simp add*: *inner_commute*)
show *inner* (*x* + *y*) *z* = *inner x z* + *inner y z*
unfolding *inner_vec_def*
by (*simp add*: *inner_add_left sum.distrib*)
show *inner* (*scaleR r x*) *y* = *r* ∗ *inner x y*
unfolding *inner_vec_def*
by (*simp add*: *sum_distrib_left*)
show *0* ≤ *inner x x*
unfolding *inner_vec_def*
by (*simp add*: *sum_nonneg*)
show *inner x x* = *0* ⟷ *x* = *0*
unfolding *inner_vec_def*
by (*simp add*: *vec_eq_iff sum_nonneg_eq_0_iff*)
show *norm x* = *sqrt* (*inner x x*)
unfolding *inner_vec_def norm_vec_def L2_set_def*
by (*simp add*: *power2_norm_eq_inner*)
qed

end

1.8.10 Euclidean space

Vectors pointing along a single axis.

definition *axis k x* = (χ *i*. *if i* = *k then x else 0*)

lemma *axis_nth* [*simp*]: *axis i x* $ *i* = *x*

unfolding *axis_def* **by** *simp*

lemma *axis_eq_axis*: *axis i x* = *axis j y* ⟷ *x* = *y* ∧ *i* = *j* ∨ *x* = *0* ∧ *y* = *0*
unfolding *axis_def vec_eq_iff* **by** *auto*

lemma *inner_axis_axis*:
inner (*axis i x*) (*axis j y*) = (*if i* = *j then inner x y else 0*)
by (*simp add*: *inner_vec_def axis_def sum.neutral sum.remove* [*of* _ *j*])

lemma *inner_axis*: *inner x* (*axis i y*) = *inner* (*x* $ *i*) *y*
by (*simp add*: *inner_vec_def axis_def sum.remove* [**where** *x*=*i*])

lemma *inner_axis'*: *inner*(*axis i y*) *x* = *inner y* (*x* $ *i*)
by (*simp add*: *inner_axis inner_commute*)

instantiation *vec* :: (*euclidean_space*, *finite*) *euclidean_space*
begin

definition *Basis* = (⋃ *i*. ⋃ *u*∈*Basis*. {*axis i u*})

instance proof
show (*Basis* :: (*'a* ^ *'b*) *set*) ≠ {}
unfolding *Basis_vec_def* **by** *simp*
next
show *finite* (*Basis* :: (*'a* ^ *'b*) *set*)
unfolding *Basis_vec_def* **by** *simp*
next
fix *u v* :: *'a* ^ *'b*
assume *u* ∈ *Basis* **and** *v* ∈ *Basis*
thus *inner u v* = (*if u* = *v then 1 else 0*)
unfolding *Basis_vec_def*
by (*auto simp add*: *inner_axis_axis axis_eq_axis inner_Basis*)
next
fix *x* :: *'a* ^ *'b*
show (∀ *u*∈*Basis*. *inner x u* = *0*) ⟷ *x* = *0*
unfolding *Basis_vec_def*
by (*simp add*: *inner_axis euclidean_all_zero_iff vec_eq_iff*)
qed

proposition *DIM_cart* [*simp*]: *DIM*(*'a*^*'b*) = *CARD*(*'b*) * *DIM*(*'a*)
proof −
have *card* (⋃ *i*::*'b*. ⋃ *u*::*'a*∈*Basis*. {*axis i u*}) = (∑ *i*::*'b*∈*UNIV*. *card* (⋃ *u*::*'a*∈*Basis*. {*axis i u*}))
by (*rule card_UN_disjoint*) (*auto simp*: *axis_eq_axis*)
also have ... = *CARD*(*'b*) * *DIM*(*'a*)
by (*subst card_UN_disjoint*) (*auto simp*: *axis_eq_axis*)
finally show *?thesis*
by (*simp add*: *Basis_vec_def*)
qed

end

lemma *norm_axis_1* [*simp*]: *norm* (*axis m* (*1*::*real*)) = *1*
by (*simp add*: *inner_axis'* *norm_eq_1*)

lemma *sum_norm_allsubsets_bound_cart*:
fixes *f*:: *'a* ⇒ *real* ^*'n*
assumes *fP*: *finite P* **and** *fPs*: ⋀*Q*. *Q* ⊆ *P* ⟹ *norm* (*sum f Q*) ≤ *e*
shows *sum* (λ*x*. *norm* (*f x*)) *P* ≤ *2* ∗ *real CARD*(*'n*) ∗ *e*
using *sum_norm_allsubsets_bound*[*OF assms*]
by *simp*

lemma *cart_eq_inner_axis*: *a* $ *i* = *inner a* (*axis i 1*)
by (*simp add*: *inner_axis*)

lemma *axis_eq_0_iff* [*simp*]:
shows *axis m x* = *0* ⟷ *x* = *0*
by (*simp add*: *axis_def vec_eq_iff*)

lemma *axis_in_Basis_iff* [*simp*]: *axis i a* ∈ *Basis* ⟷ *a* ∈ *Basis*
by (*auto simp*: *Basis_vec_def axis_eq_axis*)

Mapping each basis element to the corresponding finite index

definition *axis_index* :: (*'a*::*comm_ring_1*)^*'n* ⇒ *'n* **where** *axis_index v* ≡ *SOME i*. *v* = *axis i 1*

lemma *axis_inverse*:
fixes *v* :: *real*^*'n*
assumes *v* ∈ *Basis*
shows ∃ *i*. *v* = *axis i 1*
proof −
have *v* ∈ (⋃ *n*. ⋃ *r*∈*Basis*. {*axis n r*})
using *assms Basis_vec_def* **by** *blast*
then show *?thesis*
by (*force simp add*: *vec_eq_iff*)
qed

lemma *axis_index*:
fixes *v* :: *real*^*'n*
assumes *v* ∈ *Basis*
shows *v* = *axis* (*axis_index v*) *1*
by (*metis* (*mono_tags*) *assms axis_inverse axis_index_def someI_ex*)

lemma *axis_index_axis* [*simp*]:
fixes *UU* :: *real*^*'n*
shows (*axis_index* (*axis u 1* :: *real*^*'n*)) = (*u*::*'n*)
by (*simp add*: *axis_eq_axis axis_index_def*)

1.8.11 A naive proof procedure to lift really trivial arithmetic stuff from the basis of the vector space

lemma *sum_cong_aux*:
(⋀*x*. *x* ∈ *A* ⟹ *f x* = *g x*) ⟹ *sum f A* = *sum g A*
by (*auto intro*: *sum.cong*)

hide_fact (**open**) *sum_cong_aux*

method_setup *vector* = ‹
let
val ss1 =
simpset_of (*put_simpset HOL_basic_ss* ***context***
addsimps [@{*thm sum.distrib*} *RS sym*,
@{*thm sum_subtractf*} *RS sym*, @{*thm sum_distrib_left*},
@{*thm sum_distrib_right*}, @{*thm sum_negf*} *RS sym*])
val ss2 =
simpset_of (***context*** *addsimps*
[@{*thm plus_vec_def*}, @{*thm times_vec_def*},
@{*thm minus_vec_def*}, @{*thm uminus_vec_def*},
@{*thm one_vec_def*}, @{*thm zero_vec_def*}, @{*thm vec_def*},
@{*thm scaleR_vec_def*}, @{*thm vector_scalar_mult_def*}])
fun vector_arith_tac ctxt ths =
simp_tac (*put_simpset ss1 ctxt*)
THEN' (*fn i => resolve_tac ctxt* @{*thms Finite_Cartesian_Product.sum_cong_aux*}
i
ORELSE resolve_tac ctxt @{*thms sum.neutral*} *i*
ORELSE simp_tac (*put_simpset HOL_basic_ss ctxt addsimps* [@{*thm vec_eq_iff*}]) *i*)
(∗ *THEN' TRY o clarify_tac HOL_cs THEN'* (*TRY o rtac* @{*thm iffI*}) ∗)
THEN' asm_full_simp_tac (*put_simpset ss2 ctxt addsimps ths*)
in
Attrib.thms >> (*fn ths => fn ctxt => SIMPLE_METHOD'* (*vector_arith_tac ctxt ths*))
end
› *lift trivial vector statements to real arith statements*

lemma *vec_0*[*simp*]: *vec 0* = *0* **by** *vector*
lemma *vec_1*[*simp*]: *vec 1* = *1* **by** *vector*

lemma *vec_inj*[*simp*]: *vec x* = *vec y* ⟷ *x* = *y* **by** *vector*

lemma *vec_in_image_vec*: *vec x* ∈ (*vec* ‘ *S*) ⟷ *x* ∈ *S* **by** *auto*

lemma *vec_add*: *vec*(*x* + *y*) = *vec x* + *vec y* **by** *vector*
lemma *vec_sub*: *vec*(*x* − *y*) = *vec x* − *vec y* **by** *vector*
lemma *vec_cmul*: *vec*(*c* ∗ *x*) = *c* ∗*s vec x* **by** *vector*
lemma *vec_neg*: *vec*(− *x*) = − *vec x* **by** *vector*

lemma *vec_scaleR*: *vec*(*c* ∗ *x*) = *c* ∗*R vec x*

by *vector*

lemma *vec_sum*:
assumes *finite S*
shows *vec*(*sum f S*) = *sum* (*vec* ∘ *f*) *S*
using *assms*
proof *induct*
case *empty*
then show *?case* **by** *simp*
next
case *insert*
then show *?case* **by** (*auto simp add*: *vec_add*)
qed

Obvious "component-pushing".

lemma *vec_component* [*simp*]: *vec x* $ *i* = *x*
by *vector*

lemma *vector_mult_component* [*simp*]: (*x* ∗ *y*)$*i* = *x*$*i* ∗ *y*$*i*
by *vector*

lemma *vector_smult_component* [*simp*]: (*c* ∗*s* *y*)$*i* = *c* ∗ (*y*$*i*)
by *vector*

lemma *cond_component*: (*if b then x else y*)$*i* = (*if b then x*$*i else y*$*i*) **by** *vector*

lemmas *vector_component* =
vec_component vector_add_component vector_mult_component
vector_smult_component vector_minus_component vector_uminus_component
vector_scaleR_component cond_component

1.8.12 Some frequently useful arithmetic lemmas over vectors

instance *vec* :: (*semigroup_mult*, *finite*) *semigroup_mult*
by *standard* (*vector mult.assoc*)

instance *vec* :: (*monoid_mult*, *finite*) *monoid_mult*
by *standard vector*+

instance *vec* :: (*ab_semigroup_mult*, *finite*) *ab_semigroup_mult*
by *standard* (*vector mult.commute*)

instance *vec* :: (*comm_monoid_mult*, *finite*) *comm_monoid_mult*
by *standard vector*

instance *vec* :: (*semiring*, *finite*) *semiring*
by *standard* (*vector field_simps*)+

instance *vec* :: (*semiring_0*, *finite*) *semiring_0*
  by *standard* (*vector field_simps*)+
instance *vec* :: (*semiring_1*, *finite*) *semiring_1*
  by *standard vector*
instance *vec* :: (*comm_semiring*, *finite*) *comm_semiring*
  by *standard* (*vector field_simps*)+

instance *vec* :: (*comm_semiring_0*, *finite*) *comm_semiring_0* **..**
instance *vec* :: (*semiring_0_cancel*, *finite*) *semiring_0_cancel* **..**
instance *vec* :: (*comm_semiring_0_cancel*, *finite*) *comm_semiring_0_cancel* **..**
instance *vec* :: (*ring*, *finite*) *ring* **..**
instance *vec* :: (*semiring_1_cancel*, *finite*) *semiring_1_cancel* **..**
instance *vec* :: (*comm_semiring_1*, *finite*) *comm_semiring_1* **..**

instance *vec* :: (*ring_1*, *finite*) *ring_1* **..**

instance *vec* :: (*real_algebra*, *finite*) *real_algebra*
  by *standard* (*simp_all add*: *vec_eq_iff*)

instance *vec* :: (*real_algebra_1*, *finite*) *real_algebra_1* **..**

lemma *of_nat_index*: (*of_nat n* :: *'a*::*semiring_1* ^*'n*)$*i* = *of_nat n*
proof (*induct n*)
  case *0*
  then show *?case* **by** *vector*
next
  case *Suc*
  then show *?case* **by** *vector*
qed

lemma *one_index* [*simp*]: (*1* :: *'a* :: *one* ^ *'n*) $ *i* = *1*
  by *vector*

lemma *neg_one_index* [*simp*]: (*− 1* :: *'a* :: {*one*, *uminus*} ^ *'n*) $ *i* = *− 1*
  by *vector*

instance *vec* :: (*semiring_char_0*, *finite*) *semiring_char_0*
proof
  fix *m n* :: *nat*
  show *inj* (*of_nat* :: *nat* ⇒ *'a* ^ *'b*)
    by (*auto intro*!: *injI simp add*: *vec_eq_iff of_nat_index*)
qed

instance *vec* :: (*numeral*, *finite*) *numeral* **..**
instance *vec* :: (*semiring_numeral*, *finite*) *semiring_numeral* **..**

lemma *numeral_index* [*simp*]: *numeral w* $ *i* = *numeral w*
  by (*induct w*) (*simp_all only*: *numeral.simps vector_add_component one_index*)

lemma *neg_numeral_index* [*simp*]: $- \mathit{numeral}\ w\ \$\ i = - \mathit{numeral}\ w$
by (*simp only*: *vector_uminus_component numeral_index*)

instance *vec* :: (*comm_ring_1*, *finite*) *comm_ring_1* **..**
instance *vec* :: (*ring_char_0*, *finite*) *ring_char_0* **..**

lemma *vector_smult_assoc*: $a *s (b *s x) = ((a::'a::\mathit{semigroup\_mult}) * b) *s x$
by (*vector mult.assoc*)
lemma *vector_sadd_rdistrib*: $((a::'a::\mathit{semiring}) + b) *s x = a *s x + b *s x$
by (*vector field_simps*)
lemma *vector_add_ldistrib*: $(c::'a::\mathit{semiring}) *s (x + y) = c *s x + c *s y$
by (*vector field_simps*)
lemma *vector_smult_lzero*[*simp*]: $(0::'a::\mathit{mult\_zero}) *s x = 0$ **by** *vector*
lemma *vector_smult_lid*[*simp*]: $(1::'a::\mathit{monoid\_mult}) *s x = x$ **by** *vector*
lemma *vector_ssub_ldistrib*: $(c::'a::\mathit{ring}) *s (x - y) = c *s x - c *s y$
by (*vector field_simps*)
lemma *vector_smult_rneg*: $(c::'a::\mathit{ring}) *s -x = -(c *s x)$ **by** *vector*
lemma *vector_smult_lneg*: $- (c::'a::\mathit{ring}) *s x = -(c *s x)$ **by** *vector*
lemma *vector_sneg_minus1*: $-x = (-1::'a::\mathit{ring\_1}) *s x$ **by** *vector*
lemma *vector_smult_rzero*[*simp*]: $c *s 0 = (0::'a::\mathit{mult\_zero}\ \hat{}\ 'n)$ **by** *vector*
lemma *vector_sub_rdistrib*: $((a::'a::\mathit{ring}) - b) *s x = a *s x - b *s x$
by (*vector field_simps*)

lemma *vec_eq*[*simp*]: $(\mathit{vec}\ m = \mathit{vec}\ n) \longleftrightarrow (m = n)$
by (*simp add*: *vec_eq_iff*)

lemma *Vector_Spaces_linear_vec* [*simp*]: *Vector_Spaces.linear* $(*)$ *vector_scalar_mult vec*
by *unfold_locales* (*vector algebra_simps*)+

lemma *vector_mul_eq_0*[*simp*]: $(a *s x = 0) \longleftrightarrow a = (0::'a::\mathit{idom}) \vee x = 0$
by *vector*

lemma *vector_mul_lcancel*[*simp*]: $a *s x = a *s y \longleftrightarrow a = (0::'a::\mathit{field}) \vee x = y$
by (*metis eq_iff_diff_eq_0 vector_mul_eq_0 vector_ssub_ldistrib*)

lemma *vector_mul_rcancel*[*simp*]: $a *s x = b *s x \longleftrightarrow (a::'a::\mathit{field}) = b \vee x = 0$
by (*subst eq_iff_diff_eq_0*, *subst vector_sub_rdistrib* [*symmetric*]) *simp*

lemma *scalar_mult_eq_scaleR* [*abs_def*]: $c *s x = c *_R x$
unfolding *scaleR_vec_def vector_scalar_mult_def* **by** *simp*

lemma *dist_mul*[*simp*]: $\mathit{dist}\ (c *s x)\ (c *s y) = |c| * \mathit{dist}\ x\ y$
unfolding *dist_norm scalar_mult_eq_scaleR*
unfolding *scaleR_right_diff_distrib*[*symmetric*] **by** *simp*

lemma *sum_component* [*simp*]:
fixes $f::\ 'a \Rightarrow ('b::\mathit{comm\_monoid\_add})\ \hat{}\ 'n$
shows $(\mathit{sum}\ f\ S)\$i = \mathit{sum}\ (\lambda x.\ (f\ x)\$i)\ S$

proof (*cases finite S*)
case *True*
then show *?thesis* **by** *induct simp_all*
next
case *False*
then show *?thesis* **by** *simp*
qed

lemma *sum_eq*: *sum f S* = (χ *i*. *sum* (λ*x*. (*f x*)$*i*) *S*)
by (*simp add*: *vec_eq_iff*)

lemma *sum_cmul*:
fixes *f*:: *'c* ⇒ (*'a*::*semiring_1*)^*'n*
shows *sum* (λ*x*. *c* ∗*s f x*) *S* = *c* ∗*s sum f S*
by (*simp add*: *vec_eq_iff sum_distrib_left*)

lemma *linear_vec* [*simp*]: *linear vec*
using *Vector_Spaces_linear_vec*
by *unfold_locales* (*vector algebra_simps*)+

1.8.13 Matrix operations

Matrix notation. NB: an MxN matrix is of type ((*'a*, *'n*) *vec*, *'m*) *vec*, not ((*'a*, *'m*) *vec*, *'n*) *vec*

definition *map_matrix*::(*'a* ⇒ *'b*) ⇒ ((*'a*, *'i*::*finite*)*vec*, *'j*::*finite*) *vec* ⇒ ((*'b*, *'i*)*vec*, *'j*) *vec* **where**
map_matrix f x = (χ *i j*. *f* (*x* $ *i* $ *j*))

lemma *nth_map_matrix*[*simp*]: *map_matrix f x* $ *i* $ *j* = *f* (*x* $ *i* $ *j*)
by (*simp add*: *map_matrix_def*)

definition *matrix_matrix_mult* :: (*'a*::*semiring_1*) ^*'n*^*'m* ⇒ *'a* ^*'p*^*'n* ⇒ *'a* ^ *'p* ^*'m*
(**infixl** ∗∗ *70*)
where *m* ∗∗ *m'* == (χ *i j*. *sum* (λ*k*. ((*m*$*i*)$*k*) ∗ ((*m'*$*k*)$*j*)) (*UNIV* :: *'n set*)) ::*'a* ^ *'p* ^*'m*

definition *matrix_vector_mult* :: (*'a*::*semiring_1*) ^*'n*^*'m* ⇒ *'a* ^*'n* ⇒ *'a* ^ *'m*
(**infixl** ∗*v* *70*)
where *m* ∗*v x* ≡ (χ *i*. *sum* (λ*j*. ((*m*$*i*)$*j*) ∗ (*x*$*j*)) (*UNIV* ::*'n set*)) :: *'a*^*'m*

definition *vector_matrix_mult* :: *'a* ^ *'m* ⇒ (*'a*::*semiring_1*) ^*'n*^*'m* ⇒ *'a* ^*'n*
(**infixl** *v*∗ *70*)
where *v v*∗ *m* == (χ *j*. *sum* (λ*i*. ((*m*$*i*)$*j*) ∗ (*v*$*i*)) (*UNIV* :: *'m set*)) :: *'a*^*'n*

definition (*mat*::*'a*::*zero* => *'a* ^*'n*^*'n*) *k* = (χ *i j*. *if i* = *j then k else 0*)
definition *transpose* **where**
(*transpose*::*'a*^*'n*^*'m* ⇒ *'a*^*'m*^*'n*) *A* = (χ *i j*. ((*A*$*j*)$*i*))
definition (*row*::*'m* => *'a* ^*'n*^*'m* ⇒ *'a* ^*'n*) *i A* = (χ *j*. ((*A*$*i*)$*j*))

definition (*column*::'*n* =>'*a*^'*n*^'*m* =>'*a*^'*m*) *j A* = (χ *i*. ((*A*$*i*)$*j*))
definition *rows*(*A*::'*a*^'*n*^'*m*) = { *row i A* | *i*. *i* ∈ (*UNIV* :: '*m set*)}
definition *columns*(*A*::'*a*^'*n*^'*m*) = { *column i A* | *i*. *i* ∈ (*UNIV* :: '*n set*)}

lemma *times0_left* [*simp*]: (*0*::'*a*::*semiring_1*^'*n*^'*m*) ** (*A*::'*a* ^'*p*^'*n*) = *0*
by (*simp add*: *matrix_matrix_mult_def zero_vec_def*)

lemma *times0_right* [*simp*]: (*A*::'*a*::*semiring_1*^'*n*^'*m*) ** (*0*::'*a* ^'*p*^'*n*) = *0*
by (*simp add*: *matrix_matrix_mult_def zero_vec_def*)

lemma *mat_0*[*simp*]: *mat 0* = *0* **by** (*vector mat_def*)
lemma *matrix_add_ldistrib*: (*A* ** (*B* + *C*)) = (*A* ** *B*) + (*A* ** *C*)
by (*vector matrix_matrix_mult_def sum.distrib*[*symmetric*] *field_simps*)

lemma *matrix_mul_lid* [*simp*]:
fixes *A* :: '*a*::*semiring_1* ^ '*m* ^ '*n*
shows *mat 1* ** *A* = *A*
unfolding *matrix_matrix_mult_def mat_def*
by (*auto simp*: *if_distrib if_distribR sum.delta'*[*OF finite*] *cong*: *if_cong*)

lemma *matrix_mul_rid* [*simp*]:
fixes *A* :: '*a*::*semiring_1* ^ '*m* ^ '*n*
shows *A* ** *mat 1* = *A*
unfolding *matrix_matrix_mult_def mat_def*
by (*auto simp*: *if_distrib if_distribR sum.delta'*[*OF finite*] *cong*: *if_cong*)

proposition *matrix_mul_assoc*: *A* ** (*B* ** *C*) = (*A* ** *B*) ** *C*
apply (*vector matrix_matrix_mult_def sum_distrib_left sum_distrib_right mult.assoc*)
apply (*subst sum.swap*)
apply *simp*
done

proposition *matrix_vector_mul_assoc*: *A* *v (*B* *v *x*) = (*A* ** *B*) *v *x*
apply (*vector matrix_matrix_mult_def matrix_vector_mult_def*
sum_distrib_left sum_distrib_right mult.assoc)
apply (*subst sum.swap*)
apply *simp*
done

proposition *scalar_matrix_assoc*:
fixes *A* :: ('*a*::*real_algebra_1*)^'*m*^'*n*
shows *k* *$_R$ (*A* ** *B*) = (*k* *$_R$ *A*) ** *B*
by (*simp add*: *matrix_matrix_mult_def sum_distrib_left mult_ac vec_eq_iff scaleR_sum_right*)

proposition *matrix_scalar_ac*:
fixes *A* :: ('*a*::*real_algebra_1*)^'*m*^'*n*
shows *A* ** (*k* *$_R$ *B*) = *k* *$_R$ *A* ** *B*
by (*simp add*: *matrix_matrix_mult_def sum_distrib_left mult_ac vec_eq_iff*)

lemma *matrix_vector_mul_lid* [*simp*]: *mat 1 ∗v x = (x::'a::semiring_1 ^ 'n)*
apply (*vector matrix_vector_mult_def mat_def*)
apply (*simp add*: *if_distrib if_distribR cong del*: *if_weak_cong*)
done

lemma *matrix_transpose_mul*:
transpose(A ∗∗ B) = transpose B ∗∗ transpose (*A::'a::comm_semiring_1^_ ^_*)
by (*simp add*: *matrix_matrix_mult_def transpose_def vec_eq_iff mult.commute*)

lemma *matrix_mult_transpose_dot_column*:
shows *transpose A ∗∗ A* = (*χ i j. inner* (*column i A*) (*column j A*))
by (*simp add*: *matrix_matrix_mult_def vec_eq_iff transpose_def column_def inner_vec_def*)

lemma *matrix_mult_transpose_dot_row*:
shows *A ∗∗ transpose A* = (*χ i j. inner* (*row i A*) (*row j A*))
by (*simp add*: *matrix_matrix_mult_def vec_eq_iff transpose_def row_def inner_vec_def*)

lemma *matrix_eq*:
fixes *A B* :: *'a::semiring_1 ^ 'n ^ 'm*
shows *A* = *B* ⟷ (∀ *x. A ∗v x* = *B ∗v x*) (**is** *?lhs* ⟷ *?rhs*)
proof
assume *?rhs*
then show *?lhs*
apply (*subst vec_eq_iff*)
apply (*clarsimp simp add*: *matrix_vector_mult_def if_distrib if_distribR vec_eq_iff cong*: *if_cong*)
apply (*erule_tac x=axis ia 1* **in** *allE*)
apply (*erule_tac x=i* **in** *allE*)
apply (*auto simp add*: *if_distrib if_distribR axis_def*
sum.delta[*OF finite*] *cong del*: *if_weak_cong*)
done
qed *auto*

lemma *matrix_vector_mul_component*: (*A ∗v x*)$*k* = *inner* (*A$k*) *x*
by (*simp add*: *matrix_vector_mult_def inner_vec_def*)

lemma *dot_lmul_matrix*: *inner* ((*x::real ^_*) *v∗ A*) *y* = *inner x* (*A ∗v y*)
apply (*simp add*: *inner_vec_def matrix_vector_mult_def vector_matrix_mult_def sum_distrib_right sum_distrib_left ac_simps*)
apply (*subst sum.swap*)
apply *simp*
done

lemma *transpose_mat* [*simp*]: *transpose* (*mat n*) = *mat n*
by (*vector transpose_def mat_def*)

lemma *transpose_transpose* [*simp*]: *transpose*(*transpose A*) = *A*
by (*vector transpose_def*)

lemma *row_transpose* [*simp*]: *row i* (*transpose A*) = *column i A*
by (*simp add*: *row_def column_def transpose_def vec_eq_iff*)

lemma *column_transpose* [*simp*]: *column i* (*transpose A*) = *row i A*
by (*simp add*: *row_def column_def transpose_def vec_eq_iff*)

lemma *rows_transpose* [*simp*]: *rows*(*transpose A*) = *columns A*
by (*auto simp add*: *rows_def columns_def intro*: *set_eqI*)

lemma *columns_transpose* [*simp*]: *columns*(*transpose A*) = *rows A*
by (*metis transpose_transpose rows_transpose*)

lemma *transpose_scalar*: *transpose* (*k* $*_R$ *A*) = *k* $*_R$ *transpose A*
unfolding *transpose_def*
by (*simp add*: *vec_eq_iff*)

lemma *transpose_iff* [*iff*]: *transpose A* = *transpose B* ⟷ *A* = *B*
by (*metis transpose_transpose*)

lemma *matrix_mult_sum*:
(*A*::'*a*::*comm_semiring_1*^'*n*^'*m*) *v* *x* = *sum* (λ*i*. (*x*$*i*) *s* *column i A*) (*UNIV*::
'*n set*)
by (*simp add*: *matrix_vector_mult_def vec_eq_iff column_def mult.commute*)

lemma *vector_componentwise*:
(*x*::'*a*::*ring_1*^'*n*) = (χ *j*. ∑ *i*∈*UNIV*. (*x*$*i*) * (*axis i 1* :: '*a*^'*n*) $ *j*)
by (*simp add*: *axis_def if_distrib sum.If_cases vec_eq_iff*)

lemma *basis_expansion*: *sum* (λ*i*. (*x*$*i*) *s* *axis i 1*) *UNIV* = (*x*::('*a*::*ring_1*) ^'*n*)
by (*auto simp add*: *axis_def vec_eq_iff if_distrib sum.If_cases cong del*: *if_weak_cong*)

Correspondence between matrices and linear operators.

definition *matrix* :: ('*a*::{*plus*,*times*, *one*, *zero*}^'*m* ⇒ '*a* ^ '*n*) ⇒ '*a*^'*m*^'*n*
where *matrix f* = (χ *i j*. (*f*(*axis j 1*))$*i*)

lemma *matrix_id_mat_1*: *matrix id* = *mat 1*
by (*simp add*: *mat_def matrix_def axis_def*)

lemma *matrix_scaleR*: (*matrix* ((*$*_R*) *r*)) = *mat r*
by (*simp add*: *mat_def matrix_def axis_def if_distrib cong*: *if_cong*)

lemma *matrix_vector_mul_linear*[*intro*, *simp*]: *linear* (λ*x*. *A* *v* (*x*::'*a*::*real_algebra_1*
^ _))
by (*simp add*: *linear_iff matrix_vector_mult_def vec_eq_iff field_simps sum_distrib_left*
sum.distrib scaleR_right.sum)

lemma *vector_matrix_left_distrib* [*algebra_simps*]:
shows $(x + y)\ v*\ A = x\ v*\ A + y\ v*\ A$
unfolding *vector_matrix_mult_def*
by (*simp add*: *algebra_simps sum.distrib vec_eq_iff*)

lemma *matrix_vector_right_distrib* [*algebra_simps*]:
$A *v (x + y) = A *v x + A *v y$
by (*vector matrix_vector_mult_def sum.distrib distrib_left*)

lemma *matrix_vector_mult_diff_distrib* [*algebra_simps*]:
fixes $A :: 'a::ring\_1\hat{}'n\hat{}'m$
shows $A *v (x - y) = A *v x - A *v y$
by (*vector matrix_vector_mult_def sum_subtractf right_diff_distrib*)

lemma *matrix_vector_mult_scaleR*[*algebra_simps*]:
fixes $A :: real\hat{}'n\hat{}'m$
shows $A *v (c *_R x) = c *_R (A *v x)$
using *linear_iff matrix_vector_mul_linear* **by** *blast*

lemma *matrix_vector_mult_0_right* [*simp*]: $A *v 0 = 0$
by (*simp add*: *matrix_vector_mult_def vec_eq_iff*)

lemma *matrix_vector_mult_0* [*simp*]: $0 *v w = 0$
by (*simp add*: *matrix_vector_mult_def vec_eq_iff*)

lemma *matrix_vector_mult_add_rdistrib* [*algebra_simps*]:
$(A + B) *v x = (A *v x) + (B *v x)$
by (*vector matrix_vector_mult_def sum.distrib distrib_right*)

lemma *matrix_vector_mult_diff_rdistrib* [*algebra_simps*]:
fixes $A :: 'a :: ring\_1\hat{}'n\hat{}'m$
shows $(A - B) *v x = (A *v x) - (B *v x)$
by (*vector matrix_vector_mult_def sum_subtractf left_diff_distrib*)

lemma *matrix_vector_column*:
$(A::'a::comm\_semiring\_1\hat{}'n\hat{}\_)\ *v\ x = sum\ (\lambda i.\ (x\$i)\ *s\ ((transpose\ A)\$i))$
$(UNIV :: 'n\ set)$
by (*simp add*: *matrix_vector_mult_def transpose_def vec_eq_iff mult.commute*)

1.8.14 Inverse matrices (not necessarily square)

definition
$invertible(A::'a::semiring\_1\hat{}'n\hat{}'m) \longleftrightarrow (\exists A'::'a\hat{}'m\hat{}'n.\ A ** A' = mat\ 1 \wedge A' ** A = mat\ 1)$

definition
matrix_inv$(A:: 'a::semiring\_1\hat{}'n\hat{}'m) =$
$(SOME\ A'::'a\hat{}'m\hat{}'n.\ A ** A' = mat\ 1 \wedge A' ** A = mat\ 1)$

lemma *inj_matrix_vector_mult*:
fixes A::'a::*field*^'n^'m
assumes *invertible* A
shows *inj* ((∗v) A)
by (*metis assms inj_on_inverseI invertible_def matrix_vector_mul_assoc matrix_vector_mul_lid*)

lemma *scalar_invertible*:
fixes A :: ('a::*real_algebra_1*)^'m^'n
assumes $k \neq 0$ **and** *invertible* A
shows *invertible* (k ∗$_R$ A)
proof −
obtain A' **where** A ∗∗ $A' = mat\ 1$ **and** A' ∗∗ $A = mat\ 1$
using *assms* **unfolding** *invertible_def* **by** *auto*
with ‹$k \neq 0$›
have (k ∗$_R$ A) ∗∗ ((1/k) ∗$_R$ A') = *mat 1* ((1/k) ∗$_R$ A') ∗∗ (k ∗$_R$ A) = *mat 1*
by (*simp_all add*: *assms matrix_scalar_ac*)
thus *invertible* (k ∗$_R$ A)
unfolding *invertible_def* **by** *auto*
qed

proposition *scalar_invertible_iff*:
fixes A :: ('a::*real_algebra_1*)^'m^'n
assumes $k \neq 0$ **and** *invertible* A
shows *invertible* (k ∗$_R$ A) ⟷ $k \neq 0 \wedge$ *invertible* A
by (*simp add*: *assms scalar_invertible*)

lemma *vector_transpose_matrix* [*simp*]: x *v∗ transpose* $A = A$ ∗*v* x
unfolding *transpose_def vector_matrix_mult_def matrix_vector_mult_def*
by *simp*

lemma *transpose_matrix_vector* [*simp*]: *transpose* A ∗*v* $x = x$ *v∗* A
unfolding *transpose_def vector_matrix_mult_def matrix_vector_mult_def*
by *simp*

lemma *vector_scalar_commute*:
fixes A :: 'a::{*field*}^'m^'n
shows A ∗*v* (c ∗*s* x) = c ∗*s* (A ∗*v* x)
by (*simp add*: *vector_scalar_mult_def matrix_vector_mult_def mult_ac sum_distrib_left*)

lemma *scalar_vector_matrix_assoc*:
fixes k :: 'a::{*field*} **and** x :: 'a::{*field*}^'n **and** A :: 'a^'m^'n
shows (k ∗*s* x) *v∗* $A = k$ ∗*s* (x *v∗* A)
by (*metis transpose_matrix_vector vector_scalar_commute*)

lemma *vector_matrix_mult_0* [*simp*]: *0 v∗* $A = 0$
unfolding *vector_matrix_mult_def* **by** (*simp add*: *zero_vec_def*)

lemma *vector_matrix_mult_0_right* [*simp*]: x *v∗ 0* = *0*

unfolding *vector_matrix_mult_def* **by** (*simp add*: *zero_vec_def*)

lemma *vector_matrix_mul_rid* [*simp*]:
fixes *v* :: ('*a*::*semiring_1*)^'*n*
shows *v v∗ mat 1* = *v*
by (*metis matrix_vector_mul_lid transpose_mat vector_transpose_matrix*)

lemma *scaleR_vector_matrix_assoc*:
fixes *k* :: *real* **and** *x* :: *real*^'*n* **and** *A* :: *real*^'*m*^'*n*
shows (*k* ∗*R x*) *v∗ A* = *k* ∗*R* (*x v∗ A*)
by (*metis matrix_vector_mult_scaleR transpose_matrix_vector*)

proposition *vector_scaleR_matrix_ac*:
fixes *k* :: *real* **and** *x* :: *real*^'*n* **and** *A* :: *real*^'*m*^'*n*
shows *x v∗* (*k* ∗*R A*) = *k* ∗*R* (*x v∗ A*)
proof −
have *x v∗* (*k* ∗*R A*) = (*k* ∗*R x*) *v∗ A*
unfolding *vector_matrix_mult_def*
by (*simp add*: *algebra_simps*)
with *scaleR_vector_matrix_assoc*
show *x v∗* (*k* ∗*R A*) = *k* ∗*R* (*x v∗ A*)
by *auto*
qed

end

1.9 Linear Algebra on Finite Cartesian Products

theory *Cartesian_Space*
imports
HOL−Combinatorics.Transposition
Finite_Cartesian_Product
Linear_Algebra
begin

1.9.1 Type ('*a*, '*n*) *vec* and fields as vector spaces

definition *cart_basis* = {*axis i 1* | *i*. *i*∈*UNIV*}

lemma *finite_cart_basis*: *finite* (*cart_basis*) **unfolding** *cart_basis_def*
using *finite_Atleast_Atmost_nat* **by** *fastforce*

lemma *card_cart_basis*: *card* (*cart_basis*::('*a*::*zero_neq_one*^'*i*) *set*) = *CARD*('*i*)
unfolding *cart_basis_def Setcompr_eq_image*
by (*rule card_image*) (*auto simp*: *inj_on_def axis_eq_axis*)

interpretation *vec*: *vector_space* (∗*s*)
by *unfold_locales* (*vector algebra_simps*)+

```
lemma independent_cart_basis: vec.independent (cart_basis)
proof (rule vec.independent_if_scalars_zero)
  show finite (cart_basis) using finite_cart_basis .
  fix f::('a, 'b) vec ⇒ 'a and x::('a, 'b) vec
  assume eq_0: (∑ x∈cart_basis. f x *s x) = 0 and x_in: x ∈ cart_basis
  obtain i where x: x = axis i 1 using x_in unfolding cart_basis_def by auto
  have sum_eq_0: (∑ x∈(cart_basis) - {x}. f x * (x $ i)) = 0
  proof (intro sum.neutral ballI)
    fix y assume y: y ∈ cart_basis - {x}
    obtain a where a: y = axis a 1 and a_not_i: a ≠ i
      using y x unfolding cart_basis_def by auto
    have y $ i = 0 unfolding a axis_def using a_not_i by auto
    thus f y * y $ i = 0 by simp
  qed
  have 0 = (∑ x∈cart_basis. f x *s x) $ i using eq_0 by simp
  also have ... = (∑ x∈cart_basis. (f x *s x) $ i) unfolding sum_component ..
 also have ... = (∑ x∈cart_basis. f x * (x $ i)) unfolding vector_smult_component
..
  also have ... = f x * (x $ i) + (∑ x∈(cart_basis) - {x}. f x * (x $ i))
    by (rule sum.remove[OF finite_cart_basis x_in])
  also have ... =  f x * (x $ i) unfolding sum_eq_0 by simp
  also have ... = f x unfolding x axis_def by auto
  finally show f x = 0 ..
qed

lemma span_cart_basis [simp]: vec.span (cart_basis) = UNIV
proof -
  have x ∈ vec.span cart_basis for x :: ('a, 'b) vec
  proof -
    let ?f=λv. x $ (THE i. v = axis i 1)
    have x $ i = (∑ v∈cart_basis. x $ (THE i. v = axis i 1) *s v) $ i for i::'b
    proof -
      let ?w = axis i (1::'a)
      have the_eq_i: (THE a. ?w = axis a 1) = i
        by (rule the_equality, auto simp: axis_eq_axis)
      have sum_eq_0: (∑ v∈(cart_basis) - {?w}. x $ (THE i. v = axis i 1) * v
$ i) = 0
      proof (intro sum.neutral ballI)
        fix y:: ('a, 'b) vec
        assume y: y ∈ cart_basis - {?w}
        obtain j where j: y = axis j 1 and i_not_j: i ≠ j
          using y unfolding cart_basis_def by auto
        have the_eq_j: (THE i. y = axis i 1) = j
          by (simp add: axis_eq_axis j)
        show x $ (THE i. y = axis i 1) * y $ i = 0
          by (simp add: axis_def i_not_j j)
      qed
      have (∑ v∈cart_basis. x $ (THE i. v = axis i 1) *s v) $ i
          = (∑ v∈cart_basis. x $ (THE i. v = axis i 1) * v $ i)
```

```
      by force
    also have ... = x $ (THE a. ?w = axis a 1) * ?w $ i + (∑ v∈(cart_basis)
- {?w}. x $ (THE i. v = axis i 1) * v $ i)
      by (rule sum.remove[OF finite_cart_basis], auto simp add: cart_basis_def)
    also have ... = x $ (THE a. ?w = axis a 1) * ?w $ i
      unfolding sum_eq_0 by simp
    also have ... = x $ i
      unfolding the_eq_i unfolding axis_def by auto
    finally show ?thesis by simp
  qed
  then show x ∈ vec.span (cart_basis)
    by (metis (no_types, lifting) vec.span_base vec.span_scale vec.span_sum
vec_eq_iff)
 qed
 then show ?thesis by auto
qed


interpretation vec: finite_dimensional_vector_space (*s) cart_basis
  by (unfold_locales, auto simp add: finite_cart_basis independent_cart_basis
span_cart_basis)

lemma matrix_vector_mul_linear_gen[intro, simp]:
 Vector_Spaces.linear (*s) (*s) ((*v) A)
 by unfold_locales
  (vector matrix_vector_mult_def sum.distrib algebra_simps)+

lemma span_vec_eq: vec.span X = span X
 and dim_vec_eq: vec.dim X = dim X
 and dependent_vec_eq: vec.dependent X = dependent X
 and subspace_vec_eq: vec.subspace X = subspace X
 for X::(real^'n) set
 unfolding span_raw_def dim_raw_def dependent_raw_def subspace_raw_def
 by (auto simp: scalar_mult_eq_scaleR)

lemma linear_componentwise:
 fixes f:: 'a::field ^'m ⇒ 'a ^ 'n
 assumes lf: Vector_Spaces.linear (*s) (*s) f
 shows (f x)$j = sum (λi. (x$i) * (f (axis i 1)$j)) (UNIV :: 'm set) (is ?lhs =
?rhs)
proof −
 interpret lf: Vector_Spaces.linear (*s) (*s) f
  using lf .
 let ?M = (UNIV :: 'm set)
 let ?N = (UNIV :: 'n set)
 have fM: finite ?M by simp
 have ?rhs = (sum (λi. (x$i) *s (f (axis i 1))) ?M)$j
  unfolding sum_component by simp
 then show ?thesis
```

unfolding *lf.sum*[*symmetric*] *lf.scale*[*symmetric*]
unfolding *basis_expansion* **by** *auto*
qed

interpretation *vec*: *Vector_Spaces.linear* (∗*s*) (∗*s*) (∗*v*) *A*
using *matrix_vector_mul_linear_gen*.

interpretation *vec*: *finite_dimensional_vector_space_pair* (∗*s*) *cart_basis* (∗*s*) *cart_basis* ..

lemma *matrix_works*:
assumes *lf*: *Vector_Spaces.linear* (∗*s*) (∗*s*) *f*
shows *matrix f* ∗*v x* = *f* (*x*::'*a*::*field* ^ '*n*)
proof −
have ∀ *i*. (∑ *j*∈*UNIV*. *x* $ *j* ∗ *f* (*axis j 1*) $ *i*) = *f x* $ *i*
by (*simp add*: *Cartesian_Space.linear_componentwise lf*)
then show *?thesis*
by (*simp add*: *matrix_def matrix_vector_mult_def vec_eq_iff mult.commute*)
qed

lemma *matrix_of_matrix_vector_mul*[*simp*]: *matrix*(λ*x*. *A* ∗*v* (*x* :: '*a*::*field* ^ '*n*))
= *A*
by (*simp add*: *matrix_eq matrix_works*)

lemma *matrix_compose_gen*:
assumes *lf*: *Vector_Spaces.linear* (∗*s*) (∗*s*) (*f*::'*a*::{*field*}^'*n* ⇒ '*a*^'*m*)
and *lg*: *Vector_Spaces.linear* (∗*s*) (∗*s*) (*g*::'*a*^'*m* ⇒ '*a*^_)
shows *matrix* (*g o f*) = *matrix g* ∗∗ *matrix f*
using *lf lg Vector_Spaces.linear_compose*[*OF lf lg*] *matrix_works*[*OF Vector_Spaces.linear_compose*[*OF lf lg*]]
by (*simp add*: *matrix_eq matrix_works matrix_vector_mul_assoc*[*symmetric*] *o_def*)

lemma *matrix_compose*:
assumes *linear* (*f*::*real*^'*n* ⇒ *real*^'*m*) *linear* (*g*::*real*^'*m* ⇒ *real*^_)
shows *matrix* (*g o f*) = *matrix g* ∗∗ *matrix f*
using *matrix_compose_gen*[*of f g*] *assms*
by (*simp add*: *linear_def scalar_mult_eq_scaleR*)

lemma *left_invertible_transpose*:
(∃(*B*). *B* ∗∗ *transpose* (*A*) = *mat* (*1*::'*a*::*comm_semiring_1*)) ⟷ (∃(*B*). *A* ∗∗ *B* = *mat 1*)
by (*metis matrix_transpose_mul transpose_mat transpose_transpose*)

lemma *right_invertible_transpose*:
(∃(*B*). *transpose* (*A*) ∗∗ *B* = *mat* (*1*::'*a*::*comm_semiring_1*)) ⟷ (∃(*B*). *B* ∗∗ *A* = *mat 1*)
by (*metis matrix_transpose_mul transpose_mat transpose_transpose*)

lemma *linear_matrix_vector_mul_eq*:
Vector_Spaces.linear $(*_s)$ $(*_s)$ $f \longleftrightarrow$ *linear* $(f :: real^{\wedge}{}'n \Rightarrow real^{\wedge}{}'m)$
by (*simp add*: *scalar_mult_eq_scaleR linear_def*)

lemma *matrix_vector_mul*[*simp*]:
Vector_Spaces.linear $(*_s)$ $(*_s)$ $g \Longrightarrow (\lambda y.\ matrix\ g *_v y) = g$
linear $f \Longrightarrow (\lambda x.\ matrix\ f *_v x) = f$
bounded_linear $f \Longrightarrow (\lambda x.\ matrix\ f *_v x) = f$
for $f :: real^{\wedge}{}'n \Rightarrow real^{\wedge}{}'m$
by (*simp_all add*: *ext matrix_works linear_matrix_vector_mul_eq linear_linear*)

lemma *matrix_left_invertible_injective*:
fixes $A :: 'a{::}field^{\wedge}{}'n^{\wedge}{}'m$
shows $(\exists B.\ B ** A = mat\ 1) \longleftrightarrow inj\ ((*_v)\ A)$
proof *safe*
fix *B*
assume *B*: $B ** A = mat\ 1$
show *inj* $((*_v)\ A)$
unfolding *inj_on_def*
by (*metis B matrix_vector_mul_assoc matrix_vector_mul_lid*)
next
assume *inj* $((*_v)\ A)$
from *vec.linear_injective_left_inverse*[*OF matrix_vector_mul_linear_gen this*]
obtain *g* **where** *Vector_Spaces.linear* $(*_s)$ $(*_s)$ *g* **and** $g \circ (*_v)\ A = id$
by *blast*
then have *matrix g* $** A = mat\ 1$
by (*metis matrix_compose_gen matrix_id_mat_1 matrix_of_matrix_vector_mul vec.linear_axioms*)
then show $\exists B.\ B ** A = mat\ 1$
by *metis*
qed

lemma *matrix_left_invertible_ker*:
$(\exists B.\ (B::'a::\{field\}\ ^{\wedge}{}'m^{\wedge}{}'n) ** (A::'a::\{field\}\ ^{\wedge}{}'n^{\wedge}{}'m) = mat\ 1) \longleftrightarrow (\forall x.\ A *_v x = 0 \longrightarrow x = 0)$
by (*simp add*: *matrix_left_invertible_injective vec.inj_iff_eq_0*)

lemma *matrix_right_invertible_surjective*:
$(\exists B.\ (A::'a::field^{\wedge}{}'n^{\wedge}{}'m) ** (B::'a::field^{\wedge}{}'m^{\wedge}{}'n) = mat\ 1) \longleftrightarrow surj\ (\lambda x.\ A *_v x)$
proof −
have $\bigwedge B\ x.\ A ** B = mat\ 1 \Longrightarrow \exists y.\ x = A *_v y$
by (*metis matrix_vector_mul_assoc matrix_vector_mul_lid*)
moreover have $\forall x.\ \exists xa.\ x = A *_v xa \Longrightarrow \exists B.\ A ** B = mat\ 1$
by (*metis* (*mono_tags*, *lifting*) *matrix_compose_gen matrix_id_mat_1 matrix_of_matrix_vector_mul surj_def vec.linear_axioms vec.linear_surjective_right_inverse*)
ultimately show *?thesis*
by (*auto simp*: *image_def set_eq_iff*)
qed

lemma *matrix_left_invertible_independent_columns*:
fixes $A :: 'a::\{field\}^'n^'m$
shows $(\exists (B::'a^'m^'n).\ B ** A = mat\ 1) \longleftrightarrow$
$(\forall c.\ sum\ (\lambda i.\ c\ i *s column\ i\ A)\ (UNIV :: 'n\ set) = 0 \longrightarrow (\forall i.\ c\ i = 0))$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof −
let *?U* = *UNIV* :: *'n set*
have *c i* = *0*
if $\forall x.\ A *v x = 0 \longrightarrow x = 0$ *sum* $(\lambda i.\ c\ i *s column\ i\ A)$ *?U* = *0* **for** *c i*
by (*metis* (*no_types*) *UNIV_I matrix_mult_sum vec_lambda_eta vec_nth_cases zero_vec_def that*)
moreover have *x* = *0* **if** *A* $*v$ *x* = *0 ?rhs* **for** *x*
by (*metis* (*full_types*) *matrix_mult_sum that vec_eq_iff zero_index*)
ultimately show *?thesis*
unfolding *matrix_left_invertible_ker* **by** *auto*
qed

lemma *matrix_right_invertible_independent_rows*:
fixes $A :: 'a::\{field\}^'n^'m$
shows $(\exists (B::'a^'m^'n).\ A ** B = mat\ 1) \longleftrightarrow$
$(\forall c.\ sum\ (\lambda i::'m.\ c\ i *s row\ i\ A)\ UNIV = 0 \longrightarrow (\forall i.\ c\ i = 0))$
by (*simp add*: *matrix_left_invertible_independent_columns flip*: *left_invertible_transpose*)

lemma *matrix_right_invertible_span_columns*:
$(\exists (B::'a::field^'n^'m).\ (A::'a^'m^'n) ** B = mat\ 1) \longleftrightarrow$
vec.span (*columns A*) = *UNIV* (**is** *?lhs* = *?rhs*)
proof −
let *?U* = *UNIV* :: *'m set*
have *fU*: *finite ?U* **by** *simp*
have *lhseq*: *?lhs* $\longleftrightarrow (\forall y.\ \exists (x::'a^'m).\ sum\ (\lambda i.\ (x\$i) *s column\ i\ A)\ ?U = y)$
unfolding *matrix_right_invertible_surjective matrix_mult_sum surj_def*
by (*simp add*: *eq_commute*)
have *rhseq*: *?rhs* $\longleftrightarrow (\forall x.\ x \in$ *vec.span* (*columns A*)) **by** *blast*
{ assume *h*: *?lhs*
{ fix *x*:: *'a* $^'n$
obtain *y* :: *'a* $^'m$ **where** *y*: *sum* $(\lambda i.\ (y\$i) *s column\ i\ A)$ *?U* = *x*
using *h lhseq* **by** *blast*
then have *x* ∈ *vec.span* (*columns A*)
by (*metis* (*mono_tags*, *lifting*) *columns_def mem_Collect_eq vec.span_base vec.span_scale vec.span_sum*)
}
then have *?rhs* **unfolding** *rhseq* **by** *blast* **}**
moreover
{ assume *h*:*?rhs*
let *?P* = $\lambda(y::'a^'n).\ \exists (x::'a^'m).\ sum\ (\lambda i.\ (x\$i) *s column\ i\ A)\ ?U = y$
{ fix *y*
have *y* ∈ *vec.span* (*columns A*)
unfolding *h* **by** *blast*
then have *?P y*

```
    proof (induction rule: vec.span_induct_alt)
      case base
      then show ?case
        by (metis (full_types) matrix_mult_sum matrix_vector_mult_0_right)
    next
      case (step c y1 y2)
      from step obtain i where i: i ∈ ?U y1 = column i A
        unfolding columns_def by blast
      obtain x:: 'a ^'m where x: sum (λi. (x$i) *s column i A) ?U = y2
        using step by blast
      let ?x = (χ j. if j = i then c + (x$i) else (x$j))::'a^'m
      show ?case
        proof (rule exI[where x= ?x], vector, auto simp add: i x[symmetric]
if_distrib distrib_left if_distribR cong del: if_weak_cong)
        fix j
        have th: ∀ xa ∈ ?U. (if xa = i then (c + (x$i)) * ((column xa A)$j)
          else (x$xa) * ((column xa A$j))) = (if xa = i then c * ((column i A)$j)
else 0) + ((x$xa) * ((column xa A)$j))
          using i(1) by (simp add: field_simps)
        have sum (λxa. if xa = i then (c + (x$i)) * ((column xa A)$j)
            else (x$xa) * ((column xa A$j))) ?U = sum (λxa. (if xa = i then c *
((column i A)$j) else 0) + ((x$xa) * ((column xa A)$j))) ?U
          using th by force
        also have ... = sum (λxa. if xa = i then c * ((column i A)$j) else 0) ?U
+ sum (λxa. ((x$xa) * ((column xa A)$j))) ?U
          by (simp add: sum.distrib)
        also have ... = c * ((column i A)$j) + sum (λxa. ((x$xa) * ((column xa
A)$j))) ?U
          unfolding sum.delta[OF fU] using i(1) by simp
        finally show sum (λxa. if xa = i then (c + (x$i)) * ((column xa A)$j)
                                  else (x$xa) * ((column xa A$j))) ?U
                     = c * ((column i A)$j) + sum (λxa. ((x$xa) * ((column xa
A)$j))) ?U .
      qed
    qed
  }
  then have ?lhs unfolding lhseq ..
 }
 ultimately show ?thesis by blast
qed

lemma matrix_left_invertible_span_rows_gen:
  (∃(B::'a^'m^'n). B ** (A::'a::field^'n^'m) = mat 1) ⟷ vec.span (rows A) =
UNIV
 by (metis columns_transpose matrix_right_invertible_span_columns right_invertible_transpose)

lemma matrix_left_invertible_span_rows:
  (∃(B::real^'m^'n). B ** (A::real^'n^'m) = mat 1) ⟷ span (rows A) = UNIV
 using matrix_left_invertible_span_rows_gen[of A] by (simp add: span_vec_eq)
```

lemma *matrix_left_right_inverse*:
fixes A A' :: $'a$::{*field*}$^{\sim}n^{\sim}n$
shows A ** $A' = mat\ 1 \longleftrightarrow A'$ ** $A = mat\ 1$
proof −
{ fix A A' :: $'a\ ^{\sim}n^{\sim}n$
assume AA': A ** $A' = mat\ 1$
have sA: *surj* ((∗*v*) A)
using AA' *matrix_right_invertible_surjective* **by** *auto*
obtain f' :: $'a\ ^{\sim}n \Rightarrow 'a\ ^{\sim}n$
where f': *Vector_Spaces.linear* (∗*s*) (∗*s*) $f' \forall x.\ f'\ (A *v\ x) = x \forall x.\ A *v\ f'$ $x = x$
using sA *vec.linear_surjective_isomorphism* **by** *blast*
have *matrix* f' ** $A = mat\ 1$
by (*metis* f' *matrix_eq matrix_vector_mul_assoc matrix_vector_mul_lid matrix_works*)
hence A' ** $A = mat\ 1$
by (*metis* AA' *matrix_mul_assoc matrix_mul_lid*)
}
then show *?thesis* **by** *blast*
qed

lemma *invertible_left_inverse*:
fixes A :: $'a$::{*field*}$^{\sim}n^{\sim}n$
shows *invertible* $A \longleftrightarrow (\exists (B::'a^{\sim}n^{\sim}n).\ B$ ** $A = mat\ 1)$
by (*metis invertible_def matrix_left_right_inverse*)

lemma *invertible_right_inverse*:
fixes A :: $'a$::{*field*}$^{\sim}n^{\sim}n$
shows *invertible* $A \longleftrightarrow (\exists (B::'a^{\sim}n^{\sim}n).\ A$** $B = mat\ 1)$
by (*metis invertible_def matrix_left_right_inverse*)

lemma *invertible_mult*:
assumes *inv_A*: *invertible* A
and *inv_B*: *invertible* B
shows *invertible* (A**B)
proof −
obtain A' **where** AA': A ** $A' = mat\ 1$ **and** $A'A$: A' ** $A = mat\ 1$
using *inv_A* **unfolding** *invertible_def* **by** *blast*
obtain B' **where** BB': B ** $B' = mat\ 1$ **and** $B'B$: B' ** $B = mat\ 1$
using *inv_B* **unfolding** *invertible_def* **by** *blast*
have A ** B ** (B' ** $A') = mat\ 1$
by (*metis* AA' BB' *matrix_mul_assoc matrix_mul_rid*)
moreover have B' ** A' ** (A ** $B) = mat\ 1$
by (*metis* $A'A$ $B'B$ *matrix_mul_assoc matrix_mul_rid*)
ultimately show *?thesis*
using *invertible_def* **by** *blast*
qed

lemma *transpose_invertible*:
fixes $A :: real^{\wedge}{}'n^{\wedge}{}'n$
assumes *invertible A*
shows *invertible* (*transpose A*)
by (*meson assms invertible_def matrix_left_right_inverse right_invertible_transpose*)

lemma *vector_matrix_mul_assoc*:
fixes $v :: ('a::comm\_semiring\_1)^{\wedge}{}'n$
shows $(v\ v*\ M)\ v*\ N = v\ v*\ (M\ **\ N)$
by (*metis* (*no_types*, *opaque_lifting*) *matrix_transpose_mul matrix_vector_mul_assoc transpose_matrix_vector*)

lemma *matrix_scaleR_vector_ac*:
fixes $A :: real^{\wedge}('m::finite)^{\wedge}{}'n$
shows $A\ *v\ (k\ *_R\ v) = k\ *_R\ A\ *v\ v$
by (*metis matrix_vector_mult_scaleR transpose_scalar vector_scaleR_matrix_ac vector_transpose_matrix*)

lemma *scaleR_matrix_vector_assoc*:
fixes $A :: real^{\wedge}('m::finite)^{\wedge}{}'n$
shows $k\ *_R\ (A\ *v\ v) = k\ *_R\ A\ *v\ v$
by (*metis matrix_scaleR_vector_ac matrix_vector_mult_scaleR*)

1.9.2 Some interesting theorems and interpretations

locale *linear_first_finite_dimensional_vector_space* =
l?: *Vector_Spaces.linear scaleB scaleC f* +
B?: *finite_dimensional_vector_space scaleB BasisB*
for *scaleB* :: ('*a*::*field* => '*b*::*ab_group_add* => '*b*) (**infixr** $*b$ *75*)
and *scaleC* :: ('*a* => '*c*::*ab_group_add* => '*c*) (**infixr** $*c$ *75*)
and *BasisB* :: ('*b set*)
and *f* :: ('*b*=>'*c*)

lemma *vec_dim_card*: *vec.dim* (*UNIV*::('*a*::{*field*}$^{\wedge}$'*n*) *set*) = *CARD* ('*n*)
by (*simp add*: *card_cart_basis*)

interpretation *vector_space_over_itself*: *vector_space* (∗) :: '*a*::*field* ⇒ '*a* ⇒ '*a*
by *unfold_locales* (*simp_all add*: *algebra_simps*)

lemmas [*simp del*] = *vector_space_over_itself.scale_scale*

interpretation *vector_space_over_itself*: *finite_dimensional_vector_space*
(∗) :: '*a*::*field* => '*a* => '*a* {*1*}
by *unfold_locales* (*auto simp*: *vector_space_over_itself.span_singleton*)

lemma *dimension_eq_1*[*code_unfold*]: *vector_space_over_itself.dimension TYPE*('*a*::*field*)= *1*
unfolding *vector_space_over_itself.dimension_def* **by** *simp*

lemma *dim_subset_UNIV_cart_gen*:
fixes $S :: ('a::field^{\frown}'n)\ set$
shows $vec.dim\ S \leq CARD('n)$
by (*metis vec.dim_eq_full vec.dim_subset_UNIV vec.span_UNIV vec_dim_card*)

lemma *dim_subset_UNIV_cart*:
fixes $S :: (real^{\frown}'n)\ set$
shows $dim\ S \leq CARD('n)$
using *dim_subset_UNIV_cart_gen*[*of S*] **by** (*simp add*: *dim_vec_eq*)

Two sometimes fruitful ways of looking at matrix-vector multiplication.

lemma *matrix_mult_dot*: $A *v x = (\chi\ i.\ inner\ (A\$i)\ x)$
by (*simp add*: *matrix_vector_mult_def inner_vec_def*)

lemma *adjoint_matrix*: $adjoint(\lambda x.\ (A::real^{\frown}'n^{\frown}'m) *v\ x) = (\lambda x.\ transpose\ A *v\ x)$
by (*metis adjoint_unique dot_lmul_matrix vector_transpose_matrix*)

lemma *matrix_adjoint*:
assumes *lf*: $linear\ (f :: real^{\frown}'n \Rightarrow real\ ^{\frown}'m)$
shows $matrix(adjoint\ f) = transpose(matrix\ f)$
by (*metis adjoint_matrix assms matrix_of_matrix_vector_mul matrix_vector_mul*(*2*))

1.9.3 Rank of a matrix

Equivalence of row and column rank is taken from George Mackiw's paper, Mathematics Magazine 1995, p. 285.

lemma *matrix_vector_mult_in_columnspace_gen*:
fixes $A :: 'a::field^{\frown}'n^{\frown}'m$
shows $(A *v x) \in vec.span(columns\ A)$
unfolding *columns_def*
by (*metis* (*mono_tags*, *lifting*) *matrix_mult_sum mem_Collect_eq vec.span_base vec.span_scale vec.span_sum*)

lemma *matrix_vector_mult_in_columnspace*:
fixes $A :: real^{\frown}'n^{\frown}'m$
shows $(A *v x) \in span(columns\ A)$
using *matrix_vector_mult_in_columnspace_gen*[*of A x*] **by** (*simp add*: *span_vec_eq*)

lemma *subspace_orthogonal_to_vector*: $subspace\ \{y.\ orthogonal\ x\ y\}$
by (*simp add*: *subspace_def orthogonal_clauses*)

lemma *orthogonal_nullspace_rowspace*:
fixes $A :: real^{\frown}'n^{\frown}'m$
assumes *0*: $A *v x = 0$ **and** *y*: $y \in span(rows\ A)$
shows $orthogonal\ x\ y$
using y

proof (*induction rule*: *span_induct*)
case *base*
then show *?case*
by (*simp add*: *subspace_orthogonal_to_vector*)
next
case (*step v*)
then obtain *i* **where** *v* = *row i A*
by (*auto simp*: *rows_def*)
with *0* **show** *?case*
unfolding *orthogonal_def inner_vec_def matrix_vector_mult_def row_def*
by (*simp add*: *mult.commute*) (*metis* (*no_types*) *vec_lambda_beta zero_index*)
qed

lemma *nullspace_inter_rowspace*:
fixes *A* :: *real^'n^'m*
shows *A* ∗*v x* = *0* ∧ *x* ∈ *span*(*rows A*) ⟷ *x* = *0*
using *orthogonal_nullspace_rowspace orthogonal_self span_zero matrix_vector_mult_0_right*
by *blast*

lemma *matrix_vector_mul_injective_on_rowspace*:
fixes *A* :: *real^'n^'m*
shows ⟦*A* ∗*v x* = *A* ∗*v y*; *x* ∈ *span*(*rows A*); *y* ∈ *span*(*rows A*)⟧ ⟹ *x* = *y*
using *nullspace_inter_rowspace* [*of A x*−*y*]
by (*metis diff_eq_diff_eq diff_self matrix_vector_mult_diff_distrib span_diff*)

definition *rank* :: *'a*::*field^'n^'m*⇒*nat*
where *row_rank_def_gen*: *rank A* ≡ *vec.dim*(*rows A*)

lemma *row_rank_def*: *rank A* = *dim* (*rows A*) **for** *A*::*real^'n^'m*
by (*auto simp*: *row_rank_def_gen dim_vec_eq*)

lemma *dim_rows_le_dim_columns*:
fixes *A* :: *real^'n^'m*
shows *dim*(*rows A*) ≤ *dim*(*columns A*)
proof −
have *dim* (*span* (*rows A*)) ≤ *dim* (*span* (*columns A*))
proof −
obtain *B* **where** *independent B span*(*rows A*) ⊆ *span B*
and *B*: *B* ⊆ *span*(*rows A*)*card B* = *dim* (*span*(*rows A*))
using *basis_exists* [*of span*(*rows A*)] **by** *metis*
with *span_subspace* **have** *eq*: *span B* = *span*(*rows A*)
by *auto*
then have *inj*: *inj_on* ((∗*v*) *A*) (*span B*)
by (*simp add*: *inj_on_def matrix_vector_mul_injective_on_rowspace*)
then have *ind*: *independent* ((∗*v*) *A* ‘ *B*)
by (*rule linear_independent_injective_image* [*OF Finite_Cartesian_Product.matrix_vector_mul_li*
‹*independent B*›])
have *dim* (*span* (*rows A*)) ≤ *card* ((∗*v*) *A* ‘ *B*)
by (*metis B*(*2*) *card_image inj inj_on_subset order.refl span_superset*)

```
    also have ... ≤ dim (span (columns A))
      using _ ind
    by (rule independent_card_le_dim) (auto intro!: matrix_vector_mult_in_columnspace)
    finally show ?thesis .
  qed
  then show ?thesis
    by (simp)
qed

lemma column_rank_def:
  fixes A :: real^'n^'m
  shows rank A = dim(columns A)
  unfolding row_rank_def
  by (metis columns_transpose dim_rows_le_dim_columns le_antisym rows_transpose)

lemma rank_transpose:
  fixes A :: real^'n^'m
  shows rank(transpose A) = rank A
  by (metis column_rank_def row_rank_def rows_transpose)

lemma matrix_vector_mult_basis:
  fixes A :: real^'n^'m
  shows A *v (axis k 1) = column k A
  by (simp add: cart_eq_inner_axis column_def matrix_mult_dot)

lemma columns_image_basis:
  fixes A :: real^'n^'m
  shows columns A = (*v) A ` (range (λi. axis i 1))
  by (force simp: columns_def matrix_vector_mult_basis [symmetric])

lemma rank_dim_range:
  fixes A :: real^'n^'m
  shows rank A = dim(range (λx. A *v x))
  unfolding column_rank_def
  by (smt (verit, best) columns_image_basis dim_span image_subset_iff iso_tuple_UNIV_I
matrix_vector_mult_in_columnspace span_eq)

lemma rank_bound:
  fixes A :: real^'n^'m
  shows rank A ≤ min CARD('m) (CARD('n))
  by (metis (mono_tags, lifting) dim_subset_UNIV_cart min.bounded_iff
      column_rank_def row_rank_def)

lemma full_rank_injective:
  fixes A :: real^'n^'m
  shows rank A = CARD('n) ⟷ inj ((*v) A)
  by (simp add: matrix_left_invertible_injective [symmetric] matrix_left_invertible_span_rows
row_rank_def
      dim_eq_full [symmetric] card_cart_basis vec.dimension_def)
```

lemma *full_rank_surjective*:
fixes $A :: real^{\prime}n^{\prime}m$
shows *rank* $A = CARD('m) \longleftrightarrow$ *surj* $((*v)\ A)$
by (*metis* (*no_types*, *opaque_lifting*) *dim_eq_full dim_vec_eq rank_dim_range span_vec_eq vec.span_UNIV vec.span_image vec_dim_card*)

lemma *rank_I*: *rank*(*mat 1*::$real^{\prime}n^{\prime}n$) = $CARD('n)$
by (*simp add*: *full_rank_injective inj_on_def*)

lemma *less_rank_noninjective*:
fixes $A :: real^{\prime}n^{\prime}m$
shows *rank* $A < CARD('n) \longleftrightarrow \neg$ *inj* $((*v)\ A)$
using *less_le rank_bound* **by** (*auto simp*: *full_rank_injective* [*symmetric*])

lemma *matrix_nonfull_linear_equations_eq*:
fixes $A :: real^{\prime}n^{\prime}m$
shows $(\exists x.\ (x \neq 0) \wedge A *v\ x = 0) \longleftrightarrow$ *rank* $A \neq CARD('n)$
by (*meson matrix_left_invertible_injective full_rank_injective matrix_left_invertible_ker*)

lemma *rank_eq_0*: *rank* $A = 0 \longleftrightarrow A = 0$ **and** *rank_0* [*simp*]: *rank* (*0*::$real^{\prime}n^{\prime}m$) = *0*
for $A :: real^{\prime}n^{\prime}m$
by (*auto simp*: *rank_dim_range matrix_eq*)

lemma *rank_mul_le_right*:
fixes $A :: real^{\prime}n^{\prime}m$ **and** $B :: real^{\prime}p^{\prime}n$
shows *rank*$(A ** B) \leq$ *rank* B
proof −
have *rank*$(A ** B) \leq$ *dim* $((*v)\ A$ ‘ *range* $((*v)\ B))$
by (*auto simp*: *rank_dim_range image_comp o_def matrix_vector_mul_assoc*)
also have ... $\leq$ *rank* B
by (*simp add*: *rank_dim_range dim_image_le*)
finally show *?thesis* **.**
qed

lemma *rank_mul_le_left*:
fixes $A :: real^{\prime}n^{\prime}m$ **and** $B :: real^{\prime}p^{\prime}n$
shows *rank*$(A ** B) \leq$ *rank* A
by (*metis matrix_transpose_mul rank_mul_le_right rank_transpose*)

1.9.4 Lemmas for working on *real^1/2/3/4*

lemma *exhaust_2*:
fixes $x :: 2$
shows $x = 1 \vee x = 2$
proof (*induct x*)
case (*of_int z*)
then have $z = 0 \mid z = 1$

```
    by fastforce
  then show ?case
    by auto
qed

lemma forall_2: (∀ i::2. P i) ⟷ P 1 ∧ P 2
  by (metis exhaust_2)

lemma exhaust_3:
  fixes x :: 3
  shows x = 1 ∨ x = 2 ∨ x = 3
proof (induct x)
  case (of_int z)
  then have z = 0 ∨ z = 1 ∨ z = 2 by fastforce
  then show ?case by auto
qed

lemma forall_3: (∀ i::3. P i) ⟷ P 1 ∧ P 2 ∧ P 3
  by (metis exhaust_3)

lemma exhaust_4:
  fixes x :: 4
  shows x = 1 ∨ x = 2 ∨ x = 3 ∨ x = 4
proof (induct x)
  case (of_int z)
  then have z = 0 ∨ z = 1 ∨ z = 2 ∨ z = 3 by fastforce
  then show ?case by auto
qed

lemma forall_4: (∀ i::4. P i) ⟷ P 1 ∧ P 2 ∧ P 3 ∧ P 4
  by (metis exhaust_4)

lemma UNIV_1 [simp]: UNIV = {1::1}
  by (auto simp add: num1_eq_iff)

lemma UNIV_2: UNIV = {1::2, 2::2}
  using exhaust_2 by auto

lemma UNIV_3: UNIV = {1::3, 2::3, 3::3}
  using exhaust_3 by auto

lemma UNIV_4: UNIV = {1::4, 2::4, 3::4, 4::4}
  using exhaust_4 by auto

lemma sum_1: sum f (UNIV::1 set) = f 1
  unfolding UNIV_1 by simp

lemma sum_2: sum f (UNIV::2 set) = f 1 + f 2
  unfolding UNIV_2 by simp
```

lemma *sum_3*: *sum f* (*UNIV*::*3 set*) = *f 1* + *f 2* + *f 3*
unfolding *UNIV_3* **by** (*simp add*: *ac_simps*)

lemma *sum_4*: *sum f* (*UNIV*::*4 set*) = *f 1* + *f 2* + *f 3* + *f 4*
unfolding *UNIV_4* **by** (*simp add*: *ac_simps*)

1.9.5 The collapse of the general concepts to dimension one

lemma *vector_one*: (*x*::*'a* ^*1*) = (χ *i*. (*x*$*1*))
by (*simp add*: *vec_eq_iff*)

lemma *forall_one*: (∀(*x*::*'a* ^*1*). *P x*) ⟷ (∀*x*. *P*(χ *i*. *x*))
by (*metis vector_one*)

lemma *norm_vector_1*: *norm* (*x* :: _^*1*) = *norm* (*x*$*1*)
by (*simp add*: *norm_vec_def*)

lemma *dist_vector_1*:
fixes *x* :: *'a*::*real_normed_vector*^*1*
shows *dist x y* = *dist* (*x*$*1*) (*y*$*1*)
by (*simp add*: *dist_norm norm_vector_1*)

lemma *norm_real*: *norm*(*x*::*real* ^ *1*) = |*x*$*1*|
by (*simp add*: *norm_vector_1*)

lemma *dist_real*: *dist*(*x*::*real* ^ *1*) *y* = |(*x*$*1*) − (*y*$*1*)|
by (*auto simp add*: *norm_real dist_norm*)

1.9.6 Routine results connecting the types (*real*, *1*) *vec* and *real*

lemma *vector_one_nth* [*simp*]:
fixes *x* :: *'a*^*1* **shows** *vec* (*x* $ *1*) = *x*
by (*metis vec_def vector_one*)

lemma *tendsto_at_within_vector_1*:
fixes *S* :: *'a* :: *metric_space set*
assumes (*f* ⟶ *fx*) (*at x within S*)
shows ((λ*y*::*'a*^*1*. χ *i*. *f* (*y* $ *1*)) ⟶ (*vec fx*::*'a*^*1*)) (*at* (*vec x*) *within vec* ‘ *S*)
proof (*rule topological_tendstoI*)
fix *T* :: (*'a*^*1*) *set*
assume *open T vec fx* ∈ *T*
have ∀ *F x in at x within S*. *f x* ∈ (λ*x*. *x* $ *1*) ‘ *T*
using ‹*open T*› ‹*vec fx* ∈ *T*› *assms open_image_vec_nth tendsto_def* **by** *fastforce*
then show ∀ *F x*::*'a*^*1 in at* (*vec x*) *within vec* ‘ *S*. (χ *i*. *f* (*x* $ *1*)) ∈ *T*
unfolding *eventually_at dist_norm* [*symmetric*]

by (*rule ex_forward*)
(*use* ‹*open T*› **in**
‹*fastforce simp*: *dist_norm dist_vec_def L2_set_def image_iff vector_one open_vec_def*›)
qed

lemma *has_derivative_vector_1*:
assumes *der_g*: (*g has_derivative* (λ*x*. *x* ∗ *g*′ *a*)) (*at a within S*)
shows ((λ*x*. *vec* (*g* (*x* $ *1*))) *has_derivative* (∗$_R$) (*g*′ *a*))
(*at* ((*vec a*)::*real*^*1*) *within vec* ‘ *S*)
using *der_g*
apply (*clarsimp simp*: *Deriv.has_derivative_within bounded_linear_scaleR_right norm_vector_1*)
apply (*drule tendsto_at_within_vector_1*, *vector*)
apply (*auto simp*: *algebra_simps eventually_at tendsto_def*)
done

1.9.7 Explicit vector construction from lists

definition *vector l* = (χ *i*. *foldr* (λ*x f n*. *fun_upd* (*f* (*n*+*1*)) *n x*) *l* (λ*n x*. *0*) *1 i*)

lemma *vector_1* [*simp*]: (*vector*[*x*]) $*1* = *x*
unfolding *vector_def* **by** *simp*

lemma *vector_2* [*simp*]: (*vector*[*x*,*y*]) $*1* = *x* (*vector*[*x*,*y*] :: ′*a*^*2*)$*2* = (*y*::′*a*::*zero*)
unfolding *vector_def* **by** *simp_all*

lemma *vector_3* [*simp*]:
(*vector* [*x*,*y*,*z*] ::(′*a*::*zero*)^*3*)$*1* = *x*
(*vector* [*x*,*y*,*z*] ::(′*a*::*zero*)^*3*)$*2* = *y*
(*vector* [*x*,*y*,*z*] ::(′*a*::*zero*)^*3*)$*3* = *z*
unfolding *vector_def* **by** *simp_all*

lemma *forall_vector_1*: (∀ *v*::′*a*::*zero*^*1*. *P v*) ⟷ (∀ *x*. *P*(*vector*[*x*]))
by (*metis vector_1 vector_one*)

lemma *forall_vector_2*: (∀ *v*::′*a*::*zero*^*2*. *P v*) ⟷ (∀ *x y*. *P*(*vector*[*x*, *y*]))
proof −
have *P v* **if** ⋀*x y*. *P* (*vector* [*x*, *y*]) **for** *v*
proof −
have *vector* [*v*$*1*, *v*$*2*] = *v*
by (*smt* (*verit*, *best*) *exhaust_2 vec_eq_iff vector_2*)
then show *?thesis*
by (*metis that*)
qed
then show *?thesis* **by** *auto*
qed

lemma *forall_vector_3*: (∀ *v*::′*a*::*zero*^*3*. *P v*) ⟷ (∀ *x y z*. *P*(*vector*[*x*, *y*, *z*]))

proof −
have *P v* **if** ⋀*x y z*. *P* (*vector* [*x*, *y*, *z*]) **for** *v*
proof −
have *vector* [*v*$*1*, *v*$*2*, *v*$*3*] = *v*
by (*smt* (*verit*, *best*) *exhaust_3 vec_eq_iff vector_3*)
then show *?thesis*
by (*metis that*)
qed
then show *?thesis* **by** *auto*
qed

1.9.8 lambda skolemization on cartesian products

lemma *lambda_skolem*: (∀ *i*. ∃ *x*. *P i x*) ⟷ (∃ *x*::*'a* ^ *'n*. ∀ *i*. *P i* (*x* $ *i*))
by (*metis vec_lambda_beta*)

The same result in terms of square matrices.

Considering an n-element vector as an n-by-1 or 1-by-n matrix.

definition *rowvector v* = (*χ i j*. (*v*$*j*))

definition *columnvector v* = (*χ i j*. (*v*$*i*))

lemma *transpose_columnvector*: *transpose*(*columnvector v*) = *rowvector v*
by (*simp add*: *transpose_def rowvector_def columnvector_def vec_eq_iff*)

lemma *transpose_rowvector*: *transpose*(*rowvector v*) = *columnvector v*
by (*simp add*: *transpose_def columnvector_def rowvector_def vec_eq_iff*)

lemma *dot_rowvector_columnvector*: *columnvector* (*A* ∗*v v*) = *A* ∗∗ *columnvector v*
by (*vector columnvector_def matrix_matrix_mult_def matrix_vector_mult_def*)

lemma *dot_matrix_product*:
(*x*::*real*^*'n*) · *y* = (((*rowvector x* ::*real*^*'n*^*1*) ∗∗ (*columnvector y* :: *real*^*1*^*'n*))$*1*)$*1*
by (*vector matrix_matrix_mult_def rowvector_def columnvector_def inner_vec_def*)

lemma *dot_matrix_vector_mul*:
fixes *A B* :: *real* ^*'n* ^*'n* **and** *x y* :: *real* ^*'n*
shows (*A* ∗*v x*) · (*B* ∗*v y*) =
(((*rowvector x* :: *real*^*'n*^*1*) ∗∗ ((*transpose A* ∗∗ *B*) ∗∗ (*columnvector y* :: *real* ^*1*^*'n*)))$*1*)$*1*
by (*metis dot_lmul_matrix dot_matrix_product dot_rowvector_columnvector matrix_mul_assoc vector_transpose_matrix*)

lemma *dim_substandard_cart*: *vec.dim* {*x*::*'a*::*field*^*'n*. ∀ *i*. *i* ∉ *d* ⟶ *x*$*i* = *0*} = *card d*
(**is** *vec.dim ?A* = _)
proof (*rule vec.dim_unique*)

```
  let ?B = ((λx. axis x 1) ‘ d)
  have subset_basis: ?B ⊆ cart_basis
    by (auto simp: cart_basis_def)
  show ?B ⊆ ?A
    by (auto simp: axis_def)
  show vec.independent ((λx. axis x 1) ‘ d)
    using subset_basis
    by (rule vec.independent_mono[OF vec.independent_Basis])
  have x ∈ vec.span ?B if ∀ i. i ∉ d ⟶ x $ i = 0 for x::'a^'n
  proof −
    have finite ?B
      using subset_basis finite_cart_basis
      by (rule finite_subset)
    have x = (∑ i∈UNIV. x $ i *s axis i 1)
      by (rule basis_expansion[symmetric])
    also have ... = (∑ i∈d. (x $ i) *s axis i 1)
      by (rule sum.mono_neutral_cong_right) (auto simp: that)
    also have ... ∈ vec.span ?B
      by (simp add: vec.span_sum vec.span_clauses)
    finally show x ∈ vec.span ?B .
  qed
  then show ?A ⊆ vec.span ?B by auto
qed (simp add: card_image inj_on_def axis_eq_axis)

lemma affinity_inverses:
  assumes m0: m ≠ (0::'a::field)
  shows (λx. m *s x + c) ∘ (λx. inverse(m) *s x + (−(inverse(m) *s c))) = id
  (λx. inverse(m) *s x + (−(inverse(m) *s c))) ∘ (λx. m *s x + c) = id
  using m0
  by (auto simp add: fun_eq_iff vector_add_ldistrib diff_conv_add_uminus simp
del: add_uminus_conv_diff)

lemma vector_affinity_eq:
  assumes m0: (m::'a::field) ≠ 0
  shows m *s x + c = y ⟷ x = inverse m *s y + −(inverse m *s c)
proof
  assume h: m *s x + c = y
  hence m *s x = y − c by (simp add: field_simps)
  hence inverse m *s (m *s x) = inverse m *s (y − c) by simp
  then show x = inverse m *s y + − (inverse m *s c)
    by (simp add: m0 vec.scale_right_diff_distrib)
next
  assume h: x = inverse m *s y + − (inverse m *s c)
  show m *s x + c = y unfolding h
    using m0 by (simp add: vector_smult_assoc vector_ssub_ldistrib)
qed

lemma vector_eq_affinity:
    (m::'a::field) ≠ 0 ==> (y = m *s x + c ⟷ inverse(m) *s y + −(inverse(m)
```

$*s\ c) = x)$
by (*metis vector_affinity_eq*)

lemma *vector_cart*:
fixes $f :: real\hat{}'n \Rightarrow real$
shows $(\chi\ i.\ f\ (axis\ i\ 1)) = (\sum i{\in}Basis.\ f\ i *_R i)$
unfolding *euclidean_eq_iff*[**where** $'a{=}real\hat{}'n$]
by *simp* (*simp add*: *Basis_vec_def inner_axis*)

lemma *const_vector_cart*:$((\chi\ i.\ d)::real\hat{}'n) = (\sum i{\in}Basis.\ d *_R i)$
by (*rule vector_cart*)

1.9.9 Explicit formulas for low dimensions

lemma *prod_neutral_const*: *prod f* $\{(1::nat)..1\} = f\ 1$
by *simp*

lemma *prod_2*: *prod f* $\{(1::nat)..2\} = f\ 1 * f\ 2$
by (*simp add*: *eval_nat_numeral atLeastAtMostSuc_conv mult.commute*)

lemma *prod_3*: *prod f* $\{(1::nat)..3\} = f\ 1 * f\ 2 * f\ 3$
by (*simp add*: *eval_nat_numeral atLeastAtMostSuc_conv mult.commute*)

1.9.10 Orthogonality of a matrix

definition *orthogonal_matrix* $(Q::'a::semiring\_1\hat{}'n\hat{}'n) \longleftrightarrow$
$transpose\ Q ** Q = mat\ 1 \wedge Q ** transpose\ Q = mat\ 1$

lemma *orthogonal_matrix*: *orthogonal_matrix* $(Q::real\ \hat{}'n\hat{}'n) \longleftrightarrow transpose\ Q$
$** Q = mat\ 1$
by (*metis matrix_left_right_inverse orthogonal_matrix_def*)

lemma *orthogonal_matrix_id*: *orthogonal_matrix* $(mat\ 1 :: \_\hat{}'n\hat{}'n)$
by (*simp add*: *orthogonal_matrix_def*)

proposition *orthogonal_matrix_mul*:
fixes $A :: real\ \hat{}'n\hat{}'n$
assumes *orthogonal_matrix A orthogonal_matrix B*
shows *orthogonal_matrix*$(A ** B)$
using *assms*
by (*simp add*: *orthogonal_matrix matrix_transpose_mul matrix_left_right_inverse*
matrix_mul_assoc)

proposition *orthogonal_transformation_matrix*:
fixes $f::real\hat{}'n \Rightarrow real\hat{}'n$
shows *orthogonal_transformation f* $\longleftrightarrow$ *linear f* $\wedge$ *orthogonal_matrix*$(matrix\ f)$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof −
let *?mf* = *matrix f*
let *?ot* = *orthogonal_transformation f*

```
  let ?U = UNIV :: 'n set
  have fU: finite ?U by simp
  let ?m1 = mat 1 :: real ^'n^'n
  {
    assume ot: ?ot
    from ot have lf: Vector_Spaces.linear (*s) (*s) f and fd: ⋀v w. f v ∙ f w = v
∙ w
        unfolding orthogonal_transformation_def orthogonal_matrix linear_def
scalar_mult_eq_scaleR
      by blast+
    {
      fix i j
      let ?A = transpose ?mf ** ?mf
     have th0: ⋀b (x::'a::comm_ring_1). (if b then 1 else 0)*x = (if b then x else
0)
        ⋀b (x::'a::comm_ring_1). x*(if b then 1 else 0) = (if b then x else 0)
        by simp_all
      from fd[of axis i 1 axis j 1,
        simplified matrix_works[OF lf, symmetric] dot_matrix_vector_mul]
      have ?A$i$j = ?m1 $ i $ j
        by (simp add: inner_vec_def matrix_matrix_mult_def columnvector_def
rowvector_def
            th0 sum.delta[OF fU] mat_def axis_def)
    }
    then have orthogonal_matrix ?mf
      unfolding orthogonal_matrix
      by vector
    with lf have ?rhs
      unfolding linear_def scalar_mult_eq_scaleR
      by blast
  }
  moreover
  have ?lhs if Vector_Spaces.linear (*s) (*s) f and orthogonal_matrix ?mf
   using that unfolding orthogonal_matrix_def norm_eq orthogonal_transformation
    by (metis dot_matrix_product dot_matrix_vector_mul linear_matrix_vector_mul_eq
matrix_mul_lid matrix_vector_mul(2))
  ultimately show ?thesis
    by (auto simp: linear_def scalar_mult_eq_scaleR)
qed
```

1.9.11 Finding an Orthogonal Matrix

We can find an orthogonal matrix taking any unit vector to any other.

```
lemma orthogonal_matrix_transpose [simp]:
    orthogonal_matrix(transpose A) ⟷ orthogonal_matrix A
  by (auto simp: orthogonal_matrix_def)

lemma orthogonal_matrix_orthonormal_columns:
  fixes A :: real^'n^'n
```

shows *orthogonal_matrix A* $\longleftrightarrow$
$(\forall i.\ norm(column\ i\ A) = 1) \wedge$
$(\forall i\ j.\ i \neq j \longrightarrow orthogonal\ (column\ i\ A)\ (column\ j\ A))$
by (*auto simp*: *orthogonal_matrix matrix_mult_transpose_dot_column vec_eq_iff mat_def norm_eq_1 orthogonal_def*)

lemma *orthogonal_matrix_orthonormal_rows*:
fixes $A :: real^{'n}{}^{'n}$
shows *orthogonal_matrix A* $\longleftrightarrow$
$(\forall i.\ norm(row\ i\ A) = 1) \wedge$
$(\forall i\ j.\ i \neq j \longrightarrow orthogonal\ (row\ i\ A)\ (row\ j\ A))$
using *orthogonal_matrix_orthonormal_columns* [*of transpose A*] **by** *simp*

proposition *orthogonal_matrix_exists_basis*:
fixes $a :: real^{'n}$
assumes *norm a = 1*
obtains *A* **where** *orthogonal_matrix A A* $*v$ (*axis k 1*) = *a*
proof −
obtain *S* **where** $a \in S$ *pairwise orthogonal S* **and** *noS*: $\bigwedge x.\ x \in S \Longrightarrow norm\ x = 1$
and *independent S card S = CARD('n) span S = UNIV*
using *vector_in_orthonormal_basis assms* **by** *force*
then obtain *f0* **where** *bij_betw f0* (*UNIV*::*'n set*) *S*
by (*metis finite_class.finite_UNIV finite_same_card_bij finiteI_independent*)
then obtain *f* **where** *f*: *bij_betw f* (*UNIV*::*'n set*) *S* **and** *a*: *a = f k*
using *bij_swap_iff* [*of f0 k inv f0 a*]
by (*metis UNIV_I* ‹*a* ∈ *S*› *bij_betw_inv_into_right bij_betw_swap_iff swap_apply*(*1*))
show *thesis*
proof
have [*simp*]: $\bigwedge i.\ norm\ (f\ i) = 1$
using *bij_betwE* [*OF* ‹*bij_betw f UNIV S*›] **by** (*blast intro*: *noS*)
have [*simp*]: $\bigwedge i\ j.\ i \neq j \Longrightarrow orthogonal\ (f\ i)\ (f\ j)$
using ‹*pairwise orthogonal S*› ‹*bij_betw f UNIV S*›
by (*auto simp*: *pairwise_def bij_betw_def inj_on_def*)
show *orthogonal_matrix* (χ *i j. f j* $ *i*)
by (*simp add*: *orthogonal_matrix_orthonormal_columns column_def*)
show (χ *i j. f j* $ *i*) $*v$ *axis k 1 = a*
by (*simp add*: *matrix_vector_mult_def axis_def a if_distrib cong*: *if_cong*)
qed
qed

lemma *orthogonal_transformation_exists_1*:
fixes $a\ b :: real^{'n}$
assumes *norm a = 1 norm b = 1*
obtains *f* **where** *orthogonal_transformation f f a = b*
proof −
obtain *k A B* **where** *AB*: *orthogonal_matrix A orthogonal_matrix B* **and** *eq*: *A* $*v$ (*axis k 1*) = *a B* $*v$ (*axis k 1*) = *b*
using *orthogonal_matrix_exists_basis assms* **by** *metis*

```
  let ?f = λx. (B ** transpose A) *v x
  show thesis
  proof
    show orthogonal_transformation ?f
    by (simp add: AB orthogonal_matrix_mul orthogonal_transformation_matrix)
  next
    show ?f a = b
      using ‹orthogonal_matrix A› unfolding orthogonal_matrix_def
      by (metis eq matrix_mul_rid matrix_vector_mul_assoc)
  qed
qed

proposition orthogonal_transformation_exists:
  fixes a b :: real^'n
  assumes norm a = norm b
  obtains f where orthogonal_transformation f f a = b
proof (cases a = 0 ∨ b = 0)
  case True
  with assms show ?thesis
    using that by force
next
  case False
  then obtain f where f: orthogonal_transformation f and eq: f (a /R norm a)
= (b /R norm b)
   by (auto intro: orthogonal_transformation_exists_1 [of a /R norm a b /R norm
b])
  show ?thesis
    using False assms eq f orthogonal_transformation_scaleR that by fastforce
qed
```

1.9.12 Scaling and isometry

```
proposition scaling_linear:
  fixes f :: 'a::real_inner ⇒ 'a::real_inner
  assumes f0: f 0 = 0
    and fd: ∀x y. dist (f x) (f y) = c * dist x y
  shows linear f
proof −
  {
    fix v w
    have norm (f x) = c * norm x for x
      by (metis dist_0_norm f0 fd)
    then have f v • f w = c^2 * (v • w)
      unfolding dot_norm_neg dist_norm[symmetric]
      by (simp add: fd power2_eq_square field_simps)
  }
  then show ?thesis
    unfolding linear_iff vector_eq[where 'a='a] scalar_mult_eq_scaleR
    by (simp add: inner_add field_simps)
```

qed

lemma *isometry_linear*:
f (*0*::*'a*::*real_inner*) = (*0*::*'a*) ⟹ ∀ *x y*. *dist*(*f x*) (*f y*) = *dist x y* ⟹ *linear f*
by (*rule scaling_linear*[**where** *c=1*]) *simp_all*

Hence another formulation of orthogonal transformation

proposition *orthogonal_transformation_isometry*:
orthogonal_transformation f ⟷ *f*(*0*::*'a*::*real_inner*) = (*0*::*'a*) ∧ (∀ *x y*. *dist*(*f x*) (*f y*) = *dist x y*)
unfolding *orthogonal_transformation*
by (*metis dist_0_norm dist_norm isometry_linear linear_0 linear_diff*)

Can extend an isometry from unit sphere:

lemma *isometry_sphere_extend*:
fixes *f*:: *'a*::*real_inner* ⇒ *'a*
assumes *f1*: ⋀*x*. *norm x* = *1* ⟹ *norm* (*f x*) = *1*
and *fd1*: ⋀*x y*. ⟦*norm x* = *1*; *norm y* = *1*⟧ ⟹ *dist* (*f x*) (*f y*) = *dist x y*
shows ∃ *g*. *orthogonal_transformation g* ∧ (∀ *x*. *norm x* = *1* ⟶ *g x* = *f x*)
proof −
{
fix *x y x' y' u v u' v'* :: *'a*
assume *H*: *x* = *norm x* ∗*R u y* = *norm y* ∗*R v*
x' = *norm x* ∗*R u' y'* = *norm y* ∗*R v'*
and *J*: *norm u* = *1 norm u'* = *1 norm v* = *1 norm v'* = *1 norm*(*u'* − *v'*) = *norm*(*u* − *v*)
then have ∗: *u* · *v* = *u'* · *v'* + *v'* · *u'* − *v* · *u*
by (*simp add*: *norm_eq norm_eq_1 inner_add inner_diff*)
have *norm* (*norm x* ∗*R u'* − *norm y* ∗*R v'*) = *norm* (*norm x* ∗*R u* − *norm y* ∗*R v*)
using *J* **by** (*simp add*: *norm_eq norm_eq_1 inner_diff* ∗ *field_simps*)
then have *norm*(*x'* − *y'*) = *norm*(*x* − *y*)
using *H* **by** *metis*
}
note *norm_eq* = *this*
let *?g* = λ*x*. *if x* = *0 then 0 else norm x* ∗*R f* (*x* /*R norm x*)
have *thfg*: *?g x* = *f x* **if** *norm x* = *1* **for** *x*
using *that* **by** *auto*
have *thd*: *dist* (*?g x*) (*?g y*) = *dist x y* **for** *x y*
proof (*cases x=0* ∨ *y=0*)
case *False*
show *dist* (*?g x*) (*?g y*) = *dist x y*
unfolding *dist_norm*
proof (*rule norm_eq*)
show *x* = *norm x* ∗*R* (*x* /*R norm x*) *y* = *norm y* ∗*R* (*y* /*R norm y*)
norm (*f* (*x* /*R norm x*)) = *1 norm* (*f* (*y* /*R norm y*)) = *1*
using *False f1* **by** *auto*
qed (*use False* **in** ‹*auto simp*: *field_simps intro*: *f1 fd1*[*unfolded dist_norm*]›)
qed (*auto simp*: *f1*)

```
  show ?thesis
    unfolding orthogonal_transformation_isometry
    by (rule exI[where x= ?g]) (metis thfg thd)
qed
```

1.9.13 Induction on matrix row operations

```
lemma induct_matrix_row_operations:
  fixes P :: real^'n^'n ⇒ bool
  assumes zero_row: ⋀A i. row i A = 0 ⟹ P A
    and diagonal: ⋀A. (⋀i j. i ≠ j ⟹ A$i$j = 0) ⟹ P A
     and swap_cols: ⋀A m n. ⟦P A; m ≠ n⟧ ⟹ P(χ i j. A $ i $ Transposition.transpose m n j)
    and row_op: ⋀A m n c. ⟦P A; m ≠ n⟧
                ⟹ P(χ i. if i = m then row m A + c *R row n A else row i A)
  shows P A
proof −
  have P A if (⋀i j. ⟦j ∈ −K;  i ≠ j⟧ ⟹ A$i$j = 0) for A K
  proof −
    have finite K
      by simp
    then show ?thesis using that
    proof (induction arbitrary: A rule: finite_induct)
      case empty
      with diagonal show ?case
        by simp
    next
      case (insert k K)
      note insertK = insert
      have P A if kk: A$k$k ≠ 0
        and 0: ⋀i j. ⟦j ∈ − insert k K; i ≠ j⟧ ⟹ A$i$j = 0
               ⋀i. ⟦i ∈ −L; i ≠ k⟧ ⟹ A$i$k = 0 for A L
      proof −
        have finite L
          by simp
        then show ?thesis using 0 kk
        proof (induction arbitrary: A rule: finite_induct)
          case (empty B)
          show ?case
          proof (rule insertK)
            fix i j
            assume i ∈ − K j ≠ i
            show B $ j $ i = 0
              using ‹j ≠ i› ‹i ∈ − K› empty
             by (metis ComplD ComplI Compl_eq_Diff_UNIV Diff_empty UNIV_I
insert_iff)
          qed
        next
          case (insert l L B)
```

```
        show ?case
        proof (cases k = l)
          case True
          with insert show ?thesis
            by auto
        next
          case False
          let ?C = χ i. if i = l then row l B − (B $ l $ k / B $ k $ k) *R row k
B else row i B
          have 1: ⟦j ∈ − insert k K; i ≠ j⟧ ⟹ ?C $ i $ j = 0 for j i
            by (auto simp: insert.prems(1) row_def)
          have 2: ?C $ i $ k = 0
            if i ∈ − L i ≠ k for i
          proof (cases i=l)
            case True
            with that insert.prems show ?thesis
              by (simp add: row_def)
          next
            case False
            with that show ?thesis
              by (simp add: insert.prems(2) row_def)
          qed
          have 3: ?C $ k $ k ≠ 0
            by (auto simp: insert.prems row_def ‹k ≠ l›)
          have PC: P ?C
            using insert.IH [OF 1 2 3] by auto
          have eqB: (χ i. if i = l then row l ?C + (B $ l $ k / B $ k $ k) *R row
k ?C else row i ?C) = B
            using ‹k ≠ l› by (simp add: vec_eq_iff row_def)
          show ?thesis
           using row_op [OF PC, of l k, where c = B$l$k / B$k$k] eqB ‹k ≠ l›
            by (simp add: cong: if_cong)
         qed
       qed
     qed
     then have nonzero_hyp: P A
       if kk: A$k$k ≠ 0 and zeroes: ⋀i j. j ∈ − insert k K ∧ i≠j ⟹ A$i$j = 0
for A
       by (auto simp: intro!: kk zeroes)
     show ?case
     proof (cases row k A = 0)
       case True
       with zero_row show ?thesis by auto
     next
       case False
       then obtain l where l: A$k$l ≠ 0
         by (auto simp: row_def zero_vec_def vec_eq_iff)
       show ?thesis
       proof (cases k = l)
```

```
        case True
        with l nonzero_hyp insert.prems show ?thesis
          by blast
      next
        case False
        have *: A $ i $ Transposition.transpose k l j = 0 if j ≠ k j ∉ K i ≠ j for
i j
          using False l insert.prems that
          by (auto simp add: Transposition.transpose_def)
            have P (χ i j. (χ i j. A $ i $ Transposition.transpose k l j) $ i $
Transposition.transpose k l j)
          by (rule swap_cols [OF nonzero_hyp False]) (auto simp: l *)
        moreover
        have (χ i j. (χ i j. A $ i $ Transposition.transpose k l j) $ i $ Transpos-
ition.transpose k l j) = A
          by simp
        ultimately show ?thesis
          by simp
      qed
    qed
  qed
 qed
 then show ?thesis
   by blast
qed

lemma induct_matrix_elementary:
 fixes P :: real^'n^'n ⇒ bool
 assumes mult: ⋀A B. ⟦P A; P B⟧ ⟹ P(A ** B)
   and zero_row: ⋀A i. row i A = 0 ⟹ P A
   and diagonal: ⋀A. (⋀i j. i ≠ j ⟹ A$i$j = 0) ⟹ P A
   and swap1: ⋀m n. m ≠ n ⟹ P(χ i j. mat 1 $ i $ Transposition.transpose m
n j)
   and idplus: ⋀m n c. m ≠ n ⟹ P(χ i j. if i = m ∧ j = n then c else of_bool
(i = j))
 shows P A
proof −
 have swap: P (χ i j. A $ i $ Transposition.transpose m n j) (is P ?C)
   if P A m ≠ n for A m n
 proof −
   have A ** (χ i j. mat 1 $ i $ Transposition.transpose m n j) = ?C
   by (simp add: matrix_matrix_mult_def mat_def vec_eq_iff if_distrib sum.delta_remove)
   then show ?thesis
     using mult swap1 that by metis
 qed
 have row: P (χ i. if i = m then row m A + c *_R row n A else row i A) (is P
?C)
   if P A m ≠ n for A m n c
 proof −
```

```
    let ?B = χ i j. if i = m ∧ j = n then c else of_bool (i = j)
    have ?B ** A = ?C
      using ‹m ≠ n› unfolding matrix_matrix_mult_def row_def of_bool_def
      by (auto simp: vec_eq_iff if_distrib [of λx. x * y for y] sum.remove cong:
if_cong)
    then show ?thesis
      by (rule subst) (auto simp: that mult idplus)
  qed
  show ?thesis
    by (rule induct_matrix_row_operations [OF zero_row diagonal swap row])
qed

lemma induct_matrix_elementary_alt:
  fixes P :: real^'n^'n ⇒ bool
  assumes mult: ⋀A B. ⟦P A; P B⟧ ⟹ P(A ** B)
    and zero_row: ⋀A i. row i A = 0 ⟹ P A
    and diagonal: ⋀A. (⋀i j. i ≠ j ⟹ A$i$j = 0) ⟹ P A
    and swap1: ⋀m n. m ≠ n ⟹ P(χ i j. mat 1 $ i $ Transposition.transpose m
n j)
    and idplus: ⋀m n. m ≠ n ⟹ P(χ i j. of_bool (i = m ∧ j = n ∨ i = j))
  shows P A
proof −
  have ∗: P (χ i j. if i = m ∧ j = n then c else of_bool (i = j))
    if m ≠ n for m n c
  proof (cases c = 0)
    case True
    with diagonal show ?thesis by auto
  next
    case False
    then have eq: (χ i j. if i = m ∧ j = n then c else of_bool (i = j)) =
                  (χ i j. if i = j then (if j = n then inverse c else 1) else 0) **
                  (χ i j. of_bool (i = m ∧ j = n ∨ i = j)) **
                  (χ i j. if i = j then if j = n then c else 1 else 0)
      using ‹m ≠ n›
      apply (simp add: matrix_matrix_mult_def vec_eq_iff of_bool_def if_distrib
[of λx. y * x for y] cong: if_cong)
      apply (simp add: if_if_eq_conj sum.neutral conj_commute cong: conj_cong)
      done
    show ?thesis
      unfolding eq by (intro mult idplus that) (auto intro: diagonal)
  qed
  show ?thesis
    by (rule induct_matrix_elementary) (auto intro: assms ∗)
qed

lemma matrix_vector_mult_matrix_matrix_mult_compose:
  (∗v) (A ** B) = (∗v) A ∘ (∗v) B
  by (auto simp: matrix_vector_mul_assoc)
```

lemma *induct_linear_elementary*:
fixes $f :: real^{\prime}n \Rightarrow real^{\prime}n$
assumes *linear f*
and *comp*: $\bigwedge f\ g.\ [\![linear\ f;\ linear\ g;\ P\ f;\ P\ g]\!] \Longrightarrow P(f \circ g)$
and *zeroes*: $\bigwedge f\ i.\ [\![linear\ f;\ \bigwedge x.\ (f\ x)\ \$\ i = 0]\!] \Longrightarrow P\ f$
and *const*: $\bigwedge c.\ P(\lambda x.\ \chi\ i.\ c\ i * x\$i)$
and *swap*: $\bigwedge m\ n::{}'n.\ m \neq n \Longrightarrow P(\lambda x.\ \chi\ i.\ x\ \$\ Transposition.transpose\ m\ n\ i)$
and *idplus*: $\bigwedge m\ n::{}'n.\ m \neq n \Longrightarrow P(\lambda x.\ \chi\ i.\ if\ i = m\ then\ x\$m + x\$n\ else\ x\$i)$
shows *P f*
proof −
have $P\ ((*v)\ A)$ **for** *A*
proof (*rule induct_matrix_elementary_alt*)
fix *A B*
assume $P\ ((*v)\ A)$ **and** $P\ ((*v)\ B)$
then show $P\ ((*v)\ (A ** B))$
by (*auto simp add*: *matrix_vector_mult_matrix_matrix_mult_compose intro*!: *comp*)
next
fix $A :: real^{\prime}n^{\prime}n$ **and** *i*
assume *row i A = 0*
show $P\ ((*v)\ A)$
using *matrix_vector_mul_linear*
by (*rule zeroes*[**where** *i=i*])
(*metis ‹row i A = 0› inner_zero_left matrix_vector_mul_component row_def vec_lambda_eta*)
next
fix $A :: real^{\prime}n^{\prime}n$
assume *0*: $\bigwedge i\ j.\ i \neq j \Longrightarrow A\ \$\ i\ \$\ j = 0$
have $A\ \$\ i\ \$\ i * x\ \$\ i = (\sum j{\in}UNIV.\ A\ \$\ i\ \$\ j * x\ \$\ j)$ **for** *x* **and** $i :: {}'n$
by (*simp add*: *0 comm_monoid_add_class.sum.remove* [**where** *x=i*])
then have $(\lambda x.\ \chi\ i.\ A\ \$\ i\ \$\ i * x\ \$\ i) = ((*v)\ A)$
by (*auto simp*: *0 matrix_vector_mult_def*)
then show $P\ ((*v)\ A)$
using *const* [*of* $\lambda i.\ A\ \$\ i\ \$\ i$] **by** *simp*
next
fix *m n* :: $'n$
assume $m \neq n$
have *eq*: $(\sum j{\in}UNIV.\ if\ i = Transposition.transpose\ m\ n\ j\ then\ x\ \$\ j\ else\ 0) =$
$(\sum j{\in}UNIV.\ if\ j = Transposition.transpose\ m\ n\ i\ then\ x\ \$\ j\ else\ 0)$
for *i* **and** *x* :: $real^{\prime}n$
by (*rule sum.cong*) (*auto simp add*: *swap_id_eq*)
have $(\lambda x::real^{\prime}n.\ \chi\ i.\ x\ \$\ Transposition.transpose\ m\ n\ i) = ((*v)\ (\chi\ i\ j.\ if\ i = Transposition.transpose\ m\ n\ j\ then\ 1\ else\ 0))$
by (*auto simp*: *mat_def matrix_vector_mult_def eq if_distrib* [*of* $\lambda x.\ x * y$ **for** *y*] *cong*: *if_cong*)
with *swap* [*OF* ‹*m* ≠ *n*›] **show** $P\ ((*v)\ (\chi\ i\ j.\ mat\ 1\ \$\ i\ \$\ Transposition.transpose\ m\ n\ j))$
by (*simp add*: *mat_def matrix_vector_mult_def*)

next
fix $m\ n :: 'n$
assume $m \neq n$
then have $x\ \$\ m + x\ \$\ n = (\sum j{\in}UNIV.\ of\_bool\ (j = n \vee m = j) * x\ \$\ j)$ **for** $x :: real^{\frown}'n$
by (*auto simp*: *of_bool_def if_distrib* [*of* $\lambda x.\ x * y$ **for** y] *sum.remove cong*: *if_cong*)
then have $(\lambda x{::}real^{\frown}'n.\ \chi\ i.\ if\ i = m\ then\ x\ \$\ m + x\ \$\ n\ else\ x\ \$\ i) =$
$((*v)\ (\chi\ i\ j.\ of\_bool\ (i = m \wedge j = n \vee i = j)))$
unfolding *matrix_vector_mult_def of_bool_def*
by (*auto simp*: *vec_eq_iff if_distrib* [*of* $\lambda x.\ x * y$ **for** y] *cong*: *if_cong*)
then show $P\ ((*v)\ (\chi\ i\ j.\ of\_bool\ (i = m \wedge j = n \vee i = j)))$
using *idplus* [*OF* ‹$m \neq n$›] **by** *simp*
qed
then show *?thesis*
by (*metis* ‹*linear f*› *matrix_vector_mul*(*2*))
qed

end

1.10 Traces and Determinants of Square Matrices

theory *Determinants*
imports
HOL−Combinatorics.Permutations
Cartesian_Space
begin

1.10.1 Trace

definition $trace :: 'a{::}semiring\_1^{\frown}'n^{\frown}'n \Rightarrow 'a$
where $trace\ A = sum\ (\lambda i.\ ((A\$i)\$i))\ (UNIV{::}'n\ set)$

lemma *trace_0*: $trace\ (mat\ 0) = 0$
by (*simp add*: *trace_def mat_def*)

lemma *trace_I*: $trace\ (mat\ 1 :: 'a{::}semiring\_1^{\frown}'n^{\frown}'n) = of\_nat(CARD('n))$
by (*simp add*: *trace_def mat_def*)

lemma *trace_add*: $trace\ ((A{::}'a{::}comm\_semiring\_1^{\frown}'n^{\frown}'n) + B) = trace\ A + trace\ B$
by (*simp add*: *trace_def sum.distrib*)

lemma *trace_sub*: $trace\ ((A{::}'a{::}comm\_ring\_1^{\frown}'n^{\frown}'n) - B) = trace\ A - trace\ B$
by (*simp add*: *trace_def sum_subtractf*)

lemma *trace_mul_sym*: $trace\ ((A{::}'a{::}comm\_semiring\_1^{\frown}'n^{\frown}'m) ** B) = trace\ (B{**}A)$
apply (*simp add*: *trace_def matrix_matrix_mult_def*)

```
  apply (subst sum.swap)
  apply (simp add: mult.commute)
  done
```

Definition of determinant

```
definition det:: 'a::comm_ring_1^'n^'n ⇒ 'a where
  det A =
    sum (λp. of_int (sign p) * prod (λi. A$i$p i) (UNIV :: 'n set))
      {p. p permutes (UNIV :: 'n set)}
```

Basic determinant properties

```
lemma det_transpose [simp]: det (transpose A) = det (A::'a::comm_ring_1 ^'n^'n)
proof −
  let ?di = λA i j. A$i$j
  let ?U = (UNIV :: 'n set)
  have fU: finite ?U by simp
  {
    fix p
    assume p: p ∈ {p. p permutes ?U}
    from p have pU: p permutes ?U
      by blast
    have sth: sign (inv p) = sign p
      by (metis sign_inverse fU p mem_Collect_eq permutation_permutes)
    from permutes_inj[OF pU]
    have pi: inj_on p ?U
      by (blast intro: subset_inj_on)
    from permutes_image[OF pU]
    have prod (λi. ?di (transpose A) i (inv p i)) ?U =
      prod (λi. ?di (transpose A) i (inv p i)) (p ' ?U)
      by simp
    also have ... = prod ((λi. ?di (transpose A) i (inv p i)) ∘ p) ?U
      unfolding prod.reindex[OF pi] ..
    also have ... = prod (λi. ?di A i (p i)) ?U
    proof −
      have ((λi. ?di (transpose A) i (inv p i)) ∘ p) i = ?di A i (p i) if i ∈ ?U for i
        using that permutes_inv_o[OF pU] permutes_in_image[OF pU]
        unfolding transpose_def by (simp add: fun_eq_iff)
      then show prod ((λi. ?di (transpose A) i (inv p i)) ∘ p) ?U = prod (λi. ?di
A i (p i)) ?U
        by (auto intro: prod.cong)
    qed
    finally have of_int (sign (inv p)) * (prod (λi. ?di (transpose A) i (inv p i))
?U) =
      of_int (sign p) * (prod (λi. ?di A i (p i)) ?U)
      using sth by simp
  }
  then show ?thesis
    unfolding det_def
```

```
    by (subst sum_permutations_inverse) (blast intro: sum.cong)
qed

lemma  det_lowerdiagonal:
  fixes A :: 'a::comm_ring_1^('n::{finite,wellorder})^('n::{finite,wellorder})
  assumes ld: ⋀i j. i < j ⟹ A$i$j = 0
  shows det A = prod (λi. A$i$i) (UNIV:: 'n set)
proof −
  let ?U = UNIV:: 'n set
  let ?PU = {p. p permutes ?U}
  let ?pp = λp. of_int (sign p) * prod (λi. A$i$p i) (UNIV :: 'n set)
  have fU: finite ?U
    by simp
  have id0: {id} ⊆ ?PU
    by (auto simp: permutes_id)
  have p0: ∀ p ∈ ?PU − {id}. ?pp p = 0
  proof
    fix p
    assume p ∈ ?PU − {id}
    then obtain i where i: p i > i
      by clarify (meson leI permutes_natset_le)
    from ld[OF i] have ∃ i ∈ ?U. A$i$p i = 0
      by blast
    with prod_zero[OF fU] show ?pp p = 0
      by force
  qed
 from sum.mono_neutral_cong_left[OF finite_permutations[OF fU] id0 p0] show
?thesis
    unfolding det_def by (simp add: sign_id)
qed

lemma  det_upperdiagonal:
  fixes A :: 'a::comm_ring_1^'n::{finite,wellorder}^'n::{finite,wellorder}
  assumes ld: ⋀i j. i > j ⟹ A$i$j = 0
  shows det A = prod (λi. A$i$i) (UNIV:: 'n set)
proof −
  let ?U = UNIV:: 'n set
  let ?PU = {p. p permutes ?U}
  let ?pp = (λp. of_int (sign p) * prod (λi. A$i$p i) (UNIV :: 'n set))
  have fU: finite ?U
    by simp
  have id0: {id} ⊆ ?PU
    by (auto simp: permutes_id)
  have p0: ∀ p ∈ ?PU −{id}. ?pp p = 0
  proof
    fix p
    assume p: p ∈ ?PU − {id}
    then obtain i where i: p i < i
      by clarify (meson leI permutes_natset_ge)
```

```
    from ld[OF i] have ∃ i ∈ ?U. A$i$p i = 0
      by blast
    with prod_zero[OF fU]  show ?pp p = 0
      by force
  qed
 from sum.mono_neutral_cong_left[OF finite_permutations[OF fU] id0 p0] show
?thesis
    unfolding det_def by (simp add: sign_id)
qed

proposition  det_diagonal:
  fixes A :: 'a::comm_ring_1^'n^'n
  assumes ld: ⋀i j. i ≠ j ⟹ A$i$j = 0
  shows det A = prod (λi. A$i$i) (UNIV::'n set)
proof −
  let ?U = UNIV:: 'n set
  let ?PU = {p. p permutes ?U}
  let ?pp = λp. of_int (sign p) * prod (λi. A$i$p i) (UNIV :: 'n set)
  have fU: finite ?U by simp
  from finite_permutations[OF fU] have fPU: finite ?PU .
  have id0: {id} ⊆ ?PU
    by (auto simp: permutes_id)
  have p0: ∀ p ∈ ?PU − {id}. ?pp p = 0
  proof
    fix p
    assume p: p ∈ ?PU − {id}
    then obtain i where i: p i ≠ i
      by fastforce
    with ld have ∃ i ∈ ?U. A$i$p i = 0
      by (metis UNIV_I)
    with prod_zero [OF fU] show ?pp p = 0
      by force
  qed
  from sum.mono_neutral_cong_left[OF fPU id0 p0] show ?thesis
    unfolding det_def by (simp add: sign_id)
qed

lemma  det_I [simp]: det (mat 1 :: 'a::comm_ring_1^'n^'n) = 1
  by (simp add: det_diagonal mat_def)

lemma  det_0 [simp]: det (mat 0 :: 'a::comm_ring_1^'n^'n) = 0
  by (simp add: det_def prod_zero power_0_left)

lemma  det_permute_rows:
  fixes A :: 'a::comm_ring_1^'n^'n
  assumes p: p permutes (UNIV :: 'n::finite set)
  shows det (χ i. A$p i :: 'a^'n^'n) = of_int (sign p) * det A
proof −
  let ?U = UNIV :: 'n set
```

```
  let ?PU = {p. p permutes ?U}
  have *: (∑ q∈?PU. of_int (sign (q ∘ p)) * (∏ i∈?U. A $ p i $ (q ∘ p) i)) =
        (∑ n∈?PU. of_int (sign p) * of_int (sign n) * (∏ i∈?U. A $ i $ n i))
  proof (rule sum.cong)
    fix q
    assume qPU: q ∈ ?PU
    have fU: finite ?U
      by simp
    from qPU have q: q permutes ?U
      by blast
    have prod (λi. A$p i$ (q ∘ p) i) ?U = prod ((λi. A$p i$(q ∘ p) i) ∘ inv p) ?U
      by (simp only: prod.permute[OF permutes_inv[OF p], symmetric])
    also have ... = prod (λi. A $ (p ∘ inv p) i $ (q ∘ (p ∘ inv p)) i) ?U
      by (simp only: o_def)
    also have ... = prod (λi. A$i$q i) ?U
      by (simp only: o_def permutes_inverses[OF p])
    finally have thp: prod (λi. A$p i$ (q ∘ p) i) ?U = prod (λi. A$i$q i) ?U
      by blast
    from p q have pp: permutation p and qp: permutation q
      by (metis fU permutation_permutes)+
    show of_int (sign (q ∘ p)) * prod (λi. A$ p i$ (q ∘ p) i) ?U =
        of_int (sign p) * of_int (sign q) * prod (λi. A$i$q i) ?U
      by (simp only: thp sign_compose[OF qp pp] mult.commute of_int_mult)
  qed auto
  show ?thesis
    apply (simp add: det_def sum_distrib_left mult.assoc[symmetric])
    apply (subst sum_permutations_compose_right[OF p])
    apply (rule *)
    done
qed

lemma det_permute_columns:
  fixes A :: 'a::comm_ring_1^'n^'n
  assumes p: p permutes (UNIV :: 'n set)
  shows det(χ i j. A$i$ p j :: 'a^'n^'n) = of_int (sign p) * det A
proof −
  let ?Ap = χ i j. A$i$ p j :: 'a^'n^'n
  let ?At = transpose A
  have of_int (sign p) * det A = det (transpose (χ i. transpose A $ p i))
    unfolding det_permute_rows[OF p, of ?At] det_transpose ..
  moreover
  have ?Ap = transpose (χ i. transpose A $ p i)
    by (simp add: transpose_def vec_eq_iff)
  ultimately show ?thesis
    by simp
qed

lemma det_identical_columns:
  fixes A :: 'a::comm_ring_1^'n^'n
```

```
  assumes jk: j ≠ k
    and r: column j A = column k A
  shows det A = 0
proof −
  let ?U=UNIV::'n set
  let ?t_jk=Transposition.transpose j k
  let ?PU={p. p permutes ?U}
  let ?S1={p. p∈?PU ∧ evenperm p}
  let ?S2={(?t_jk ∘ p) |p. p ∈?S1}
  let ?f=λp. of_int (sign p) * (∏ i∈UNIV. A $ i $ p i)
  let ?g=λp. ?t_jk ∘ p
  have g_S1: ?S2 = ?g‘ ?S1 by auto
  have inj_g: inj_on ?g ?S1
  proof (unfold inj_on_def, auto)
    fix x y assume x: x permutes ?U and even_x: evenperm x
      and y: y permutes ?U and even_y: evenperm y and eq: ?t_jk ∘ x = ?t_jk
∘ y
    show x = y by (metis (opaque_lifting, no_types) comp_assoc eq id_comp
swap_id_idempotent)
  qed
  have tjk_permutes: ?t_jk permutes ?U
    by (auto simp add: permutes_def dest: transpose_eq_imp_eq) (meson trans-
pose_involutory)
  have tjk_eq: ∀ i l. A $ i $ ?t_jk l  =  A $ i $ l
    using r jk
   unfolding column_def vec_eq_iff by (simp add: Transposition.transpose_def)

  have sign_tjk: sign ?t_jk = −1 using sign_swap_id[of j k] jk by auto
  {fix x
    assume x: x∈ ?S1
    have sign (?t_jk ∘ x) = sign (?t_jk) * sign x
      by (metis (lifting) finite_class.finite_UNIV mem_Collect_eq
          permutation_permutes permutation_swap_id sign_compose x)
    also have ... = − sign x using sign_tjk by simp
    also have ... ≠ sign x unfolding sign_def by simp
    finally have sign (?t_jk ∘ x) ≠ sign x and (?t_jk ∘ x) ∈ ?S2
      using x by force+
  }
  hence disjoint: ?S1 ∩ ?S2 = {}
    by (force simp: sign_def)
  have PU_decomposition: ?PU = ?S1 ∪ ?S2
  proof (auto)
    fix x
      assume x: x permutes ?U and ∀ p. p permutes ?U ⟶ x = Transposi-
tion.transpose j k ∘ p ⟶ ¬ evenperm p
   then obtain p where p: p permutes UNIV and x_eq: x = Transposition.transpose
j k ∘ p
      and odd_p: ¬ evenperm p
     by (metis (mono_tags) id_o o_assoc permutes_compose swap_id_idempotent
```

```
tjk_permutes)
    thus evenperm x
      by (meson evenperm_comp evenperm_swap finite_class.finite_UNIV
          jk permutation_permutes permutation_swap_id)
  next
    fix p assume p: p permutes ?U
   show Transposition.transpose j k ∘ p permutes UNIV by (metis p permutes_compose
tjk_permutes)
  qed
  have sum ?f ?S2 = sum ((λp. of_int (sign p) * (∏ i∈UNIV. A $ i $ p i))
  ∘ (∘) (Transposition.transpose j k)) {p ∈ {p. p permutes UNIV}. evenperm p}
    unfolding g_S1 by (rule sum.reindex[OF inj_g])
  also have ... = sum (λp. of_int (sign (?t_jk ∘ p)) * (∏ i∈UNIV. A $ i $ p i))
?S1
    unfolding o_def by (rule sum.cong, auto simp: tjk_eq)
  also have ... = sum (λp. − ?f p) ?S1
  proof (rule sum.cong, auto)
    fix x assume x: x permutes ?U
      and even_x: evenperm x
    hence perm_x: permutation x and perm_tjk: permutation ?t_jk
      using permutation_permutes[of x] permutation_permutes[of ?t_jk] permuta-
tion_swap_id
      by (metis finite_code)+
    have (sign (?t_jk ∘ x)) = − (sign x)
      unfolding sign_compose[OF perm_tjk perm_x] sign_tjk by auto
    thus of_int (sign (?t_jk ∘ x)) * (∏ i∈UNIV. A $ i $ x i)
      = − (of_int (sign x) * (∏ i∈UNIV. A $ i $ x i))
      by auto
  qed
  also have ... = − sum ?f ?S1 unfolding sum_negf ..
  finally have *: sum ?f ?S2 = − sum ?f ?S1 .
  have det A = (∑ p | p permutes UNIV. of_int (sign p) * (∏ i∈UNIV. A $ i $
p i))
    unfolding det_def ..
  also have ... = sum ?f ?S1 + sum ?f ?S2
    by (subst PU_decomposition, rule sum.union_disjoint[OF _ _ disjoint], auto)
  also have ... = sum ?f ?S1 − sum ?f ?S1 unfolding * by auto
  also have ... = 0 by simp
  finally show det A = 0 by simp
qed

lemma det_identical_rows:
  fixes A :: 'a::comm_ring_1^'n^'n
  assumes ij: i ≠ j and r: row i A = row j A
  shows det A = 0
  by (metis column_transpose det_identical_columns det_transpose ij r)

lemma det_zero_row:
  fixes A :: 'a::{idom, ring_char_0}^'n^'n and F :: 'b::{field}^'m^'m
```

shows *row i A = 0 ⟹ det A = 0* **and** *row j F = 0 ⟹ det F = 0*
by (*force simp*: *row_def det_def vec_eq_iff sign_nz intro*!: *sum.neutral*)+

lemma *det_zero_column*:
fixes *A* :: *'a*::{*idom*, *ring_char_0*}^'*n*^'*n* **and** *F* :: *'b*::{*field*}^'*m*^'*m*
shows *column i A = 0 ⟹ det A = 0* **and** *column j F = 0 ⟹ det F = 0*
unfolding *atomize_conj atomize_imp*
by (*metis det_transpose det_zero_row row_transpose*)

lemma *det_row_add*:
fixes *a b c* :: *'n*::*finite* ⇒ _ ^ *'n*
shows *det*((χ *i*. *if i = k then a i + b i else c i*)::*'a*::*comm_ring_1*^'*n*^'*n*) =
det((χ *i*. *if i = k then a i else c i*)::*'a*::*comm_ring_1*^'*n*^'*n*) +
det((χ *i*. *if i = k then b i else c i*)::*'a*::*comm_ring_1*^'*n*^'*n*)
unfolding *det_def vec_lambda_beta sum.distrib*[*symmetric*]
proof (*rule sum.cong*)
let *?U = UNIV* :: *'n set*
let *?pU* = {*p*. *p permutes ?U*}
let *?f* = (λ*i*. *if i = k then a i + b i else c i*)::*'n* ⇒ *'a*::*comm_ring_1*^'*n*
let *?g* = (λ *i*. *if i = k then a i else c i*)::*'n* ⇒ *'a*::*comm_ring_1*^'*n*
let *?h* = (λ *i*. *if i = k then b i else c i*)::*'n* ⇒ *'a*::*comm_ring_1*^'*n*
fix *p*
assume *p*: *p* ∈ *?pU*
let *?Uk = ?U* − {*k*}
from *p* **have** *pU*: *p permutes ?U*
by *blast*
have *kU*: *?U = insert k ?Uk*
by *blast*
have *eq*: *prod* (λ*i*. *?f i* $ *p i*) *?Uk = prod* (λ*i*. *?g i* $ *p i*) *?Uk*
prod (λ*i*. *?f i* $ *p i*) *?Uk = prod* (λ*i*. *?h i* $ *p i*) *?Uk*
by *auto*
have *Uk*: *finite ?Uk k* ∉ *?Uk*
by *auto*
have *prod* (λ*i*. *?f i* $ *p i*) *?U = prod* (λ*i*. *?f i* $ *p i*) (*insert k ?Uk*)
unfolding *kU*[*symmetric*] **..**
also have ... = *?f k* $ *p k* ∗ *prod* (λ*i*. *?f i* $ *p i*) *?Uk*
by (*rule prod.insert*) *auto*
also have ... = (*a k* $ *p k* ∗ *prod* (λ*i*. *?f i* $ *p i*) *?Uk*) + (*b k*$ *p k* ∗ *prod* (λ*i*. *?f i* $ *p i*) *?Uk*)
by (*simp add*: *field_simps*)
also have ... = (*a k* $ *p k* ∗ *prod* (λ*i*. *?g i* $ *p i*) *?Uk*) + (*b k*$ *p k* ∗ *prod* (λ*i*. *?h i* $ *p i*) *?Uk*)
by (*metis eq*)
also have ... = *prod* (λ*i*. *?g i* $ *p i*) (*insert k ?Uk*) + *prod* (λ*i*. *?h i* $ *p i*) (*insert k ?Uk*)
unfolding *prod.insert*[*OF Uk*] **by** *simp*
finally have *prod* (λ*i*. *?f i* $ *p i*) *?U = prod* (λ*i*. *?g i* $ *p i*) *?U* + *prod* (λ*i*. *?h i* $ *p i*) *?U*
unfolding *kU*[*symmetric*] **.**

```
  then show of_int (sign p) * prod (λi. ?f i $ p i) ?U =
    of_int (sign p) * prod (λi. ?g i $ p i) ?U + of_int (sign p) * prod (λi. ?h i $
p i) ?U
    by (simp add: field_simps)
qed auto

lemma det_row_mul:
  fixes a b :: 'n::finite ⇒ _ ^ 'n
  shows det((χ i. if i = k then c *s a i else b i)::'a::comm_ring_1^'n^'n) =
    c * det((χ i. if i = k then a i else b i)::'a::comm_ring_1^'n^'n)
  unfolding det_def vec_lambda_beta sum_distrib_left
proof (rule sum.cong)
  let ?U = UNIV :: 'n set
  let ?pU = {p. p permutes ?U}
  let ?f = (λi. if i = k then c*s a i else b i)::'n ⇒ 'a::comm_ring_1^'n
  let ?g = (λ i. if i = k then a i else b i)::'n ⇒ 'a::comm_ring_1^'n
  fix p
  assume p: p ∈ ?pU
  let ?Uk = ?U - {k}
  from p have pU: p permutes ?U
    by blast
  have kU: ?U = insert k ?Uk
    by blast
  have eq: prod (λi. ?f i $ p i) ?Uk = prod (λi. ?g i $ p i) ?Uk
    by auto
  have Uk: finite ?Uk k ∉ ?Uk
    by auto
  have prod (λi. ?f i $ p i) ?U = prod (λi. ?f i $ p i) (insert k ?Uk)
    unfolding kU[symmetric] ..
  also have ... = ?f k $ p k  * prod (λi. ?f i $ p i) ?Uk
    by (rule prod.insert) auto
  also have ... = (c*s a k) $ p k * prod (λi. ?f i $ p i) ?Uk
    by (simp add: field_simps)
  also have ... = c* (a k $ p k * prod (λi. ?g i $ p i) ?Uk)
    unfolding eq by (simp add: ac_simps)
  also have ... = c* (prod (λi. ?g i $ p i) (insert k ?Uk))
    unfolding prod.insert[OF Uk] by simp
  finally have prod (λi. ?f i $ p i) ?U = c* (prod (λi. ?g i $ p i) ?U)
    unfolding kU[symmetric] .
  then show of_int (sign p) * prod (λi. ?f i $ p i) ?U = c * (of_int (sign p) *
prod (λi. ?g i $ p i) ?U)
    by (simp add: field_simps)
qed auto

lemma det_row_0:
  fixes b :: 'n::finite ⇒ _ ^ 'n
  shows det((χ i. if i = k then 0 else b i)::'a::comm_ring_1^'n^'n) = 0
  using det_row_mul[of k 0 λi. 1 b]
  apply simp
```

```
  apply (simp only: vector_smult_lzero)
  done

lemma det_row_operation:
  fixes A :: 'a::{comm_ring_1}^'n^'n
  assumes ij: i ≠ j
  shows det (χ k. if k = i then row i A + c *s row j A else row k A) = det A
proof −
  let ?Z = (χ k. if k = i then row j A else row k A) :: 'a ^'n^'n
  have th: row i ?Z = row j ?Z by (vector row_def)
  have th2: ((χ k. if k = i then row i A else row k A) :: 'a^'n^'n) = A
    by (vector row_def)
  show ?thesis
    unfolding det_row_add [of i] det_row_mul[of i] det_identical_rows[OF ij th]
th2
    by simp
qed

lemma det_row_span:
  fixes A :: 'a::{field}^'n^'n
  assumes x: x ∈ vec.span {row j A |j. j ≠ i}
  shows det (χ k. if k = i then row i A + x else row k A) = det A
  using x
proof (induction rule: vec.span_induct_alt)
  case base
  have (if k = i then row i A + 0 else row k A) = row k A for k
    by simp
  then show ?case
    by (simp add: row_def)
next
  case (step c z y)
  then obtain j where j: z = row j A i ≠ j
    by blast
  let ?w = row i A + y
  have th0: row i A + (c*s z + y) = ?w + c*s z
    by vector
  let ?d = λx. det (χ k. if k = i then x else row k A)
  have thz: ?d z = 0
    apply (rule det_identical_rows[OF j(2)])
    using j
    apply (vector row_def)
    done
  have ?d (row i A + (c*s z + y)) = ?d (?w + c*s z)
    unfolding th0 ..
  then have ?d (row i A + (c*s z + y)) = det A
    unfolding thz step.IH det_row_mul[of i] det_row_add[of i] by simp
  then show ?case
    unfolding scalar_mult_eq_scaleR .
qed
```

lemma *matrix_id* [*simp*]: *det* (*matrix id*) = *1*
by (*simp add*: *matrix_id_mat_1*)

proposition *det_matrix_scaleR* [*simp*]: *det* (*matrix* (((∗$_R$) *r*)) :: *real*^'*n*^'*n*) = *r* ^ *CARD*('*n*::*finite*)
apply (*subst det_diagonal*)
apply (*auto simp*: *matrix_def mat_def*)
apply (*simp add*: *cart_eq_inner_axis inner_axis_axis*)
done

May as well do this, though it's a bit unsatisfactory since it ignores exact duplicates by considering the rows/columns as a set.

lemma *det_dependent_rows*:
fixes *A*:: '*a*::{*field*}^'*n*^'*n*
assumes *d*: *vec.dependent* (*rows A*)
shows *det A* = *0*
proof −
let *?U* = *UNIV* :: '*n set*
from *d* **obtain** *i* **where** *i*: *row i A* ∈ *vec.span* (*rows A* − {*row i A*})
unfolding *vec.dependent_def rows_def* **by** *blast*
show *?thesis*
proof (*cases* ∀ *i j*. *i* ≠ *j* ⟶ *row i A* ≠ *row j A*)
case *True*
with *i* **have** *vec.span* (*rows A* − {*row i A*}) ⊆ *vec.span* {*row j A* |*j*. *j* ≠ *i*}
by (*auto simp*: *rows_def intro*!: *vec.span_mono*)
then have − *row i A* ∈ *vec.span* {*row j A*|*j*. *j* ≠ *i*}
by (*meson i subsetCE vec.span_neg*)
from *det_row_span*[*OF this*]
have *det A* = *det* (χ *k*. *if k* = *i then 0* ∗*s 1 else row k A*)
unfolding *right_minus vector_smult_lzero* **..**
with *det_row_mul*[*of i 0* λ*i*. *1*]
show *?thesis* **by** *simp*
next
case *False*
then obtain *j k* **where** *jk*: *j* ≠ *k row j A* = *row k A*
by *auto*
from *det_identical_rows*[*OF jk*] **show** *?thesis* **.**
qed
qed

lemma *det_dependent_columns*:
assumes *d*: *vec.dependent* (*columns* (*A*::*real*^'*n*^'*n*))
shows *det A* = *0*
by (*metis d det_dependent_rows rows_transpose det_transpose*)

Multilinearity and the multiplication formula

lemma *Cart_lambda_cong*: (⋀*x*. *f x* = *g x*) ⟹ (*vec_lambda f*::'*a*^'*n*) = (*vec_lambda g* :: '*a*^'*n*)

by *auto*

lemma *det_linear_row_sum*:
assumes *fS*: *finite S*
shows *det* ((χ *i*. *if i* = *k then sum* (*a i*) *S else c i*)::′*a*::*comm_ring_1*^′*n*^′*n*) =
sum (λ*j*. *det* ((χ *i*. *if i* = *k then a i j else c i*)::′*a*^′*n*^′*n*)) *S*
using *fS* **by** (*induct rule*: *finite_induct*; *simp add*: *det_row_0 det_row_add cong*: *if_cong*)

lemma *finite_bounded_functions*:
assumes *fS*: *finite S*
shows *finite* {*f*. (∀ *i* ∈ {*1*.. (*k*::*nat*)}. *f i* ∈ *S*) ∧ (∀ *i*. *i* ∉ {*1* .. *k*} ⟶ *f i* = *i*)}
proof (*induct k*)
case *0*
have ∗: {*f*. ∀ *i*. *f i* = *i*} = {*id*}
by *auto*
show *?case*
by (*auto simp*: ∗)
next
case (*Suc k*)
let *?f* = λ(*y*::*nat*,*g*) *i*. *if i* = *Suc k then y else g i*
let *?S* = *?f* ‘ (*S* × {*f*. (∀ *i*∈{*1*..*k*}. *f i* ∈ *S*) ∧ (∀ *i*. *i* ∉ {*1*..*k*} ⟶ *f i* = *i*)})
have *?S* = {*f*. (∀ *i*∈{*1*.. *Suc k*}. *f i* ∈ *S*) ∧ (∀ *i*. *i* ∉ {*1*.. *Suc k*} ⟶ *f i* = *i*)}
apply (*auto simp*: *image_iff*)
apply (*rename_tac f*)
apply (*rule_tac x*=*f* (*Suc k*) **in** *bexI*)
apply (*rule_tac x* = λ*i*. *if i* = *Suc k then i else f i* **in** *exI*, *auto*)
done
with *finite_imageI*[*OF finite_cartesian_product*[*OF fS Suc.hyps*(*1*)], *of ?f*]
show *?case*
by *metis*
qed

lemma *det_linear_rows_sum_lemma*:
assumes *fS*: *finite S*
and *fT*: *finite T*
shows *det* ((χ *i*. *if i* ∈ *T then sum* (*a i*) *S else c i*):: ′*a*::*comm_ring_1*^′*n*^′*n*) =
sum (λ*f*. *det*((χ *i*. *if i* ∈ *T then a i* (*f i*) *else c i*)::′*a*^′*n*^′*n*))
{*f*. (∀ *i* ∈ *T*. *f i* ∈ *S*) ∧ (∀ *i*. *i* ∉ *T* ⟶ *f i* = *i*)}
using *fT*
proof (*induct T arbitrary*: *a c set*: *finite*)
case *empty*
have *th0*: ⋀*x y*. (χ *i*. *if i* ∈ {} *then x i else y i*) = (χ *i*. *y i*)
by *vector*
from *empty.prems* **show** *?case*
unfolding *th0* **by** (*simp add*: *eq_id_iff*)
next
case (*insert z T a c*)

```
  let ?F = λT. {f. (∀ i ∈ T. f i ∈ S) ∧ (∀ i. i ∉ T ⟶ f i = i)}
  let ?h = λ(y,g) i. if i = z then y else g i
  let ?k = λh. (h(z),(λi. if i = z then i else h i))
  let ?s = λ k a c f. det((χ i. if i ∈ T then a i (f i) else c i)::'a^'n^'n)
  let ?c = λj i. if i = z then a i j else c i
  have thif: ⋀a b c d. (if a ∨ b then c else d) = (if a then c else if b then c else d)
    by simp
  have thif2: ⋀a b c d e. (if a then b else if c then d else e) =
    (if c then (if a then b else d) else (if a then b else e))
    by simp
  from ‹z ∉ T› have nz: ⋀i. i ∈ T ⟹ i ≠ z
    by auto
  have det (χ i. if i ∈ insert z T then sum (a i) S else c i) =
    det (χ i. if i = z then sum (a i) S else if i ∈ T then sum (a i) S else c i)
    unfolding insert_iff thif ..
  also have ... = (∑j∈S. det (χ i. if i ∈ T then sum (a i) S else if i = z then a
i j else c i))
    unfolding det_linear_row_sum[OF fS]
    by (subst thif2) (simp add: nz cong: if_cong)
  finally have tha:
    det (χ i. if i ∈ insert z T then sum (a i) S else c i) =
    (∑(j, f)∈S × ?F T. det (χ i. if i ∈ T then a i (f i)
                            else if i = z then a i j
                            else c i))
    unfolding insert.hyps unfolding sum.cartesian_product by blast
  show ?case unfolding tha
    using ‹z ∉ T›
    by (intro sum.reindex_bij_witness[where i=?k and j=?h])
      (auto intro!: cong[OF refl[of det]] simp: vec_eq_iff)
qed

lemma det_linear_rows_sum:
  fixes S :: 'n::finite set
  assumes fS: finite S
  shows det (χ i. sum (a i) S) =
    sum (λf. det (χ i. a i (f i) :: 'a::comm_ring_1 ^ 'n^'n)) {f. ∀ i. f i ∈ S}
proof −
  have th0: ⋀x y. ((χ i. if i ∈ (UNIV:: 'n set) then x i else y i) :: 'a^'n^'n) = (χ
i. x i)
    by vector
  from det_linear_rows_sum_lemma[OF fS, of UNIV :: 'n set a, unfolded th0,
OF finite]
  show ?thesis by simp
qed

lemma matrix_mul_sum_alt:
  fixes A B :: 'a::comm_ring_1^'n^'n
  shows A ** B = (χ i. sum (λk. A$i$k *s B $ k) (UNIV :: 'n set))
  by (vector matrix_matrix_mult_def sum_component)
```

lemma *det_rows_mul*:
det((χ *i*. *c i* ∗*s a i*)::*'a*::*comm_ring_1*^*'n*^*'n*) =
prod (λ*i*. *c i*) (*UNIV*:: *'n set*) ∗ *det*((χ *i*. *a i*)::*'a*^*'n*^*'n*)
proof (*simp add*: *det_def sum_distrib_left cong add*: *prod.cong*, *rule sum.cong*)
let *?U* = *UNIV* :: *'n set*
let *?PU* = {*p*. *p permutes ?U*}
fix *p*
assume *pU*: *p* ∈ *?PU*
let *?s* = *of_int* (*sign p*)
from *pU* **have** *p*: *p permutes ?U*
by *blast*
have *prod* (λ*i*. *c i* ∗ *a i* $ *p i*) *?U* = *prod c ?U* ∗ *prod* (λ*i*. *a i* $ *p i*) *?U*
unfolding *prod.distrib* **..**
then show *?s* ∗ (∏ *xa*∈*?U*. *c xa* ∗ *a xa* $ *p xa*) =
prod c ?U ∗ (*?s*∗ (∏ *xa*∈*?U*. *a xa* $ *p xa*))
by (*simp add*: *field_simps*)
qed *rule*

proposition *det_mul*:
fixes *A B* :: *'a*::*comm_ring_1*^*'n*^*'n*
shows *det* (*A* ∗∗ *B*) = *det A* ∗ *det B*
proof −
let *?U* = *UNIV* :: *'n set*
let *?F* = {*f*. (∀ *i* ∈ *?U*. *f i* ∈ *?U*) ∧ (∀ *i*. *i* ∉ *?U* ⟶ *f i* = *i*)}
let *?PU* = {*p*. *p permutes ?U*}
have *p* ∈ *?F* **if** *p permutes ?U* **for** *p*
by *simp*
then have *PUF*: *?PU* ⊆ *?F* **by** *blast*
{
fix *f*
assume *fPU*: *f* ∈ *?F* − *?PU*
have *fUU*: *f* ‘ *?U* ⊆ *?U*
using *fPU* **by** *auto*
from *fPU* **have** *f*: ∀ *i* ∈ *?U*. *f i* ∈ *?U* ∀ *i*. *i* ∉ *?U* ⟶ *f i* = *i* ¬(∀ *y*. ∃ !*x*. *f x*
= *y*)
unfolding *permutes_def* **by** *auto*

let *?A* = (χ *i*. *A*$*i*$*f i* ∗*s B*$*f i*) :: *'a*^*'n*^*'n*
let *?B* = (χ *i*. *B*$*f i*) :: *'a*^*'n*^*'n*
{
assume *fni*: ¬ *inj_on f ?U*
then obtain *i j* **where** *ij*: *f i* = *f j i* ≠ *j*
unfolding *inj_on_def* **by** *blast*
then have *row i ?B* = *row j ?B*
by (*vector row_def*)
with *det_identical_rows*[*OF ij*(*2*)]
have *det* (χ *i*. *A*$*i*$*f i* ∗*s B*$*f i*) = *0*
unfolding *det_rows_mul* **by** *force*

```
    }
    moreover
    {
      assume fi: inj_on f ?U
      from f fi have fith: ⋀i j. f i = f j ⟹ i = j
        unfolding inj_on_def by metis
       note fs = fi[unfolded surjective_iff_injective_gen[OF finite finite refl fUU,
symmetric]]
      have ∃!x. f x = y for y
        using fith fs by blast
      with f(3) have det (χ i. A$i$f i *s B$f i) = 0
        by blast
    }
    ultimately have det (χ i. A$i$f i *s B$f i) = 0
      by blast
  }
  then have zth: ∀ f∈ ?F - ?PU. det (χ i. A$i$f i *s B$f i) = 0
    by simp
  {
    fix p
    assume pU: p ∈ ?PU
    from pU have p: p permutes ?U
      by blast
    let ?s = λp. of_int (sign p)
    let ?f = λq. ?s p * (∏ i∈ ?U. A $ i $ p i) * (?s q * (∏ i∈ ?U. B $ i $ q i))
    have (sum (λq. ?s q *
        (∏ i∈ ?U. (χ i. A $ i $ p i *s B $ p i :: 'a^'n^'n) $ i $ q i)) ?PU) =
      (sum (λq. ?s p * (∏ i∈ ?U. A $ i $ p i) * (?s q * (∏ i∈ ?U. B $ i $ q i)))
?PU)
      unfolding sum_permutations_compose_right[OF permutes_inv[OF p], of ?f]
    proof (rule sum.cong)
      fix q
      assume qU: q ∈ ?PU
      then have q: q permutes ?U
        by blast
      from p q have pp: permutation p and pq: permutation q
        unfolding permutation_permutes by auto
      have th00: of_int (sign p) * of_int (sign p) = (1::'a)
        ⋀a. of_int (sign p) * (of_int (sign p) * a) = a
        unfolding mult.assoc[symmetric]
        unfolding of_int_mult[symmetric]
        by (simp_all add: sign_idempotent)
      have ths: ?s q = ?s p * ?s (q ∘ inv p)
        using pp pq permutation_inverse[OF pp] sign_inverse[OF pp]
        by (simp add: th00 ac_simps sign_idempotent sign_compose)
      have th001: prod (λi. B$i$ q (inv p i)) ?U = prod ((λi. B$i$ q (inv p i)) ∘
p) ?U
        by (rule prod.permute[OF p])
      have thp: prod (λi. (χ i. A$i$p i *s B$p i :: 'a^'n^'n) $i $ q i) ?U =
```

prod (λ*i*. *A*$*i*$*p i*) *?U* ∗ *prod* (λ*i*. *B*$*i*$ *q* (*inv p i*)) *?U*
unfolding *th001 prod.distrib*[*symmetric*] *o_def permutes_inverses*[*OF p*]
apply (*rule prod.cong*[*OF refl*])
using *permutes_in_image*[*OF q*]
apply *vector*
done
show *?s q* ∗ *prod* (λ*i*. (((χ *i*. *A*$*i*$*p i* ∗*s B*$*p i*) :: *'a*^*'n*^*'n*)$*i*$*q i*)) *?U* =
?s p ∗ (*prod* (λ*i*. *A*$*i*$*p i*) *?U*) ∗ (*?s* (*q* ∘ *inv p*) ∗ *prod* (λ*i*. *B*$*i*$(*q* ∘ *inv p*)
i) *?U*)
using *ths thp pp pq permutation_inverse*[*OF pp*] *sign_inverse*[*OF pp*]
by (*simp add*: *sign_nz th00 field_simps sign_idempotent sign_compose*)
qed *rule*
}
then have *th2*: *sum* (λ*f*. *det* (χ *i*. *A*$*i*$*f i* ∗*s B*$*f i*)) *?PU* = *det A* ∗ *det B*
unfolding *det_def sum_product*
by (*rule sum.cong* [*OF refl*])
have *det* (*A*∗∗*B*) = *sum* (λ*f*. *det* (χ *i*. *A* $ *i* $ *f i* ∗*s B* $ *f i*)) *?F*
unfolding *matrix_mul_sum_alt det_linear_rows_sum*[*OF finite*]
by *simp*
also have ... = *sum* (λ*f*. *det* (χ *i*. *A*$*i*$*f i* ∗*s B*$*f i*)) *?PU*
using *sum.mono_neutral_cong_left*[*OF finite PUF zth*, *symmetric*]
unfolding *det_rows_mul* **by** *auto*
finally show *?thesis* **unfolding** *th2* .
qed

1.10.2 Relation to invertibility

proposition *invertible_det_nz*:
fixes *A*::*'a*::{*field*}^*'n*^*'n*
shows *invertible A* ⟷ *det A* ≠ *0*
proof (*cases invertible A*)
case *True*
then obtain *B* :: *'a*^*'n*^*'n* **where** *B*: *A* ∗∗ *B* = *mat 1*
unfolding *invertible_right_inverse* **by** *blast*
then have *det* (*A* ∗∗ *B*) = *det* (*mat 1* :: *'a*^*'n*^*'n*)
by *simp*
then show *?thesis*
by (*metis True det_I det_mul mult_zero_left one_neq_zero*)
next
case *False*
let *?U* = *UNIV* :: *'n set*
have *fU*: *finite ?U*
by *simp*
from *False* **obtain** *c i* **where** *c*: *sum* (λ*i*. *c i* ∗*s row i A*) *?U* = *0* **and** *iU*: *i* ∈
?U **and** *ci*: *c i* ≠ *0*
unfolding *invertible_right_inverse matrix_right_invertible_independent_rows*
by *blast*
have *thr0*: − *row i A* = *sum* (λ*j*. (*1* / *c i*) ∗*s* (*c j* ∗*s row j A*)) (*?U* − {*i*})
unfolding *sum_cmul* **using** *c ci*

```
    by (auto simp: sum.remove[OF fU iU] eq_vector_fraction_iff add_eq_0_iff)
  have thr: − row i A ∈ vec.span {row j A| j. j ≠ i}
    unfolding thr0 by (auto intro: vec.span_base vec.span_scale vec.span_sum)
  let ?B = (χ k. if k = i then 0 else row k A) :: 'a^'n^'n
  have thrb: row i ?B = 0 using iU by (vector row_def)
  have det A = 0
    unfolding det_row_span[OF thr, symmetric] right_minus
    unfolding det_zero_row(2)[OF thrb] ..
  then show ?thesis
    by (simp add: False)
qed


lemma det_nz_iff_inj_gen:
  fixes f :: 'a::field^'n ⇒ 'a::field^'n
  assumes Vector_Spaces.linear (∗s) (∗s) f
  shows det (matrix f) ≠ 0 ⟷ inj f
proof
  assume det (matrix f) ≠ 0
  then show inj f
    using assms invertible_det_nz inj_matrix_vector_mult by force
next
  assume inj f
  show det (matrix f) ≠ 0
    using vec.linear_injective_left_inverse [OF assms ‹inj f›]
   by (metis assms invertible_det_nz invertible_left_inverse matrix_compose_gen
matrix_id_mat_1)
qed

lemma det_nz_iff_inj:
  fixes f :: real^'n ⇒ real^'n
  assumes linear f
  shows det (matrix f) ≠ 0 ⟷ inj f
  using det_nz_iff_inj_gen[of f] assms
  unfolding linear_matrix_vector_mul_eq .

lemma det_eq_0_rank:
  fixes A :: real^'n^'n
  shows det A = 0 ⟷ rank A < CARD('n)
  using invertible_det_nz [of A]
 by (auto simp: matrix_left_invertible_injective invertible_left_inverse less_rank_noninjective)
```

Invertibility of matrices and corresponding linear functions

```
lemma matrix_left_invertible_gen:
  fixes f :: 'a::field^'m ⇒ 'a::field^'n
  assumes Vector_Spaces.linear (∗s) (∗s) f
  shows ((∃ B. B ∗∗ matrix f = mat 1) ⟷ (∃ g. Vector_Spaces.linear (∗s) (∗s)
g ∧ g ∘ f = id))
```

proof *safe*
fix *B*
assume *1*: *B ** matrix f = mat 1*
show ∃ *g*. *Vector_Spaces.linear (∗s) (∗s) g ∧ g ∘ f = id*
proof (*intro exI conjI*)
show *Vector_Spaces.linear (∗s) (∗s) (λy. B ∗v y)*
by *simp*
show *((∗v) B) ∘ f = id*
unfolding *o_def*
by (*metis assms 1 eq_id_iff matrix_vector_mul(1) matrix_vector_mul_assoc matrix_vector_mul_lid*)
qed
next
fix *g*
assume *Vector_Spaces.linear (∗s) (∗s) g g ∘ f = id*
then have *matrix g ** matrix f = mat 1*
by (*metis assms matrix_compose_gen matrix_id_mat_1*)
then show ∃ *B*. *B ** matrix f = mat 1* **..**
qed

lemma *matrix_left_invertible*:
linear f ⟹ *((∃ B. B ** matrix f = mat 1)* ⟷ *(∃ g. linear g ∧ g ∘ f = id))* **for** *f*::*real^'m* ⇒ *real^'n*
using *matrix_left_invertible_gen*[*of f*]
by (*auto simp*: *linear_matrix_vector_mul_eq*)

lemma *matrix_right_invertible_gen*:
fixes *f* :: *'a*::*field^'m* ⇒ *'a^'n*
assumes *Vector_Spaces.linear (∗s) (∗s) f*
shows *((∃ B. matrix f ** B = mat 1)* ⟷ *(∃ g. Vector_Spaces.linear (∗s) (∗s) g ∧ f ∘ g = id))*
proof *safe*
fix *B*
assume *1*: *matrix f ** B = mat 1*
show ∃ *g*. *Vector_Spaces.linear (∗s) (∗s) g ∧ f ∘ g = id*
proof (*intro exI conjI*)
show *Vector_Spaces.linear (∗s) (∗s) ((∗v) B)*
by *simp*
show *f ∘ (∗v) B = id*
using *1 assms comp_apply eq_id_iff vec.linear_id matrix_id_mat_1 matrix_vector_mul_assoc matrix_works*
by (*metis* (*no_types*, *opaque_lifting*))
qed
next
fix *g*
assume *Vector_Spaces.linear (∗s) (∗s) g* **and** *f ∘ g = id*
then have *matrix f ** matrix g = mat 1*
by (*metis assms matrix_compose_gen matrix_id_mat_1*)
then show ∃ *B*. *matrix f ** B = mat 1* **..**

qed

lemma *matrix_right_invertible*:
linear f $\Longrightarrow$ ((∃ *B*. *matrix f* ** *B* = *mat 1*) $\longleftrightarrow$ (∃ *g*. *linear g* ∧ *f* ∘ *g* = *id*)) **for** *f*::*real*^'*m* ⇒ *real*^'*n*
using *matrix_right_invertible_gen*[*of f*]
by (*auto simp*: *linear_matrix_vector_mul_eq*)

lemma *matrix_invertible_gen*:
fixes *f* :: '*a*::*field*^'*m* ⇒ '*a*::*field*^'*n*
assumes *Vector_Spaces.linear* (∗*s*) (∗*s*) *f*
shows *invertible* (*matrix f*) $\longleftrightarrow$ (∃ *g*. *Vector_Spaces.linear* (∗*s*) (∗*s*) *g* ∧ *f* ∘ *g* = *id* ∧ *g* ∘ *f* = *id*)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*metis assms invertible_def left_right_inverse_eq matrix_left_invertible_gen matrix_right_invertible_gen*)
next
assume *?rhs* **then show** *?lhs*
by (*metis assms invertible_def matrix_compose_gen matrix_id_mat_1*)
qed

lemma *matrix_invertible*:
linear f $\Longrightarrow$ *invertible* (*matrix f*) $\longleftrightarrow$ (∃ *g*. *linear g* ∧ *f* ∘ *g* = *id* ∧ *g* ∘ *f* = *id*)
for *f*::*real*^'*m* ⇒ *real*^'*n*
using *matrix_invertible_gen*[*of f*]
by (*auto simp*: *linear_matrix_vector_mul_eq*)

lemma *invertible_eq_bij*:
fixes *m* :: '*a*::*field*^'*m*^'*n*
shows *invertible m* $\longleftrightarrow$ *bij* ((∗*v*) *m*)
using *matrix_invertible_gen*[*OF matrix_vector_mul_linear_gen*, *of m*, *simplified matrix_of_matrix_vector_mul*]
by (*metis bij_betw_def left_right_inverse_eq matrix_vector_mul_linear_gen o_bij*
vec.linear_injective_left_inverse vec.linear_surjective_right_inverse)

1.10.3 Cramer's rule

lemma *cramer_lemma_transpose*:
fixes *A*:: '*a*::{*field*}^'*n*^'*n*
and *x* :: '*a*::{*field*}^'*n*
shows *det* ((χ *i*. *if i* = *k then sum* (λ*i*. *x*$*i* ∗*s row i A*) (*UNIV*::'*n set*)
else row i A)::'*a*::{*field*}^'*n*^'*n*) = *x*$*k* ∗ *det A*
(**is** *?lhs* = *?rhs*)
proof −
let *?U* = *UNIV* :: '*n set*
let *?Uk* = *?U* − {*k*}

```
  have U: ?U = insert k ?Uk
    by blast
  have kUk: k ∉ ?Uk
    by simp
  have th00: ⋀k s. x$k *s row k A + s = (x$k - 1) *s row k A + row k A + s
    by (vector field_simps)
  have th001: ⋀f k . (λx. if x = k then f k else f x) = f
    by auto
  have (χ i. row i A) = A by (vector row_def)
  then have thd1: det (χ i. row i A) = det A
    by simp
  have thd0: det (χ i. if i = k then row k A + (∑ i ∈ ?Uk. x $ i *s row i A) else
row i A) = det A
    by (force intro: det_row_span vec.span_sum vec.span_scale vec.span_base)
  show ?lhs = x$k * det A
    apply (subst U)
    unfolding sum.insert[OF finite kUk]
    apply (subst th00)
    unfolding add.assoc
    apply (subst det_row_add)
    unfolding thd0
    unfolding det_row_mul
    unfolding th001[of k λi. row i A]
    unfolding thd1
    apply (simp add: field_simps)
    done
qed

proposition cramer_lemma:
  fixes A :: 'a::{field}^'n^'n
  shows det((χ i j. if j = k then (A *v x)$i else A$i$j):: 'a::{field}^'n^'n) = x$k
* det A
proof -
  let ?U = UNIV :: 'n set
  have *: ⋀c. sum (λi. c i *s row i (transpose A)) ?U = sum (λi. c i *s column i
A) ?U
    by (auto intro: sum.cong)
  show ?thesis
    unfolding matrix_mult_sum
   unfolding cramer_lemma_transpose[of k x transpose A, unfolded det_transpose,
symmetric]
    unfolding *[of λi. x$i]
    apply (subst det_transpose[symmetric])
    apply (rule cong[OF refl[of det]])
    apply (vector transpose_def column_def row_def)
    done
qed

proposition cramer:
```

```
  fixes A ::'a::{field}^'n^'n
  assumes d0: det A ≠ 0
  shows A *v x = b ⟷ x = (χ k. det(χ i j. if j=k then b$i else A$i$j) / det A)
proof −
  from d0 obtain B where B: A ** B = mat 1 B ** A = mat 1
    unfolding invertible_det_nz[symmetric] invertible_def
    by blast
  have (A ** B) *v b = b
    by (simp add: B)
  then have A *v (B *v b) = b
    by (simp add: matrix_vector_mul_assoc)
  then have xe: ∃ x. A *v x = b
    by blast
  {
    fix x
    assume x: A *v x = b
    have x = (χ k. det(χ i j. if j=k then b$i else A$i$j) / det A)
      unfolding x[symmetric]
      using d0 by (simp add: vec_eq_iff cramer_lemma field_simps)
  }
  with xe show ?thesis
    by auto
qed

lemma det_1: det (A::'a::comm_ring_1^1^1) = A$1$1
  by (simp add: det_def sign_id)

lemma det_2: det (A::'a::comm_ring_1^2^2) = A$1$1 * A$2$2 − A$1$2 *
A$2$1
proof −
  have f12: finite {2::2} 1 ∉ {2::2} by auto
  show ?thesis
    unfolding det_def UNIV_2
    unfolding sum_over_permutations_insert[OF f12]
    unfolding permutes_sing
    by (simp add: sign_swap_id sign_id swap_id_eq)
qed

lemma det_3:
  det (A::'a::comm_ring_1^3^3) =
    A$1$1 * A$2$2 * A$3$3 +
    A$1$2 * A$2$3 * A$3$1 +
    A$1$3 * A$2$1 * A$3$2 −
    A$1$1 * A$2$3 * A$3$2 −
    A$1$2 * A$2$1 * A$3$3 −
    A$1$3 * A$2$2 * A$3$1
proof −
  have f123: finite {2::3, 3} 1 ∉ {2::3, 3}
    by auto
```

have *f23*: *finite* {*3*::*3*} *2* ∉ {*3*::*3*}
by *auto*

show *?thesis*
unfolding *det_def UNIV_3*
unfolding *sum_over_permutations_insert*[*OF f123*]
unfolding *sum_over_permutations_insert*[*OF f23*]
unfolding *permutes_sing*
by (*simp add*: *sign_swap_id permutation_swap_id sign_compose sign_id swap_id_eq*)
qed

proposition *det_orthogonal_matrix*:
fixes *Q*:: *'a*::*linordered_idom*^*'n*^*'n*
assumes *oQ*: *orthogonal_matrix Q*
shows *det Q* = *1* ∨ *det Q* = − *1*
proof −
have *Q* ** *transpose Q* = *mat 1*
by (*metis oQ orthogonal_matrix_def*)
then have *det* (*Q* ** *transpose Q*) = *det* (*mat 1*:: *'a*^*'n*^*'n*)
by *simp*
then have *det Q* * *det Q* = *1*
by (*simp add*: *det_mul*)
then show *?thesis*
by (*simp add*: *square_eq_1_iff*)
qed

proposition *orthogonal_transformation_det* [*simp*]:
fixes *f* :: *real*^*'n* ⇒ *real*^*'n*
shows *orthogonal_transformation f* ⟹ |*det* (*matrix f*)| = *1*
using *det_orthogonal_matrix orthogonal_transformation_matrix* **by** *fastforce*

1.10.4 Rotation, reflection, rotoinversion

definition *rotation_matrix Q* ⟷ *orthogonal_matrix Q* ∧ *det Q* = *1*
definition *rotoinversion_matrix Q* ⟷ *orthogonal_matrix Q* ∧ *det Q* = − *1*

lemma *orthogonal_rotation_or_rotoinversion*:
fixes *Q* :: *'a*::*linordered_idom*^*'n*^*'n*
shows *orthogonal_matrix Q* ⟷ *rotation_matrix Q* ∨ *rotoinversion_matrix Q*
by (*metis rotoinversion_matrix_def rotation_matrix_def det_orthogonal_matrix*)

Slightly stronger results giving rotation, but only in two or more dimensions

lemma *rotation_matrix_exists_basis*:
fixes *a* :: *real*^*'n*
assumes *2*: *2* ≤ *CARD*(*'n*) **and** *norm a* = *1*
obtains *A* **where** *rotation_matrix A A* *v (*axis k 1*) = *a*
proof −
obtain *A* **where** *orthogonal_matrix A* **and** *A*: *A* *v (*axis k 1*) = *a*

```
    using orthogonal_matrix_exists_basis assms by metis
  with orthogonal_rotation_or_rotoinversion
  consider rotation_matrix A | rotoinversion_matrix A
    by metis
  then show thesis
  proof cases
    assume rotation_matrix A
    then show ?thesis
      using ‹A *v axis k 1 = a› that by auto
  next
    from obtain_subset_with_card_n[OF 2] obtain h i::'n where h ≠ i
      by (fastforce simp add: eval_nat_numeral card_Suc_eq)
    then obtain j where j ≠ k
      by (metis (full_types))
    let ?TA = transpose A
    let ?A = χ i. if i = j then - 1 *R (?TA $ i) else ?TA $i
    assume rotoinversion_matrix A
    then have [simp]: det A = -1
      by (simp add: rotoinversion_matrix_def)
    show ?thesis
    proof
      have [simp]: row i (χ i. if i = j then - 1 *R ?TA $ i else ?TA $ i) = (if i =
j then - row i ?TA else row i ?TA) for i
        by (auto simp: row_def)
      have orthogonal_matrix ?A
        unfolding orthogonal_matrix_orthonormal_rows
      using ‹orthogonal_matrix A› by (auto simp: orthogonal_matrix_orthonormal_columns
orthogonal_clauses)
      then show rotation_matrix (transpose ?A)
        unfolding rotation_matrix_def
      by (simp add: det_row_mul[of j _ λi. ?TA $ i, unfolded scalar_mult_eq_scaleR])
      show transpose ?A *v axis k 1 = a
      using ‹j ≠ k› A by (simp add: matrix_vector_column axis_def scalar_mult_eq_scaleR
if_distrib [of λz. z *R c for c] cong: if_cong)
    qed
  qed
qed

lemma rotation_exists_1:
  fixes a :: real^'n
  assumes 2 ≤ CARD('n) norm a = 1 norm b = 1
  obtains f where orthogonal_transformation f det(matrix f) = 1 f a = b
proof -
  obtain k::'n where True
    by simp
  obtain A B where AB: rotation_matrix A rotation_matrix B
              and eq: A *v (axis k 1) = a B *v (axis k 1) = b
    using rotation_matrix_exists_basis assms by metis
  let ?f = λx. (B ** transpose A) *v x
```

```
  show thesis
  proof
    show orthogonal_transformation ?f
      using AB orthogonal_matrix_mul orthogonal_transformation_matrix rota-
tion_matrix_def matrix_vector_mul_linear by force
    show det (matrix ?f) = 1
      using AB by (auto simp: det_mul rotation_matrix_def)
    show ?f a = b
      using AB unfolding orthogonal_matrix_def rotation_matrix_def
      by (metis eq matrix_mul_rid matrix_vector_mul_assoc)
  qed
qed

lemma rotation_exists:
  fixes a :: real^'n
  assumes 2: 2 ≤ CARD('n) and eq: norm a = norm b
  obtains f where orthogonal_transformation f det(matrix f) = 1 f a = b
proof (cases a = 0 ∨ b = 0)
  case True
  with assms have a = 0 b = 0
    by auto
  then show ?thesis
    by (metis eq_id_iff matrix_id orthogonal_transformation_id that)
next
  case False
  then obtain f where f: orthogonal_transformation f det (matrix f) = 1
    and f': f (a /R norm a) = b /R norm b
    using rotation_exists_1 [of a /R norm a b /R norm b, OF 2] by auto
  then interpret linear f by (simp add: orthogonal_transformation)
  have f a = b
    using f' False
    by (simp add: eq scale)
  with f show thesis ..
qed

lemma rotation_rightward_line:
  fixes a :: real^'n
  obtains f where orthogonal_transformation f 2 ≤ CARD('n) ⟹ det(matrix
f) = 1
                  f(norm a *R axis k 1) = a
proof (cases CARD('n) = 1)
  case True
  obtain f where orthogonal_transformation f f (norm a *R axis k (1::real)) = a
  proof (rule orthogonal_transformation_exists)
    show norm (norm a *R axis k (1::real)) = norm a
      by simp
  qed auto
  then show thesis
    using True that by auto
```

next
case *False*
obtain *f* **where** *orthogonal_transformation f det(matrix f) = 1 f (norm a* $*_R$ *axis k 1) = a*
proof (*rule rotation_exists*)
show *2* $\leq$ *CARD('n)*
using *False one_le_card_finite* [**where** *'a='n*] **by** *linarith*
show *norm (norm a* $*_R$ *axis k (1::real)) = norm a*
by *simp*
qed *auto*
then show *thesis*
using *that* **by** *blast*
qed

end

1.11 Operators involving abstract topology

theory *Abstract_Topology*
imports
Complex_Main
HOL−Library.Set_Idioms
HOL−Library.FuncSet
begin

1.11.1 General notion of a topology as a value

definition *istopology* :: (*'a set* $\Rightarrow$ *bool*) $\Rightarrow$ *bool* **where**
istopology L $\equiv$ ($\forall$ *S T. L S* $\longrightarrow$ *L T* $\longrightarrow$ *L (S* $\cap$ *T))* $\wedge$ ($\forall \mathcal{K}$. ($\forall$ *K*$\in\mathcal{K}$. *L K*) $\longrightarrow$ *L* ($\bigcup \mathcal{K}$))

typedef *'a topology* = {*L*::(*'a set*) $\Rightarrow$ *bool. istopology L*}
morphisms *openin topology*
unfolding *istopology_def* **by** *blast*

lemma *istopology_openin*[*iff*]: *istopology*(*openin U*)
using *openin*[*of U*] **by** *blast*

lemma *istopology_open*[*iff*]: *istopology open*
by (*auto simp*: *istopology_def*)

lemma *topology_inverse'* [*simp*]: *istopology U* $\Longrightarrow$ *openin* (*topology U*) = *U*
using *topology_inverse*[*unfolded mem_Collect_eq*] **.**

lemma *topology_inverse_iff*: *istopology U* $\longleftrightarrow$ *openin* (*topology U*) = *U*
by (*metis istopology_openin topology_inverse'*)

lemma *topology_eq*: *T1* = *T2* $\longleftrightarrow$ ($\forall$ *S. openin T1 S* $\longleftrightarrow$ *openin T2 S*)
proof

assume *T1 = T2*
then show $\forall S.$ *openin T1 S* $\longleftrightarrow$ *openin T2 S* **by** *simp*
next
assume *H*: $\forall S.$ *openin T1 S* $\longleftrightarrow$ *openin T2 S*
then have *openin T1 = openin T2* **by** (*simp add*: *fun_eq_iff*)
then have *topology* (*openin T1*) = *topology* (*openin T2*) **by** *simp*
then show *T1 = T2* **unfolding** *openin_inverse* **.**
qed

The "universe": the union of all sets in the topology.

definition *topspace T* = $\bigcup\{S.$ *openin T S*$\}$

Main properties of open sets

proposition *openin_clauses*:
fixes *U* :: *'a topology*
shows
openin U {}
$\bigwedge S\ T.$ *openin U S* $\Longrightarrow$ *openin U T* $\Longrightarrow$ *openin U* (*S*$\cap$*T*)
$\bigwedge K.$ ($\forall S \in K.$ *openin U S*) $\Longrightarrow$ *openin U* ($\bigcup K$)
using *openin*[*of U*] **unfolding** *istopology_def* **by** *auto*

lemma *openin_subset*: *openin U S* $\Longrightarrow$ *S* $\subseteq$ *topspace U*
unfolding *topspace_def* **by** *blast*

lemma *openin_empty*[*simp*]: *openin U* {}
by (*rule openin_clauses*)

lemma *openin_Int*[*intro*]: *openin U S* $\Longrightarrow$ *openin U T* $\Longrightarrow$ *openin U* (*S* $\cap$ *T*)
by (*rule openin_clauses*)

lemma *openin_Union*[*intro*]: ($\bigwedge S.$ *S* $\in$ *K* $\Longrightarrow$ *openin U S*) $\Longrightarrow$ *openin U* ($\bigcup K$)
using *openin_clauses* **by** *blast*

lemma *openin_Un*[*intro*]: *openin U S* $\Longrightarrow$ *openin U T* $\Longrightarrow$ *openin U* (*S* $\cup$ *T*)
using *openin_Union*[*of* {*S*,*T*} *U*] **by** *auto*

lemma *openin_topspace*[*intro*, *simp*]: *openin U* (*topspace U*)
by (*force simp*: *openin_Union topspace_def*)

lemma *openin_subopen*: *openin U S* $\longleftrightarrow$ ($\forall x \in S.\ \exists T.$ *openin U T* $\wedge$ *x* $\in$ *T* $\wedge$ *T* $\subseteq$ *S*)
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
assume *?lhs*
then show *?rhs* **by** *auto*
next
assume *H*: *?rhs*
let *?t* = $\bigcup\{T.$ *openin U T* $\wedge$ *T* $\subseteq$ *S*$\}$

have *openin U ?t* **by** (*force simp*: *openin_Union*)
also have *?t* = *S* **using** *H* **by** *auto*
finally show *openin U S* **.**
qed

lemma *openin_INT* [*intro*]:
assumes *finite I*
⋀*i*. *i* ∈ *I* ⟹ *openin T* (*U i*)
shows *openin T* ((⋂ *i* ∈ *I*. *U i*) ∩ *topspace T*)
using *assms* **by** (*induct*, *auto simp*: *inf_sup_aci*(*2*) *openin_Int*)

lemma *openin_INT2* [*intro*]:
assumes *finite I I* ≠ {}
⋀*i*. *i* ∈ *I* ⟹ *openin T* (*U i*)
shows *openin T* (⋂ *i* ∈ *I*. *U i*)
proof −
have (⋂ *i* ∈ *I*. *U i*) ⊆ *topspace T*
using ‹*I* ≠ {}› *openin_subset*[*OF assms*(*3*)] **by** *auto*
then show *?thesis*
using *openin_INT*[*of* _ _ *U*, *OF assms*(*1*) *assms*(*3*)] **by** (*simp add*: *inf.absorb2 inf_commute*)
qed

lemma *openin_Inter* [*intro*]:
assumes *finite* ℱ ℱ ≠ {} ⋀*X*. *X* ∈ ℱ ⟹ *openin T X* **shows** *openin T* (⋂ ℱ)
by (*metis* (*full_types*) *assms openin_INT2 image_ident*)

lemma *openin_Int_Inter*:
assumes *finite* ℱ *openin T U* ⋀*X*. *X* ∈ ℱ ⟹ *openin T X* **shows** *openin T* (*U* ∩ ⋂ ℱ)
using *openin_Inter* [*of insert U* ℱ] *assms* **by** *auto*

Closed sets

definition *closedin* :: *'a topology* ⇒ *'a set* ⇒ *bool* **where**
closedin U S ⟷ *S* ⊆ *topspace U* ∧ *openin U* (*topspace U* − *S*)

lemma *closedin_subset*: *closedin U S* ⟹ *S* ⊆ *topspace U*
by (*metis closedin_def*)

lemma *closedin_empty*[*simp*]: *closedin U* {}
by (*simp add*: *closedin_def*)

lemma *closedin_topspace*[*intro*, *simp*]: *closedin U* (*topspace U*)
by (*simp add*: *closedin_def*)

lemma *closedin_Un*[*intro*]: *closedin U S* ⟹ *closedin U T* ⟹ *closedin U* (*S* ∪ *T*)
by (*auto simp*: *Diff_Un closedin_def*)

lemma *Diff_Inter*[*intro*]: $A - \bigcap S = \bigcup \{A - s | s.\ s \in S\}$
by *auto*

lemma *closedin_Union*:
assumes *finite S* $\bigwedge T.\ T \in S \Longrightarrow$ *closedin U T*
shows *closedin U* $(\bigcup S)$
using *assms* **by** *induction auto*

lemma *closedin_Inter*[*intro*]:
assumes *Ke*: $K \neq \{\}$
and *Kc*: $\bigwedge S.\ S \in K \Longrightarrow$ *closedin U S*
shows *closedin U* $(\bigcap K)$
using *Ke Kc* **unfolding** *closedin_def Diff_Inter* **by** *auto*

lemma *closedin_INT*[*intro*]:
assumes $A \neq \{\}$ $\bigwedge x.\ x \in A \Longrightarrow$ *closedin U* $(B\ x)$
shows *closedin U* $(\bigcap x \in A.\ B\ x)$
using *assms* **by** *blast*

lemma *closedin_Int*[*intro*]: *closedin U S* $\Longrightarrow$ *closedin U T* $\Longrightarrow$ *closedin U* $(S \cap T)$
using *closedin_Inter*[*of* $\{S,T\}$ *U*] **by** *auto*

lemma *openin_closedin_eq*: *openin U S* $\longleftrightarrow$ $S \subseteq$ *topspace U* $\wedge$ *closedin U* (*topspace U* $-$ *S*)
by (*metis Diff_subset closedin_def double_diff equalityD1 openin_subset*)

lemma *topology_finer_closedin*:
topspace X $=$ *topspace Y* $\Longrightarrow$ $(\forall S.$ *openin Y S* $\longrightarrow$ *openin X S*$)$ $\longleftrightarrow$ $(\forall S.$ *closedin Y S* $\longrightarrow$ *closedin X S*$)$
by (*metis closedin_def openin_closedin_eq*)

lemma *openin_closedin*: $S \subseteq$ *topspace U* $\Longrightarrow$ (*openin U S* $\longleftrightarrow$ *closedin U* (*topspace U* $-$ *S*))
by (*simp add*: *openin_closedin_eq*)

lemma *openin_diff*[*intro*]:
assumes *oS*: *openin U S*
and *cT*: *closedin U T*
shows *openin U* $(S - T)$
by (*metis Int_Diff cT closedin_def inf.orderE oS openin_Int openin_subset*)

lemma *closedin_diff*[*intro*]:
assumes *oS*: *closedin U S*
and *cT*: *openin U T*
shows *closedin U* $(S - T)$
by (*metis Int_Diff cT closedin_Int closedin_subset inf.orderE oS openin_closedin_eq*)

lemma *all_openin*: $(\forall\, U.\ \mathit{openin}\ X\ U \longrightarrow P\ U) \longleftrightarrow (\forall\, U.\ \mathit{closedin}\ X\ U \longrightarrow P\ (\mathit{topspace}\ X - U))$
by (*metis Diff_Diff_Int closedin_def inf.absorb_iff2 openin_closedin_eq*)

lemma *all_closedin*: $(\forall\, U.\ \mathit{closedin}\ X\ U \longrightarrow P\ U) \longleftrightarrow (\forall\, U.\ \mathit{openin}\ X\ U \longrightarrow P\ (\mathit{topspace}\ X - U))$
by (*metis Diff_Diff_Int closedin_subset inf.absorb_iff2 openin_closedin_eq*)

lemma *ex_openin*: $(\exists\, U.\ \mathit{openin}\ X\ U \wedge P\ U) \longleftrightarrow (\exists\, U.\ \mathit{closedin}\ X\ U \wedge P\ (\mathit{topspace}\ X - U))$
by (*metis Diff_Diff_Int closedin_def inf.absorb_iff2 openin_closedin_eq*)

lemma *ex_closedin*: $(\exists\, U.\ \mathit{closedin}\ X\ U \wedge P\ U) \longleftrightarrow (\exists\, U.\ \mathit{openin}\ X\ U \wedge P\ (\mathit{topspace}\ X - U))$
by (*metis Diff_Diff_Int closedin_subset inf.absorb_iff2 openin_closedin_eq*)

1.11.2 The discrete topology

definition *discrete_topology* **where** $\mathit{discrete\_topology}\ U \equiv \mathit{topology}\ (\lambda S.\ S \subseteq U)$

lemma *openin_discrete_topology* [*simp*]: $\mathit{openin}\ (\mathit{discrete\_topology}\ U)\ S \longleftrightarrow S \subseteq U$
proof −
have *istopology* $(\lambda S.\ S \subseteq U)$
by (*auto simp*: *istopology_def*)
then show *?thesis*
by (*simp add*: *discrete_topology_def topology_inverse′*)
qed

lemma *topspace_discrete_topology* [*simp*]: $\mathit{topspace}(\mathit{discrete\_topology}\ U) = U$
by (*meson openin_discrete_topology openin_subset openin_topspace order_refl subset_antisym*)

lemma *closedin_discrete_topology* [*simp*]: $\mathit{closedin}\ (\mathit{discrete\_topology}\ U)\ S \longleftrightarrow S \subseteq U$
by (*simp add*: *closedin_def*)

lemma *discrete_topology_unique*:
$\mathit{discrete\_topology}\ U = X \longleftrightarrow \mathit{topspace}\ X = U \wedge (\forall\, x \in U.\ \mathit{openin}\ X\ \{x\})$ (**is** *?lhs = ?rhs*)
proof
assume *R*: *?rhs*
then have $\mathit{openin}\ X\ S$ **if** $S \subseteq U$ **for** S
using *openin_subopen subsetD that* **by** *fastforce*
then show *?lhs*
by (*metis R openin_discrete_topology openin_subset topology_eq*)
qed *auto*

lemma *discrete_topology_unique_alt*:

discrete_topology U = X ⟷ topspace X ⊆ U ∧ (∀x ∈ U. openin X {x})
using *openin_subset*
by (*auto simp*: *discrete_topology_unique*)

lemma *subtopology_eq_discrete_topology_empty*:
X = discrete_topology {} ⟷ topspace X = {}
using *discrete_topology_unique* [*of* {} *X*] **by** *auto*

lemma *subtopology_eq_discrete_topology_sing*:
X = discrete_topology {a} ⟷ topspace X = {a}
by (*metis discrete_topology_unique openin_topspace singletonD*)

abbreviation *trivial_topology* **where** *trivial_topology ≡ discrete_topology {}*

lemma *null_topspace_iff_trivial* [*simp*]: *topspace X = {} ⟷ X = trivial_topology*
by (*simp add*: *subtopology_eq_discrete_topology_empty*)

1.11.3 Subspace topology

definition *subtopology* :: *'a topology ⇒ 'a set ⇒ 'a topology*
where *subtopology U V = topology* (*λT. ∃S. T = S ∩ V ∧ openin U S*)

lemma *istopology_subtopology*: *istopology* (*λT. ∃S. T = S ∩ V ∧ openin U S*)
(**is** *istopology ?L*)
proof −
have *?L {}* **by** *blast*
{
fix *A B*
assume *A*: *?L A* **and** *B*: *?L B*
from *A B* **obtain** *Sa* **and** *Sb* **where** *Sa*: *openin U Sa A = Sa ∩ V* **and** *Sb*: *openin U Sb B = Sb ∩ V*
by *blast*
have *A ∩ B = (Sa ∩ Sb) ∩ V openin U (Sa ∩ Sb)*
using *Sa Sb* **by** *blast+*
then have *?L (A ∩ B)* **by** *blast*
}
moreover
{
fix *K*
assume *K*: *K ⊆ Collect ?L*
have *th0*: *Collect ?L = (λS. S ∩ V) ‘ Collect (openin U)*
by *blast*
from *K*[*unfolded th0 subset_image_iff*]
obtain *Sk* **where** *Sk*: *Sk ⊆ Collect (openin U) K = (λS. S ∩ V) ‘ Sk*
by *blast*
have *⋃K = (⋃Sk) ∩ V*
using *Sk* **by** *auto*
moreover have *openin U (⋃Sk)*
using *Sk* **by** (*auto simp*: *subset_eq*)

ultimately have *?L* $(\bigcup K)$ **by** *blast*
}
ultimately show *?thesis*
unfolding *subset_eq mem_Collect_eq istopology_def* **by** *auto*
qed

lemma *openin_subtopology*: *openin* (*subtopology U V*) $S \longleftrightarrow (\exists T.$ *openin U T* $\wedge$ $S = T \cap V)$
unfolding *subtopology_def topology_inverse'*[*OF istopology_subtopology*]
by *auto*

lemma *subset_openin_subtopology*:
$[\![$*openin X S*; $S \subseteq T]\!] \Longrightarrow$ *openin* (*subtopology X T*) *S*
by (*metis inf.orderE openin_subtopology*)

lemma *openin_subtopology_Int*:
openin X S $\Longrightarrow$ *openin* (*subtopology X T*) $(S \cap T)$
using *openin_subtopology* **by** *auto*

lemma *openin_subtopology_Int2*:
openin X T $\Longrightarrow$ *openin* (*subtopology X S*) $(S \cap T)$
using *openin_subtopology* **by** *auto*

lemma *openin_subtopology_diff_closed*:
$[\![S \subseteq$ *topspace X*; *closedin X T*$]\!] \Longrightarrow$ *openin* (*subtopology X S*) $(S - T)$
unfolding *closedin_def openin_subtopology*
by (*rule_tac x=topspace X* $-$ *T* **in** *exI*) *auto*

lemma *openin_relative_to*: (*openin X relative_to S*) = *openin* (*subtopology X S*)
by (*force simp*: *relative_to_def openin_subtopology*)

lemma *topspace_subtopology* [*simp*]: *topspace* (*subtopology U V*) = *topspace U* $\cap$ *V*
by (*auto simp*: *topspace_def openin_subtopology*)

lemma *topspace_subtopology_subset*:
$S \subseteq$ *topspace X* $\Longrightarrow$ *topspace*(*subtopology X S*) = *S*
by (*simp add*: *inf.absorb_iff2*)

lemma *closedin_subtopology*: *closedin* (*subtopology U V*) $S \longleftrightarrow (\exists T.$ *closedin U T* $\wedge$ $S = T \cap V)$
unfolding *closedin_def topspace_subtopology*
by (*auto simp*: *openin_subtopology*)

lemma *closedin_subtopology_Int_closed*:
closedin X T $\Longrightarrow$ *closedin* (*subtopology X S*) $(S \cap T)$
using *closedin_subtopology inf_commute* **by** *blast*

lemma *closedin_subset_topspace*:

⟦*closedin X S*; *S* ⊆ *T*⟧ ⟹ *closedin* (*subtopology X T*) *S*
using *closedin_subtopology* **by** *fastforce*

lemma *closedin_relative_to*:
(*closedin X relative_to S*) = *closedin* (*subtopology X S*)
by (*force simp*: *relative_to_def closedin_subtopology*)

lemma *openin_subtopology_refl*: *openin* (*subtopology U V*) *V* ⟷ *V* ⊆ *topspace U*
unfolding *openin_subtopology*
by *auto* (*metis IntD1 in_mono openin_subset*)

lemma *subtopology_trivial_iff*: *subtopology X S* = *trivial_topology* ⟷ *X* = *trivial_topology* ∨ *topspace X* ∩ *S* = {}
by (*auto simp flip*: *null_topspace_iff_trivial*)

lemma *subtopology_subtopology*:
subtopology (*subtopology X S*) *T* = *subtopology X* (*S* ∩ *T*)
proof −
have *eq*: ⋀*T'*. (∃*S'*. *T'* = *S'* ∩ *T* ∧ (∃*T*. *openin X T* ∧ *S'* = *T* ∩ *S*)) = (∃*Sa*. *T'* = *Sa* ∩ (*S* ∩ *T*) ∧ *openin X Sa*)
by (*metis inf_assoc*)
have *subtopology* (*subtopology X S*) *T* = *topology* (λ*Ta*. ∃*Sa*. *Ta* = *Sa* ∩ *T* ∧ *openin* (*subtopology X S*) *Sa*)
by (*simp add*: *subtopology_def*)
also have ... = *subtopology X* (*S* ∩ *T*)
by (*simp add*: *openin_subtopology eq*) (*simp add*: *subtopology_def*)
finally show *?thesis* **.**
qed

lemma *openin_subtopology_alt*:
openin (*subtopology X U*) *S* ⟷ *S* ∈ (λ*T*. *U* ∩ *T*) ‘ *Collect* (*openin X*)
by (*simp add*: *image_iff inf_commute openin_subtopology*)

lemma *closedin_subtopology_alt*:
closedin (*subtopology X U*) *S* ⟷ *S* ∈ (λ*T*. *U* ∩ *T*) ‘ *Collect* (*closedin X*)
by (*simp add*: *image_iff inf_commute closedin_subtopology*)

lemma *subtopology_superset*:
assumes *UV*: *topspace U* ⊆ *V*
shows *subtopology U V* = *U*
proof −
{ fix *S*
have *openin U S* **if** *openin U T S* = *T* ∩ *V* **for** *T*
by (*metis Int_subset_iff assms inf.orderE openin_subset that*)
then have (∃*T*. *openin U T* ∧ *S* = *T* ∩ *V*) ⟷ *openin U S*
by (*metis assms inf.orderE inf_assoc openin_subset*)
}
then show *?thesis*

unfolding *topology_eq openin_subtopology* **by** *blast*
qed

lemma *subtopology_topspace*[*simp*]: *subtopology U* (*topspace U*) = *U*
by (*simp add*: *subtopology_superset*)

lemma *subtopology_UNIV*[*simp*]: *subtopology U UNIV* = *U*
by (*simp add*: *subtopology_superset*)

lemma *subtopology_empty_iff_trivial* [*simp*]: *subtopology X* {} = *trivial_topology*
by (*simp add*: *subtopology_eq_discrete_topology_empty*)

lemma *subtopology_restrict*:
subtopology X (*topspace X* ∩ *S*) = *subtopology X S*
by (*metis subtopology_subtopology subtopology_topspace*)

lemma *openin_subtopology_empty*:
openin (*subtopology U* {}) *S* ⟷ *S* = {}
by (*metis Int_empty_right openin_empty openin_subtopology*)

lemma *closedin_subtopology_empty*:
closedin (*subtopology U* {}) *S* ⟷ *S* = {}
by (*metis Int_empty_right closedin_empty closedin_subtopology*)

lemma *closedin_subtopology_refl* [*simp*]:
closedin (*subtopology U X*) *X* ⟷ *X* ⊆ *topspace U*
by (*metis closedin_def closedin_topspace inf.absorb_iff2 le_inf_iff topspace_subtopology*)

lemma *closedin_topspace_empty* [*simp*]: *closedin trivial_topology S* ⟷ *S* = {}
by (*simp add*: *closedin_def*)

lemma *openin_topspace_empty* [*simp*]:
openin trivial_topology S ⟷ *S* = {}
by (*simp add*: *openin_closedin_eq*)

lemma *openin_imp_subset*:
openin (*subtopology U S*) *T* ⟹ *T* ⊆ *S*
by (*metis Int_iff openin_subtopology subsetI*)

lemma *closedin_imp_subset*:
closedin (*subtopology U S*) *T* ⟹ *T* ⊆ *S*
by (*simp add*: *closedin_def*)

lemma *openin_open_subtopology*:
openin X S ⟹ *openin* (*subtopology X S*) *T* ⟷ *openin X T* ∧ *T* ⊆ *S*
by (*metis inf.orderE openin_Int openin_imp_subset openin_subtopology*)

lemma *closedin_closed_subtopology*:
closedin X S ⟹ (*closedin* (*subtopology X S*) *T* ⟷ *closedin X T* ∧ *T* ⊆ *S*)

by (*metis closedin_Int closedin_imp_subset closedin_subtopology inf.orderE*)

lemma *closedin_trans_full*:
⟦*closedin* (*subtopology X U*) *S*; *closedin X U*⟧ ⟹ *closedin X S*
using *closedin_closed_subtopology* **by** *blast*

lemma *openin_subtopology_Un*:
⟦*openin* (*subtopology X T*) *S*; *openin* (*subtopology X U*) *S*⟧
⟹ *openin* (*subtopology X* (*T* ∪ *U*)) *S*
by (*simp add*: *openin_subtopology*) *blast*

lemma *closedin_subtopology_Un*:
⟦*closedin* (*subtopology X T*) *S*; *closedin* (*subtopology X U*) *S*⟧
⟹ *closedin* (*subtopology X* (*T* ∪ *U*)) *S*
by (*simp add*: *closedin_subtopology*) *blast*

lemma *openin_trans_full*:
⟦*openin* (*subtopology X U*) *S*; *openin X U*⟧ ⟹ *openin X S*
by (*simp add*: *openin_open_subtopology*)

1.11.4 The canonical topology from the underlying type class

abbreviation *euclidean* :: *'a*::*topological_space topology*
where *euclidean* ≡ *topology open*

abbreviation *top_of_set* :: *'a*::*topological_space set* ⇒ *'a topology*
where *top_of_set* ≡ *subtopology* (*topology open*)

lemma *open_openin*: *open S* ⟷ *openin euclidean S*
by *simp*

declare *open_openin* [*symmetric*, *simp*]

lemma *topspace_euclidean* [*simp*]: *topspace euclidean* = *UNIV*
by (*force simp*: *topspace_def*)

lemma *topspace_euclidean_subtopology*[*simp*]: *topspace* (*top_of_set S*) = *S*
by (*simp*)

lemma *closed_closedin*: *closed S* ⟷ *closedin euclidean S*
by (*simp add*: *closed_def closedin_def Compl_eq_Diff_UNIV*)

declare *closed_closedin* [*symmetric*, *simp*]

lemma *openin_subtopology_self* [*simp*]: *openin* (*top_of_set S*) *S*
by (*metis openin_topspace topspace_euclidean_subtopology*)

lemma *euclidean_nontrivial* [*simp*]: *euclidean* ≠ *trivial_topology*
by (*simp add*: *subtopology_eq_discrete_topology_empty*)

The most basic facts about the usual topology and metric on R

abbreviation *euclideanreal* :: *real topology*
where *euclideanreal* ≡ *topology open*

1.11.5 Basic "localization" results are handy for connectedness.

lemma *openin_open*: *openin* (*top_of_set U*) *S* ⟷ (∃ *T*. *open T* ∧ (*S* = *U* ∩ *T*))
by (*auto simp*: *openin_subtopology*)

lemma *openin_Int_open*:
⟦*openin* (*top_of_set U*) *S*; *open T*⟧
⟹ *openin* (*top_of_set U*) (*S* ∩ *T*)
by (*metis open_Int Int_assoc openin_open*)

lemma *openin_open_Int*[*intro*]: *open S* ⟹ *openin* (*top_of_set U*) (*U* ∩ *S*)
by (*auto simp*: *openin_open*)

lemma *open_openin_trans*[*trans*]:
open S ⟹ *open T* ⟹ *T* ⊆ *S* ⟹ *openin* (*top_of_set S*) *T*
by (*metis Int_absorb1 openin_open_Int*)

lemma *open_subset*: *S* ⊆ *T* ⟹ *open S* ⟹ *openin* (*top_of_set T*) *S*
by (*auto simp*: *openin_open*)

lemma *closedin_closed*: *closedin* (*top_of_set U*) *S* ⟷ (∃ *T*. *closed T* ∧ *S* = *U* ∩ *T*)
by (*simp add*: *closedin_subtopology Int_ac*)

lemma *closedin_closed_Int*: *closed S* ⟹ *closedin* (*top_of_set U*) (*U* ∩ *S*)
by (*metis closedin_closed*)

lemma *closed_subset*: *S* ⊆ *T* ⟹ *closed S* ⟹ *closedin* (*top_of_set T*) *S*
by (*auto simp*: *closedin_closed*)

lemma *closedin_closed_subset*:
⟦*closedin* (*top_of_set U*) *V*; *T* ⊆ *U*; *S* = *V* ∩ *T*⟧
⟹ *closedin* (*top_of_set T*) *S*
by (*metis* (*no_types*, *lifting*) *Int_assoc Int_commute closedin_closed inf.orderE*)

lemma *finite_imp_closedin*:
fixes *S* :: *'a*::*t1_space set*
shows ⟦*finite S*; *S* ⊆ *T*⟧ ⟹ *closedin* (*top_of_set T*) *S*
by (*simp add*: *finite_imp_closed closed_subset*)

lemma *closedin_singleton* [*simp*]:
fixes *a* :: *'a*::*t1_space*
shows *closedin* (*top_of_set U*) {*a*} ⟷ *a* ∈ *U*

using *closedin_subset* **by** (*force intro*: *closed_subset*)

lemma *openin_euclidean_subtopology_iff*:
fixes *S U* :: *'a*::*metric_space set*
shows *openin* (*top_of_set U*) *S* ⟷
S ⊆ *U* ∧ (∀ *x*∈*S*. ∃ *e*>*0*. ∀ *x'*∈*U*. *dist x' x* < *e* ⟶ *x'*∈ *S*)
(**is** *?lhs* ⟷ *?rhs*)
proof
assume *?lhs*
then show *?rhs*
unfolding *openin_open open_dist* **by** *blast*
next
define *T* **where** *T* = {*x*. ∃ *a*∈*S*. ∃ *d*>*0*. (∀ *y*∈*U*. *dist y a* < *d* ⟶ *y* ∈ *S*) ∧ *dist x a* < *d*}
have *1*: ∀ *x*∈*T*. ∃ *e*>*0*. ∀ *y*. *dist y x* < *e* ⟶ *y* ∈ *T*
unfolding *T_def*
apply *clarsimp*
apply (*rule_tac x*=*d* − *dist x a* **in** *exI*)
by (*metis add_0_left dist_commute dist_triangle_lt less_diff_eq*)
assume *?rhs* **then have** *2*: *S* = *U* ∩ *T*
unfolding *T_def*
by *auto* (*metis dist_self*)
from *1 2* **show** *?lhs*
unfolding *openin_open open_dist* **by** *fast*
qed

lemma *connected_openin*:
connected S ⟷
¬(∃ *E1 E2*. *openin* (*top_of_set S*) *E1* ∧
openin (*top_of_set S*) *E2* ∧
S ⊆ *E1* ∪ *E2* ∧ *E1* ∩ *E2* = {} ∧ *E1* ≠ {} ∧ *E2* ≠ {})
unfolding *connected_def openin_open disjoint_iff_not_equal* **by** *blast*

lemma *connected_openin_eq*:
connected S ⟷
¬(∃ *E1 E2*. *openin* (*top_of_set S*) *E1* ∧
openin (*top_of_set S*) *E2* ∧
E1 ∪ *E2* = *S* ∧ *E1* ∩ *E2* = {} ∧
E1 ≠ {} ∧ *E2* ≠ {})
unfolding *connected_openin*
by (*metis* (*no_types*, *lifting*) *Un_subset_iff openin_imp_subset subset_antisym*)

lemma *connected_closedin*:
connected S ⟷
(∄ *E1 E2*.
closedin (*top_of_set S*) *E1* ∧
closedin (*top_of_set S*) *E2* ∧
S ⊆ *E1* ∪ *E2* ∧ *E1* ∩ *E2* = {} ∧ *E1* ≠ {} ∧ *E2* ≠ {})
(**is** *?lhs* = *?rhs*)

proof
assume *?lhs*
then show *?rhs*
by (*auto simp add*: *connected_closed closedin_closed*)
next
assume *R*: *?rhs*
then show *?lhs*
proof (*clarsimp simp add*: *connected_closed closedin_closed*)
fix *A B*
assume *s_sub*: $S \subseteq A \cup B$ $B \cap S \neq \{\}$
and *disj*: $A \cap B \cap S = \{\}$
and *cl*: *closed A closed B*
have $S - A = B \cap S$
using *Diff_subset_conv Un_Diff_Int disj s_sub*(*1*) **by** *auto*
then show $A \cap S = \{\}$
by (*metis Int_Diff_Un Int_Diff_disjoint R cl closedin_closed_Int dual_order.refl inf_commute s_sub*(*2*))
qed
qed

lemma *connected_closedin_eq*:
connected S $\longleftrightarrow$
$\neg(\exists$ *E1 E2*.
closedin (*top_of_set S*) *E1* $\wedge$
closedin (*top_of_set S*) *E2* $\wedge$
E1 $\cup$ *E2* $= S \wedge$ *E1* $\cap$ *E2* $= \{\} \wedge$
E1 $\neq \{\} \wedge$ *E2* $\neq \{\}$)
unfolding *connected_closedin*
by (*metis Un_subset_iff closedin_imp_subset subset_antisym*)

These "transitivity" results are handy too

lemma *openin_trans*[*trans*]:
openin (*top_of_set T*) *S* $\Longrightarrow$ *openin* (*top_of_set U*) *T* $\Longrightarrow$
openin (*top_of_set U*) *S*
by (*metis openin_Int_open openin_open*)

lemma *openin_open_trans*: *openin* (*top_of_set T*) *S* $\Longrightarrow$ *open T* $\Longrightarrow$ *open S*
by (*auto simp*: *openin_open intro*: *openin_trans*)

lemma *closedin_trans*[*trans*]:
closedin (*top_of_set T*) *S* $\Longrightarrow$ *closedin* (*top_of_set U*) *T* $\Longrightarrow$
closedin (*top_of_set U*) *S*
by (*auto simp*: *closedin_closed closed_Inter Int_assoc*)

lemma *closedin_closed_trans*: *closedin* (*top_of_set T*) *S* $\Longrightarrow$ *closed T* $\Longrightarrow$ *closed S*
by (*auto simp*: *closedin_closed intro*: *closedin_trans*)

lemma *openin_subtopology_Int_subset*:

⟦*openin* (*top_of_set u*) (*u* ∩ *S*); *v* ⊆ *u*⟧ ⟹ *openin* (*top_of_set v*) (*v* ∩ *S*)
by (*auto simp*: *openin_subtopology*)

lemma *openin_open_eq*: *open s* ⟹ (*openin* (*top_of_set s*) *t* ⟷ *open t* ∧ *t* ⊆ *s*)
using *open_subset openin_open_trans openin_subset* **by** *fastforce*

1.11.6 Derived set (set of limit points)

definition *derived_set_of* :: *'a topology* ⇒ *'a set* ⇒ *'a set* (**infixl** *derived'_set'_of* 80)
where *X derived_set_of S* ≡
{*x* ∈ *topspace X*.
(∀ *T*. *x* ∈ *T* ∧ *openin X T* ⟶ (∃ *y*≠*x*. *y* ∈ *S* ∧ *y* ∈ *T*))}

lemma *derived_set_of_restrict* [*simp*]:
X derived_set_of (*topspace X* ∩ *S*) = *X derived_set_of S*
by (*simp add*: *derived_set_of_def*) (*metis openin_subset subset_iff*)

lemma *in_derived_set_of*:
x ∈ *X derived_set_of S* ⟷ *x* ∈ *topspace X* ∧ (∀ *T*. *x* ∈ *T* ∧ *openin X T* ⟶ (∃ *y*≠*x*. *y* ∈ *S* ∧ *y* ∈ *T*))
by (*simp add*: *derived_set_of_def*)

lemma *derived_set_of_subset_topspace*:
X derived_set_of S ⊆ *topspace X*
by (*auto simp add*: *derived_set_of_def*)

lemma *derived_set_of_subtopology*:
(*subtopology X U*) *derived_set_of S* = *U* ∩ (*X derived_set_of* (*U* ∩ *S*))
by (*simp add*: *derived_set_of_def openin_subtopology*) *blast*

lemma *derived_set_of_subset_subtopology*:
(*subtopology X S*) *derived_set_of T* ⊆ *S*
by (*simp add*: *derived_set_of_subtopology*)

lemma *derived_set_of_empty* [*simp*]: *X derived_set_of* {} = {}
by (*auto simp*: *derived_set_of_def*)

lemma *derived_set_of_mono*:
S ⊆ *T* ⟹ *X derived_set_of S* ⊆ *X derived_set_of T*
unfolding *derived_set_of_def* **by** *blast*

lemma *derived_set_of_Un*:
X derived_set_of (*S* ∪ *T*) = *X derived_set_of S* ∪ *X derived_set_of T* (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs*
by (*clarsimp simp*: *in_derived_set_of*) (*metis IntE IntI openin_Int*)

show *?rhs* ⊆ *?lhs*
by (*simp add*: *derived_set_of_mono*)
qed

lemma *derived_set_of_Union*:
finite $\mathcal{F}$ ⟹ *X derived_set_of* ($\bigcup \mathcal{F}$) = ($\bigcup S \in \mathcal{F}$. *X derived_set_of S*)
proof (*induction* $\mathcal{F}$ *rule*: *finite_induct*)
case (*insert S* $\mathcal{F}$)
then show *?case*
by (*simp add*: *derived_set_of_Un*)
qed *auto*

lemma *derived_set_of_topspace*:
X derived_set_of (*topspace X*) = {*x* ∈ *topspace X*. ¬ *openin X* {*x*}} (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs*
by (*auto simp*: *in_derived_set_of*)
show *?rhs* ⊆ *?lhs*
by (*clarsimp simp*: *in_derived_set_of*) (*metis openin_closedin_eq openin_subopen singletonD subset_eq*)
qed

lemma *discrete_topology_unique_derived_set*:
discrete_topology U = *X* ⟷ *topspace X* = *U* ∧ *X derived_set_of U* = {}
by (*auto simp*: *discrete_topology_unique derived_set_of_topspace*)

lemma *subtopology_eq_discrete_topology_eq*:
subtopology X U = *discrete_topology U* ⟷ *U* ⊆ *topspace X* ∧ *U* ∩ *X derived_set_of U* = {}
using *discrete_topology_unique_derived_set* [*of U subtopology X U*]
by (*auto simp*: *eq_commute derived_set_of_subtopology*)

lemma *subtopology_eq_discrete_topology*:
S ⊆ *topspace X* ∧ *S* ∩ *X derived_set_of S* = {}
⟹ *subtopology X S* = *discrete_topology S*
by (*simp add*: *subtopology_eq_discrete_topology_eq*)

lemma *subtopology_eq_discrete_topology_gen*:
assumes *S* ∩ *X derived_set_of S* = {}
shows *subtopology X S* = *discrete_topology*(*topspace X* ∩ *S*)
proof −
have *subtopology X S* = *subtopology X* (*topspace X* ∩ *S*)
by (*simp add*: *subtopology_restrict*)
then show *?thesis*
using *assms* **by** (*simp add*: *inf.assoc subtopology_eq_discrete_topology_eq*)
qed

lemma *subtopology_discrete_topology* [*simp*]:

subtopology (*discrete_topology U*) *S* = *discrete_topology*(*U* ∩ *S*)
proof −
have (λ*T*. ∃ *Sa*. *T* = *Sa* ∩ *S* ∧ *Sa* ⊆ *U*) = (λ*Sa*. *Sa* ⊆ *U* ∧ *Sa* ⊆ *S*)
by *force*
then show *?thesis*
by (*simp add*: *subtopology_def*) (*simp add*: *discrete_topology_def*)
qed

lemma *openin_Int_derived_set_of_subset*:
openin X S ⟹ *S* ∩ *X derived_set_of T* ⊆ *X derived_set_of* (*S* ∩ *T*)
by (*auto simp*: *derived_set_of_def*)

lemma *openin_Int_derived_set_of_eq*:
assumes *openin X S*
shows *S* ∩ *X derived_set_of T* = *S* ∩ *X derived_set_of* (*S* ∩ *T*) (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs*
by (*simp add*: *assms openin_Int_derived_set_of_subset*)
show *?rhs* ⊆ *?lhs*
by (*metis derived_set_of_mono inf_commute inf_le1 inf_mono order_refl*)
qed

1.11.7 Closure with respect to a topological space

definition *closure_of* :: *'a topology* ⇒ *'a set* ⇒ *'a set* (**infixr** *closure'_of 80*)
where *X closure_of S* ≡ {*x* ∈ *topspace X*. ∀ *T*. *x* ∈ *T* ∧ *openin X T* ⟶ (∃ *y* ∈ *S*. *y* ∈ *T*)}

lemma *closure_of_restrict*: *X closure_of S* = *X closure_of* (*topspace X* ∩ *S*)
unfolding *closure_of_def*
using *openin_subset* **by** *blast*

lemma *in_closure_of*:
x ∈ *X closure_of S* ⟷
x ∈ *topspace X* ∧ (∀ *T*. *x* ∈ *T* ∧ *openin X T* ⟶ (∃ *y*. *y* ∈ *S* ∧ *y* ∈ *T*))
by (*auto simp*: *closure_of_def*)

lemma *closure_of*: *X closure_of S* = *topspace X* ∩ (*S* ∪ *X derived_set_of S*)
by (*fastforce simp*: *in_closure_of in_derived_set_of*)

lemma *closure_of_alt*: *X closure_of S* = *topspace X* ∩ *S* ∪ *X derived_set_of S*
using *derived_set_of_subset_topspace* [*of X S*]
unfolding *closure_of_def in_derived_set_of*
by *safe* (*auto simp*: *in_derived_set_of*)

lemma *derived_set_of_subset_closure_of*:
X derived_set_of S ⊆ *X closure_of S*
by (*fastforce simp*: *closure_of_def in_derived_set_of*)

lemma *closure_of_subtopology*:
(*subtopology X U*) *closure_of S* = *U* ∩ (*X closure_of* (*U* ∩ *S*))
unfolding *closure_of_def topspace_subtopology openin_subtopology*
by *safe* (*metis* (*full_types*) *IntI Int_iff inf.commute*)+

lemma *closure_of_empty* [*simp*]: *X closure_of* {} = {}
by (*simp add*: *closure_of_alt*)

lemma *closure_of_topspace* [*simp*]: *X closure_of topspace X* = *topspace X*
by (*simp add*: *closure_of*)

lemma *closure_of_UNIV* [*simp*]: *X closure_of UNIV* = *topspace X*
by (*simp add*: *closure_of*)

lemma *closure_of_subset_topspace*: *X closure_of S* ⊆ *topspace X*
by (*simp add*: *closure_of*)

lemma *closure_of_subset_subtopology*: (*subtopology X S*) *closure_of T* ⊆ *S*
by (*simp add*: *closure_of_subtopology*)

lemma *closure_of_mono*: *S* ⊆ *T* ⟹ *X closure_of S* ⊆ *X closure_of T*
by (*fastforce simp add*: *closure_of_def*)

lemma *closure_of_subtopology_subset*:
(*subtopology X U*) *closure_of S* ⊆ (*X closure_of S*)
unfolding *closure_of_subtopology*
by *clarsimp* (*meson closure_of_mono contra_subsetD inf.cobounded2*)

lemma *closure_of_subtopology_mono*:
T ⊆ *U* ⟹ (*subtopology X T*) *closure_of S* ⊆ (*subtopology X U*) *closure_of S*
unfolding *closure_of_subtopology*
by *auto* (*meson closure_of_mono inf_mono subset_iff*)

lemma *closure_of_Un* [*simp*]: *X closure_of* (*S* ∪ *T*) = *X closure_of S* ∪ *X closure_of T*
by (*simp add*: *Un_assoc Un_left_commute closure_of_alt derived_set_of_Un inf_sup_distrib1*)

lemma *closure_of_Union*:
finite $\mathcal{F}$ ⟹ *X closure_of* (⋃$\mathcal{F}$) = (⋃*S* ∈ $\mathcal{F}$. *X closure_of S*)
by (*induction* $\mathcal{F}$ *rule*: *finite_induct*) *auto*

lemma *closure_of_subset*: *S* ⊆ *topspace X* ⟹ *S* ⊆ *X closure_of S*
by (*auto simp*: *closure_of_def*)

lemma *closure_of_subset_Int*: *topspace X* ∩ *S* ⊆ *X closure_of S*
by (*auto simp*: *closure_of_def*)

lemma *closure_of_subset_eq*: $S \subseteq$ *topspace* $X \wedge X$ *closure_of* $S \subseteq S \longleftrightarrow$ *closedin* X S
proof −
have *openin* X (*topspace* $X - S$)
if $\bigwedge x.\ [\![x \in$ *topspace* $X; \forall T.\ x \in T \wedge$ *openin* X $T \longrightarrow S \cap T \neq \{\}]\!] \Longrightarrow x \in S$
apply (*subst openin_subopen*)
by (*metis Diff_iff Diff_mono Diff_triv inf.commute openin_subset order_refl that*)
then show *?thesis*
by (*auto simp*: *closedin_def closure_of_def disjoint_iff_not_equal*)
qed

lemma *closure_of_eq*: X *closure_of* $S = S \longleftrightarrow$ *closedin* X S
by (*metis closure_of_subset closure_of_subset_eq closure_of_subset_topspace subset_antisym*)

lemma *closedin_contains_derived_set*:
closedin X $S \longleftrightarrow X$ *derived_set_of* $S \subseteq S \wedge S \subseteq$ *topspace* X
proof (*intro iffI conjI*)
show *closedin* X $S \Longrightarrow X$ *derived_set_of* $S \subseteq S$
using *closure_of_eq derived_set_of_subset_closure_of* **by** *fastforce*
show *closedin* X $S \Longrightarrow S \subseteq$ *topspace* X
using *closedin_subset* **by** *blast*
show X *derived_set_of* $S \subseteq S \wedge S \subseteq$ *topspace* $X \Longrightarrow$ *closedin* X S
by (*metis closure_of closure_of_eq inf.absorb_iff2 sup.orderE*)
qed

lemma *derived_set_subset_gen*:
X *derived_set_of* $S \subseteq S \longleftrightarrow$ *closedin* X (*topspace* $X \cap S$)
by (*simp add*: *closedin_contains_derived_set derived_set_of_subset_topspace*)

lemma *derived_set_subset*: $S \subseteq$ *topspace* $X \Longrightarrow (X$ *derived_set_of* $S \subseteq S \longleftrightarrow$ *closedin* X S)
by (*simp add*: *closedin_contains_derived_set*)

lemma *closedin_derived_set*:
closedin (*subtopology* X T) $S \longleftrightarrow$
$S \subseteq$ *topspace* $X \wedge S \subseteq T \wedge (\forall x.\ x \in X$ *derived_set_of* $S \wedge x \in T \longrightarrow x \in S$)
by (*auto simp*: *closedin_contains_derived_set derived_set_of_subtopology Int_absorb1*)

lemma *closedin_Int_closure_of*:
closedin (*subtopology* X S) $T \longleftrightarrow S \cap X$ *closure_of* $T = T$
by (*metis Int_left_absorb closure_of_eq closure_of_subtopology*)

lemma *closure_of_closedin*: *closedin* X $S \Longrightarrow X$ *closure_of* $S = S$
by (*simp add*: *closure_of_eq*)

lemma *closure_of_eq_diff*: X *closure_of* $S =$ *topspace* $X - \bigcup \{T.$ *openin* X $T \wedge$

disjnt S T}
by (*auto simp*: *closure_of_def disjnt_iff*)

lemma *closedin_closure_of* [*simp*]: *closedin X* (*X closure_of S*)
unfolding *closure_of_eq_diff* **by** *blast*

lemma *closure_of_closure_of* [*simp*]: *X closure_of* (*X closure_of S*) = *X closure_of S*
by (*simp add*: *closure_of_eq*)

lemma *closure_of_hull*:
assumes *S* ⊆ *topspace X* **shows** *X closure_of S* = (*closedin X*) *hull S*
by (*metis assms closedin_closure_of closure_of_eq closure_of_mono closure_of_subset hull_unique*)

lemma *closure_of_minimal*:
⟦*S* ⊆ *T*; *closedin X T*⟧ ⟹ (*X closure_of S*) ⊆ *T*
by (*metis closure_of_eq closure_of_mono*)

lemma *closure_of_minimal_eq*:
⟦*S* ⊆ *topspace X*; *closedin X T*⟧ ⟹ (*X closure_of S*) ⊆ *T* ⟷ *S* ⊆ *T*
by (*meson closure_of_minimal closure_of_subset subset_trans*)

lemma *closure_of_unique*:
⟦*S* ⊆ *T*; *closedin X T*;
⋀*T'*. ⟦*S* ⊆ *T'*; *closedin X T'*⟧ ⟹ *T* ⊆ *T'*⟧
⟹ *X closure_of S* = *T*
by (*meson closedin_closure_of closedin_subset closure_of_minimal closure_of_subset eq_iff order.trans*)

lemma *closure_of_eq_empty_gen*: *X closure_of S* = {} ⟷ *disjnt* (*topspace X*) *S*
unfolding *disjnt_def closure_of_restrict* [**where** *S*=*S*]
using *closure_of* **by** *fastforce*

lemma *closure_of_eq_empty*: *S* ⊆ *topspace X* ⟹ *X closure_of S* = {} ⟷ *S* = {}
using *closure_of_subset* **by** *fastforce*

lemma *openin_Int_closure_of_subset*:
assumes *openin X S*
shows *S* ∩ *X closure_of T* ⊆ *X closure_of* (*S* ∩ *T*)
proof −
have *S* ∩ *X derived_set_of T* = *S* ∩ *X derived_set_of* (*S* ∩ *T*)
by (*meson assms openin_Int_derived_set_of_eq*)
moreover have *S* ∩ (*S* ∩ *T*) = *S* ∩ *T*
by *fastforce*
ultimately show *?thesis*
by (*metis closure_of_alt inf.cobounded2 inf_left_commute inf_sup_distrib1*)

qed

lemma *closure_of_openin_Int_closure_of*:
assumes *openin X S*
shows *X closure_of* (*S* ∩ *X closure_of T*) = *X closure_of* (*S* ∩ *T*)
proof
show *X closure_of* (*S* ∩ *X closure_of T*) ⊆ *X closure_of* (*S* ∩ *T*)
by (*simp add*: *assms closure_of_minimal openin_Int_closure_of_subset*)
next
show *X closure_of* (*S* ∩ *T*) ⊆ *X closure_of* (*S* ∩ *X closure_of T*)
by (*metis Int_subset_iff assms closure_of_alt closure_of_mono inf_mono openin_subset subset_refl sup.coboundedI1*)
qed

lemma *openin_Int_closure_of_eq*:
assumes *openin X S* **shows** *S* ∩ *X closure_of T* = *S* ∩ *X closure_of* (*S* ∩ *T*) (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs*
by (*simp add*: *assms openin_Int_closure_of_subset*)
show *?rhs* ⊆ *?lhs*
by (*metis closure_of_mono inf_commute inf_le1 inf_mono order_refl*)
qed

lemma *openin_Int_closure_of_eq_empty*:
assumes *openin X S* **shows** *S* ∩ *X closure_of T* = {} ⟷ *S* ∩ *T* = {} (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⟹ *?rhs*
unfolding *disjoint_iff*
by (*meson assms in_closure_of in_mono openin_subset*)
show *?rhs* ⟹ *?lhs*
by (*simp add*: *assms openin_Int_closure_of_eq*)
qed

lemma *closure_of_openin_Int_superset*:
openin X S ∧ *S* ⊆ *X closure_of T*
⟹ *X closure_of* (*S* ∩ *T*) = *X closure_of S*
by (*metis closure_of_openin_Int_closure_of inf.orderE*)

lemma *closure_of_openin_subtopology_Int_closure_of*:
assumes *S*: *openin* (*subtopology X U*) *S* **and** *T* ⊆ *U*
shows *X closure_of* (*S* ∩ *X closure_of T*) = *X closure_of* (*S* ∩ *T*) (**is** *?lhs* = *?rhs*)
proof
obtain *S0* **where** *S0*: *openin X S0 S* = *S0* ∩ *U*
using *assms* **by** (*auto simp*: *openin_subtopology*)
then show *?lhs* ⊆ *?rhs*
proof −

```
    have S0 ∩ X closure_of T = S0 ∩ X closure_of (S0 ∩ T)
      by (meson S0(1) openin_Int_closure_of_eq)
    moreover have S0 ∩ T = S0 ∩ U ∩ T
      using ‹T ⊆ U› by fastforce
    ultimately have S ∩ X closure_of T ⊆ X closure_of (S ∩ T)
      using S0(2) by auto
    then show ?thesis
      by (meson closedin_closure_of closure_of_minimal)
  qed
next
  show ?rhs ⊆ ?lhs
  proof −
    have T ∩ S ⊆ T ∪ X derived_set_of T
      by force
    then show ?thesis
      by (smt (verit, del_insts) Int_iff in_closure_of inf.orderE openin_subset
subsetI)
  qed
qed

lemma closure_of_subtopology_open:
    openin X U ∨ S ⊆ U ⟹ (subtopology X U) closure_of S = U ∩ X closure_of
S
  by (metis closure_of_subtopology inf_absorb2 openin_Int_closure_of_eq)

lemma discrete_topology_closure_of:
    (discrete_topology U) closure_of S = U ∩ S
  by (metis closedin_discrete_topology closure_of_restrict closure_of_unique dis-
crete_topology_unique inf_sup_ord(1) order_refl)
```

Interior with respect to a topological space.

```
definition interior_of :: 'a topology ⇒ 'a set ⇒ 'a set (infixr interior'_of 80)
  where X interior_of S ≡ {x. ∃ T. openin X T ∧ x ∈ T ∧ T ⊆ S}

lemma interior_of_restrict:
   X interior_of S = X interior_of (topspace X ∩ S)
  using openin_subset by (auto simp: interior_of_def)

lemma interior_of_eq: (X interior_of S = S) ⟷ openin X S
  unfolding interior_of_def  using openin_subopen by blast

lemma interior_of_openin: openin X S ⟹ X interior_of S = S
  by (simp add: interior_of_eq)

lemma interior_of_empty [simp]: X interior_of {} = {}
  by (simp add: interior_of_eq)

lemma interior_of_topspace [simp]: X interior_of (topspace X) = topspace X
  by (simp add: interior_of_eq)
```

lemma *openin_interior_of* [*simp*]: *openin X* (*X interior_of S*)
unfolding *interior_of_def*
using *openin_subopen* **by** *fastforce*

lemma *interior_of_interior_of* [*simp*]:
X interior_of X interior_of S = *X interior_of S*
by (*simp add*: *interior_of_eq*)

lemma *interior_of_subset*: *X interior_of S* ⊆ *S*
by (*auto simp*: *interior_of_def*)

lemma *interior_of_subset_closure_of*: *X interior_of S* ⊆ *X closure_of S*
by (*metis closure_of_subset_Int dual_order.trans interior_of_restrict interior_of_subset*)

lemma *subset_interior_of_eq*: *S* ⊆ *X interior_of S* ⟷ *openin X S*
by (*metis interior_of_eq interior_of_subset subset_antisym*)

lemma *interior_of_mono*: *S* ⊆ *T* ⟹ *X interior_of S* ⊆ *X interior_of T*
by (*auto simp*: *interior_of_def*)

lemma *interior_of_maximal*: ⟦*T* ⊆ *S*; *openin X T*⟧ ⟹ *T* ⊆ *X interior_of S*
by (*auto simp*: *interior_of_def*)

lemma *interior_of_maximal_eq*: *openin X T* ⟹ *T* ⊆ *X interior_of S* ⟷ *T* ⊆ *S*
by (*meson interior_of_maximal interior_of_subset order_trans*)

lemma *interior_of_unique*:
⟦*T* ⊆ *S*; *openin X T*; ⋀*T'*. ⟦*T'* ⊆ *S*; *openin X T'*⟧ ⟹ *T'* ⊆ *T*⟧ ⟹ *X interior_of S* = *T*
by (*simp add*: *interior_of_maximal_eq interior_of_subset subset_antisym*)

lemma *interior_of_subset_topspace*: *X interior_of S* ⊆ *topspace X*
by (*simp add*: *openin_subset*)

lemma *interior_of_subset_subtopology*: (*subtopology X S*) *interior_of T* ⊆ *S*
by (*meson openin_imp_subset openin_interior_of*)

lemma *interior_of_Int*: *X interior_of* (*S* ∩ *T*) = *X interior_of S* ∩ *X interior_of T* (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs*
by (*simp add*: *interior_of_mono*)
show *?rhs* ⊆ *?lhs*
by (*meson inf_mono interior_of_maximal interior_of_subset openin_Int openin_interior_of*)
qed

lemma *interior_of_Inter_subset*: X *interior_of* $(\bigcap \mathcal{F}) \subseteq (\bigcap S \in \mathcal{F}.\ X$ *interior_of* $S)$
by (*simp add*: *INT_greatest Inf_lower interior_of_mono*)

lemma *union_interior_of_subset*:
X *interior_of* $S \cup X$ *interior_of* $T \subseteq X$ *interior_of* $(S \cup T)$
by (*simp add*: *interior_of_mono*)

lemma *interior_of_eq_empty*:
X *interior_of* $S = \{\} \longleftrightarrow (\forall T.\ openin\ X\ T \wedge T \subseteq S \longrightarrow T = \{\})$
by (*metis bot.extremum_uniqueI interior_of_maximal interior_of_subset openin_interior_of*)

lemma *interior_of_eq_empty_alt*:
X *interior_of* $S = \{\} \longleftrightarrow (\forall T.\ openin\ X\ T \wedge T \neq \{\} \longrightarrow T - S \neq \{\})$
by (*auto simp*: *interior_of_eq_empty*)

lemma *interior_of_Union_openin_subsets*:
$\bigcup \{T.\ openin\ X\ T \wedge T \subseteq S\} = X$ *interior_of* S
by (*rule interior_of_unique* [*symmetric*]) *auto*

lemma *interior_of_complement*:
X *interior_of* $(topspace\ X - S) = topspace\ X - X$ *closure_of* S
by (*auto simp*: *interior_of_def closure_of_def*)

lemma *interior_of_closure_of*:
X *interior_of* $S = topspace\ X - X$ *closure_of* $(topspace\ X - S)$
unfolding *interior_of_complement* [*symmetric*]
by (*metis Diff_Diff_Int interior_of_restrict*)

lemma *closure_of_interior_of*:
X *closure_of* $S = topspace\ X - X$ *interior_of* $(topspace\ X - S)$
by (*simp add*: *interior_of_complement Diff_Diff_Int closure_of*)

lemma *closure_of_complement*: X *closure_of* $(topspace\ X - S) = topspace\ X - X$ *interior_of* S
unfolding *interior_of_def closure_of_def*
by (*blast dest*: *openin_subset*)

lemma *interior_of_eq_empty_complement*:
X *interior_of* $S = \{\} \longleftrightarrow X$ *closure_of* $(topspace\ X - S) = topspace\ X$
using *interior_of_subset_topspace* [*of X S*] *closure_of_complement* **by** *fastforce*

lemma *closure_of_eq_topspace*:
X *closure_of* $S = topspace\ X \longleftrightarrow X$ *interior_of* $(topspace\ X - S) = \{\}$
using *closure_of_subset_topspace* [*of X S*] *interior_of_complement* **by** *fastforce*

lemma *interior_of_subtopology_subset*:
$U \cap X$ *interior_of* $S \subseteq (subtopology\ X\ U)$ *interior_of* S
by (*auto simp*: *interior_of_def openin_subtopology*)

lemma *interior_of_subtopology_subsets*:
$T \subseteq U \Longrightarrow T \cap (\mathit{subtopology}\ X\ U)\ \mathit{interior\_of}\ S \subseteq (\mathit{subtopology}\ X\ T)\ \mathit{interior\_of}\ S$
by (*metis inf.absorb_iff2 interior_of_subtopology_subset subtopology_subtopology*)

lemma *interior_of_subtopology_mono*:
$[\![S \subseteq T;\ T \subseteq U]\!] \Longrightarrow (\mathit{subtopology}\ X\ U)\ \mathit{interior\_of}\ S \subseteq (\mathit{subtopology}\ X\ T)\ \mathit{interior\_of}\ S$
by (*metis dual_order.trans inf.orderE inf_commute interior_of_subset interior_of_subtopology_subsets*)

lemma *interior_of_subtopology_open*:
assumes *openin X U*
shows $(\mathit{subtopology}\ X\ U)\ \mathit{interior\_of}\ S = U \cap X\ \mathit{interior\_of}\ S$ (**is** *?lhs = ?rhs*)
proof
show *?lhs ⊆ ?rhs*
by (*meson assms interior_of_maximal interior_of_subset le_infI openin_interior_of openin_open_subtopology*)
show *?rhs ⊆ ?lhs*
by (*simp add*: *interior_of_subtopology_subset*)
qed

lemma *dense_intersects_open*:
$X\ \mathit{closure\_of}\ S = \mathit{topspace}\ X \longleftrightarrow (\forall T.\ \mathit{openin}\ X\ T \wedge T \neq \{\} \longrightarrow S \cap T \neq \{\})$
proof −
have $X\ \mathit{closure\_of}\ S = \mathit{topspace}\ X \longleftrightarrow (\mathit{topspace}\ X - X\ \mathit{interior\_of}\ (\mathit{topspace}\ X - S) = \mathit{topspace}\ X)$
by (*simp add*: *closure_of_interior_of*)
also have $\ldots \longleftrightarrow X\ \mathit{interior\_of}\ (\mathit{topspace}\ X - S) = \{\}$
by (*simp add*: *closure_of_complement interior_of_eq_empty_complement*)
also have $\ldots \longleftrightarrow (\forall T.\ \mathit{openin}\ X\ T \wedge T \neq \{\} \longrightarrow S \cap T \neq \{\})$
unfolding *interior_of_eq_empty_alt*
using *openin_subset* **by** *fastforce*
finally show *?thesis* **.**
qed

lemma *interior_of_closedin_union_empty_interior_of*:
assumes *closedin X S* **and** *disj*: $X\ \mathit{interior\_of}\ T = \{\}$
shows $X\ \mathit{interior\_of}\ (S \cup T) = X\ \mathit{interior\_of}\ S$
proof −
have $X\ \mathit{closure\_of}\ (\mathit{topspace}\ X - T) = \mathit{topspace}\ X$
by (*metis Diff_Diff_Int disj closure_of_eq_topspace closure_of_restrict interior_of_closure_of*)
then show *?thesis*
unfolding *interior_of_closure_of*
by (*metis Diff_Un Diff_subset assms(1) closedin_def closure_of_openin_Int_superset*)
qed

lemma *interior_of_union_eq_empty*:
closedin X S
$\Longrightarrow$ (*X interior_of* (*S* $\cup$ *T*) = {} $\longleftrightarrow$
X interior_of S = {} $\wedge$ *X interior_of T* = {})
by (*metis interior_of_closedin_union_empty_interior_of le_sup_iff subset_empty union_interior_of_subset*)

lemma *discrete_topology_interior_of* [*simp*]:
(*discrete_topology U*) *interior_of S* = *U* $\cap$ *S*
by (*simp add*: *interior_of_restrict* [*of* _ *S*] *interior_of_eq*)

1.11.8 Frontier with respect to topological space

definition *frontier_of* :: *'a topology* $\Rightarrow$ *'a set* $\Rightarrow$ *'a set* (**infixr** *frontier'_of 80*)
where *X frontier_of S* $\equiv$ *X closure_of S* $-$ *X interior_of S*

lemma *frontier_of_closures*:
X frontier_of S = *X closure_of S* $\cap$ *X closure_of* (*topspace X* $-$ *S*)
by (*metis Diff_Diff_Int closure_of_complement closure_of_subset_topspace double_diff frontier_of_def interior_of_subset_closure_of*)

lemma *interior_of_union_frontier_of* [*simp*]:
X interior_of S $\cup$ *X frontier_of S* = *X closure_of S*
by (*simp add*: *frontier_of_def interior_of_subset_closure_of subset_antisym*)

lemma *frontier_of_restrict*: *X frontier_of S* = *X frontier_of* (*topspace X* $\cap$ *S*)
by (*metis closure_of_restrict frontier_of_def interior_of_restrict*)

lemma *closedin_frontier_of*: *closedin X* (*X frontier_of S*)
by (*simp add*: *closedin_Int frontier_of_closures*)

lemma *frontier_of_subset_topspace*: *X frontier_of S* $\subseteq$ *topspace X*
by (*simp add*: *closedin_frontier_of closedin_subset*)

lemma *frontier_of_subset_subtopology*: (*subtopology X S*) *frontier_of T* $\subseteq$ *S*
by (*metis* (*no_types*) *closedin_derived_set closedin_frontier_of*)

lemma *frontier_of_subtopology_subset*:
U $\cap$ (*subtopology X U*) *frontier_of S* $\subseteq$ (*X frontier_of S*)
proof $-$
have *U* $\cap$ *X interior_of S* $-$ *subtopology X U interior_of S* = {}
by (*simp add*: *interior_of_subtopology_subset*)
moreover have *X closure_of S* $\cap$ *subtopology X U closure_of S* = *subtopology X U closure_of S*
by (*meson closure_of_subtopology_subset inf.absorb_iff2*)
ultimately show *?thesis*
unfolding *frontier_of_def*
by *blast*

qed

lemma *frontier_of_subtopology_mono*:
  $[\![S \subseteq T;\ T \subseteq U]\!] \Longrightarrow$ (*subtopology X T*) *frontier_of S* $\subseteq$ (*subtopology X U*) *frontier_of S*
  by (*simp add*: *frontier_of_def Diff_mono closure_of_subtopology_mono interior_of_subtopology_mono*)

lemma *clopenin_eq_frontier_of*:
  closedin X S $\wedge$ *openin X S* $\longleftrightarrow$ *S* $\subseteq$ *topspace X* $\wedge$ *X frontier_of S* = {}
proof (*cases S* $\subseteq$ *topspace X*)
 case *True*
 then show *?thesis*
  by (*metis Diff_eq_empty_iff closure_of_eq closure_of_subset_eq frontier_of_def interior_of_eq interior_of_subset interior_of_union_frontier_of sup_bot_right*)
next
 case *False*
 then show *?thesis*
  by (*simp add*: *frontier_of_closures openin_closedin_eq*)
qed

lemma *frontier_of_eq_empty*:
  S $\subseteq$ *topspace X* $\Longrightarrow$ (*X frontier_of S* = {} $\longleftrightarrow$ *closedin X S* $\wedge$ *openin X S*)
 by (*simp add*: *clopenin_eq_frontier_of*)

lemma *frontier_of_openin*:
  openin X S $\Longrightarrow$ *X frontier_of S* = *X closure_of S* $-$ *S*
 by (*metis* (*no_types*) *frontier_of_def interior_of_eq*)

lemma *frontier_of_openin_straddle_Int*:
 assumes *openin X U U* $\cap$ *X frontier_of S* $\neq$ {}
 shows *U* $\cap$ *S* $\neq$ {} *U* $-$ *S* $\neq$ {}
proof $-$
 have *U* $\cap$ (*X closure_of S* $\cap$ *X closure_of* (*topspace X* $-$ *S*)) $\neq$ {}
  using *assms* **by** (*simp add*: *frontier_of_closures*)
 then show *U* $\cap$ *S* $\neq$ {}
  using *assms openin_Int_closure_of_eq_empty* **by** *fastforce*
 show *U* $-$ *S* $\neq$ {}
 proof $-$
  have $\exists$ *A*. *X closure_of* (*A* $-$ *S*) $\cap$ *U* $\neq$ {}
   using ‹*U* $\cap$ (*X closure_of S* $\cap$ *X closure_of* (*topspace X* $-$ *S*)) $\neq$ {}› **by** *blast*
  then have $\neg$ *U* $\subseteq$ *S*
   by (*metis Diff_disjoint Diff_eq_empty_iff Int_Diff assms*(*1*) *inf_commute openin_Int_closure_of_eq_empty*)
  then show *?thesis*
   by *blast*
 qed
qed

lemma *frontier_of_subset_closedin*: *closedin X S* $\Longrightarrow$ (*X frontier_of S*) $\subseteq$ *S*
using *closure_of_eq frontier_of_def* **by** *fastforce*

lemma *frontier_of_empty* [*simp*]: *X frontier_of* {} = {}
by (*simp add*: *frontier_of_def*)

lemma *frontier_of_topspace* [*simp*]: *X frontier_of topspace X* = {}
by (*simp add*: *frontier_of_def*)

lemma *frontier_of_subset_eq*:
assumes *S* $\subseteq$ *topspace X*
shows (*X frontier_of S*) $\subseteq$ *S* $\longleftrightarrow$ *closedin X S*
proof
show *X frontier_of S* $\subseteq$ *S* $\Longrightarrow$ *closedin X S*
by (*metis assms closure_of_subset_eq interior_of_subset interior_of_union_frontier_of le_sup_iff*)
show *closedin X S* $\Longrightarrow$ *X frontier_of S* $\subseteq$ *S*
by (*simp add*: *frontier_of_subset_closedin*)
qed

lemma *frontier_of_complement*: *X frontier_of* (*topspace X* − *S*) = *X frontier_of S*
by (*metis Diff_Diff_Int closure_of_restrict frontier_of_closures inf_commute*)

lemma *frontier_of_disjoint_eq*:
assumes *S* $\subseteq$ *topspace X*
shows ((*X frontier_of S*) $\cap$ *S* = {} $\longleftrightarrow$ *openin X S*)
proof
assume *X frontier_of S* $\cap$ *S* = {}
then have *closedin X* (*topspace X* − *S*)
using *assms closure_of_subset frontier_of_def interior_of_eq interior_of_subset*
by *fastforce*
then show *openin X S*
using *assms* **by** (*simp add*: *openin_closedin*)
next
show *openin X S* $\Longrightarrow$ *X frontier_of S* $\cap$ *S* = {}
by (*simp add*: *Diff_Diff_Int closedin_def frontier_of_openin inf.absorb_iff2 inf_commute*)
qed

lemma *frontier_of_disjoint_eq_alt*:
S $\subseteq$ (*topspace X* − *X frontier_of S*) $\longleftrightarrow$ *openin X S*
proof (*cases S* $\subseteq$ *topspace X*)
case *True*
show *?thesis*
using *True frontier_of_disjoint_eq* **by** *auto*
next
case *False*

then show *?thesis*
by (*meson Diff_subset openin_subset subset_trans*)
qed

lemma *frontier_of_Int*:
X frontier_of (*S* ∩ *T*) =
X closure_of (*S* ∩ *T*) ∩ (*X frontier_of S* ∪ *X frontier_of T*)
proof −
have ∗: *U* ⊆ *S* ∧ *U* ⊆ *T* ⟹ *U* ∩ (*S* ∩ *A* ∪ *T* ∩ *B*) = *U* ∩ (*A* ∪ *B*) **for** *U S T A B* :: *'a set*
by *blast*
show *?thesis*
by (*simp add*: *frontier_of_closures closure_of_mono Diff_Int* ∗ *flip*: *closure_of_Un*)
qed

lemma *frontier_of_Int_subset*: *X frontier_of* (*S* ∩ *T*) ⊆ *X frontier_of S* ∪ *X frontier_of T*
by (*simp add*: *frontier_of_Int*)

lemma *frontier_of_Int_closedin*:
assumes *closedin X S closedin X T*
shows *X frontier_of*(*S* ∩ *T*) = *X frontier_of S* ∩ *T* ∪ *S* ∩ *X frontier_of T* (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs*
using *assms* **by** (*force simp add*: *frontier_of_Int closedin_Int closure_of_closedin*)
show *?rhs* ⊆ *?lhs*
using *assms frontier_of_subset_closedin*
by (*auto simp add*: *frontier_of_Int closedin_Int closure_of_closedin*)
qed

lemma *frontier_of_Un_subset*: *X frontier_of*(*S* ∪ *T*) ⊆ *X frontier_of S* ∪ *X frontier_of T*
by (*metis Diff_Un frontier_of_Int_subset frontier_of_complement*)

lemma *frontier_of_Union_subset*:
finite $\mathcal{F}$ ⟹ *X frontier_of* (⋃$\mathcal{F}$) ⊆ (⋃ *T* ∈ $\mathcal{F}$. *X frontier_of T*)
proof (*induction* $\mathcal{F}$ *rule*: *finite_induct*)
case (*insert A* $\mathcal{F}$)
then show *?case*
using *frontier_of_Un_subset* **by** *fastforce*
qed *simp*

lemma *frontier_of_frontier_of_subset*:
X frontier_of (*X frontier_of S*) ⊆ *X frontier_of S*
by (*simp add*: *closedin_frontier_of frontier_of_subset_closedin*)

lemma *frontier_of_subtopology_open*:

openin X U $\Longrightarrow$ (*subtopology X U*) *frontier_of S* = *U* $\cap$ *X frontier_of S*
by (*simp add*: *Diff_Int_distrib closure_of_subtopology_open frontier_of_def interior_of_subtopology_open*)

lemma *discrete_topology_frontier_of* [*simp*]:
(*discrete_topology U*) *frontier_of S* = {}
by (*simp add*: *Diff_eq discrete_topology_closure_of frontier_of_closures*)

lemma *openin_subset_topspace_eq*:
assumes *disjnt S* (*X frontier_of U*)
shows *openin* (*subtopology X U*) *S* $\longleftrightarrow$ *openin X S* $\wedge$ *S* $\subseteq$ *U*
proof
assume *S*: *openin* (*subtopology X U*) *S*
show *openin X S* $\wedge$ *S* $\subseteq$ *U*
proof
show *S* $\subseteq$ *U*
using *S openin_imp_subset* **by** *blast*
have *disjnt S* (*X frontier_of* (*topspace X* $\cap$ *U*))
by (*metis assms frontier_of_restrict*)
moreover
have *openin* (*subtopology X* (*topspace X* $\cap$ *U*)) *S*
by (*simp add*: *S subtopology_restrict*)
moreover
have *openin X S*
if *dis*: *disjnt S* (*X frontier_of U*) **and** *ope*: *openin* (*subtopology X U*) *S* **and** *U* $\subseteq$ *topspace X*
for *S U*
proof −
obtain *T* **where** *T*: *openin X T S* = *T* $\cap$ *U*
using *ope* **by** (*auto simp*: *openin_subtopology*)
have *T* $\cap$ *U* = *T* $\cap$ *X interior_of U*
using *that T interior_of_subset in_closure_of* **by** (*fastforce simp*: *disjnt_iff frontier_of_def*)
then show *?thesis*
using ‹*S* = *T* $\cap$ *U*› ‹*openin X T*› **by** *auto*
qed
ultimately show *openin X S*
by *blast*
qed
qed (*metis inf.absorb_iff1 openin_subtopology_Int*)

1.11.9 Locally finite collections

definition *locally_finite_in*
where
locally_finite_in X $\mathcal{A}$ $\longleftrightarrow$
($\bigcup \mathcal{A}$ $\subseteq$ *topspace X*) $\wedge$
($\forall$ *x* $\in$ *topspace X*. $\exists$ *V*. *openin X V* $\wedge$ *x* $\in$ *V* $\wedge$ *finite* {*U* $\in$ $\mathcal{A}$. *U* $\cap$ *V* $\neq$ {}})

lemma *finite_imp_locally_finite_in*:
  ⟦*finite* $\mathcal{A}$; $\bigcup\mathcal{A} \subseteq$ *topspace* X⟧ ⟹ *locally_finite_in* X $\mathcal{A}$
  by (*auto simp*: *locally_finite_in_def*)

lemma *locally_finite_in_subset*:
  assumes *locally_finite_in* X $\mathcal{A}$ $\mathcal{B} \subseteq \mathcal{A}$
  shows *locally_finite_in* X $\mathcal{B}$
proof −
  have *finite* ($\mathcal{A} \cap \{U.\ U \cap V \neq \{\}\}$) ⟹ *finite* ($\mathcal{B} \cap \{U.\ U \cap V \neq \{\}\}$) **for** V
    by (*meson* ‹$\mathcal{B} \subseteq \mathcal{A}$› *finite_subset inf_le1 inf_le2 le_inf_iff subset_trans*)
  then show *?thesis*
    using *assms* **unfolding** *locally_finite_in_def Int_def* **by** *fastforce*
qed

lemma *locally_finite_in_refinement*:
  assumes $\mathcal{A}$: *locally_finite_in* X $\mathcal{A}$ **and** f: $\bigwedge S.\ S \in \mathcal{A} \Longrightarrow f\ S \subseteq S$
  shows *locally_finite_in* X (f ‘ $\mathcal{A}$)
proof −
  show *?thesis*
    unfolding *locally_finite_in_def*
  proof *safe*
    fix x
    assume $x \in$ *topspace* X
    then obtain V **where** *openin* X V $x \in V$ *finite* $\{U \in \mathcal{A}.\ U \cap V \neq \{\}\}$
      using $\mathcal{A}$ **unfolding** *locally_finite_in_def* **by** *blast*
    moreover have $\{U \in \mathcal{A}.\ f\ U \cap V \neq \{\}\} \subseteq \{U \in \mathcal{A}.\ U \cap V \neq \{\}\}$ **for** V
      using f **by** *blast*
    ultimately have *finite* $\{U \in \mathcal{A}.\ f\ U \cap V \neq \{\}\}$
      using *finite_subset* **by** *blast*
    moreover have f ‘ $\{U \in \mathcal{A}.\ f\ U \cap V \neq \{\}\} = \{U \in f$ ‘ $\mathcal{A}.\ U \cap V \neq \{\}\}$
      by *blast*
    ultimately have *finite* $\{U \in f$ ‘ $\mathcal{A}.\ U \cap V \neq \{\}\}$
      by (*metis* (*no_types*, *lifting*) *finite_imageI*)
    then show ∃ V. *openin* X $V \wedge x \in V \wedge$ *finite* $\{U \in f$ ‘ $\mathcal{A}.\ U \cap V \neq \{\}\}$
      using ‹*openin* X V› ‹$x \in V$› **by** *blast*
  next
    show $\bigwedge x\ xa.$ ⟦$xa \in \mathcal{A}$; $x \in f\ xa$⟧ ⟹ $x \in$ *topspace* X
      by (*meson Sup_upper* $\mathcal{A}$ f *locally_finite_in_def subset_iff*)
  qed
qed

lemma *locally_finite_in_subtopology*:
  assumes $\mathcal{A}$: *locally_finite_in* X $\mathcal{A}$ $\bigcup\mathcal{A} \subseteq S$
  shows *locally_finite_in* (*subtopology* X S) $\mathcal{A}$
  unfolding *locally_finite_in_def*
proof *safe*
  fix x
  assume x: $x \in$ *topspace* (*subtopology* X S)

then obtain V **where** *openin* X V $x \in V$ **and** *fin*: *finite* $\{U \in \mathcal{A}.\ U \cap V \neq \{\}\}$
using $\mathcal{A}$ **unfolding** *locally_finite_in_def topspace_subtopology* **by** *blast*
show $\exists V.$ *openin* (*subtopology* X S) $V \wedge x \in V \wedge$ *finite* $\{U \in \mathcal{A}.\ U \cap V \neq \{\}\}$
proof (*intro exI conjI*)
show *openin* (*subtopology* X S) $(S \cap V)$
by (*simp add*: ‹*openin* X V› *openin_subtopology_Int2*)
have $\{U \in \mathcal{A}.\ U \cap (S \cap V) \neq \{\}\} \subseteq \{U \in \mathcal{A}.\ U \cap V \neq \{\}\}$
by *auto*
with *fin* **show** *finite* $\{U \in \mathcal{A}.\ U \cap (S \cap V) \neq \{\}\}$
using *finite_subset* **by** *auto*
show $x \in S \cap V$
using x ‹$x \in V$› **by** (*simp*)
qed
next
show $\bigwedge x\ A.\ [\![x \in A;\ A \in \mathcal{A}]\!] \Longrightarrow x \in$ *topspace* (*subtopology* X S)
using *assms* **unfolding** *locally_finite_in_def topspace_subtopology* **by** *blast*
qed

lemma *closedin_locally_finite_Union*:
assumes *clo*: $\bigwedge S.\ S \in \mathcal{A} \Longrightarrow$ *closedin* X S **and** $\mathcal{A}$: *locally_finite_in* X $\mathcal{A}$
shows *closedin* X $(\bigcup \mathcal{A})$
using $\mathcal{A}$ **unfolding** *locally_finite_in_def closedin_def*
proof *clarify*
show *openin* X (*topspace* $X - \bigcup \mathcal{A}$)
proof (*subst openin_subopen*, *clarify*)
fix x
assume $x \in$ *topspace* X **and** $x \notin \bigcup \mathcal{A}$
then obtain V **where** *openin* X V $x \in V$ **and** *fin*: *finite* $\{U \in \mathcal{A}.\ U \cap V \neq \{\}\}$
using $\mathcal{A}$ **unfolding** *locally_finite_in_def* **by** *blast*
let *?T* $= V - \bigcup \{S \in \mathcal{A}.\ S \cap V \neq \{\}\}$
show $\exists T.$ *openin* X $T \wedge x \in T \wedge T \subseteq$ *topspace* $X - \bigcup \mathcal{A}$
proof (*intro exI conjI*)
show *openin* X *?T*
by (*metis* (*no_types*, *lifting*) *fin* ‹*openin* X V› *clo closedin_Union mem_Collect_eq openin_diff*)
show $x \in$ *?T*
using ‹$x \notin \bigcup \mathcal{A}$› ‹$x \in V$› **by** *auto*
show *?T* $\subseteq$ *topspace* $X - \bigcup \mathcal{A}$
using ‹*openin* X V› *openin_subset* **by** *auto*
qed
qed
qed

lemma *locally_finite_in_closure*:
assumes $\mathcal{A}$: *locally_finite_in* X $\mathcal{A}$
shows *locally_finite_in* X $((\lambda S.\ X$ *closure_of* $S)$ ‘ $\mathcal{A})$

using $\mathcal{A}$ **unfolding** *locally_finite_in_def*
proof (*intro conjI*; *clarsimp*)
fix x A
assume $x \in X$ *closure_of* A
then show $x \in$ *topspace* X
by (*meson in_closure_of*)
next
fix x
assume $x \in$ *topspace* X **and** $\bigcup \mathcal{A} \subseteq$ *topspace* X
then obtain V **where** V: *openin* X V $x \in V$ **and** *fin*: *finite* $\{U \in \mathcal{A}.\ U \cap V \neq \{\}\}$
using $\mathcal{A}$ **unfolding** *locally_finite_in_def* **by** *blast*
have *eq*: $\{y \in f\ `\ \mathcal{A}.\ Q\ y\} = f\ `\ \{x.\ x \in \mathcal{A} \wedge Q(f\ x)\}$ **for** f **and** $Q :: 'a\ set \Rightarrow bool$
by *blast*
have *eq2*: $\{A \in \mathcal{A}.\ X$ *closure_of* $A \cap V \neq \{\}\} = \{A \in \mathcal{A}.\ A \cap V \neq \{\}\}$
using *openin_Int_closure_of_eq_empty* V **by** *blast*
have *finite* $\{U \in (closure\_of)\ X\ `\ \mathcal{A}.\ U \cap V \neq \{\}\}$
by (*simp add*: *eq eq2 fin*)
with V **show** $\exists V.$ *openin* X $V \wedge x \in V \wedge$ *finite* $\{U \in (closure\_of)\ X\ `\ \mathcal{A}.\ U \cap V \neq \{\}\}$
by *blast*
qed

lemma *closedin_Union_locally_finite_closure*:
locally_finite_in $X\ \mathcal{A} \Longrightarrow$ *closedin* $X\ (\bigcup((\lambda S.\ X$ *closure_of* $S)\ `\ \mathcal{A}))$
by (*metis* (*mono_tags*) *closedin_closure_of closedin_locally_finite_Union imageE locally_finite_in_closure*)

lemma *closure_of_Union_subset*: $\bigcup((\lambda S.\ X$ *closure_of* $S)\ `\ \mathcal{A}) \subseteq X$ *closure_of* $(\bigcup \mathcal{A})$
by (*simp add*: *SUP_le_iff Sup_upper closure_of_mono*)

lemma *closure_of_locally_finite_Union*:
assumes *locally_finite_in* $X\ \mathcal{A}$
shows X *closure_of* $(\bigcup \mathcal{A}) = \bigcup((\lambda S.\ X$ *closure_of* $S)\ `\ \mathcal{A})$
proof (*rule closure_of_unique*)
show $\bigcup\ \mathcal{A} \subseteq \bigcup\ ((closure\_of)\ X\ `\ \mathcal{A})$
using *assms* **by** (*simp add*: *SUP_upper2 Sup_le_iff closure_of_subset locally_finite_in_def*)
show *closedin* $X\ (\bigcup\ ((closure\_of)\ X\ `\ \mathcal{A}))$
using *assms* **by** (*simp add*: *closedin_Union_locally_finite_closure*)
show $\bigwedge T'.\ [\![\bigcup\ \mathcal{A} \subseteq T'$; *closedin* $X\ T']\!] \Longrightarrow \bigcup\ ((closure\_of)\ X\ `\ \mathcal{A}) \subseteq T'$
by (*simp add*: *Sup_le_iff closure_of_minimal*)
qed

1.11.10 Continuous maps

We will need to deal with continuous maps in terms of topologies and not in terms of type classes, as defined below.

definition *continuous_map* **where**
continuous_map X Y f ≡
f ∈ *topspace X* → *topspace Y* ∧
(∀ *U*. *openin Y U* ⟶ *openin X* {*x* ∈ *topspace X*. *f x* ∈ *U*})

lemma *continuous_map*:
continuous_map X Y f ⟷
f ‘ (*topspace X*) ⊆ *topspace Y* ∧ (∀ *U*. *openin Y U* ⟶ *openin X* {*x* ∈ *topspace X*. *f x* ∈ *U*})
by (*auto simp*: *continuous_map_def*)

lemma *continuous_map_image_subset_topspace*:
continuous_map X Y f ⟹ *f* ‘ (*topspace X*) ⊆ *topspace Y*
by (*auto simp*: *continuous_map_def*)

lemma *continuous_map_funspace*:
continuous_map X Y f ⟹ *f* ∈ *topspace X* → *topspace Y*
by (*auto simp*: *continuous_map_def*)

lemma *continuous_map_on_empty* [*simp*]: *continuous_map trivial_topology Y f*
by (*auto simp*: *continuous_map_def*)

lemma *continuous_map_on_empty2* [*simp*]: *continuous_map X trivial_topology f* ⟷ *X* = *trivial_topology*
using *continuous_map_image_subset_topspace* **by** *fastforce*

lemma *continuous_map_closedin*:
continuous_map X Y f ⟷
f ∈ *topspace X* → *topspace Y* ∧
(∀ *C*. *closedin Y C* ⟶ *closedin X* {*x* ∈ *topspace X*. *f x* ∈ *C*})
proof −
have (∀ *U*. *openin Y U* ⟶ *openin X* {*x* ∈ *topspace X*. *f x* ∈ *U*}) =
(∀ *C*. *closedin Y C* ⟶ *closedin X* {*x* ∈ *topspace X*. *f x* ∈ *C*})
if *f* ∈ *topspace X* → *topspace Y*
proof −
have *eq*: {*x* ∈ *topspace X*. *f x* ∈ *topspace Y* ∧ *f x* ∉ *C*} = (*topspace X* − {*x* ∈ *topspace X*. *f x* ∈ *C*}) **for** *C*
using *that* **by** *blast*
show *?thesis*
proof (*intro iffI allI impI*)
fix *C*
assume ∀ *U*. *openin Y U* ⟶ *openin X* {*x* ∈ *topspace X*. *f x* ∈ *U*} **and** *closedin Y C*
then show *closedin X* {*x* ∈ *topspace X*. *f x* ∈ *C*}
by (*auto simp add*: *closedin_def eq*)

```
    next
      fix U
      assume ∀C. closedin Y C ⟶ closedin X {x ∈ topspace X. f x ∈ C} and
openin Y U
      then show openin X {x ∈ topspace X. f x ∈ U}
        by (auto simp add: openin_closedin_eq eq)
    qed
  qed
  then show ?thesis
    by (auto simp: continuous_map_def)
qed

lemma openin_continuous_map_preimage:
  ⟦continuous_map X Y f; openin Y U⟧ ⟹ openin X {x ∈ topspace X. f x ∈ U}
  by (simp add: continuous_map_def)

lemma closedin_continuous_map_preimage:
  ⟦continuous_map X Y f; closedin Y C⟧ ⟹ closedin X {x ∈ topspace X. f x ∈
C}
  by (simp add: continuous_map_closedin)

lemma openin_continuous_map_preimage_gen:
  assumes continuous_map X Y f openin X U openin Y V
  shows openin X {x ∈ U. f x ∈ V}
proof −
  have eq: {x ∈ U. f x ∈ V} = U ∩ {x ∈ topspace X. f x ∈ V}
    using assms(2) openin_closedin_eq by fastforce
  show ?thesis
    unfolding eq
    using assms openin_continuous_map_preimage by fastforce
qed

lemma closedin_continuous_map_preimage_gen:
  assumes continuous_map X Y f closedin X U closedin Y V
  shows closedin X {x ∈ U. f x ∈ V}
proof −
  have eq: {x ∈ U. f x ∈ V} = U ∩ {x ∈ topspace X. f x ∈ V}
    using assms(2) closedin_def by fastforce
  show ?thesis
    unfolding eq
    using assms closedin_continuous_map_preimage by fastforce
qed

lemma continuous_map_image_closure_subset:
  assumes continuous_map X Y f
  shows f ‘ (X closure_of S) ⊆ Y closure_of f ‘ S
proof −
  have ∗: f ‘ (topspace X) ⊆ topspace Y
    by (meson assms continuous_map)
```

```
  have X closure_of T ⊆ {x ∈ X closure_of T. f x ∈ Y closure_of (f ‘ T)}
    if T ⊆ topspace X for T
  proof (rule closure_of_minimal)
    show T ⊆ {x ∈ X closure_of T. f x ∈ Y closure_of f ‘ T}
      using closure_of_subset ∗ that  by (fastforce simp: in_closure_of)
  next
    show closedin X {x ∈ X closure_of T. f x ∈ Y closure_of f ‘ T}
      using assms closedin_continuous_map_preimage_gen by fastforce
  qed
  then show ?thesis
   by (smt (verit, ccfv_threshold) assms continuous_map image_eqI image_subset_iff
in_closure_of mem_Collect_eq)
qed

lemma continuous_map_subset_aux1: continuous_map X Y f ⟹
      (∀ S. f ‘ (X closure_of S) ⊆ Y closure_of f ‘ S)
  using continuous_map_image_closure_subset by blast

lemma continuous_map_subset_aux2:
  assumes ∀ S. S ⊆ topspace X ⟶ f ‘ (X closure_of S) ⊆ Y closure_of f ‘ S
  shows continuous_map X Y f
  unfolding continuous_map_closedin
proof (intro conjI ballI allI impI)
  show f ∈ topspace X → topspace Y
    using assms closure_of_subset_topspace by fastforce
next
  fix C
  assume closedin Y C
  then show closedin X {x ∈ topspace X. f x ∈ C}
  proof (clarsimp simp flip: closure_of_subset_eq, intro conjI)
    fix x
    assume x: x ∈ X closure_of {x ∈ topspace X. f x ∈ C}
      and C ⊆ topspace Y and Y closure_of C ⊆ C
    show x ∈ topspace X
      by (meson x in_closure_of)
    have {a ∈ topspace X. f a ∈ C} ⊆ topspace X
      by simp
    moreover have Y closure_of f ‘ {a ∈ topspace X. f a ∈ C} ⊆ C
      by (simp add: ‹closedin Y C› closure_of_minimal image_subset_iff)
    ultimately show f x ∈ C
      using x assms by blast
  qed
qed

lemma continuous_map_eq_image_closure_subset:
    continuous_map X Y f ⟷ (∀ S. f ‘ (X closure_of S) ⊆ Y closure_of f ‘ S)
  using continuous_map_subset_aux1 continuous_map_subset_aux2 by metis

lemma continuous_map_eq_image_closure_subset_alt:
```

continuous_map X Y f ⟷ (∀ *S*. *S* ⊆ *topspace X* ⟶ *f* ‘ (*X closure_of S*) ⊆ *Y closure_of f* ‘ *S*)
using *continuous_map_subset_aux1 continuous_map_subset_aux2* **by** *metis*

lemma *continuous_map_eq_image_closure_subset_gen*:
continuous_map X Y f ⟷
f ∈ *topspace X* → *topspace Y* ∧
(∀ *S*. *f* ‘ (*X closure_of S*) ⊆ *Y closure_of f* ‘ *S*)
by (*meson Pi_iff continuous_map continuous_map_eq_image_closure_subset image_subset_iff*)

lemma *continuous_map_closure_preimage_subset*:
continuous_map X Y f
⟹ *X closure_of* {*x* ∈ *topspace X*. *f x* ∈ *T*}
⊆ {*x* ∈ *topspace X*. *f x* ∈ *Y closure_of T*}
unfolding *continuous_map_closedin*
by (*rule closure_of_minimal*) (*use in_closure_of* **in** ‹*fastforce*+›)

lemma *continuous_map_frontier_frontier_preimage_subset*:
assumes *continuous_map X Y f*
shows *X frontier_of* {*x* ∈ *topspace X*. *f x* ∈ *T*} ⊆ {*x* ∈ *topspace X*. *f x* ∈ *Y frontier_of T*}
proof −
have *eq*: *topspace X* − {*x* ∈ *topspace X*. *f x* ∈ *T*} = {*x* ∈ *topspace X*. *f x* ∈ *topspace Y* − *T*}
using *assms* **unfolding** *continuous_map_def* **by** *blast*
have *X closure_of* {*x* ∈ *topspace X*. *f x* ∈ *T*} ⊆ {*x* ∈ *topspace X*. *f x* ∈ *Y closure_of T*}
by (*simp add*: *assms continuous_map_closure_preimage_subset*)
moreover
have *X closure_of* (*topspace X* − {*x* ∈ *topspace X*. *f x* ∈ *T*}) ⊆ {*x* ∈ *topspace X*. *f x* ∈ *Y closure_of* (*topspace Y* − *T*)}
using *continuous_map_closure_preimage_subset* [*OF assms*] *eq* **by** *presburger*
ultimately show *?thesis*
by (*auto simp*: *frontier_of_closures*)
qed

lemma *topology_finer_continuous_id*:
assumes *topspace X* = *topspace Y*
shows (∀ *S*. *openin X S* ⟶ *openin Y S*) ⟷ *continuous_map Y X id* (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⟹ *?rhs*
unfolding *continuous_map_def*
using *assms openin_subopen openin_subset* **by** *fastforce*
show *?rhs* ⟹ *?lhs*
unfolding *continuous_map_def*
using *assms openin_subopen topspace_def* **by** *fastforce*

qed

lemma *continuous_map_const* [*simp*]:
  continuous_map X Y (λ*x*. *C*) ⟷ *X* = *trivial_topology* ∨ *C* ∈ *topspace Y*
proof (*cases X* = *trivial_topology*)
 case *nontriv*: *False*
 show *?thesis*
 proof (*cases C* ∈ *topspace Y*)
  case *True*
  with *openin_subopen* **show** *?thesis*
   by (*auto simp*: *continuous_map_def*)
 next
  case *False*
  with *nontriv* **show** *?thesis*
   using *continuous_map_image_subset_topspace discrete_topology_unique image_subset_iff* **by** *fastforce*
 qed
qed *auto*

declare *continuous_map_const* [*THEN iffD2*, *continuous_intros*]

lemma *continuous_map_compose* [*continuous_intros*]:
 assumes *f*: *continuous_map X X' f* **and** *g*: *continuous_map X' X'' g*
 shows *continuous_map X X''* (*g* ∘ *f*)
 unfolding *continuous_map_def*
proof (*intro conjI ballI allI impI*)
 show *g* ∘ *f* ∈ *topspace X* → *topspace X''*
  using *assms* **unfolding** *continuous_map_def* **by** *force*
next
 fix *U*
 assume *openin X'' U*
 have *eq*: {*x* ∈ *topspace X*. (*g* ∘ *f*) *x* ∈ *U*} = {*x* ∈ *topspace X*. *f x* ∈ {*y*. *y* ∈ *topspace X'* ∧ *g y* ∈ *U*}}
  using *continuous_map_image_subset_topspace f* **by** *force*
 show *openin X* {*x* ∈ *topspace X*. (*g* ∘ *f*) *x* ∈ *U*}
  unfolding *eq*
  using *assms* **unfolding** *continuous_map_def*
  using ‹*openin X'' U*› **by** *blast*
qed

lemma *continuous_map_eq*:
 assumes *continuous_map X X' f* **and** ⋀*x*. *x* ∈ *topspace X* ⟹ *f x* = *g x*
 shows *continuous_map X X' g*
proof −
 have *eq*: {*x* ∈ *topspace X*. *f x* ∈ *U*} = {*x* ∈ *topspace X*. *g x* ∈ *U*} **for** *U*
  using *assms* **by** *auto*
 show *?thesis*
  using *assms* **by** (*force simp add*: *continuous_map_def eq*)
qed

lemma *restrict_continuous_map* [*simp*]:
  topspace X ⊆ S ⟹ continuous_map X X' (restrict f S) ⟷ continuous_map X X' f
 by (*auto simp*: *elim*!: *continuous_map_eq*)

lemma *continuous_map_in_subtopology*:
 continuous_map X (subtopology X' S) f ⟷ continuous_map X X' f ∧ f ‘ (topspace X) ⊆ S
 (**is** *?lhs = ?rhs*)
proof
 assume *L*: *?lhs*
 show *?rhs*
 proof −
  have *⋀A. f ‘ (X closure_of A) ⊆ subtopology X' S closure_of f ‘ A*
   by (*meson L continuous_map_image_closure_subset*)
  then show *?thesis*
  by (*metis* (*no_types*) *closure_of_subset_subtopology closure_of_subtopology_subset closure_of_topspace continuous_map_eq_image_closure_subset order.trans*)
 qed
next
 assume *R*: *?rhs*
 then have *eq*: *{x ∈ topspace X. f x ∈ U} = {x ∈ topspace X. f x ∈ U ∧ f x ∈ S}* **for** *U*
  by *auto*
 show *?lhs*
  using *R*
  unfolding *continuous_map*
  by (*auto simp*: *openin_subtopology eq*)
qed

lemma *continuous_map_from_subtopology*:
  continuous_map X Y f ⟹ continuous_map (subtopology X S) Y f
 by (*auto simp*: *continuous_map openin_subtopology*)

lemma *continuous_map_into_fulltopology*:
 continuous_map X (subtopology Y T) f ⟹ continuous_map X Y f
 by (*auto simp*: *continuous_map_in_subtopology*)

lemma *continuous_map_into_subtopology*:
 ⟦continuous_map X Y f; f ∈ topspace X → T⟧ ⟹ continuous_map X (subtopology Y T) f
 by (*auto simp*: *continuous_map_in_subtopology*)

lemma *continuous_map_from_subtopology_mono*:
  ⟦continuous_map (subtopology X T) Y f; S ⊆ T⟧
   ⟹ continuous_map (subtopology X S) Y f
 by (*metis inf.absorb_iff2 continuous_map_from_subtopology subtopology_subtopology*)

lemma *continuous_map_from_discrete_topology* [*simp*]:
continuous_map (*discrete_topology U*) *X f* ⟷ *f* ∈ *U* → *topspace X*
by (*auto simp*: *continuous_map_def*)

lemma *continuous_map_iff_continuous* [*simp*]: *continuous_map* (*top_of_set S*) *euclidean g* = *continuous_on S g*
by (*fastforce simp add*: *continuous_map openin_subtopology continuous_on_open_invariant*)

lemma *continuous_map_iff_continuous2* [*simp*]: *continuous_map euclidean euclidean g* = *continuous_on UNIV g*
by (*metis continuous_map_iff_continuous subtopology_UNIV*)

lemma *continuous_map_openin_preimage_eq*:
continuous_map X Y f ⟷
f ∈ (*topspace X*) → *topspace Y* ∧ (∀ *U*. *openin Y U* ⟶ *openin X* (*topspace X* ∩ *f* −‘ *U*))
by (*auto simp*: *continuous_map_def vimage_def Int_def*)

lemma *continuous_map_closedin_preimage_eq*:
continuous_map X Y f ⟷
f ∈ (*topspace X*) → *topspace Y* ∧ (∀ *U*. *closedin Y U* ⟶ *closedin X* (*topspace X* ∩ *f* −‘ *U*))
by (*auto simp*: *continuous_map_closedin vimage_def Int_def*)

lemma *continuous_map_square_root*: *continuous_map euclideanreal euclideanreal sqrt*
by (*simp add*: *continuous_at_imp_continuous_on isCont_real_sqrt*)

lemma *continuous_map_sqrt* [*continuous_intros*]:
continuous_map X euclideanreal f ⟹ *continuous_map X euclideanreal* (λ*x*. *sqrt*(*f x*))
by (*meson continuous_map_compose continuous_map_eq continuous_map_square_root o_apply*)

lemma *continuous_map_id* [*simp*, *continuous_intros*]: *continuous_map X X id*
unfolding *continuous_map_def* **using** *openin_subopen topspace_def* **by** *fastforce*

declare *continuous_map_id* [*unfolded id_def*, *simp*, *continuous_intros*]

lemma *continuous_map_id_subt* [*simp*]: *continuous_map* (*subtopology X S*) *X id*
by (*simp add*: *continuous_map_from_subtopology*)

declare *continuous_map_id_subt* [*unfolded id_def*, *simp*]

lemma *continuous_map_alt*:
continuous_map T1 T2 f

$= ((\forall U.\ openin\ T2\ U \longrightarrow openin\ T1\ (f -` U \cap topspace\ T1)) \wedge f \in topspace\ T1 \to topspace\ T2)$
by (*auto simp*: *continuous_map_def vimage_def image_def Collect_conj_eq inf_commute*)

lemma *continuous_map_open* [*intro*]:
$continuous\_map\ T1\ T2\ f \Longrightarrow openin\ T2\ U \Longrightarrow openin\ T1\ (f-`U \cap topspace(T1))$
unfolding *continuous_map_alt* **by** *auto*

lemma *continuous_map_preimage_topspace* [*intro*]:
assumes *continuous_map T1 T2 f*
shows $f-`(topspace\ T2) \cap topspace\ T1 = topspace\ T1$
using *assms* **unfolding** *continuous_map_def* **by** *auto*

1.11.11 Open and closed maps (not a priori assumed continuous)

definition *open_map* :: *'a topology* ⇒ *'b topology* ⇒ (*'a* ⇒ *'b*) ⇒ *bool*
where $open\_map\ X1\ X2\ f \equiv \forall U.\ openin\ X1\ U \longrightarrow openin\ X2\ (f `` U)$

definition *closed_map* :: *'a topology* ⇒ *'b topology* ⇒ (*'a* ⇒ *'b*) ⇒ *bool*
where $closed\_map\ X1\ X2\ f \equiv \forall U.\ closedin\ X1\ U \longrightarrow closedin\ X2\ (f `` U)$

lemma *open_map_imp_subset_topspace*:
$open\_map\ X1\ X2\ f \Longrightarrow f \in (topspace\ X1) \to topspace\ X2$
unfolding *open_map_def* **using** *openin_subset* **by** *fastforce*

lemma *open_map_on_empty* [*simp*]: *open_map trivial_topology Y f*
by (*simp add*: *open_map_def*)

lemma *closed_map_on_empty*:
closed_map trivial_topology Y f
by (*simp add*: *closed_map_def closedin_topspace_empty*)

lemma *closed_map_const*:
$closed\_map\ X\ Y\ (\lambda x.\ c) \longleftrightarrow X = trivial\_topology \vee closedin\ Y\ \{c\}$
by (*metis closed_map_def closed_map_on_empty closedin_topspace discrete_topology_unique equals0D image_constant_conv*)

lemma *open_map_imp_subset*:
$[\![open\_map\ X1\ X2\ f;\ S \subseteq topspace\ X1]\!] \Longrightarrow f \in S \to topspace\ X2$
using *open_map_imp_subset_topspace* **by** *fastforce*

lemma *topology_finer_open_id*:
$(\forall S.\ openin\ X\ S \longrightarrow openin\ X'\ S) \longleftrightarrow open\_map\ X\ X'\ id$
unfolding *open_map_def* **by** *auto*

lemma *open_map_id*: *open_map X X id*
unfolding *open_map_def* **by** *auto*

lemma *open_map_eq*:
  ⟦*open_map X X′ f*; ⋀*x*. *x* ∈ *topspace X* ⟹ *f x* = *g x*⟧ ⟹ *open_map X X′ g*
 unfolding *open_map_def*
 by (*metis image_cong openin_subset subset_iff*)

lemma *open_map_inclusion_eq*:
 open_map (*subtopology X S*) *X id* ⟷ *openin X* (*topspace X* ∩ *S*)
 by (*metis openin_topspace openin_trans_full subtopology_restrict topology_finer_open_id topspace_subtopology*)

lemma *open_map_inclusion*:
  openin X S ⟹ *open_map* (*subtopology X S*) *X id*
 by (*simp add*: *open_map_inclusion_eq openin_Int*)

lemma *open_map_compose*:
  ⟦*open_map X X′ f*; *open_map X′ X″ g*⟧ ⟹ *open_map X X″* (*g* ∘ *f*)
 by (*metis* (*no_types*, *lifting*) *image_comp open_map_def*)

lemma *closed_map_imp_subset_topspace*:
  closed_map X1 X2 f ⟹ *f* ∈ (*topspace X1*) → *topspace X2*
 by (*simp add*: *closed_map_def closedin_def image_subset_iff_funcset*)

lemma *closed_map_imp_subset*:
  ⟦*closed_map X1 X2 f*; *S* ⊆ *topspace X1*⟧ ⟹ *f* ∈ *S* → *topspace X2*
 using *closed_map_imp_subset_topspace* **by** *blast*

lemma *topology_finer_closed_id*:
  (∀ *S*. *closedin X S* ⟶ *closedin X′ S*) ⟷ *closed_map X X′ id*
 by (*simp add*: *closed_map_def*)

lemma *closed_map_id*: *closed_map X X id*
 by (*simp add*: *closed_map_def*)

lemma *closed_map_eq*:
 ⟦*closed_map X X′ f*; ⋀*x*. *x* ∈ *topspace X* ⟹ *f x* = *g x*⟧ ⟹ *closed_map X X′*
g
 unfolding *closed_map_def*
 by (*metis image_cong closedin_subset subset_iff*)

lemma *closed_map_compose*:
  ⟦*closed_map X X′ f*; *closed_map X′ X″ g*⟧ ⟹ *closed_map X X″* (*g* ∘ *f*)
 by (*metis* (*no_types*, *lifting*) *closed_map_def image_comp*)

lemma *closed_map_inclusion_eq*:
 closed_map (*subtopology X S*) *X id* ⟷
   closedin X (*topspace X* ∩ *S*)
proof −
 have ∗: *closedin X* (*T* ∩ *S*) **if** *closedin X* (*S* ∩ *topspace X*) *closedin X T* **for** *T*

by (*smt* (*verit*, *best*) *closedin_Int closure_of_subset_eq inf_sup_aci le_iff_inf that*)
then show *?thesis*
by (*fastforce simp add*: *closed_map_def Int_commute closedin_subtopology_alt intro*: ∗)
qed

lemma *closed_map_inclusion*: *closedin X S* ⟹ *closed_map* (*subtopology X S*) *X id*
by (*simp add*: *closed_map_inclusion_eq closedin_Int*)

lemma *open_map_into_subtopology*:
⟦*open_map X X' f*; *f* ∈ *topspace X* → *S*⟧ ⟹ *open_map X* (*subtopology X' S*) *f*
unfolding *open_map_def openin_subtopology*
using *openin_subset* **by** *fastforce*

lemma *closed_map_into_subtopology*:
⟦*closed_map X X' f*; *f* ∈ *topspace X* → *S*⟧ ⟹ *closed_map X* (*subtopology X' S*) *f*
unfolding *closed_map_def closedin_subtopology*
using *closedin_subset* **by** *fastforce*

lemma *open_map_into_discrete_topology*:
open_map X (*discrete_topology U*) *f* ⟷ *f* ∈ (*topspace X*) → *U*
unfolding *open_map_def openin_discrete_topology* **using** *openin_subset* **by** *blast*

lemma *closed_map_into_discrete_topology*:
closed_map X (*discrete_topology U*) *f* ⟷ *f* ∈ (*topspace X*) → *U*
unfolding *closed_map_def closedin_discrete_topology* **using** *closedin_subset* **by** *blast*

lemma *bijective_open_imp_closed_map*:
⟦*open_map X X' f*; *f* ‘ (*topspace X*) = *topspace X'*; *inj_on f* (*topspace X*)⟧ ⟹ *closed_map X X' f*
unfolding *open_map_def closed_map_def closedin_def*
by *auto* (*metis Diff_subset inj_on_image_set_diff*)

lemma *bijective_closed_imp_open_map*:
⟦*closed_map X X' f*; *f* ‘ (*topspace X*) = *topspace X'*; *inj_on f* (*topspace X*)⟧ ⟹ *open_map X X' f*
unfolding *closed_map_def open_map_def openin_closedin_eq*
by *auto* (*metis Diff_subset inj_on_image_set_diff*)

lemma *open_map_from_subtopology*:
⟦*open_map X X' f*; *openin X U*⟧ ⟹ *open_map* (*subtopology X U*) *X' f*
unfolding *open_map_def openin_subtopology_alt* **by** *blast*

lemma *closed_map_from_subtopology*:
  ⟦*closed_map X X′ f*; *closedin X U*⟧ ⟹ *closed_map* (*subtopology X U*) *X′ f*
 unfolding *closed_map_def closedin_subtopology_alt* **by** *blast*

lemma *open_map_restriction*:
 assumes *f*: *open_map X X′ f* **and** *U*: {*x* ∈ *topspace X*. *f x* ∈ *V*} = *U*
 shows *open_map* (*subtopology X U*) (*subtopology X′ V*) *f*
 unfolding *open_map_def*
proof *clarsimp*
 fix *W*
 assume *openin* (*subtopology X U*) *W*
 then obtain *T* **where** *openin X T W* = *T* ∩ *U*
  by (*meson openin_subtopology*)
 with *f U* **have** *f* ‘ *W* = (*f* ‘ *T*) ∩ *V*
  unfolding *open_map_def openin_closedin_eq* **by** *auto*
 then show *openin* (*subtopology X′ V*) (*f* ‘ *W*)
  by (*metis* ‹*openin X T*› *f open_map_def openin_subtopology_Int*)
qed

lemma *closed_map_restriction*:
 assumes *f*: *closed_map X X′ f* **and** *U*: {*x* ∈ *topspace X*. *f x* ∈ *V*} = *U*
 shows *closed_map* (*subtopology X U*) (*subtopology X′ V*) *f*
 unfolding *closed_map_def*
proof *clarsimp*
 fix *W*
 assume *closedin* (*subtopology X U*) *W*
 then obtain *T* **where** *closedin X T W* = *T* ∩ *U*
  by (*meson closedin_subtopology*)
 with *f U* **have** *f* ‘ *W* = (*f* ‘ *T*) ∩ *V*
  unfolding *closed_map_def closedin_def* **by** *auto*
 then show *closedin* (*subtopology X′ V*) (*f* ‘ *W*)
  by (*metis* ‹*closedin X T*› *closed_map_def closedin_subtopology f*)
qed

lemma *closed_map_closure_of_image*:
  closed_map X Y f ⟷
    f ∈ *topspace X* → *topspace Y* ∧
    (∀ *S*. *S* ⊆ *topspace X* ⟶ *Y closure_of* (*f* ‘ *S*) ⊆ *f* ‘ (*X closure_of S*)) (**is** *?lhs=?rhs*)
proof
 assume *?lhs*
 then show *?rhs*
  by (*simp add*: *closed_map_def closed_map_imp_subset_topspace closure_of_minimal closure_of_subset image_mono*)
next
 assume *?rhs*
 then show *?lhs*
  by (*metis closed_map_def closed_map_into_discrete_topology closure_of_eq*

closure_of_subset_eq topspace_discrete_topology)
qed

lemma *open_map_interior_of_image_subset*:
open_map X Y f ⟷ (∀ *S*. *image f* (*X interior_of S*) ⊆ *Y interior_of* (*f* ‘ *S*))
by (*metis image_mono interior_of_eq interior_of_maximal interior_of_subset open_map_def openin_interior_of subset_antisym*)

lemma *open_map_interior_of_image_subset_alt*:
open_map X Y f ⟷ (∀ *S*⊆*topspace X*. *f* ‘ (*X interior_of S*) ⊆ *Y interior_of f* ‘ *S*)
by (*metis interior_of_eq open_map_def open_map_interior_of_image_subset openin_subset subset_interior_of_eq*)

lemma *open_map_interior_of_image_subset_gen*:
open_map X Y f ⟷
(*f* ∈ *topspace X* → *topspace Y* ∧ (∀ *S*. *f* ‘ (*X interior_of S*) ⊆ *Y interior_of f* ‘ *S*))
by (*metis open_map_imp_subset_topspace open_map_interior_of_image_subset*)

lemma *open_map_preimage_neighbourhood*:
open_map X Y f ⟷
(*f* ∈ *topspace X* → *topspace Y* ∧
(∀ *U T*. *closedin X U* ∧ *T* ⊆ *topspace Y* ∧
{*x* ∈ *topspace X*. *f x* ∈ *T*} ⊆ *U* ⟶
(∃ *V*. *closedin Y V* ∧ *T* ⊆ *V* ∧ {*x* ∈ *topspace X*. *f x* ∈ *V*} ⊆ *U*))) (**is** *?lhs*=*?rhs*)
proof
assume *L*: *?lhs*
show *?rhs*
proof (*intro conjI strip*)
show *f* ∈ *topspace X* → *topspace Y*
by (*simp add*: ‹*open_map X Y f*› *open_map_imp_subset_topspace*)
next
fix *U T*
assume *UT*: *closedin X U* ∧ *T* ⊆ *topspace Y* ∧ {*x* ∈ *topspace X*. *f x* ∈ *T*} ⊆ *U*
have *closedin Y* (*topspace Y* − *f* ‘ (*topspace X* − *U*))
by (*meson UT L open_map_def openin_closedin_eq openin_diff openin_topspace*)
with *UT*
show ∃ *V*. *closedin Y V* ∧ *T* ⊆ *V* ∧ {*x* ∈ *topspace X*. *f x* ∈ *V*} ⊆ *U*
using *image_iff* **by** *auto*
qed
next
assume *R*: *?rhs*
show *?lhs*
unfolding *open_map_def*
proof (*intro strip*)
fix *U* **assume** *openin X U*

```
    have {x ∈ topspace X. f x ∈ topspace Y − f ‘ U} ⊆ topspace X − U
      by blast
    then obtain V where V: closedin Y V
      and sub: topspace Y − f ‘ U ⊆ V {x ∈ topspace X. f x ∈ V} ⊆ topspace X
− U
      using R ‹openin X U› by (meson Diff_subset openin_closedin_eq)
    then have f ‘ U ⊆ topspace Y − V
      using R ‹openin X U› openin_subset by fastforce
    with sub have f ‘ U = topspace Y − V
      by auto
    then show openin Y (f ‘ U)
      using V(1) by auto
  qed
qed


lemma closed_map_preimage_neighbourhood:
   closed_map X Y f ⟷
        f ∈ topspace X → topspace Y ∧
        (∀ U T. openin X U ∧ T ⊆ topspace Y ∧
              {x ∈ topspace X. f x ∈ T} ⊆ U
              ⟶ (∃ V. openin Y V ∧ T ⊆ V ∧
                    {x ∈ topspace X. f x ∈ V} ⊆ U)) (is ?lhs=?rhs)
proof
  assume L: ?lhs
  show ?rhs
  proof (intro conjI strip)
    show f ∈ topspace X → topspace Y
      by (simp add: L closed_map_imp_subset_topspace)
  next
    fix U T
    assume UT: openin X U ∧ T ⊆ topspace Y ∧ {x ∈ topspace X. f x ∈ T} ⊆ U
    then have openin Y (topspace Y − f ‘ (topspace X − U))
      by (meson L closed_map_def closedin_def closedin_diff closedin_topspace)
    then show ∃ V. openin Y V ∧ T ⊆ V ∧ {x ∈ topspace X. f x ∈ V} ⊆ U
      using UT image_iff by auto
  qed
next
  assume R: ?rhs
  show ?lhs
    unfolding closed_map_def
  proof (intro strip)
    fix U assume closedin X U
    have {x ∈ topspace X. f x ∈ topspace Y − f ‘ U} ⊆ topspace X − U
      by blast
    then obtain V where V: openin Y V
      and sub: topspace Y − f ‘ U ⊆ V {x ∈ topspace X. f x ∈ V} ⊆ topspace X
− U
      using R Diff_subset ‹closedin X U› unfolding closedin_def
```

```
    by (smt (verit, ccfv_threshold) Collect_mem_eq Collect_mono_iff)
  then have f ‘ U ⊆ topspace Y − V
    using R ‹closedin X U› closedin_subset by fastforce
  with sub have f ‘ U = topspace Y − V
    by auto
  with V show closedin Y (f ‘ U)
    by auto
 qed
qed

lemma closed_map_fibre_neighbourhood:
 closed_map X Y f ⟷
   f ∈ (topspace X) → topspace Y ∧
   (∀ U y. openin X U ∧ y ∈ topspace Y ∧ {x ∈ topspace X. f x = y} ⊆ U
   ⟶ (∃ V. openin Y V ∧ y ∈ V ∧ {x ∈ topspace X. f x ∈ V} ⊆ U))
 unfolding closed_map_preimage_neighbourhood
proof (intro conj_cong refl all_cong1)
 fix U
 assume f ∈ topspace X → topspace Y
 show (∀ T. openin X U ∧ T ⊆ topspace Y ∧ {x ∈ topspace X. f x ∈ T} ⊆ U
       ⟶ (∃ V. openin Y V ∧ T ⊆ V ∧ {x ∈ topspace X. f x ∈ V} ⊆ U))
     = (∀ y. openin X U ∧ y ∈ topspace Y ∧ {x ∈ topspace X. f x = y} ⊆ U
       ⟶ (∃ V. openin Y V ∧ y ∈ V ∧ {x ∈ topspace X. f x ∈ V} ⊆ U))
   (is (∀ T. ?P T) ⟷ (∀ y. ?Q y))
 proof
   assume L [rule_format]: ∀ T. ?P T
   show ∀ y. ?Q y
   proof
     fix y show ?Q y
       using L [of {y}] by blast
   qed
 next
   assume R: ∀ y. ?Q y
   show ∀ T. ?P T
   proof (cases openin X U)
     case True
     note [[unify_search_bound=3]]
     obtain V where V: ⋀y. ⟦y ∈ topspace Y; {x ∈ topspace X. f x = y} ⊆ U⟧
⟹
                       openin Y (V y) ∧ y ∈ V y ∧ {x ∈ topspace X. f x ∈ V y} ⊆ U
       using R by (simp add: True) meson
     show ?thesis
     proof clarify
       fix T
       assume openin X U and T ⊆ topspace Y and {x ∈ topspace X. f x ∈ T}
⊆ U
       with V show ∃ V. openin Y V ∧ T ⊆ V ∧ {x ∈ topspace X. f x ∈ V} ⊆
U
         by (rule_tac x=⋃ y∈T. V y in exI) fastforce
```

qed
qed *auto*
qed
qed

lemma *open_map_in_subtopology*:
openin Y S
$\Longrightarrow$ *open_map X* (*subtopology Y S*) *f* $\longleftrightarrow$ *open_map X Y f* $\wedge$ *f* $\in$ *topspace X* $\rightarrow$ *S*
by (*metis image_subset_iff_funcset open_map_def open_map_into_subtopology openin_imp_subset openin_topspace openin_trans_full*)

lemma *open_map_from_open_subtopology*:
$[\![$*openin Y S*; *open_map X* (*subtopology Y S*) *f*$]\!] \Longrightarrow$ *open_map X Y f*
using *open_map_in_subtopology* **by** *blast*

lemma *closed_map_in_subtopology*:
closedin Y S
$\Longrightarrow$ *closed_map X* (*subtopology Y S*) *f* $\longleftrightarrow$ (*closed_map X Y f* $\wedge$ *f* $\in$ *topspace X* $\rightarrow$ *S*)
by (*metis closed_map_def closed_map_imp_subset_topspace closed_map_into_subtopology closedin_closed_subtopology closedin_subset topspace_subtopology_subset*)

lemma *closed_map_from_closed_subtopology*:
$[\![$*closedin Y S*; *closed_map X* (*subtopology Y S*) *f*$]\!] \Longrightarrow$ *closed_map X Y f*
using *closed_map_in_subtopology* **by** *blast*

lemma *closed_map_from_composition_left*:
assumes *cmf*: *closed_map X Z* (*g* $\circ$ *f*) **and** *contf*: *continuous_map X Y f* **and** *fim*: *f* ‘ *topspace X* = *topspace Y*
shows *closed_map Y Z g*
unfolding *closed_map_def*
proof (*intro strip*)
fix *U* **assume** *closedin Y U*
then have *closedin X* {*x* $\in$ *topspace X*. *f x* $\in$ *U*}
using *contf closedin_continuous_map_preimage* **by** *blast*
then have *closedin Z* ((*g* $\circ$ *f*) ‘ {*x* $\in$ *topspace X*. *f x* $\in$ *U*})
using *cmf closed_map_def* **by** *blast*
moreover
have $\bigwedge$*y*. *y* $\in$ *U* $\Longrightarrow$ $\exists$ *x* $\in$ *topspace X*. *f x* $\in$ *U* $\wedge$ *g y* = *g* (*f x*)
by (*smt* (*verit*, *ccfv_SIG*) ‹*closedin Y U*› *closedin_subset fim image_iff subsetD*)
then have (*g* $\circ$ *f*) ‘ {*x* $\in$ *topspace X*. *f x* $\in$ *U*} = *g*‘*U* **by** *auto*
ultimately show *closedin Z* (*g* ‘ *U*)
by *metis*
qed

identical proof as the above

lemma *open_map_from_composition_left*:
assumes *cmf*: *open_map X Z* (*g* ∘ *f*) **and** *contf*: *continuous_map X Y f* **and** *fim*: *f* ‘ *topspace X* = *topspace Y*
shows *open_map Y Z g*
unfolding *open_map_def*
proof (*intro strip*)
fix *U* **assume** *openin Y U*
then have *openin X* {*x* ∈ *topspace X*. *f x* ∈ *U*}
using *contf openin_continuous_map_preimage* **by** *blast*
then have *openin Z* ((*g* ∘ *f*) ‘ {*x* ∈ *topspace X*. *f x* ∈ *U*})
using *cmf open_map_def* **by** *blast*
moreover
have ⋀*y*. *y* ∈ *U* ⟹ ∃ *x* ∈ *topspace X*. *f x* ∈ *U* ∧ *g y* = *g* (*f x*)
by (*smt* (*verit*, *ccfv_SIG*) ‹*openin Y U*› *openin_subset fim image_iff subsetD*)
then have (*g* ∘ *f*) ‘ {*x* ∈ *topspace X*. *f x* ∈ *U*} = *g*‘*U* **by** *auto*
ultimately show *openin Z* (*g* ‘ *U*)
by *metis*
qed

lemma *closed_map_from_composition_right*:
assumes *cmf*: *closed_map X Z* (*g* ∘ *f*) *f* ∈ *topspace X* → *topspace Y continuous_map Y Z g inj_on g* (*topspace Y*)
shows *closed_map X Y f*
unfolding *closed_map_def*
proof (*intro strip*)
fix *C* **assume** *closedin X C*
have ⋀*y c*. ⟦*y* ∈ *topspace Y*; *g y* = *g* (*f c*); *c* ∈ *C*⟧ ⟹ *y* ∈ *f* ‘ *C*
using ‹*closedin X C*› *assms closedin_subset inj_onD* **by** *fastforce*
then
have *f* ‘ *C* = {*x* ∈ *topspace Y*. *g x* ∈ (*g* ∘ *f*) ‘ *C*}
using ‹*closedin X C*› *assms*(2) *closedin_subset* **by** *fastforce*
moreover have *closedin Z* ((*g* ∘ *f*) ‘ *C*)
using ‹*closedin X C*› *cmf closed_map_def* **by** *blast*
ultimately show *closedin Y* (*f* ‘ *C*)
using *assms*(3) *closedin_continuous_map_preimage* **by** *fastforce*
qed

identical proof as the above

lemma *open_map_from_composition_right*:
assumes *open_map X Z* (*g* ∘ *f*) *f* ∈ *topspace X* → *topspace Y continuous_map Y Z g inj_on g* (*topspace Y*)
shows *open_map X Y f*
unfolding *open_map_def*
proof (*intro strip*)
fix *C* **assume** *openin X C*
have ⋀*y c*. ⟦*y* ∈ *topspace Y*; *g y* = *g* (*f c*); *c* ∈ *C*⟧ ⟹ *y* ∈ *f* ‘ *C*
using ‹*openin X C*› *assms openin_subset inj_onD* **by** *fastforce*
then
have *f* ‘ *C* = {*x* ∈ *topspace Y*. *g x* ∈ (*g* ∘ *f*) ‘ *C*}

using ‹*openin X C*› *assms*(*2*) *openin_subset* **by** *fastforce*
moreover have *openin Z* ((*g* ∘ *f*) ‘ *C*)
using ‹*openin X C*› *assms*(*1*) *open_map_def* **by** *blast*
ultimately show *openin Y* (*f* ‘ *C*)
using *assms*(*3*) *openin_continuous_map_preimage* **by** *fastforce*
qed

1.11.12 Quotient maps

definition *quotient_map* **where**
quotient_map X X′ f ⟷
f ‘ (*topspace X*) = *topspace X′* ∧
(∀ *U*. *U* ⊆ *topspace X′* ⟶ (*openin X* {*x*. *x* ∈ *topspace X* ∧ *f x* ∈ *U*} ⟷ *openin X′ U*))

lemma *quotient_map_eq*:
assumes *quotient_map X X′ f* ⋀*x*. *x* ∈ *topspace X* ⟹ *f x* = *g x*
shows *quotient_map X X′ g*
by (*smt* (*verit*) *Collect_cong assms image_cong quotient_map_def*)

lemma *quotient_map_compose*:
assumes *f*: *quotient_map X X′ f* **and** *g*: *quotient_map X′ X″ g*
shows *quotient_map X X″* (*g* ∘ *f*)
unfolding *quotient_map_def*
proof (*intro conjI allI impI*)
show (*g* ∘ *f*) ‘ *topspace X* = *topspace X″*
using *assms* **by** (*simp only*: *image_comp* [*symmetric*]) (*simp add*: *quotient_map_def*)
next
fix *U″*
assume *U″*: *U″* ⊆ *topspace X″*
define *U′* **where** *U′* ≡ {*y* ∈ *topspace X′*. *g y* ∈ *U″*}
have *U′* ⊆ *topspace X′*
by (*auto simp add*: *U′_def*)
then have *U′*: *openin X* {*x* ∈ *topspace X*. *f x* ∈ *U′*} = *openin X′ U′*
using *assms* **unfolding** *quotient_map_def* **by** *simp*
have {*x* ∈ *topspace X*. *f x* ∈ *topspace X′* ∧ *g* (*f x*) ∈ *U″*} = {*x* ∈ *topspace X*. (*g* ∘ *f*) *x* ∈ *U″*}
using *f quotient_map_def* **by** *fastforce*
then show *openin X* {*x* ∈ *topspace X*. (*g* ∘ *f*) *x* ∈ *U″*} = *openin X″ U″*
by (*smt* (*verit*, *best*) *Collect_cong U′ U′_def U″ g mem_Collect_eq quotient_map_def*)
qed

lemma *quotient_map_from_composition*:
assumes *f*: *continuous_map X X′ f* **and** *g*: *continuous_map X′ X″ g* **and** *gf*: *quotient_map X X″* (*g* ∘ *f*)
shows *quotient_map X′ X″ g*
unfolding *quotient_map_def*
proof (*intro conjI allI impI*)

```
  show g ‘ topspace X' = topspace X''
    using assms unfolding continuous_map_def quotient_map_def by fastforce
next
  fix U'' :: 'c set
  assume U'': U'' ⊆ topspace X''
  have eq: {x ∈ topspace X. g (f x) ∈ U''} = {x ∈ topspace X. f x ∈ {y. y ∈
topspace X' ∧ g y ∈ U''}}
    using continuous_map_def f by fastforce
  show openin X' {x ∈ topspace X'. g x ∈ U''} = openin X'' U''
    using assms unfolding continuous_map_def quotient_map_def
    by (metis (mono_tags, lifting) Collect_cong U'' comp_apply eq)
qed

lemma quotient_imp_continuous_map:
    quotient_map X X' f ⟹ continuous_map X X' f
  by (simp add: continuous_map openin_subset quotient_map_def)

lemma quotient_imp_surjective_map:
    quotient_map X X' f ⟹ f ‘ (topspace X) = topspace X'
  by (simp add: quotient_map_def)

lemma quotient_map_closedin:
  quotient_map X X' f ⟷
        f ‘ (topspace X) = topspace X' ∧
        (∀ U. U ⊆ topspace X' ⟶ (closedin X {x. x ∈ topspace X ∧ f x ∈ U} ⟷
closedin X' U))
proof −
  have eq: (topspace X − {x ∈ topspace X. f x ∈ U'}) = {x ∈ topspace X. f x ∈
topspace X' ∧ f x ∉ U'}
    if f ‘ topspace X = topspace X' U' ⊆ topspace X' for U'
      using that by auto
  have (∀ U⊆topspace X'. openin X {x ∈ topspace X. f x ∈ U} = openin X' U)
=
          (∀ U⊆topspace X'. closedin X {x ∈ topspace X. f x ∈ U} = closedin X'
U)
    if f ‘ topspace X = topspace X'
  proof (rule iffI; intro allI impI subsetI)
    fix U'
    assume *[rule_format]: ∀ U⊆topspace X'. openin X {x ∈ topspace X. f x ∈
U} = openin X' U
      and U': U' ⊆ topspace X'
    show closedin X {x ∈ topspace X. f x ∈ U'} = closedin X' U'
      using U' by (auto simp add: closedin_def simp flip: * [of topspace X' − U']
eq [OF that])
  next
    fix U' :: 'b set
    assume *[rule_format]: ∀ U⊆topspace X'. closedin X {x ∈ topspace X. f x ∈
U} = closedin X' U
      and U': U' ⊆ topspace X'
```

show *openin X* {*x* ∈ *topspace X*. *f x* ∈ *U'*} = *openin X' U'*
using *U'* **by** (*auto simp add*: *openin_closedin_eq simp flip*: ∗ [*of topspace X' − U'*] *eq* [*OF that*])
qed
then show *?thesis*
unfolding *quotient_map_def* **by** *force*
qed

lemma *continuous_open_imp_quotient_map*:
assumes *continuous_map X X' f* **and** *om*: *open_map X X' f* **and** *feq*: *f* ‘ (*topspace X*) = *topspace X'*
shows *quotient_map X X' f*
proof −
{ fix *U*
assume *U*: *U* ⊆ *topspace X'* **and** *openin X* {*x* ∈ *topspace X*. *f x* ∈ *U*}
then have *ope*: *openin X'* (*f* ‘ {*x* ∈ *topspace X*. *f x* ∈ *U*})
using *om* **unfolding** *open_map_def* **by** *blast*
then have *openin X' U*
using *U feq* **by** (*subst openin_subopen*) *force*
}
moreover have *openin X* {*x* ∈ *topspace X*. *f x* ∈ *U*} **if** *U* ⊆ *topspace X'* **and** *openin X' U* **for** *U*
using *that assms* **unfolding** *continuous_map_def* **by** *blast*
ultimately show *?thesis*
unfolding *quotient_map_def* **using** *assms* **by** *blast*
qed

lemma *continuous_closed_imp_quotient_map*:
assumes *continuous_map X X' f* **and** *om*: *closed_map X X' f* **and** *feq*: *f* ‘ (*topspace X*) = *topspace X'*
shows *quotient_map X X' f*
proof −
have *f* ‘ {*x* ∈ *topspace X*. *f x* ∈ *U*} = *U* **if** *U* ⊆ *topspace X'* **for** *U*
using *that feq* **by** *auto*
with *assms* **show** *?thesis*
unfolding *quotient_map_closedin closed_map_def continuous_map_closedin*
by *auto*
qed

lemma *continuous_open_quotient_map*:
⟦*continuous_map X X' f*; *open_map X X' f*⟧ ⟹ *quotient_map X X' f* ⟷ *f* ‘ (*topspace X*) = *topspace X'*
by (*meson continuous_open_imp_quotient_map quotient_map_def*)

lemma *continuous_closed_quotient_map*:
⟦*continuous_map X X' f*; *closed_map X X' f*⟧ ⟹ *quotient_map X X' f* ⟷ *f* ‘ (*topspace X*) = *topspace X'*
by (*meson continuous_closed_imp_quotient_map quotient_map_def*)

lemma *injective_quotient_map*:
assumes *inj_on f* (*topspace X*)
shows *quotient_map X X' f* ⟷
continuous_map X X' f ∧ *open_map X X' f* ∧ *closed_map X X' f* ∧ *f* ‘ (*topspace X*) = *topspace X'*
(**is** *?lhs* = *?rhs*)
proof
assume *L*: *?lhs*
have *om*: *open_map X X' f*
proof (*clarsimp simp add*: *open_map_def*)
fix *U*
assume *openin X U*
then have *U* ⊆ *topspace X*
by (*simp add*: *openin_subset*)
moreover have {*x* ∈ *topspace X*. *f x* ∈ *f* ‘ *U*} = *U*
using ‹*U* ⊆ *topspace X*› *assms inj_onD* **by** *fastforce*
ultimately show *openin X'* (*f* ‘ *U*)
using *L* **unfolding** *quotient_map_def*
by (*metis* (*no_types*, *lifting*) *Collect_cong* ‹*openin X U*› *image_mono*)
qed
then have *closed_map X X' f*
by (*simp add*: *L assms bijective_open_imp_closed_map quotient_imp_surjective_map*)
then show *?rhs*
using *L om* **by** (*simp add*: *quotient_imp_continuous_map quotient_imp_surjective_map*)
next
assume *?rhs*
then show *?lhs*
by (*simp add*: *continuous_closed_imp_quotient_map*)
qed

lemma *continuous_compose_quotient_map*:
assumes *f*: *quotient_map X X' f* **and** *g*: *continuous_map X X''* (*g* ∘ *f*)
shows *continuous_map X' X'' g*
unfolding *quotient_map_def continuous_map_def*
proof (*intro conjI ballI allI impI*)
show *g* ∈ *topspace X'* → *topspace X''*
using *assms* **unfolding** *quotient_map_def Pi_iff*
by (*metis* (*no_types*, *opaque_lifting*) *continuous_map_image_subset_topspace image_comp image_subset_iff*)
next
fix *U''* :: *'c set*
assume *U''*: *openin X'' U''*
have *f* ‘ *topspace X* = *topspace X'*
by (*simp add*: *f quotient_imp_surjective_map*)
then have *eq*: {*x* ∈ *topspace X*. *f x* ∈ *topspace X'* ∧ *g* (*f x*) ∈ *U*} = {*x* ∈ *topspace X*. *g* (*f x*) ∈ *U*} **for** *U*
by *auto*
have *openin X* {*x* ∈ *topspace X*. *f x* ∈ *topspace X'* ∧ *g* (*f x*) ∈ *U''*}
unfolding *eq* **using** *U'' g openin_continuous_map_preimage* **by** *fastforce*

then have $*$: *openin* X $\{x \in$ *topspace* X. $f\,x \in \{x \in$ *topspace* X'. $g\,x \in U''\}\}$
by *auto*
show *openin* X' $\{x \in$ *topspace* X'. $g\,x \in U''\}$
using f **unfolding** *quotient_map_def*
by (*metis* (*no_types*) *Collect_subset* $*$)
qed

lemma *continuous_compose_quotient_map_eq*:
quotient_map $X\ X'\ f \Longrightarrow$ *continuous_map* $X\ X''\ (g \circ f) \longleftrightarrow$ *continuous_map* $X'\ X''\ g$
using *continuous_compose_quotient_map continuous_map_compose quotient_imp_continuous_map*
by *blast*

lemma *quotient_map_compose_eq*:
quotient_map $X\ X'\ f \Longrightarrow$ *quotient_map* $X\ X''\ (g \circ f) \longleftrightarrow$ *quotient_map* $X'\ X''\ g$
by (*meson continuous_compose_quotient_map_eq quotient_imp_continuous_map quotient_map_compose quotient_map_from_composition*)

lemma *quotient_map_restriction*:
assumes *quo*: *quotient_map* $X\ Y\ f$ **and** U: $\{x \in$ *topspace* X. $f\,x \in V\} = U$ **and** *disj*: *openin* $Y\ V \vee$ *closedin* $Y\ V$
shows *quotient_map* (*subtopology* $X\ U$) (*subtopology* $Y\ V$) f
using *disj*
proof
assume V: *openin* $Y\ V$
with U **have** *sub*: $U \subseteq$ *topspace* $X\ V \subseteq$ *topspace* Y
by (*auto simp*: *openin_subset*)
have *fim*: f ' *topspace* $X =$ *topspace* Y
and Y: $\bigwedge U$. $U \subseteq$ *topspace* $Y \Longrightarrow$ *openin* X $\{x \in$ *topspace* X. $f\,x \in U\} =$ *openin* $Y\ U$
using *quo* **unfolding** *quotient_map_def* **by** *auto*
have *openin* $X\ U$
using $U\ V\ Y$ *sub*(*2*) **by** *blast*
show *?thesis*
unfolding *quotient_map_def*
proof (*intro conjI allI impI*)
show f ' *topspace* (*subtopology* $X\ U$) = *topspace* (*subtopology* $Y\ V$)
using *sub* U *fim* **by** (*auto*)
next
fix Y' :: *'b set*
assume $Y' \subseteq$ *topspace* (*subtopology* $Y\ V$)
then have $Y' \subseteq$ *topspace* $Y\ Y' \subseteq V$
by (*simp_all*)
then have *eq*: $\{x \in$ *topspace* X. $x \in U \wedge f\,x \in Y'\} = \{x \in$ *topspace* X. $f\,x \in Y'\}$
using U **by** *blast*
then show *openin* (*subtopology* $X\ U$) $\{x \in$ *topspace* (*subtopology* $X\ U$). $f\,x \in Y'\}$ = *openin* (*subtopology* $Y\ V$) Y'

```
    using U V Y ‹openin X U› ‹Y' ⊆ topspace Y› ‹Y' ⊆ V›
    by (simp add: openin_open_subtopology eq) (auto simp: openin_closedin_eq)
  qed
next
  assume V: closedin Y V
  with U have sub: U ⊆ topspace X V ⊆ topspace Y
    by (auto simp: closedin_subset)
  have fim: f ` topspace X = topspace Y
    and Y: ⋀U. U ⊆ topspace Y ⟹ closedin X {x ∈ topspace X. f x ∈ U} =
closedin Y U
    using quo unfolding quotient_map_closedin by auto
  have closedin X U
    using U V Y sub(2) by blast
  show ?thesis
    unfolding quotient_map_closedin
  proof (intro conjI allI impI)
    show f ` topspace (subtopology X U) = topspace (subtopology Y V)
      using sub U fim by (auto)
  next
    fix Y' :: 'b set
    assume Y' ⊆ topspace (subtopology Y V)
    then have Y' ⊆ topspace Y Y' ⊆ V
      by (simp_all)
    then have eq: {x ∈ topspace X. x ∈ U ∧ f x ∈ Y'} = {x ∈ topspace X. f x ∈
Y'}
      using U by blast
    then show closedin (subtopology X U) {x ∈ topspace (subtopology X U). f x ∈
Y'} = closedin (subtopology Y V) Y'
      using U V Y ‹closedin X U› ‹Y' ⊆ topspace Y› ‹Y' ⊆ V›
      by (simp add: closedin_closed_subtopology eq) (auto simp: closedin_def)
  qed
qed

lemma quotient_map_saturated_open:
    quotient_map X Y f ⟷
      continuous_map X Y f ∧ f ` (topspace X) = topspace Y ∧
      (∀ U. openin X U ∧ {x ∈ topspace X. f x ∈ f ` U} ⊆ U ⟶ openin Y (f `
U))
    (is ?lhs = ?rhs)
proof
  assume L: ?lhs
  then have fim: f ` topspace X = topspace Y
    and Y: ⋀U. U ⊆ topspace Y ⟹ openin Y U = openin X {x ∈ topspace X. f
x ∈ U}
    unfolding quotient_map_def by auto
  show ?rhs
  proof (intro conjI allI impI)
    show continuous_map X Y f
      by (simp add: L quotient_imp_continuous_map)
```

```
    show f ‘ topspace X = topspace Y
      by (simp add: fim)
  next
    fix U :: 'a set
    assume U: openin X U ∧ {x ∈ topspace X. f x ∈ f ‘ U} ⊆ U
    then have sub: f ‘ U ⊆ topspace Y and eq: {x ∈ topspace X. f x ∈ f ‘ U} =
U
      using fim openin_subset by fastforce+
    show openin Y (f ‘ U)
      by (simp add: sub Y eq U)
  qed
next
  assume ?rhs
  then have YX: ⋀U. openin Y U ⟹ openin X {x ∈ topspace X. f x ∈ U}
       and fim: f ‘ topspace X = topspace Y
       and XY: ⋀U. ⟦openin X U; {x ∈ topspace X. f x ∈ f ‘ U} ⊆ U⟧ ⟹ openin
Y (f ‘ U)
    by (auto simp: quotient_map_def continuous_map_def)
  show ?lhs
  proof (simp add: quotient_map_def fim, intro allI impI iffI)
    fix U :: 'b set
    assume U ⊆ topspace Y and X: openin X {x ∈ topspace X. f x ∈ U}
    have feq: f ‘ {x ∈ topspace X. f x ∈ U} = U
      using ‹U ⊆ topspace Y› fim by auto
    show openin Y U
      using XY [OF X] by (simp add: feq)
  next
    fix U :: 'b set
    assume U ⊆ topspace Y and Y: openin Y U
    show openin X {x ∈ topspace X. f x ∈ U}
      by (metis YX [OF Y])
  qed
qed

lemma quotient_map_saturated_closed:
    quotient_map X Y f ⟷
      continuous_map X Y f ∧ f ‘ (topspace X) = topspace Y ∧
      (∀ U. closedin X U ∧ {x ∈ topspace X. f x ∈ f ‘ U} ⊆ U ⟶ closedin Y (f
‘ U))
    (is ?lhs = ?rhs)
proof
  assume L: ?lhs
  then obtain fim: f ‘ topspace X = topspace Y
    and Y: ⋀U. U ⊆ topspace Y ⟹ closedin Y U = closedin X {x ∈ topspace
X. f x ∈ U}
    by (simp add: quotient_map_closedin)
  show ?rhs
  proof (intro conjI allI impI)
    show continuous_map X Y f
```

```
    by (simp add: L quotient_imp_continuous_map)
   show f ‘ topspace X = topspace Y
    by (simp add: fim)
  next
   fix U :: 'a set
   assume U: closedin X U ∧ {x ∈ topspace X. f x ∈ f ‘ U} ⊆ U
   then have sub: f ‘ U ⊆ topspace Y and eq: {x ∈ topspace X. f x ∈ f ‘ U} =
U
    using fim closedin_subset by fastforce+
   show closedin Y (f ‘ U)
    by (simp add: sub Y eq U)
  qed
next
  assume ?rhs
  then obtain YX: ⋀U. closedin Y U ⟹ closedin X {x ∈ topspace X. f x ∈ U}
   and fim: f ‘ topspace X = topspace Y
   and XY: ⋀U. ⟦closedin X U; {x ∈ topspace X. f x ∈ f ‘ U} ⊆ U⟧ ⟹ closedin
Y (f ‘ U)
   by (simp add: continuous_map_closedin)
  show ?lhs
  proof (simp add: quotient_map_closedin fim, intro allI impI iffI)
   fix U :: 'b set
   assume U ⊆ topspace Y and X: closedin X {x ∈ topspace X. f x ∈ U}
   have feq: f ‘ {x ∈ topspace X. f x ∈ U} = U
    using ‹U ⊆ topspace Y› fim by auto
   show closedin Y U
    using XY [OF X] by (simp add: feq)
  next
   fix U :: 'b set
   assume U ⊆ topspace Y and Y: closedin Y U
   show closedin X {x ∈ topspace X. f x ∈ U}
    by (metis YX [OF Y])
  qed
qed

lemma quotient_map_onto_image:
  assumes f ‘ topspace X ⊆ topspace Y and U: ⋀U. U ⊆ topspace Y ⟹ openin
X {x ∈ topspace X. f x ∈ U} = openin Y U
  shows quotient_map X (subtopology Y (f ‘ topspace X)) f
  unfolding quotient_map_def topspace_subtopology
proof (intro conjI strip)
  fix U
  assume U ⊆ topspace Y ∩ f ‘ topspace X
  with U have openin X {x ∈ topspace X. f x ∈ U} ⟹ ∃x. openin Y x ∧ U =
f ‘ topspace X ∩ x
   by fastforce
  moreover have ∃x. openin Y x ∧ U = f ‘ topspace X ∩ x ⟹ openin X {x ∈
topspace X. f x ∈ U}
   by (metis (mono_tags, lifting) Collect_cong IntE IntI U image_eqI openin_subset)
```

ultimately show *openin X* $\{x \in$ *topspace X*. *f x* $\in U\}$ = *openin* (*subtopology Y* (*f* ‘ *topspace X*)) *U*
by (*force simp*: *openin_subtopology_alt image_iff*)
qed (*use assms* **in** *auto*)

lemma *quotient_map_lift_exists*:
assumes *f*: *quotient_map X Y f* **and** *h*: *continuous_map X Z h*
and $\bigwedge x\ y.$ $[\![x \in$ *topspace X*; *y* $\in$ *topspace X*; *f x* = *f y*$]\!] \Longrightarrow$ *h x* = *h y*
obtains *g* **where** *continuous_map Y Z g* ∧ *g* ‘ *topspace Y* = *h* ‘ *topspace X*
$\bigwedge x.$ *x* $\in$ *topspace X* $\Longrightarrow$ *g*(*f x*) = *h x*
proof −
obtain *g* **where** *g*: $\bigwedge x.$ *x* $\in$ *topspace X* $\Longrightarrow$ *h x* = *g*(*f x*)
using *function_factors_left_gen*[*of* $\lambda x.$ *x* $\in$ *topspace X f h*] *assms* **by** *blast*
show *?thesis*
proof
show *g* ‘ *topspace Y* = *h* ‘ *topspace X*
using *f g* **by** (*force dest*!: *quotient_imp_surjective_map*)
show *continuous_map Y Z g*
by (*smt* (*verit*) *f g h continuous_compose_quotient_map_eq continuous_map_eq o_def*)
qed (*simp add*: *g*)
qed

1.11.13 Separated Sets

definition *separatedin* :: *'a topology* ⇒ *'a set* ⇒ *'a set* ⇒ *bool*
where *separatedin X S T* ≡
S ⊆ *topspace X* ∧ *T* ⊆ *topspace X* ∧
S ∩ *X closure_of T* = {} ∧ *T* ∩ *X closure_of S* = {}

lemma *separatedin_empty* [*simp*]:
separatedin X S {} ⟷ *S* ⊆ *topspace X*
separatedin X {} *S* ⟷ *S* ⊆ *topspace X*
by (*simp_all add*: *separatedin_def*)

lemma *separatedin_refl* [*simp*]:
separatedin X S S ⟷ *S* = {}
by (*metis closure_of_subset empty_subsetI inf.orderE separatedin_def*)

lemma *separatedin_sym*:
separatedin X S T ⟷ *separatedin X T S*
by (*auto simp*: *separatedin_def*)

lemma *separatedin_imp_disjoint*:
separatedin X S T ⟹ *disjnt S T*
by (*meson closure_of_subset disjnt_def disjnt_subset2 separatedin_def*)

lemma *separatedin_mono*:
$[\![$*separatedin X S T*; *S′* ⊆ *S*; *T′* ⊆ *T*$]\!] \Longrightarrow$ *separatedin X S′ T′*

unfolding *separatedin_def*
using *closure_of_mono* **by** *blast*

lemma *separatedin_open_sets*:
⟦*openin X S*; *openin X T*⟧ ⟹ *separatedin X S T* ⟷ *disjnt S T*
unfolding *disjnt_def separatedin_def*
by (*auto simp*: *openin_Int_closure_of_eq_empty openin_subset*)

lemma *separatedin_closed_sets*:
⟦*closedin X S*; *closedin X T*⟧ ⟹ *separatedin X S T* ⟷ *disjnt S T*
unfolding *closure_of_eq disjnt_def separatedin_def*
by (*metis closedin_def closure_of_eq inf_commute*)

lemma *separatedin_subtopology*:
separatedin (*subtopology X U*) *S T* ⟷ *S* ⊆ *U* ∧ *T* ⊆ *U* ∧ *separatedin X S T*
by (*auto simp*: *separatedin_def closure_of_subtopology Int_ac disjoint_iff elim*!: *inf.orderE*)

lemma *separatedin_discrete_topology*:
separatedin (*discrete_topology U*) *S T* ⟷ *S* ⊆ *U* ∧ *T* ⊆ *U* ∧ *disjnt S T*
by (*metis openin_discrete_topology separatedin_def separatedin_open_sets topspace_discrete_topology*)

lemma *separated_eq_distinguishable*:
separatedin X {*x*} {*y*} ⟷
x ∈ *topspace X* ∧ *y* ∈ *topspace X* ∧
(∃ *U*. *openin X U* ∧ *x* ∈ *U* ∧ (*y* ∉ *U*)) ∧
(∃ *v*. *openin X v* ∧ *y* ∈ *v* ∧ (*x* ∉ *v*))
by (*force simp*: *separatedin_def closure_of_def*)

lemma *separatedin_Un* [*simp*]:
separatedin X S (*T* ∪ *U*) ⟷ *separatedin X S T* ∧ *separatedin X S U*
separatedin X (*S* ∪ *T*) *U* ⟷ *separatedin X S U* ∧ *separatedin X T U*
by (*auto simp*: *separatedin_def*)

lemma *separatedin_Union*:
finite $\mathcal{F}$ ⟹ *separatedin X S* (⋃$\mathcal{F}$) ⟷ *S* ⊆ *topspace X* ∧ (∀ *T* ∈ $\mathcal{F}$. *separatedin X S T*)
finite $\mathcal{F}$ ⟹ *separatedin X* (⋃$\mathcal{F}$) *S* ⟷ (∀ *T* ∈ $\mathcal{F}$. *separatedin X S T*) ∧ *S* ⊆ *topspace X*
by (*auto simp*: *separatedin_def closure_of_Union*)

lemma *separatedin_openin_diff*:
⟦*openin X S*; *openin X T*⟧ ⟹ *separatedin X* (*S* − *T*) (*T* − *S*)
unfolding *separatedin_def*
by (*metis Diff_Int_distrib2 Diff_disjoint Diff_empty Diff_mono empty_Diff empty_subsetI openin_Int_closure_of_eq_empty openin_subset*)

lemma *separatedin_closedin_diff*:
assumes *closedin X S closedin X T*

shows *separatedin X* (*S* − *T*) (*T* − *S*)
proof −
have *S* − *T* ⊆ *topspace X T* − *S* ⊆ *topspace X*
using *assms closedin_subset* **by** *auto*
with *assms* **show** *?thesis*
by (*simp add*: *separatedin_def Diff_Int_distrib2 closure_of_minimal inf_absorb2*)
qed

lemma *separation_closedin_Un_gen*:
separatedin X S T ⟷
S ⊆ *topspace X* ∧ *T* ⊆ *topspace X* ∧ *disjnt S T* ∧
closedin (*subtopology X* (*S* ∪ *T*)) *S* ∧
closedin (*subtopology X* (*S* ∪ *T*)) *T*
by (*auto simp add*: *separatedin_def closedin_Int_closure_of disjnt_iff dest*: *closure_of_subset*)

lemma *separation_openin_Un_gen*:
separatedin X S T ⟷
S ⊆ *topspace X* ∧ *T* ⊆ *topspace X* ∧ *disjnt S T* ∧
openin (*subtopology X* (*S* ∪ *T*)) *S* ∧
openin (*subtopology X* (*S* ∪ *T*)) *T*
unfolding *openin_closedin_eq topspace_subtopology separation_closedin_Un_gen disjnt_def*
by (*auto simp*: *Diff_triv Int_commute Un_Diff inf_absorb1 topspace_def*)

lemma *separatedin_full*:
S ∪ *T* = *topspace X*
⟹ *separatedin X S T* ⟷ *disjnt S T* ∧ *closedin X S* ∧ *openin X S* ∧ *closedin X T* ∧ *openin X T*
by (*metis separatedin_open_sets separation_closedin_Un_gen separation_openin_Un_gen subtopology_topspace*)

1.11.14 Homeomorphisms

(1-way and 2-way versions may be useful in places)

definition *homeomorphic_map* :: *'a topology* ⇒ *'b topology* ⇒ (*'a* ⇒ *'b*) ⇒ *bool*
where
homeomorphic_map X Y f ≡ *quotient_map X Y f* ∧ *inj_on f* (*topspace X*)

definition *homeomorphic_maps* :: *'a topology* ⇒ *'b topology* ⇒ (*'a* ⇒ *'b*) ⇒ (*'b* ⇒ *'a*) ⇒ *bool*
where
homeomorphic_maps X Y f g ≡
continuous_map X Y f ∧ *continuous_map Y X g* ∧
(∀ *x* ∈ *topspace X*. *g*(*f x*) = *x*) ∧ (∀ *y* ∈ *topspace Y*. *f*(*g y*) = *y*)

lemma *homeomorphic_map_eq*:
⟦*homeomorphic_map X Y f*; ⋀*x*. *x* ∈ *topspace X* ⟹ *f x* = *g x*⟧ ⟹ *homeo-*

morphic_map X Y g
by (*meson homeomorphic_map_def inj_on_cong quotient_map_eq*)

lemma *homeomorphic_maps_eq*:
⟦*homeomorphic_maps X Y f g*;
⋀*x*. *x* ∈ *topspace X* ⟹ *f x* = *f′ x*; ⋀*y*. *y* ∈ *topspace Y* ⟹ *g y* = *g′ y*⟧
⟹ *homeomorphic_maps X Y f′ g′*
unfolding *homeomorphic_maps_def*
by (*metis continuous_map_eq continuous_map_image_subset_topspace image_subset_iff*)

lemma *homeomorphic_maps_sym*:
homeomorphic_maps X Y f g ⟷ *homeomorphic_maps Y X g f*
by (*auto simp*: *homeomorphic_maps_def*)

lemma *homeomorphic_maps_id*:
homeomorphic_maps X Y id id ⟷ *Y* = *X* (**is** *?lhs* = *?rhs*)
proof
assume *L*: *?lhs*
then have *topspace X* = *topspace Y*
by (*auto simp*: *homeomorphic_maps_def continuous_map_def*)
with *L* **show** *?rhs*
unfolding *homeomorphic_maps_def*
by (*metis topology_finer_continuous_id topology_eq*)
next
assume *?rhs*
then show *?lhs*
unfolding *homeomorphic_maps_def* **by** *auto*
qed

lemma *homeomorphic_map_id* [*simp*]: *homeomorphic_map X Y id* ⟷ *Y* = *X*
(**is** *?lhs* = *?rhs*)
proof
assume *L*: *?lhs*
then have *eq*: *topspace X* = *topspace Y*
by (*auto simp*: *homeomorphic_map_def continuous_map_def quotient_map_def*)
then have ⋀*S*. *openin X S* ⟶ *openin Y S*
by (*meson L homeomorphic_map_def injective_quotient_map topology_finer_open_id*)
then show *?rhs*
using *L* **unfolding** *homeomorphic_map_def*
by (*metis eq quotient_imp_continuous_map topology_eq topology_finer_continuous_id*)
next
assume *?rhs*
then show *?lhs*
unfolding *homeomorphic_map_def*
by (*simp add*: *closed_map_id continuous_closed_imp_quotient_map*)
qed

lemma *homeomorphic_map_compose*:

assumes *homeomorphic_map X Y f homeomorphic_map Y X'' g*
shows *homeomorphic_map X X'' (g ∘ f)*
proof −
have *inj_on g (f ‘ topspace X)*
by (*metis* (*no_types*) *assms homeomorphic_map_def quotient_imp_surjective_map*)
then show *?thesis*
using *assms* **by** (*meson comp_inj_on homeomorphic_map_def quotient_map_compose_eq*)
qed

lemma *homeomorphic_maps_compose*:
homeomorphic_maps X Y f h ∧
homeomorphic_maps Y X'' g k
⟹ homeomorphic_maps X X'' (g ∘ f) (h ∘ k)
unfolding *homeomorphic_maps_def*
by (*auto simp*: *continuous_map_compose*; *simp add*: *continuous_map_def Pi_iff*)

lemma *homeomorphic_eq_everything_map*:
homeomorphic_map X Y f ⟷
continuous_map X Y f ∧ open_map X Y f ∧ closed_map X Y f ∧
f ‘ (topspace X) = topspace Y ∧ inj_on f (topspace X)
unfolding *homeomorphic_map_def*
by (*force simp*: *injective_quotient_map intro*: *injective_quotient_map*)

lemma *homeomorphic_imp_continuous_map*:
homeomorphic_map X Y f ⟹ continuous_map X Y f
by (*simp add*: *homeomorphic_eq_everything_map*)

lemma *homeomorphic_imp_open_map*:
homeomorphic_map X Y f ⟹ open_map X Y f
by (*simp add*: *homeomorphic_eq_everything_map*)

lemma *homeomorphic_imp_closed_map*:
homeomorphic_map X Y f ⟹ closed_map X Y f
by (*simp add*: *homeomorphic_eq_everything_map*)

lemma *homeomorphic_imp_surjective_map*:
homeomorphic_map X Y f ⟹ f ‘ topspace X = topspace Y
using *homeomorphic_eq_everything_map* **by** *fastforce*

lemma *homeomorphic_imp_injective_map*:
homeomorphic_map X Y f ⟹ inj_on f (topspace X)
by (*simp add*: *homeomorphic_eq_everything_map*)

lemma *bijective_open_imp_homeomorphic_map*:
⟦*continuous_map X Y f*; *open_map X Y f*; *f ‘ (topspace X) = topspace Y*;
inj_on f (topspace X)⟧
⟹ homeomorphic_map X Y f
by (*simp add*: *homeomorphic_map_def continuous_open_imp_quotient_map*)

lemma *bijective_closed_imp_homeomorphic_map*:
⟦*continuous_map X Y f*; *closed_map X Y f*; *f ‘ (topspace X) = topspace Y*; *inj_on f (topspace X)*⟧
⟹ *homeomorphic_map X Y f*
by (*simp add*: *continuous_closed_quotient_map homeomorphic_map_def*)

lemma *open_eq_continuous_inverse_map*:
assumes *X*: ⋀*x*. *x* ∈ *topspace X* ⟹ *f x* ∈ *topspace Y* ∧ *g*(*f x*) = *x*
and *Y*: ⋀*y*. *y* ∈ *topspace Y* ⟹ *g y* ∈ *topspace X* ∧ *f*(*g y*) = *y*
shows *open_map X Y f* ⟷ *continuous_map Y X g*
proof −
have *eq*: {*x* ∈ *topspace Y*. *g x* ∈ *U*} = *f ‘ U* **if** *openin X U* **for** *U*
using *openin_subset* [*OF that*] **by** (*force simp*: *X Y image_iff*)
show *?thesis*
by (*auto simp*: *Y open_map_def continuous_map_def eq*)
qed

lemma *closed_eq_continuous_inverse_map*:
assumes *X*: ⋀*x*. *x* ∈ *topspace X* ⟹ *f x* ∈ *topspace Y* ∧ *g*(*f x*) = *x*
and *Y*: ⋀*y*. *y* ∈ *topspace Y* ⟹ *g y* ∈ *topspace X* ∧ *f*(*g y*) = *y*
shows *closed_map X Y f* ⟷ *continuous_map Y X g*
proof −
have *eq*: {*x* ∈ *topspace Y*. *g x* ∈ *U*} = *f ‘ U* **if** *closedin X U* **for** *U*
using *closedin_subset* [*OF that*] **by** (*force simp*: *X Y image_iff*)
show *?thesis*
by (*auto simp*: *Y closed_map_def continuous_map_closedin eq*)
qed

lemma *homeomorphic_maps_map*:
homeomorphic_maps X Y f g ⟷
homeomorphic_map X Y f ∧ *homeomorphic_map Y X g* ∧
(∀ *x* ∈ *topspace X*. *g*(*f x*) = *x*) ∧ (∀ *y* ∈ *topspace Y*. *f*(*g y*) = *y*)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then have *L*: *continuous_map X Y f continuous_map Y X g* ∀ *x*∈*topspace X*. *g* (*f x*) = *x* ∀ *x*′∈*topspace Y*. *f* (*g x*′) = *x*′
by (*auto simp*: *homeomorphic_maps_def*)
show *?rhs*
proof (*intro conjI bijective_open_imp_homeomorphic_map L*)
show *open_map X Y f*
using *L* **using** *open_eq_continuous_inverse_map* [*of concl*: *X Y f g*]
by (*simp add*: *continuous_map_def Pi_iff*)
show *open_map Y X g*
using *L* **using** *open_eq_continuous_inverse_map* [*of concl*: *Y X g f*]
by (*simp add*: *continuous_map_def Pi_iff*)
show *f ‘ topspace X = topspace Y g ‘ topspace Y = topspace X*
using *L* **by** (*force simp*: *continuous_map_closedin Pi_iff*)+
show *inj_on f* (*topspace X*) *inj_on g* (*topspace Y*)

using *L* **unfolding** *inj_on_def* **by** *metis+*
qed
next
assume *?rhs*
then show *?lhs*
by (*auto simp*: *homeomorphic_maps_def homeomorphic_imp_continuous_map*)
qed

lemma *homeomorphic_maps_imp_map*:
homeomorphic_maps X Y f g ⟹ *homeomorphic_map X Y f*
using *homeomorphic_maps_map* **by** *blast*

lemma *homeomorphic_map_maps*:
homeomorphic_map X Y f ⟷ (∃ *g*. *homeomorphic_maps X Y f g*)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then have *L*: *continuous_map X Y f open_map X Y f closed_map X Y f*
f ‘ (*topspace X*) = *topspace Y inj_on f* (*topspace X*)
by (*auto simp*: *homeomorphic_eq_everything_map*)
have *X*: ⋀*x*. *x* ∈ *topspace X* ⟹ *f x* ∈ *topspace Y* ∧ *inv_into* (*topspace X*) *f* (*f x*) = *x*
using *L* **by** *auto*
have *Y*: ⋀*y*. *y* ∈ *topspace Y* ⟹ *inv_into* (*topspace X*) *f y* ∈ *topspace X* ∧ *f* (*inv_into* (*topspace X*) *f y*) = *y*
by (*simp add*: *L f_inv_into_f inv_into_into*)
have *homeomorphic_maps X Y f* (*inv_into* (*topspace X*) *f*)
unfolding *homeomorphic_maps_def*
proof (*intro conjI L*)
show *continuous_map Y X* (*inv_into* (*topspace X*) *f*)
by (*simp add*: *L X Y flip*: *open_eq_continuous_inverse_map* [**where** *f*=*f*])
next
show ∀ *x*∈*topspace X*. *inv_into* (*topspace X*) *f* (*f x*) = *x*
∀ *y*∈*topspace Y*. *f* (*inv_into* (*topspace X*) *f y*) = *y*
using *X Y* **by** *auto*
qed
then show *?rhs*
by *metis*
next
assume *?rhs*
then show *?lhs*
using *homeomorphic_maps_map* **by** *blast*
qed

lemma *homeomorphic_maps_involution*:
⟦*continuous_map X X f*; ⋀*x*. *x* ∈ *topspace X* ⟹ *f*(*f x*) = *x*⟧ ⟹ *homeomorphic_maps X X f f*
by (*auto simp*: *homeomorphic_maps_def*)

lemma *homeomorphic_map_involution*:
⟦*continuous_map X X f*; ⋀*x*. *x* ∈ *topspace X* ⟹ *f*(*f x*) = *x*⟧ ⟹ *homeomorphic_map X X f*
using *homeomorphic_maps_involution homeomorphic_maps_map* **by** *blast*

lemma *homeomorphic_map_openness*:
assumes *hom*: *homeomorphic_map X Y f* **and** *U*: *U* ⊆ *topspace X*
shows *openin Y* (*f* ‘ *U*) ⟷ *openin X U*
proof −
obtain *g* **where** *homeomorphic_maps X Y f g*
using *assms* **by** (*auto simp*: *homeomorphic_map_maps*)
then have *g*: *homeomorphic_map Y X g* **and** *gf*: ⋀*x*. *x* ∈ *topspace X* ⟹ *g*(*f x*) = *x*
by (*auto simp*: *homeomorphic_maps_map*)
then have *openin X U* ⟹ *openin Y* (*f* ‘ *U*)
using *hom homeomorphic_imp_open_map open_map_def* **by** *blast*
show *openin Y* (*f* ‘ *U*) = *openin X U*
proof
assume *L*: *openin Y* (*f* ‘ *U*)
have *U* = *g* ‘ (*f* ‘ *U*)
using *U gf* **by** *force*
then show *openin X U*
by (*metis L homeomorphic_imp_open_map open_map_def g*)
next
assume *openin X U*
then show *openin Y* (*f* ‘ *U*)
using *hom homeomorphic_imp_open_map open_map_def* **by** *blast*
qed
qed

lemma *homeomorphic_map_closedness*:
assumes *hom*: *homeomorphic_map X Y f* **and** *U*: *U* ⊆ *topspace X*
shows *closedin Y* (*f* ‘ *U*) ⟷ *closedin X U*
proof −
obtain *g* **where** *homeomorphic_maps X Y f g*
using *assms* **by** (*auto simp*: *homeomorphic_map_maps*)
then have *g*: *homeomorphic_map Y X g* **and** *gf*: ⋀*x*. *x* ∈ *topspace X* ⟹ *g*(*f x*) = *x*
by (*auto simp*: *homeomorphic_maps_map*)
then have *closedin X U* ⟹ *closedin Y* (*f* ‘ *U*)
using *hom homeomorphic_imp_closed_map closed_map_def* **by** *blast*
show *closedin Y* (*f* ‘ *U*) = *closedin X U*
proof
assume *L*: *closedin Y* (*f* ‘ *U*)
have *U* = *g* ‘ (*f* ‘ *U*)
using *U gf* **by** *force*
then show *closedin X U*
by (*metis L homeomorphic_imp_closed_map closed_map_def g*)

next
assume *closedin X U*
then show *closedin Y (f ‘ U)*
using *hom homeomorphic_imp_closed_map closed_map_def* **by** *blast*
qed
qed

lemma *homeomorphic_map_openness_eq*:
homeomorphic_map X Y f ⟹ openin X U ⟷ U ⊆ topspace X ∧ openin Y (f ‘ U)
by (*meson homeomorphic_map_openness openin_closedin_eq*)

lemma *homeomorphic_map_closedness_eq*:
homeomorphic_map X Y f ⟹ closedin X U ⟷ U ⊆ topspace X ∧ closedin Y (f ‘ U)
by (*meson closedin_subset homeomorphic_map_closedness*)

lemma *all_openin_homeomorphic_image*:
assumes *homeomorphic_map X Y f*
shows (∀ *V*. *openin Y V ⟶ P V*) ⟷ (∀ *U*. *openin X U ⟶ P*(*f ‘ U*))
by (*metis* (*no_types*, *lifting*) *assms homeomorphic_eq_everything_map homeomorphic_map_openness openin_subset subset_image_iff*)

lemma *all_closedin_homeomorphic_image*:
assumes *homeomorphic_map X Y f*
shows (∀ *V*. *closedin Y V ⟶ P V*) ⟷ (∀ *U*. *closedin X U ⟶ P*(*f ‘ U*))
by (*metis* (*no_types*, *lifting*) *assms closedin_subset homeomorphic_eq_everything_map homeomorphic_map_closedness subset_image_iff*)

lemma *homeomorphic_map_derived_set_of*:
assumes *hom*: *homeomorphic_map X Y f* **and** *S*: *S ⊆ topspace X*
shows *Y derived_set_of (f ‘ S) = f ‘ (X derived_set_of S)*
proof −
have *fim*: *f ‘ (topspace X) = topspace Y* **and** *inj*: *inj_on f (topspace X)*
using *hom* **by** (*auto simp*: *homeomorphic_eq_everything_map*)
have *iff*: (∀ *T*. *x ∈ T ∧ openin X T ⟶* (∃ *y*. *y ≠ x ∧ y ∈ S ∧ y ∈ T*)) =
(∀ *T*. *T ⊆ topspace Y ⟶ f x ∈ T ⟶ openin Y T ⟶* (∃ *y*. *y ≠ f x ∧ y ∈ f ‘ S ∧ y ∈ T*))
if *x ∈ topspace X* **for** *x*
proof −
have §: (*x ∈ T ∧ openin X T*) = (*T ⊆ topspace X ∧ f x ∈ f ‘ T ∧ openin Y (f ‘ T)*) **for** *T*
by (*meson hom homeomorphic_map_openness_eq inj inj_on_image_mem_iff that*)
moreover have (∃ *y*. *y ≠ x ∧ y ∈ S ∧ y ∈ T*) = (∃ *y*. *y ≠ f x ∧ y ∈ f ‘ S ∧ y ∈ f ‘ T*) (**is** *?lhs = ?rhs*)
if *T ⊆ topspace X ∧ f x ∈ f ‘ T ∧ openin Y (f ‘ T)* **for** *T*
by (*smt* (*verit*, *del_insts*) *S ‹x ∈ topspace X› image_iff inj inj_on_def*

subsetD that)
ultimately show *?thesis*
by (*auto simp flip*: *fim simp*: *all_subset_image*)
qed
have ∗: ⟦$T = f$ ‘ S; ⋀x. $x \in S \Longrightarrow P\ x \longleftrightarrow Q(f\ x)$⟧ ⟹ {$y$. $y \in T \wedge Q\ y$} = f ‘ {$x \in S$. $P\ x$} **for** $T\ S\ P\ Q$
by *auto*
show *?thesis*
unfolding *derived_set_of_def*
by (*rule* ∗) (*use fim iff openin_subset* **in** *force*)+
qed

lemma *homeomorphic_map_closure_of*:
assumes *hom*: *homeomorphic_map X Y f* **and** *S*: $S \subseteq$ *topspace X*
shows *Y closure_of* (f ‘ S) = f ‘ (*X closure_of S*)
unfolding *closure_of*
using *homeomorphic_imp_surjective_map* [*OF hom*] *S*
by (*auto simp*: *in_derived_set_of homeomorphic_map_derived_set_of* [*OF assms*])

lemma *homeomorphic_map_interior_of*:
assumes *hom*: *homeomorphic_map X Y f* **and** *S*: $S \subseteq$ *topspace X*
shows *Y interior_of* (f ‘ S) = f ‘ (*X interior_of S*)
proof −
{ **fix** y
assume $y \in$ *topspace Y* **and** $y \notin$ *Y closure_of* (*topspace Y* − f ‘ S)
then have $y \in f$ ‘ (*topspace X* − *X closure_of* (*topspace X* − S))
using *homeomorphic_eq_everything_map* [*THEN iffD1*, *OF hom*] *homeomorphic_map_closure_of* [*OF hom*]
by (*metis DiffI Diff_subset S closure_of_subset_topspace inj_on_image_set_diff*)
}
moreover
{ **fix** x
assume $x \in$ *topspace X*
then have $f\ x \in$ *topspace Y*
using *hom homeomorphic_imp_surjective_map* **by** *blast* }
moreover
{ **fix** x
assume $x \in$ *topspace X* **and** $x \notin$ *X closure_of* (*topspace X* − S) **and** $f\ x \in$ *Y closure_of* (*topspace Y* − f ‘ S)
then have *False*
using *homeomorphic_map_closure_of* [*OF hom*] *hom*
unfolding *homeomorphic_eq_everything_map*
by (*metis Diff_subset S closure_of_subset_topspace inj_on_image_mem_iff inj_on_image_set_diff*)
}
ultimately show *?thesis*
by (*auto simp*: *interior_of_closure_of*)
qed

lemma *homeomorphic_map_frontier_of*:
assumes *hom*: *homeomorphic_map X Y f* **and** *S*: *S* ⊆ *topspace X*
shows *Y frontier_of* (*f* ‘ *S*) = *f* ‘ (*X frontier_of S*)
unfolding *frontier_of_def*
proof (*intro equalityI subsetI DiffI*)
fix *y*
assume *y* ∈ *Y closure_of f* ‘ *S* − *Y interior_of f* ‘ *S*
then show *y* ∈ *f* ‘ (*X closure_of S* − *X interior_of S*)
using *S hom homeomorphic_map_closure_of homeomorphic_map_interior_of*
by *fastforce*
next
fix *y*
assume *y* ∈ *f* ‘ (*X closure_of S* − *X interior_of S*)
then show *y* ∈ *Y closure_of f* ‘ *S*
using *S hom homeomorphic_map_closure_of* **by** *fastforce*
next
fix *x*
assume *x* ∈ *f* ‘ (*X closure_of S* − *X interior_of S*)
then obtain *y* **where** *y*: *x* = *f y y* ∈ *X closure_of S y* ∉ *X interior_of S*
by *blast*
then show *x* ∉ *Y interior_of f* ‘ *S*
using *S hom homeomorphic_map_interior_of y*(*1*)
unfolding *homeomorphic_map_def*
by (*smt* (*verit*, *ccfv_SIG*) *in_closure_of inj_on_image_mem_iff interior_of_subset_topspace*)

qed

lemma *homeomorphic_maps_subtopologies*:
⟦*homeomorphic_maps X Y f g*; *f* ‘ (*topspace X* ∩ *S*) = *topspace Y* ∩ *T*⟧
⟹ *homeomorphic_maps* (*subtopology X S*) (*subtopology Y T*) *f g*
unfolding *homeomorphic_maps_def*
by (*force simp*: *continuous_map_from_subtopology continuous_map_in_subtopology*)

lemma *homeomorphic_maps_subtopologies_alt*:
⟦*homeomorphic_maps X Y f g*; *f* ‘ (*topspace X* ∩ *S*) ⊆ *T*; *g* ‘ (*topspace Y* ∩
T) ⊆ *S*⟧
⟹ *homeomorphic_maps* (*subtopology X S*) (*subtopology Y T*) *f g*
unfolding *homeomorphic_maps_def*
by (*force simp*: *continuous_map_from_subtopology continuous_map_in_subtopology*)

lemma *homeomorphic_map_subtopologies*:
⟦*homeomorphic_map X Y f*; *f* ‘ (*topspace X* ∩ *S*) = *topspace Y* ∩ *T*⟧
⟹ *homeomorphic_map* (*subtopology X S*) (*subtopology Y T*) *f*
by (*meson homeomorphic_map_maps homeomorphic_maps_subtopologies*)

lemma *homeomorphic_map_subtopologies_alt*:
assumes *hom*: *homeomorphic_map X Y f*
and *S*: ⋀*x*. ⟦*x* ∈ *topspace X*; *f x* ∈ *topspace Y*⟧ ⟹ *f x* ∈ *T* ⟷ *x* ∈ *S*

shows *homeomorphic_map* (*subtopology X S*) (*subtopology Y T*) *f*
proof −
have *homeomorphic_maps* (*subtopology X S*) (*subtopology Y T*) *f g*
if *homeomorphic_maps X Y f g* **for** *g*
proof (*rule homeomorphic_maps_subtopologies* [*OF that*])
have *f ' (topspace X ∩ S) ⊆ topspace Y ∩ T*
using *S hom homeomorphic_imp_surjective_map* **by** *fastforce*
then show *f ' (topspace X ∩ S) = topspace Y ∩ T*
using *that* **unfolding** *homeomorphic_maps_def continuous_map_def Pi_iff*
by (*smt* (*verit*, *del_insts*) *Int_iff S image_iff subsetI subset_antisym*)
qed
then show *?thesis*
using *hom* **by** (*meson homeomorphic_map_maps*)
qed

1.11.15 Relation of homeomorphism between topological spaces

definition *homeomorphic_space* (**infixr** *homeomorphic′_space 50*)
where *X homeomorphic_space Y ≡ ∃f g. homeomorphic_maps X Y f g*

lemma *homeomorphic_space_refl* [*iff*]: *X homeomorphic_space X*
by (*meson homeomorphic_maps_id homeomorphic_space_def*)

lemma *homeomorphic_space_sym*:
X homeomorphic_space Y ⟷ Y homeomorphic_space X
unfolding *homeomorphic_space_def* **by** (*metis homeomorphic_maps_sym*)

lemma *homeomorphic_space_trans* [*trans*]:
⟦X1 homeomorphic_space X2; *X2 homeomorphic_space X3⟧ ⟹ X1 homeomorphic_space X3*
unfolding *homeomorphic_space_def* **by** (*metis homeomorphic_maps_compose*)

lemma *homeomorphic_space*:
X homeomorphic_space Y ⟷ (∃f. homeomorphic_map X Y f)
by (*simp add*: *homeomorphic_map_maps homeomorphic_space_def*)

lemma *homeomorphic_maps_imp_homeomorphic_space*:
homeomorphic_maps X Y f g ⟹ X homeomorphic_space Y
unfolding *homeomorphic_space_def* **by** *metis*

lemma *homeomorphic_map_imp_homeomorphic_space*:
homeomorphic_map X Y f ⟹ X homeomorphic_space Y
unfolding *homeomorphic_map_maps*
using *homeomorphic_space_def* **by** *blast*

lemma *homeomorphic_empty_space*:
X homeomorphic_space Y ⟹ X = trivial_topology ⟷ Y = trivial_topology
by (*meson continuous_map_on_empty2 homeomorphic_maps_def homeomorphic_space_def*)

lemma *homeomorphic_empty_space_eq*:
assumes $X = \mathit{trivial\_topology}$
shows X *homeomorphic_space* $Y \longleftrightarrow Y = \mathit{trivial\_topology}$
using *assms funcset_mem*
by (*fastforce simp*: *homeomorphic_maps_def homeomorphic_space_def continuous_map_def*)

lemma *homeomorphic_space_unfold*:
assumes X *homeomorphic_space* Y
obtains $f\ g$ **where** *homeomorphic_map* $X\ Y\ f$ *homeomorphic_map* $Y\ X\ g$
and $\bigwedge x.\ x \in \mathit{topspace}\ X \Longrightarrow g(f\ x) = x$ $\bigwedge y.\ y \in \mathit{topspace}\ Y \Longrightarrow f(g\ y) = y$
and $f \in \mathit{topspace}\ X \to \mathit{topspace}\ Y$ $g \in \mathit{topspace}\ Y \to \mathit{topspace}\ X$
using *assms* **unfolding** *homeomorphic_space_def homeomorphic_maps_map*
by (*smt* (*verit*, *best*) *Pi_I homeomorphic_imp_surjective_map image_eqI*)

1.11.16 Connected topological spaces

definition *connected_space* :: *'a topology* ⇒ *bool* **where**
connected_space $X \equiv$
$\neg(\exists\, E1\ E2.\ \mathit{openin}\ X\ E1 \wedge \mathit{openin}\ X\ E2 \wedge$
$\mathit{topspace}\ X \subseteq E1 \cup E2 \wedge E1 \cap E2 = \{\} \wedge E1 \neq \{\} \wedge E2 \neq \{\})$

definition *connectedin* :: *'a topology* ⇒ *'a set* ⇒ *bool* **where**
connectedin $X\ S \equiv S \subseteq \mathit{topspace}\ X \wedge \mathit{connected\_space}\ (\mathit{subtopology}\ X\ S)$

lemma *connected_spaceD*:
⟦*connected_space* X;
openin $X\ U$; *openin* $X\ V$; $\mathit{topspace}\ X \subseteq U \cup V$; $U \cap V = \{\}$; $U \neq \{\}$; $V \neq \{\}$⟧ ⟹ *False*
by (*auto simp*: *connected_space_def*)

lemma *connectedin_subset_topspace*: *connectedin* $X\ S \Longrightarrow S \subseteq \mathit{topspace}\ X$
by (*simp add*: *connectedin_def*)

lemma *connectedin_topspace*:
connectedin $X\ (\mathit{topspace}\ X) \longleftrightarrow \mathit{connected\_space}\ X$
by (*simp add*: *connectedin_def*)

lemma *connected_space_subtopology*:
connectedin $X\ S \Longrightarrow \mathit{connected\_space}\ (\mathit{subtopology}\ X\ S)$
by (*simp add*: *connectedin_def*)

lemma *connectedin_subtopology*:
connectedin $(\mathit{subtopology}\ X\ S)\ T \longleftrightarrow \mathit{connectedin}\ X\ T \wedge T \subseteq S$
by (*force simp*: *connectedin_def subtopology_subtopology inf_absorb2*)

lemma *connected_space_eq*:
connected_space $X \longleftrightarrow$

(∄*E1 E2*. *openin X E1* ∧ *openin X E2* ∧ *E1* ∪ *E2* = *topspace X* ∧ *E1* ∩ *E2* = {} ∧ *E1* ≠ {} ∧ *E2* ≠ {})
unfolding *connected_space_def*
by (*metis openin_Un openin_subset subset_antisym*)

lemma *connected_space_closedin*:
connected_space X ⟷
(∄*E1 E2*. *closedin X E1* ∧ *closedin X E2* ∧ *topspace X* ⊆ *E1* ∪ *E2* ∧
E1 ∩ *E2* = {} ∧ *E1* ≠ {} ∧ *E2* ≠ {}) (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then have ⋀*E1 E2*. ⟦*openin X E1*; *E1* ∩ *E2* = {}; *topspace X* ⊆ *E1* ∪ *E2*; *openin X E2*⟧ ⟹ *E1* = {} ∨ *E2* = {}
by (*simp add*: *connected_space_def*)
then show *?rhs*
unfolding *connected_space_def*
by (*metis disjnt_def separatedin_closed_sets separation_openin_Un_gen subtopology_superset*)
next
assume *R*: *?rhs*
then show *?lhs*
unfolding *connected_space_def*
by (*metis Diff_triv Int_commute separatedin_openin_diff separation_closedin_Un_gen subtopology_superset*)
qed

lemma *connected_space_closedin_eq*:
connected_space X ⟷
(∄*E1 E2*. *closedin X E1* ∧ *closedin X E2* ∧
E1 ∪ *E2* = *topspace X* ∧ *E1* ∩ *E2* = {} ∧ *E1* ≠ {} ∧ *E2* ≠ {})
by (*metis closedin_Un closedin_def connected_space_closedin subset_antisym*)

lemma *connected_space_clopen_in*:
connected_space X ⟷
(∀ *T*. *openin X T* ∧ *closedin X T* ⟶ *T* = {} ∨ *T* = *topspace X*)
proof −
have *eq*: *openin X E1* ∧ *openin X E2* ∧ *E1* ∪ *E2* = *topspace X* ∧ *E1* ∩ *E2* = {} ∧ *P*
⟷ *E2* = *topspace X* − *E1* ∧ *openin X E1* ∧ *openin X E2* ∧ *P* **for** *E1 E2 P*
using *openin_subset* **by** *blast*
show *?thesis*
unfolding *connected_space_eq eq closedin_def*
by (*auto simp*: *openin_closedin_eq*)
qed

lemma *connectedin*:
connectedin X S ⟷
S ⊆ *topspace X* ∧

```
        (∄ E1 E2.
            openin X E1 ∧ openin X E2 ∧
            S ⊆ E1 ∪ E2 ∧ E1 ∩ E2 ∩ S = {} ∧ E1 ∩ S ≠ {} ∧ E2 ∩ S ≠ {})
(is ?lhs = ?rhs)
proof −
  have ∗: (∃ E1:: 'a set. ∃ E2:: 'a set. (∃ T1:: 'a set. P1 T1 ∧ E1 = f1 T1) ∧
(∃ T2:: 'a set. P2 T2 ∧ E2 = f2 T2) ∧
            R E1 E2) ⟷ (∃ T1 T2. P1 T1 ∧ P2 T2 ∧ R(f1 T1) (f2 T2)) for P1
f1 P2 f2 R
    by auto
  show ?thesis
   unfolding connectedin_def connected_space_def openin_subtopology topspace_subtopology
∗
   by (intro conj_cong arg_cong [where f=Not] ex_cong1; blast dest!: openin_subset)
qed

lemma connectedinD:
    ⟦connectedin X S; openin X E1; openin X E2; S ⊆ E1 ∪ E2; E1 ∩ E2 ∩ S =
{}; E1 ∩ S ≠ {}; E2 ∩ S ≠ {}⟧ ⟹ False
  by (meson connectedin)

lemma connectedin_iff_connected [simp]: connectedin euclidean S ⟷ connected
S
  by (simp add: connected_def connectedin)

lemma connectedin_closedin:
   connectedin X S ⟷
       S ⊆ topspace X ∧
       ¬(∃ E1 E2. closedin X E1 ∧ closedin X E2 ∧
                  S ⊆ (E1 ∪ E2) ∧
                  (E1 ∩ E2 ∩ S = {}) ∧
                  ¬(E1 ∩ S = {}) ∧ ¬(E2 ∩ S = {}))
proof −
  have ∗: (∃ E1:: 'a set. ∃ E2:: 'a set. (∃ T1:: 'a set. P1 T1 ∧ E1 = f1 T1) ∧
(∃ T2:: 'a set. P2 T2 ∧ E2 = f2 T2) ∧
            R E1 E2) ⟷ (∃ T1 T2. P1 T1 ∧ P2 T2 ∧ R(f1 T1) (f2 T2)) for P1
f1 P2 f2 R
    by auto
  show ?thesis
   unfolding connectedin_def connected_space_closedin closedin_subtopology topspace_subtopology
∗
   by (intro conj_cong arg_cong [where f=Not] ex_cong1; blast dest!: openin_subset)
qed

lemma connectedin_empty [simp]: connectedin X {}
  by (simp add: connectedin)

lemma connected_space_trivial_topology [simp]: connected_space trivial_topology
  using connectedin_topspace by fastforce
```

lemma *connectedin_sing* [*simp*]: *connectedin X* {*a*} ⟷ *a* ∈ *topspace X*
by (*simp add*: *connectedin*)

lemma *connectedin_absolute* [*simp*]:
connectedin (*subtopology X S*) *S* ⟷ *connectedin X S*
by (*simp add*: *connectedin_subtopology*)

lemma *connectedin_Union*:
assumes $\mathcal{U}$: ⋀*S*. *S* ∈ $\mathcal{U}$ ⟹ *connectedin X S* **and** *ne*: ⋂$\mathcal{U}$ ≠ {}
shows *connectedin X* (⋃$\mathcal{U}$)
proof −
have ⋃$\mathcal{U}$ ⊆ *topspace X*
using $\mathcal{U}$ **by** (*simp add*: *Union_least connectedin_def*)
moreover have *False*
if *openin X E1 openin X E2* **and** *cover*: ⋃$\mathcal{U}$ ⊆ *E1* ∪ *E2* **and** *disj*: *E1* ∩ *E2* ∩ ⋃$\mathcal{U}$ = {}
and *overlap1*: *E1* ∩ ⋃$\mathcal{U}$ ≠ {} **and** *overlap2*: *E2* ∩ ⋃$\mathcal{U}$ ≠ {}
for *E1 E2*
proof −
have *disjS*: *E1* ∩ *E2* ∩ *S* = {} **if** *S* ∈ $\mathcal{U}$ **for** *S*
using *Diff_triv that disj* **by** *auto*
have *coverS*: *S* ⊆ *E1* ∪ *E2* **if** *S* ∈ $\mathcal{U}$ **for** *S*
using *that cover* **by** *blast*
have $\mathcal{U}$ ≠ {}
using *overlap1* **by** *blast*
obtain *a* **where** *a*: ⋀*U*. *U* ∈ $\mathcal{U}$ ⟹ *a* ∈ *U*
using *ne* **by** *force*
with ‹$\mathcal{U}$ ≠ {}› **have** *a* ∈ ⋃$\mathcal{U}$
by *blast*
then consider *a* ∈ *E1* | *a* ∈ *E2*
using ‹⋃$\mathcal{U}$ ⊆ *E1* ∪ *E2*› **by** *auto*
then show *False*
proof *cases*
case *1*
then obtain *b S* **where** *b* ∈ *E2 b* ∈ *S S* ∈ $\mathcal{U}$
using *overlap2* **by** *blast*
then show *?thesis*
using *1* ‹*openin X E1*› ‹*openin X E2*› *disjS coverS a* [*OF* ‹*S* ∈ $\mathcal{U}$›] $\mathcal{U}$[*OF* ‹*S* ∈ $\mathcal{U}$›]
unfolding *connectedin*
by (*meson disjoint_iff_not_equal*)
next
case *2*
then obtain *b S* **where** *b* ∈ *E1 b* ∈ *S S* ∈ $\mathcal{U}$
using *overlap1* **by** *blast*
then show *?thesis*
using *2* ‹*openin X E1*› ‹*openin X E2*› *disjS coverS a* [*OF* ‹*S* ∈ $\mathcal{U}$›] $\mathcal{U}$[*OF* ‹*S* ∈ $\mathcal{U}$›]

```
      unfolding connectedin
      by (meson disjoint_iff_not_equal)
    qed
  qed
  ultimately show ?thesis
    unfolding connectedin by blast
qed

lemma connectedin_Un:
    ⟦connectedin X S; connectedin X T; S ∩ T ≠ {}⟧ ⟹ connectedin X (S ∪ T)
  using connectedin_Union [of {S,T}] by auto

lemma connected_space_subconnected:
  connected_space X ⟷ (∀ x ∈ topspace X. ∀ y ∈ topspace X. ∃ S. connectedin X
S ∧ x ∈ S ∧ y ∈ S) (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    using connectedin_topspace by blast
next
  assume R [rule_format]: ?rhs
  have False if openin X U openin X V and disj: U ∩ V = {} and cover: topspace
X ⊆ U ∪ V
    and U ≠ {} V ≠ {} for U V
  proof −
    obtain u v where u ∈ U v ∈ V
      using ‹U ≠ {}› ‹V ≠ {}› by auto
    then obtain T where u ∈ T v ∈ T and T: connectedin X T
      using R [of u v] that
      by (meson ‹openin X U› ‹openin X V› subsetD openin_subset)
    then show False
      using that unfolding connectedin
      by (metis IntI ‹u ∈ U› ‹v ∈ V› empty_iff inf_bot_left subset_trans)
  qed
  then show ?lhs
    by (auto simp: connected_space_def)
qed

lemma connectedin_intermediate_closure_of:
  assumes connectedin X S S ⊆ T T ⊆ X closure_of S
  shows connectedin X T
proof −
  have S: S ⊆ topspace X and T: T ⊆ topspace X
    using assms by (meson closure_of_subset_topspace dual_order.trans)+
  have §: ⋀E1 E2. ⟦openin X E1; openin X E2; E1 ∩ S = {} ∨ E2 ∩ S = {}⟧
⟹ E1 ∩ T = {} ∨ E2 ∩ T = {}
    using assms unfolding disjoint_iff by (meson in_closure_of subsetD)
  then show ?thesis
    using assms
```

unfolding *connectedin closure_of_subset_topspace S T*
by (*metis Int_empty_right T dual_order.trans inf.orderE inf_left_commute*)
qed

lemma *connectedin_closure_of*:
connectedin X S ⟹ *connectedin X* (*X closure_of S*)
by (*meson closure_of_subset connectedin_def connectedin_intermediate_closure_of subset_refl*)

lemma *connectedin_separation*:
connectedin X S ⟷
S ⊆ *topspace X* ∧
(∄*C1 C2*. *C1* ∪ *C2* = *S* ∧ *C1* ≠ {} ∧ *C2* ≠ {} ∧ *C1* ∩ *X closure_of C2* = {} ∧ *C2* ∩ *X closure_of C1* = {})
unfolding *connectedin_def connected_space_closedin_eq closedin_Int_closure_of topspace_subtopology*
apply (*intro conj_cong refl arg_cong* [**where** *f*=*Not*])
apply (*intro ex_cong1 iffI*, *blast*)
using *closure_of_subset_Int* **by** *force*

lemma *connectedin_eq_not_separated*:
connectedin X S ⟷
S ⊆ *topspace X* ∧
(∄*C1 C2*. *C1* ∪ *C2* = *S* ∧ *C1* ≠ {} ∧ *C2* ≠ {} ∧ *separatedin X C1 C2*)
unfolding *separatedin_def* **by** (*metis connectedin_separation sup.boundedE*)

lemma *connectedin_eq_not_separated_subset*:
connectedin X S ⟷
S ⊆ *topspace X* ∧ (∄*C1 C2*. *S* ⊆ *C1* ∪ *C2* ∧ *S* ∩ *C1* ≠ {} ∧ *S* ∩ *C2* ≠ {} ∧ *separatedin X C1 C2*)
proof −
have ∀*C1 C2*. *S* ⊆ *C1* ∪ *C2* ⟶ *S* ∩ *C1* = {} ∨ *S* ∩ *C2* = {} ∨ ¬ *separatedin X C1 C2*
if ⋀*C1 C2*. *C1* ∪ *C2* = *S* ⟶ *C1* = {} ∨ *C2* = {} ∨ ¬ *separatedin X C1 C2*
proof (*intro allI*)
fix *C1 C2*
show *S* ⊆ *C1* ∪ *C2* ⟶ *S* ∩ *C1* = {} ∨ *S* ∩ *C2* = {} ∨ ¬ *separatedin X C1 C2*
using *that* [*of S* ∩ *C1 S* ∩ *C2*]
by (*auto simp*: *separatedin_mono*)
qed
then show *?thesis*
by (*metis Un_Int_eq*(*1*) *Un_Int_eq*(*2*) *connectedin_eq_not_separated order_refl*)
qed

lemma *connected_space_eq_not_separated*:
connected_space X ⟷
(∄*C1 C2*. *C1* ∪ *C2* = *topspace X* ∧ *C1* ≠ {} ∧ *C2* ≠ {} ∧ *separatedin X C1*

C2)
by (*simp add*: *connectedin_eq_not_separated flip*: *connectedin_topspace*)

lemma *connected_space_eq_not_separated_subset*:
connected_space X ⟷
(∄*C1 C2*. *topspace X* ⊆ *C1* ∪ *C2* ∧ *C1* ≠ {} ∧ *C2* ≠ {} ∧ *separatedin X C1 C2*)
by (*metis connected_space_eq_not_separated le_sup_iff separatedin_def subset_antisym*)

lemma *connectedin_subset_separated_union*:
⟦*connectedin X C*; *separatedin X S T*; *C* ⊆ *S* ∪ *T*⟧ ⟹ *C* ⊆ *S* ∨ *C* ⊆ *T*
unfolding *connectedin_eq_not_separated_subset* **by** *blast*

lemma *connectedin_nonseparated_union*:
assumes *connectedin X S connectedin X T* ¬*separatedin X S T*
shows *connectedin X* (*S* ∪ *T*)
proof −
have ⋀*C1 C2*. ⟦*T* ⊆ *C1* ∪ *C2*; *S* ⊆ *C1* ∪ *C2*⟧ ⟹
S ∩ *C1* = {} ∧ *T* ∩ *C1* = {} ∨ *S* ∩ *C2* = {} ∧ *T* ∩ *C2* = {} ∨ ¬ *separatedin X C1 C2*
using *assms*
unfolding *connectedin_eq_not_separated_subset*
by (*metis* (*no_types*, *lifting*) *assms connectedin_subset_separated_union inf.orderE separatedin_empty*(*1*) *separatedin_mono separatedin_sym*)
then show *?thesis*
unfolding *connectedin_eq_not_separated_subset*
by (*simp add*: *assms connectedin_subset_topspace Int_Un_distrib2*)
qed

lemma *connected_space_closures*:
connected_space X ⟷
(∄*e1 e2*. *e1* ∪ *e2* = *topspace X* ∧ *X closure_of e1* ∩ *X closure_of e2* = {} ∧ *e1* ≠ {} ∧ *e2* ≠ {})
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
unfolding *connected_space_closedin_eq*
by (*metis Un_upper1 Un_upper2 closedin_closure_of closure_of_Un closure_of_eq_empty closure_of_topspace*)
next
assume *?rhs*
then show *?lhs*
unfolding *connected_space_closedin_eq*
by (*metis closure_of_eq*)
qed

lemma *connectedin_Int_frontier_of*:

assumes *connectedin X S S ∩ T ≠ {} S − T ≠ {}*
shows *S ∩ X frontier_of T ≠ {}*
proof −
have *S ⊆ topspace X* **and** ∗:
⋀*E1 E2. openin X E1 ⟶ openin X E2 ⟶ E1 ∩ E2 ∩ S = {} ⟶ S ⊆ E1 ∪ E2 ⟶ E1 ∩ S = {} ∨ E2 ∩ S = {}*
using ‹*connectedin X S*› **by** (*auto simp*: *connectedin*)
moreover
have *S − (topspace X ∩ T) ≠ {}*
using *assms*(*3*) **by** *blast*
moreover
have *S ∩ topspace X ∩ T ≠ {}*
using *assms connectedin* **by** *fastforce*
moreover
have *False* **if** *S ∩ T ≠ {} S − T ≠ {} T ⊆ topspace X S ∩ X frontier_of T = {}* **for** *T*
proof −
have *null*: *S ∩ (X closure_of T − X interior_of T) = {}*
using *that* **unfolding** *frontier_of_def* **by** *blast*
have *X interior_of T ∩ (topspace X − X closure_of T) ∩ S = {}*
by (*metis Diff_disjoint inf_bot_left interior_of_Int interior_of_complement interior_of_empty*)
moreover have *S ⊆ X interior_of T ∪ (topspace X − X closure_of T)*
using *that* ‹*S ⊆ topspace X*› *null* **by** *auto*
moreover have *S ∩ X interior_of T ≠ {}*
using *closure_of_subset that*(*1*) *that*(*3*) *null* **by** *fastforce*
ultimately have *S ∩ X interior_of (topspace X − T) = {}*
by (*metis ∗ inf_commute interior_of_complement openin_interior_of*)
then have *topspace* (*subtopology X S*) *∩ X interior_of T = S*
using ‹*S ⊆ topspace X*› *interior_of_complement null* **by** *fastforce*
then show *?thesis*
using *that* **by** (*metis Diff_eq_empty_iff inf_le2 interior_of_subset subset_trans*)
qed
ultimately show *?thesis*
by (*metis Int_lower1 frontier_of_restrict inf_assoc*)
qed

lemma *connectedin_continuous_map_image*:
assumes *f*: *continuous_map X Y f* **and** *connectedin X S*
shows *connectedin Y* (*f ‘ S*)
proof −
have *S ⊆ topspace X* **and** ∗:
⋀*E1 E2. openin X E1 ⟶ openin X E2 ⟶ E1 ∩ E2 ∩ S = {} ⟶ S ⊆ E1 ∪ E2 ⟶ E1 ∩ S = {} ∨ E2 ∩ S = {}*
using ‹*connectedin X S*› **by** (*auto simp*: *connectedin*)
show *?thesis*
unfolding *connectedin connected_space_def*
proof (*intro conjI notI*; *clarify*)

```
    show f x ∈ topspace Y if  x ∈ S for x
      using ‹S ⊆ topspace X› continuous_map_image_subset_topspace f that by
blast
  next
    fix U V
    let ?U = {x ∈ topspace X. f x ∈ U}
    let ?V = {x ∈ topspace X. f x ∈ V}
    assume UV: openin Y U openin Y V f ‘ S ⊆ U ∪ V U ∩ V ∩ f ‘ S = {} U
∩ f ‘ S ≠ {} V ∩ f ‘ S ≠ {}
    then have 1: ?U ∩ ?V ∩ S = {}
      by auto
    have 2: openin X ?U openin X ?V
      using ‹openin Y U› ‹openin Y V› continuous_map f by fastforce+
    show False
      using * [of ?U ?V] UV ‹S ⊆ topspace X›
      by (auto simp: 1 2)
  qed
qed

lemma connected_space_quotient_map_image:
  ⟦quotient_map X X' q; connected_space X⟧ ⟹ connected_space X'
 by (metis connectedin_continuous_map_image connectedin_topspace quotient_imp_continuous_map
quotient_imp_surjective_map)

lemma homeomorphic_connected_space:
    X homeomorphic_space Y ⟹ connected_space X ⟷ connected_space Y
  unfolding homeomorphic_space_def homeomorphic_maps_def
  by (metis connected_space_subconnected connectedin_continuous_map_image
connectedin_topspace continuous_map_image_subset_topspace image_eqI image_subset_iff)

lemma homeomorphic_map_connectedness:
  assumes f: homeomorphic_map X Y f and U: U ⊆ topspace X
  shows connectedin Y (f ‘ U) ⟷ connectedin X U
proof −
  have 1: f ‘ U ⊆ topspace Y ⟷ U ⊆ topspace X
    using U f homeomorphic_imp_surjective_map by blast
  moreover have connected_space (subtopology Y (f ‘ U)) ⟷ connected_space
(subtopology X U)
  proof (rule homeomorphic_connected_space)
    have f ‘ U ⊆ topspace Y
      by (simp add: U 1)
    then have topspace Y ∩ f ‘ U = f ‘ U
      by (simp add: subset_antisym)
    then show subtopology Y (f ‘ U) homeomorphic_space subtopology X U
      by (metis U f homeomorphic_map_imp_homeomorphic_space homeomor-
phic_map_subtopologies homeomorphic_space_sym inf.absorb_iff2)
  qed
  ultimately show ?thesis
    by (auto simp: connectedin_def)
```

qed

lemma *homeomorphic_map_connectedness_eq*:
  homeomorphic_map X Y f
    ⟹ *connectedin X U* ⟷
      U ⊆ *topspace X* ∧ *connectedin Y* (*f* ‘ *U*)
 using *homeomorphic_map_connectedness connectedin_subset_topspace* **by** *metis*

lemma *connectedin_discrete_topology*:
  connectedin (*discrete_topology U*) *S* ⟷ *S* ⊆ *U* ∧ (∃ *a*. *S* ⊆ {*a*})
proof (*cases S* ⊆ *U*)
 case *True*
 show *?thesis*
 proof (*cases S* = {})
  case *False*
  moreover have *connectedin* (*discrete_topology U*) *S* ⟷ (∃ *a*. *S* = {*a*})
  proof
   show *connectedin* (*discrete_topology U*) *S* ⟹ ∃ *a*. *S* = {*a*}
    using *False connectedin_Int_frontier_of insert_Diff* **by** *fastforce*
  qed (*use True* **in** *auto*)
  ultimately show *?thesis*
   by *auto*
 qed *simp*
next
 case *False*
 then show *?thesis*
  by (*simp add*: *connectedin_def*)
qed

lemma *connected_space_discrete_topology*:
    connected_space (*discrete_topology U*) ⟷ (∃ *a*. *U* ⊆ {*a*})
 by (*metis connectedin_discrete_topology connectedin_topspace order_refl topspace_discrete_topology*)

1.11.17 Compact sets

definition *compactin* **where**
 compactin X S ⟷
    S ⊆ *topspace X* ∧
    (∀𝒰. (∀ *U* ∈ 𝒰. *openin X U*) ∧ *S* ⊆ ⋃𝒰
      ⟶ (∃ℱ. *finite* ℱ ∧ ℱ ⊆ 𝒰 ∧ *S* ⊆ ⋃ℱ))

definition *compact_space* **where**
  compact_space X ≡ *compactin X* (*topspace X*)

lemma *compact_space_alt*:
  compact_space X ⟷
    (∀𝒰. (∀ *U* ∈ 𝒰. *openin X U*) ∧ *topspace X* ⊆ ⋃𝒰
      ⟶ (∃ℱ. *finite* ℱ ∧ ℱ ⊆ 𝒰 ∧ *topspace X* ⊆ ⋃ℱ))
 by (*simp add*: *compact_space_def compactin_def*)

lemma *compact_space*:
  compact_space X $\longleftrightarrow$
    $(\forall \mathcal{U}.\ (\forall\ U \in \mathcal{U}.\ \textit{openin}\ X\ U) \wedge \bigcup \mathcal{U} = \textit{topspace}\ X$
      $\longrightarrow (\exists \mathcal{F}.\ \textit{finite}\ \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge \bigcup \mathcal{F} = \textit{topspace}\ X))$
 unfolding *compact_space_alt*
 using *openin_subset* **by** *fastforce*

lemma *compactinD*:
 $[\![\textit{compactin}\ X\ S;\ \bigwedge U.\ U \in \mathcal{U} \Longrightarrow \textit{openin}\ X\ U;\ S \subseteq \bigcup \mathcal{U}]\!] \Longrightarrow \exists \mathcal{F}.\ \textit{finite}\ \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge S \subseteq \bigcup \mathcal{F}$
 by (*auto simp*: *compactin_def*)

lemma *compactin_euclidean_iff* [*simp*]: *compactin euclidean S* $\longleftrightarrow$ *compact S*
 by (*simp add*: *compact_eq_Heine_Borel compactin_def*) *meson*

lemma *compactin_absolute* [*simp*]:
  compactin (*subtopology X S*) *S* $\longleftrightarrow$ *compactin X S*
proof −
 have *eq*: $(\forall\ U \in \mathcal{U}.\ \exists\ Y.\ \textit{openin}\ X\ Y \wedge\ U = Y \cap S) \longleftrightarrow \mathcal{U} \subseteq (\lambda Y.\ Y \cap S)$ ‘ $\{y.\ \textit{openin}\ X\ y\}$ **for** $\mathcal{U}$
   by *auto*
 show *?thesis*
  by (*auto simp*: *compactin_def openin_subtopology eq imp_conjL all_subset_image ex_finite_subset_image*)
qed

lemma *compactin_subspace*: *compactin X S* $\longleftrightarrow$ $S \subseteq$ *topspace X* $\wedge$ *compact_space* (*subtopology X S*)
 unfolding *compact_space_def topspace_subtopology*
 by (*metis compactin_absolute compactin_def inf.absorb2*)

lemma *compact_space_subtopology*: *compactin X S* $\Longrightarrow$ *compact_space* (*subtopology X S*)
 by (*simp add*: *compactin_subspace*)

lemma *compactin_subtopology*: *compactin* (*subtopology X S*) *T* $\longleftrightarrow$ *compactin X T* $\wedge$ $T \subseteq S$
 by (*metis compactin_subspace inf.absorb_iff2 le_inf_iff subtopology_subtopology topspace_subtopology*)

lemma *compactin_subset_topspace*: *compactin X S* $\Longrightarrow$ $S \subseteq$ *topspace X*
 by (*simp add*: *compactin_subspace*)

lemma *compactin_contractive*:
  $[\![\textit{compactin}\ X'\ S;\ \textit{topspace}\ X' = \textit{topspace}\ X;$
    $\bigwedge U.\ \textit{openin}\ X\ U \Longrightarrow \textit{openin}\ X'\ U]\!] \Longrightarrow \textit{compactin}\ X\ S$
 by (*simp add*: *compactin_def*)

lemma *finite_imp_compactin*:
  $[\![S \subseteq \text{topspace } X; \text{finite } S]\!] \Longrightarrow \text{compactin } X\ S$
 by (*metis compactin_subspace compact_space finite_UnionD inf.absorb_iff2 order_refl topspace_subtopology*)

lemma *compactin_empty* [*iff*]: *compactin X* {}
 by (*simp add*: *finite_imp_compactin*)

lemma *compact_space_trivial_topology* [*simp*]: *compact_space trivial_topology*
 by (*simp add*: *compact_space_def*)

lemma *finite_imp_compactin_eq*:
  finite S $\Longrightarrow$ (*compactin X S* $\longleftrightarrow$ $S \subseteq$ *topspace X*)
 using *compactin_subset_topspace finite_imp_compactin* **by** *blast*

lemma *compactin_sing* [*simp*]: *compactin X* {*a*} $\longleftrightarrow$ $a \in$ *topspace X*
 by (*simp add*: *finite_imp_compactin_eq*)

lemma *closed_compactin*:
 assumes *XK*: *compactin X K* **and** $C \subseteq K$ **and** *XC*: *closedin X C*
 shows *compactin X C*
 unfolding *compactin_def*
proof (*intro conjI allI impI*)
 show $C \subseteq$ *topspace X*
  by (*simp add*: *XC closedin_subset*)
next
 fix $\mathcal{U}$ :: *'a set set*
 assume $\mathcal{U}$: *Ball* $\mathcal{U}$ (*openin X*) $\wedge$ $C \subseteq \bigcup\mathcal{U}$
 have ($\forall$ *U*$\in$*insert* (*topspace X* $-$ *C*) $\mathcal{U}$. *openin X U*)
  using *XC* $\mathcal{U}$ **by** *blast*
 moreover have $K \subseteq \bigcup$(*insert* (*topspace X* $-$ *C*) $\mathcal{U}$)
  using $\mathcal{U}$ *XK compactin_subset_topspace* **by** *fastforce*
 ultimately obtain $\mathcal{F}$ **where** *finite* $\mathcal{F}$ $\mathcal{F} \subseteq$ *insert* (*topspace X* $-$ *C*) $\mathcal{U}$ $K \subseteq \bigcup\mathcal{F}$
  using *assms* **unfolding** *compactin_def* **by** *metis*
 moreover have *openin X* (*topspace X* $-$ *C*)
  using *XC* **by** *auto*
 ultimately show $\exists\mathcal{F}$. *finite* $\mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge C \subseteq \bigcup\mathcal{F}$
  using ‹*C* $\subseteq$ *K*›
  by (*rule_tac x*=$\mathcal{F}$ $-$ {*topspace X* $-$ *C*} **in** *exI*) *auto*
qed

lemma *closed_compactin_Inter*:
 $[\![$*compactin X K*; $K \in \mathcal{K}$; $\bigwedge K.\ K \in \mathcal{K} \Longrightarrow$ *closedin X K*$]\!] \Longrightarrow$ *compactin X* ($\bigcap\mathcal{K}$)
 by (*metis Inf_lower closed_compactin closedin_Inter empty_iff*)

lemma *closedin_subtopology_Int_subset*:
  $[\![$*closedin* (*subtopology X U*) ($U \cap S$); $V \subseteq U$$]\!] \Longrightarrow$ *closedin* (*subtopology X V*) ($V \cap S$)
  by (*smt* (*verit*, *ccfv_SIG*) *closedin_subtopology inf.left_commute inf.orderE*

inf_commute)

lemma *closedin_subtopology_Int_closedin*:
  ⟦*closedin* (*subtopology X U*) *S*; *closedin X T*⟧ ⟹ *closedin* (*subtopology X U*) (*S* ∩ *T*)
  by (*smt* (*verit*, *best*) *closedin_Int closedin_subtopology inf_assoc inf_commute*)

lemma *closedin_compact_space*:
  ⟦*compact_space X*; *closedin X S*⟧ ⟹ *compactin X S*
  by (*simp add*: *closed_compactin closedin_subset compact_space_def*)

lemma *compact_Int_closedin*:
  assumes *compactin X S closedin X T* **shows** *compactin X* (*S* ∩ *T*)
proof −
  have *compactin* (*subtopology X S*) (*S* ∩ *T*)
   by (*metis assms closedin_compact_space closedin_subtopology compactin_subspace inf_commute*)
  then show *?thesis*
    by (*simp add*: *compactin_subtopology*)
qed

lemma *closed_Int_compactin*: ⟦*closedin X S*; *compactin X T*⟧ ⟹ *compactin X* (*S* ∩ *T*)
  by (*metis compact_Int_closedin inf_commute*)

lemma *compactin_Un*:
  assumes *S*: *compactin X S* **and** *T*: *compactin X T* **shows** *compactin X* (*S* ∪ *T*)
  unfolding *compactin_def*
proof (*intro conjI allI impI*)
  show *S* ∪ *T* ⊆ *topspace X*
    using *assms* **by** (*auto simp*: *compactin_def*)
next
  fix *𝒰* :: *'a set set*
  assume *𝒰*: *Ball 𝒰* (*openin X*) ∧ *S* ∪ *T* ⊆ ⋃*𝒰*
  with *S* **obtain** *ℱ* **where** *𝒱*: *finite ℱ ℱ* ⊆ *𝒰 S* ⊆ ⋃*ℱ*
    unfolding *compactin_def* **by** (*meson sup.bounded_iff*)
  obtain *𝒲* **where** *finite 𝒲 𝒲* ⊆ *𝒰 T* ⊆ ⋃*𝒲*
    using *𝒰 T*
    unfolding *compactin_def* **by** (*meson sup.bounded_iff*)
  with *𝒱* **show** ∃*𝒱*. *finite 𝒱* ∧ *𝒱* ⊆ *𝒰* ∧ *S* ∪ *T* ⊆ ⋃*𝒱*
    by (*rule_tac x*=*ℱ* ∪ *𝒲* **in** *exI*) *auto*
qed

lemma *compactin_Union*:
  ⟦*finite ℱ*; ⋀*S*. *S* ∈ *ℱ* ⟹ *compactin X S*⟧ ⟹ *compactin X* (⋃*ℱ*)
by (*induction rule*: *finite_induct*) (*simp_all add*: *compactin_Un*)

lemma *compactin_subtopology_imp_compact*:

assumes *compactin* (*subtopology X S*) *K* **shows** *compactin X K*
using *assms*
proof (*clarsimp simp add*: *compactin_def*)
fix $\mathcal{U}$
define $\mathcal{V}$ **where** $\mathcal{V} \equiv (\lambda U.\ U \cap S)\ `\ \mathcal{U}$
assume $K \subseteq$ *topspace X* **and** $K \subseteq S$ **and** $\forall\, x{\in}\mathcal{U}.$ *openin X x* **and** $K \subseteq \bigcup\mathcal{U}$
then have $\forall\, V \in \mathcal{V}.$ *openin* (*subtopology X S*) *V* $K \subseteq \bigcup\mathcal{V}$
unfolding $\mathcal{V}$*_def* **by** (*auto simp*: *openin_subtopology*)
moreover
assume $\forall\mathcal{U}.$ $(\forall\, x{\in}\mathcal{U}.$ *openin* (*subtopology X S*) $x) \wedge K \subseteq \bigcup\mathcal{U} \longrightarrow (\exists\mathcal{F}.$ *finite* $\mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge K \subseteq \bigcup\mathcal{F})$
ultimately obtain $\mathcal{F}$ **where** *finite* $\mathcal{F}$ $\mathcal{F} \subseteq \mathcal{V}$ $K \subseteq \bigcup\mathcal{F}$
by *meson*
then have $\mathcal{F}$: $\exists\, U.\ U \in \mathcal{U} \wedge V = U \cap S$ **if** $V \in \mathcal{F}$ **for** V
unfolding $\mathcal{V}$*_def* **using** *that* **by** *blast*
let $?\mathcal{F} = (\lambda F.\ @U.\ U \in \mathcal{U} \wedge F = U \cap S)\ `\ \mathcal{F}$
show $\exists\mathcal{F}.$ *finite* $\mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge K \subseteq \bigcup\mathcal{F}$
proof (*intro exI conjI*)
show *finite* $?\mathcal{F}$
using ‹*finite* $\mathcal{F}$› **by** *blast*
show $?\mathcal{F} \subseteq \mathcal{U}$
using *someI_ex* [*OF* $\mathcal{F}$] **by** *blast*
show $K \subseteq \bigcup ?\mathcal{F}$
proof *clarsimp*
fix x
assume $x \in K$
then show $\exists\, V \in \mathcal{F}.\ x \in (SOME\ U.\ U \in \mathcal{U} \wedge V = U \cap S)$
using ‹$K \subseteq \bigcup\mathcal{F}$› *someI_ex* [*OF* $\mathcal{F}$]
by (*metis* (*no_types*, *lifting*) *IntD1 Union_iff subsetCE*)
qed
qed
qed

lemma *compact_imp_compactin_subtopology*:
assumes *compactin X K* $K \subseteq S$ **shows** *compactin* (*subtopology X S*) *K*
using *assms*
proof (*clarsimp simp add*: *compactin_def*)
fix $\mathcal{U}$ :: *'a set set*
define $\mathcal{V}$ **where** $\mathcal{V} \equiv \{V.\ \text{openin } X\ V \wedge (\exists\, U \in \mathcal{U}.\ U = V \cap S)\}$
assume $K \subseteq S$ **and** $K \subseteq$ *topspace X* **and** $\forall\, U{\in}\mathcal{U}.$ *openin* (*subtopology X S*) *U* **and** $K \subseteq \bigcup\mathcal{U}$
then have $\forall\, V \in \mathcal{V}.$ *openin X V* $K \subseteq \bigcup\mathcal{V}$
unfolding $\mathcal{V}$*_def* **by** (*fastforce simp*: *subset_eq openin_subtopology*)+
moreover
assume $\forall\mathcal{U}.$ $(\forall\, U{\in}\mathcal{U}.$ *openin X U*$) \wedge K \subseteq \bigcup\mathcal{U} \longrightarrow (\exists\mathcal{F}.$ *finite* $\mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge K \subseteq \bigcup\mathcal{F})$
ultimately obtain $\mathcal{F}$ **where** *finite* $\mathcal{F}$ $\mathcal{F} \subseteq \mathcal{V}$ $K \subseteq \bigcup\mathcal{F}$
by *meson*
let $?\mathcal{F} = (\lambda F.\ F \cap S)\ `\ \mathcal{F}$

show $\exists \mathcal{F}$. *finite* $\mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge K \subseteq \bigcup \mathcal{F}$
proof (*intro exI conjI*)
show *finite* $?\mathcal{F}$
using ‹*finite* $\mathcal{F}$› **by** *blast*
show $?\mathcal{F} \subseteq \mathcal{U}$
using $\mathcal{V}$*_def* ‹$\mathcal{F} \subseteq \mathcal{V}$› **by** *blast*
show $K \subseteq \bigcup ?\mathcal{F}$
using ‹$K \subseteq \bigcup \mathcal{F}$› *assms*(*2*) **by** *auto*
qed
qed

proposition *compact_space_fip*:
compact_space $X \longleftrightarrow$
$(\forall \mathcal{U}.\ (\forall\, C{\in}\mathcal{U}.\ \textit{closedin}\ X\ C) \wedge (\forall \mathcal{F}.\ \textit{finite}\ \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \longrightarrow \bigcap \mathcal{F} \neq \{\}) \longrightarrow \bigcap \mathcal{U} \neq \{\})$
(**is** _ = *?rhs*)
proof (*cases* X = *trivial_topology*)
case *True*
then show *?thesis*
by (*metis Pow_iff closedin_topspace_empty compact_space_trivial_topology finite.emptyI finite_Pow_iff finite_subset subsetI*)
next
case *False*
show *?thesis*
proof *safe*
fix $\mathcal{U}$ :: *'a set set*
assume $*$ [*rule_format*]: $\forall \mathcal{F}.$ *finite* $\mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \longrightarrow \bigcap \mathcal{F} \neq \{\}$
define $\mathcal{V}$ **where** $\mathcal{V} \equiv (\lambda S.$ *topspace* $X - S)$ ‘ $\mathcal{U}$
assume *clo*: $\forall\, C{\in}\mathcal{U}.$ *closedin* $X\ C$ **and** [*simp*]: $\bigcap \mathcal{U} = \{\}$
then have $\forall\, V \in \mathcal{V}.$ *openin* $X\ V$ *topspace* $X \subseteq \bigcup \mathcal{V}$
by (*auto simp*: $\mathcal{V}$*_def*)
moreover assume [*unfolded compact_space_alt*, *rule_format*, *of* $\mathcal{V}$]: *compact_space* X
ultimately obtain $\mathcal{F}$ **where** $\mathcal{F}$: *finite* $\mathcal{F}$ $\mathcal{F} \subseteq \mathcal{U}$ *topspace* $X \subseteq$ *topspace* $X - \bigcap \mathcal{F}$
by (*auto simp*: *ex_finite_subset_image* $\mathcal{V}$*_def*)
moreover have $\mathcal{F} \neq \{\}$
using $\mathcal{F}$ *False* **by** *force*
ultimately show *False*
using $*$ [*of* $\mathcal{F}$]
by *auto* (*metis Diff_iff Inter_iff clo closedin_def subsetD*)
next
assume R [*rule_format*]: *?rhs*
show *compact_space* X
unfolding *compact_space_alt*
proof *clarify*
fix $\mathcal{U}$ :: *'a set set*
define $\mathcal{V}$ **where** $\mathcal{V} \equiv (\lambda S.$ *topspace* $X - S)$ ‘ $\mathcal{U}$

```
      assume ∀ C∈𝒰. openin X C and topspace X ⊆ ⋃𝒰
      with False have *: ∀ V ∈ 𝒱. closedin X V 𝒰 ≠ {}
        by (auto simp: 𝒱_def)
      show ∃ℱ. finite ℱ ∧ ℱ ⊆ 𝒰 ∧ topspace X ⊆ ⋃ℱ
      proof (rule ccontr; simp)
        assume ∀ℱ⊆𝒰. finite ℱ ⟶ ¬ topspace X ⊆ ⋃ℱ
        then have ∀ℱ. finite ℱ ∧ ℱ ⊆ 𝒱 ⟶ ⋂ℱ ≠ {}
          by (simp add: 𝒱_def all_finite_subset_image)
        with ‹topspace X ⊆ ⋃𝒰› show False
          using R [of 𝒱] * by (simp add: 𝒱_def)
      qed
    qed
  qed
qed

corollary compactin_fip:
  compactin X S ⟷
    S ⊆ topspace X ∧
    (∀𝒰. (∀ C∈𝒰. closedin X C) ∧ (∀ℱ. finite ℱ ∧ ℱ ⊆ 𝒰 ⟶ S ∩ ⋂ℱ ≠ {}) ⟶
S ∩ ⋂𝒰 ≠ {})
proof (cases S = {})
  case False
  show ?thesis
  proof (cases S ⊆ topspace X)
    case True
    then have compactin X S ⟷
          (∀𝒰. 𝒰 ⊆ (λT. S ∩ T) ` {T. closedin X T} ⟶
           (∀ℱ. finite ℱ ⟶ ℱ ⊆ 𝒰 ⟶ ⋂ℱ ≠ {}) ⟶ ⋂𝒰 ≠ {})
      by (simp add: compact_space_fip compactin_subspace closedin_subtopology
image_def subset_eq Int_commute imp_conjL)
    also have ... = (∀𝒰⊆Collect (closedin X). (∀ℱ. finite ℱ ⟶ ℱ ⊆ (∩) S ` 𝒰
⟶ ⋂ℱ ≠ {}) ⟶ ⋂ ((∩) S ` 𝒰) ≠ {})
      by (simp add: all_subset_image)
    also have ... = (∀𝒰. (∀ C∈𝒰. closedin X C) ∧ (∀ℱ. finite ℱ ∧ ℱ ⊆ 𝒰 ⟶
S ∩ ⋂ℱ ≠ {}) ⟶ S ∩ ⋂𝒰 ≠ {})
    proof −
      have eq: ((∀ℱ. finite ℱ ∧ ℱ ⊆ 𝒰 ⟶ ⋂ ((∩) S ` ℱ) ≠ {}) ⟶ ⋂ ((∩) S `
𝒰) ≠ {}) ⟷
            ((∀ℱ. finite ℱ ∧ ℱ ⊆ 𝒰 ⟶ S ∩ ⋂ℱ ≠ {}) ⟶ S ∩ ⋂𝒰 ≠ {}) for
𝒰
        by simp (use ‹S ≠ {}› in blast)
      show ?thesis
        unfolding imp_conjL [symmetric] all_finite_subset_image eq by blast
    qed
    finally show ?thesis
      using True by simp
  qed (simp add: compactin_subspace)
qed force
```

corollary *compact_space_imp_nest*:
  fixes $C :: nat \Rightarrow 'a\ set$
  assumes *compact_space* X **and** *clo*: $\bigwedge n.\ closedin\ X\ (C\ n)$
    and *ne*: $\bigwedge n.\ C\ n \neq \{\}$ **and** *dec*: *decseq* C
  shows $(\bigcap n.\ C\ n) \neq \{\}$
proof −
  let $?\mathcal{U} = range\ (\lambda n.\ \bigcap m \leq n.\ C\ m)$
  have *closedin* $X\ A$ **if** $A \in ?\mathcal{U}$ **for** A
    using *that clo* **by** *auto*
  moreover have $(\bigcap n{\in}K.\ \bigcap m \leq n.\ C\ m) \neq \{\}$ **if** *finite* K **for** K
  proof −
    obtain n **where** $\bigwedge k.\ k \in K \Longrightarrow k \leq n$
      using *Max.coboundedI* ‹*finite* K› **by** *blast*
    with *dec* **have** $C\ n \subseteq (\bigcap n{\in}K.\ \bigcap m \leq n.\ C\ m)$
      unfolding *decseq_def* **by** *blast*
  with *ne* [*of* n] **show** *?thesis*
    by *blast*
  qed
  ultimately show *?thesis*
    using ‹*compact_space* X› [*unfolded compact_space_fip*, *rule_format*, *of* $?\mathcal{U}$]
   by (*simp add*: *all_finite_subset_image INT_extend_simps UN_atMost_UNIV*
del: *INT_simps*)
qed

lemma *compactin_discrete_topology*:
  compactin (*discrete_topology* X) $S \longleftrightarrow S \subseteq X \wedge$ *finite* S (**is** *?lhs* = *?rhs*)
proof (*intro iffI conjI*)
  assume L: *?lhs*
  then show $S \subseteq X$
    by (*auto simp*: *compactin_def*)
  have $*$: $\bigwedge \mathcal{U}.\ Ball\ \mathcal{U}\ (openin\ (discrete\_topology\ X)) \wedge S \subseteq \bigcup \mathcal{U} \Longrightarrow$
      $(\exists \mathcal{F}.\ finite\ \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge S \subseteq \bigcup \mathcal{F})$
    using L **by** (*auto simp*: *compactin_def*)
  show *finite* S
    using $*$ [*of* $(\lambda x.\ \{x\})$ ‘ X] ‹$S \subseteq X$›
    by *clarsimp* (*metis UN_singleton finite_subset_image infinite_super*)
next
  assume *?rhs*
  then show *?lhs*
    by (*simp add*: *finite_imp_compactin*)
qed

lemma *compact_space_discrete_topology*: *compact_space*(*discrete_topology* X) $\longleftrightarrow$
finite X
  by (*simp add*: *compactin_discrete_topology compact_space_def*)

lemma *compact_space_imp_Bolzano_Weierstrass*:
  assumes *compact_space* X *infinite* S $S \subseteq$ *topspace* X
  shows X *derived_set_of* $S \neq \{\}$

proof
assume *X*: *X derived_set_of S* = {}
then have *closedin X S*
by (*simp add*: *closedin_contains_derived_set assms*)
then have *compactin X S*
by (*rule closedin_compact_space* [*OF* ‹*compact_space X*›])
with *X* **show** *False*
by (*metis* ‹*infinite S*› *compactin_subspace compact_space_discrete_topology inf_bot_right subtopology_eq_discrete_topology_eq*)
qed

lemma *compactin_imp_Bolzano_Weierstrass*:
⟦*compactin X S*; *infinite T* ∧ *T* ⊆ *S*⟧ ⟹ *S* ∩ *X derived_set_of T* ≠ {}
using *compact_space_imp_Bolzano_Weierstrass* [*of subtopology X S*]
by (*simp add*: *compactin_subspace derived_set_of_subtopology inf_absorb2*)

lemma *compact_closure_of_imp_Bolzano_Weierstrass*:
⟦*compactin X* (*X closure_of S*); *infinite T*; *T* ⊆ *S*; *T* ⊆ *topspace X*⟧ ⟹ *X derived_set_of T* ≠ {}
using *closure_of_mono closure_of_subset compactin_imp_Bolzano_Weierstrass*
by *fastforce*

lemma *discrete_compactin_eq_finite*:
S ∩ *X derived_set_of S* = {} ⟹ *compactin X S* ⟷ *S* ⊆ *topspace X* ∧ *finite S*
by (*meson compactin_imp_Bolzano_Weierstrass finite_imp_compactin_eq order_refl*)

lemma *discrete_compact_space_eq_finite*:
X derived_set_of (*topspace X*) = {} ⟹ (*compact_space X* ⟷ *finite*(*topspace X*))
by (*metis compact_space_discrete_topology discrete_topology_unique_derived_set*)

lemma *image_compactin*:
assumes *cpt*: *compactin X S* **and** *cont*: *continuous_map X Y f*
shows *compactin Y* (*f* ‘ *S*)
unfolding *compactin_def*
proof (*intro conjI allI impI*)
show *f* ‘ *S* ⊆ *topspace Y*
using *compactin_subset_topspace cont continuous_map_image_subset_topspace cpt* **by** *blast*
next
fix *𝒰* :: *'b set set*
assume *𝒰*: *Ball 𝒰* (*openin Y*) ∧ *f* ‘ *S* ⊆ ⋃*𝒰*
define *𝒱* **where** *𝒱* ≡ (λ*U*. {*x* ∈ *topspace X*. *f x* ∈ *U*}) ‘ *𝒰*
have *S* ⊆ *topspace X*
and ∗: ⋀*𝒰*. ⟦∀ *U*∈*𝒰*. *openin X U*; *S* ⊆ ⋃*𝒰*⟧ ⟹ ∃ℱ. *finite* ℱ ∧ ℱ ⊆ *𝒰* ∧ *S* ⊆ ⋃ℱ
using *cpt* **by** (*auto simp*: *compactin_def*)

obtain $\mathcal{F}$ **where** $\mathcal{F}$: *finite* $\mathcal{F}$ $\mathcal{F} \subseteq \mathcal{V}$ $S \subseteq \bigcup \mathcal{F}$
proof −
have *1*: $\forall\, U{\in}\mathcal{V}.$ *openin* X U
unfolding $\mathcal{V}$*_def* **using** $\mathcal{U}$ *cont*[*unfolded continuous_map*] **by** *blast*
have *2*: $S \subseteq \bigcup \mathcal{V}$
unfolding $\mathcal{V}$*_def* **using** *compactin_subset_topspace cpt* $\mathcal{U}$ **by** *fastforce*
show *thesis*
using $*$ [*OF 1 2*] *that* **by** *metis*
qed
have $\forall\, v \in \mathcal{V}.\ \exists\, U.\ U \in \mathcal{U} \wedge v = \{x \in$ *topspace* $X.\ f\, x \in U\}$
using $\mathcal{V}$*_def* **by** *blast*
then obtain U **where** U: $\forall\, v \in \mathcal{V}.\ U\, v \in \mathcal{U} \wedge v = \{x \in$ *topspace* $X.\ f\, x \in U\, v\}$
by *metis*
show $\exists\, \mathcal{F}.$ *finite* $\mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge f$ ‘ $S \subseteq \bigcup \mathcal{F}$
proof (*intro conjI exI*)
show *finite* (U ‘ $\mathcal{F}$)
by (*simp add*: ‹*finite* $\mathcal{F}$›)
next
show U ‘ $\mathcal{F} \subseteq \mathcal{U}$
using ‹$\mathcal{F} \subseteq \mathcal{V}$› U **by** *auto*
next
show f ‘ $S \subseteq \bigcup$ (U ‘ $\mathcal{F}$)
using $\mathcal{F}$(*2−3*) U *UnionE subset_eq* U **by** *fastforce*
qed
qed

lemma *homeomorphic_compact_space*:
assumes X *homeomorphic_space* Y
shows *compact_space* $X \longleftrightarrow$ *compact_space* Y
using *homeomorphic_space_sym*
by (*metis assms compact_space_def homeomorphic_eq_everything_map homeomorphic_space image_compactin*)

lemma *homeomorphic_map_compactness*:
assumes *hom*: *homeomorphic_map* X Y f **and** U: $U \subseteq$ *topspace* X
shows *compactin* Y (f ‘ U) $\longleftrightarrow$ *compactin* X U
proof −
have f ‘ $U \subseteq$ *topspace* Y
using *hom* U *homeomorphic_imp_surjective_map* **by** *blast*
moreover have *homeomorphic_map* (*subtopology* X U) (*subtopology* Y (f ‘ U)) f
using U *hom homeomorphic_imp_surjective_map* **by** (*blast intro*: *homeomorphic_map_subtopologies*)
then have *compact_space* (*subtopology* Y (f ‘ U)) = *compact_space* (*subtopology* X U)
using *homeomorphic_compact_space homeomorphic_map_imp_homeomorphic_space* **by** *blast*
ultimately show *?thesis*

by (*simp add*: *compactin_subspace U*)
qed

lemma *homeomorphic_map_compactness_eq*:
homeomorphic_map X Y f
⟹ *compactin X U* ⟷ *U* ⊆ *topspace X* ∧ *compactin Y* (*f* ‘ *U*)
by (*meson compactin_subset_topspace homeomorphic_map_compactness*)

1.11.18 Embedding maps

definition *embedding_map*
where *embedding_map X Y f* ≡ *homeomorphic_map X* (*subtopology Y* (*f* ‘ (*topspace X*))) *f*

lemma *embedding_map_eq*:
⟦*embedding_map X Y f*; ⋀*x*. *x* ∈ *topspace X* ⟹ *f x* = *g x*⟧ ⟹ *embedding_map X Y g*
unfolding *embedding_map_def*
by (*metis homeomorphic_map_eq image_cong*)

lemma *embedding_map_compose*:
assumes *embedding_map X X′ f embedding_map X′ X″ g*
shows *embedding_map X X″* (*g* ∘ *f*)
proof −
have *hm*: *homeomorphic_map X* (*subtopology X′* (*f* ‘ *topspace X*)) *f homeomorphic_map X′* (*subtopology X″* (*g* ‘ *topspace X′*)) *g*
using *assms* **by** (*auto simp*: *embedding_map_def*)
then obtain *C* **where** *g* ‘ *topspace X′* ∩ *C* = (*g* ∘ *f*) ‘ *topspace X*
using *continuous_map_image_subset_topspace homeomorphic_imp_continuous_map*
by *fastforce*
then have *homeomorphic_map* (*subtopology X′* (*f* ‘ *topspace X*)) (*subtopology X″* ((*g* ∘ *f*) ‘ *topspace X*)) *g*
by (*metis hm homeomorphic_eq_everything_map homeomorphic_map_subtopologies image_comp subtopology_subtopology topspace_subtopology*)
then show *?thesis*
unfolding *embedding_map_def*
using *hm*(*1*) *homeomorphic_map_compose* **by** *blast*
qed

lemma *surjective_embedding_map*:
embedding_map X Y f ∧ *f* ‘ (*topspace X*) = *topspace Y* ⟷ *homeomorphic_map X Y f*
by (*force simp*: *embedding_map_def homeomorphic_eq_everything_map*)

lemma *embedding_map_in_subtopology*:
embedding_map X (*subtopology Y S*) *f* ⟷ *embedding_map X Y f* ∧ *f* ‘ (*topspace X*) ⊆ *S* (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⟹ *?rhs*

unfolding *embedding_map_def*
by (*metis continuous_map_in_subtopology homeomorphic_imp_continuous_map inf_absorb2 subtopology_subtopology*)
qed (*simp add*: *embedding_map_def inf.absorb_iff2 subtopology_subtopology*)

lemma *injective_open_imp_embedding_map*:
⟦*continuous_map X Y f*; *open_map X Y f*; *inj_on f* (*topspace X*)⟧ ⟹ *embedding_map X Y f*
unfolding *embedding_map_def homeomorphic_map_def*
by (*simp add*: *image_subset_iff_funcset continuous_map_in_subtopology continuous_open_quotient_map eq_iff*
open_map_imp_subset open_map_into_subtopology)

lemma *injective_closed_imp_embedding_map*:
⟦*continuous_map X Y f*; *closed_map X Y f*; *inj_on f* (*topspace X*)⟧ ⟹ *embedding_map X Y f*
unfolding *embedding_map_def homeomorphic_map_def*
by (*simp add*: *closed_map_imp_subset closed_map_into_subtopology continuous_closed_quotient_map*
continuous_map_in_subtopology dual_order.eq_iff image_subset_iff_funcset)

lemma *embedding_map_imp_homeomorphic_space*:
embedding_map X Y f ⟹ *X homeomorphic_space* (*subtopology Y* (*f ' topspace X*))
unfolding *embedding_map_def*
using *homeomorphic_space* **by** *blast*

lemma *embedding_imp_closed_map*:
⟦*embedding_map X Y f*; *closedin Y* (*f ' topspace X*)⟧ ⟹ *closed_map X Y f*
unfolding *closed_map_def*
by (*auto simp*: *closedin_closed_subtopology embedding_map_def homeomorphic_map_closedness_eq*)

lemma *embedding_imp_closed_map_eq*:
embedding_map X Y f ⟹ (*closed_map X Y f* ⟷ *closedin Y* (*f ' topspace X*))
using *closed_map_def embedding_imp_closed_map* **by** *blast*

1.11.19 Retraction and section maps

definition *retraction_maps* :: *'a topology* ⇒ *'b topology* ⇒ (*'a* ⇒ *'b*) ⇒ (*'b* ⇒ *'a*) ⇒ *bool*
where *retraction_maps X Y f g* ≡
continuous_map X Y f ∧ *continuous_map Y X g* ∧ (∀ *x* ∈ *topspace Y*. *f*(*g x*) = *x*)

definition *section_map* :: *'a topology* ⇒ *'b topology* ⇒ (*'a* ⇒ *'b*) ⇒ *bool*
where *section_map X Y f* ≡ ∃ *g*. *retraction_maps Y X g f*

definition *retraction_map* :: *'a topology* ⇒ *'b topology* ⇒ (*'a* ⇒ *'b*) ⇒ *bool*

where *retraction_map X Y f* ≡ ∃ *g*. *retraction_maps X Y f g*

lemma *retraction_maps_eq*:
⟦*retraction_maps X Y f g*; ⋀*x*. *x* ∈ *topspace X* ⟹ *f x* = *f' x*; ⋀*x*. *x* ∈ *topspace Y* ⟹ *g x* = *g' x*⟧
⟹ *retraction_maps X Y f' g'*
unfolding *retraction_maps_def*
by (*metis continuous_map_eq continuous_map_image_subset_topspace image_subset_iff*)

lemma *section_map_eq*:
⟦*section_map X Y f*; ⋀*x*. *x* ∈ *topspace X* ⟹ *f x* = *g x*⟧ ⟹ *section_map X Y g*
unfolding *section_map_def* **using** *retraction_maps_eq* **by** *blast*

lemma *retraction_map_eq*:
⟦*retraction_map X Y f*; ⋀*x*. *x* ∈ *topspace X* ⟹ *f x* = *g x*⟧ ⟹ *retraction_map X Y g*
unfolding *retraction_map_def* **using** *retraction_maps_eq* **by** *blast*

lemma *homeomorphic_imp_retraction_maps*:
homeomorphic_maps X Y f g ⟹ *retraction_maps X Y f g*
by (*simp add*: *homeomorphic_maps_def retraction_maps_def*)

lemma *section_and_retraction_eq_homeomorphic_map*:
section_map X Y f ∧ *retraction_map X Y f* ⟷ *homeomorphic_map X Y f* (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then obtain *g* **where** *homeomorphic_maps X Y f g*
unfolding *homeomorphic_maps_def retraction_map_def section_map_def*
by (*smt* (*verit*) *Pi_iff continuous_map_def retraction_maps_def*)
then show *?rhs*
using *homeomorphic_map_maps* **by** *blast*
next
assume *?rhs*
then show *?lhs*
unfolding *retraction_map_def section_map_def*
by (*meson homeomorphic_imp_retraction_maps homeomorphic_map_maps homeomorphic_maps_sym*)
qed

lemma *section_imp_embedding_map*:
section_map X Y f ⟹ *embedding_map X Y f*
unfolding *section_map_def embedding_map_def homeomorphic_map_maps retraction_maps_def homeomorphic_maps_def*
by (*force simp*: *continuous_map_in_subtopology continuous_map_from_subtopology*)

lemma *retraction_imp_quotient_map*:
assumes *retraction_map X Y f*

shows *quotient_map X Y f*
unfolding *quotient_map_def*
proof (*intro conjI subsetI allI impI*)
show *f ' topspace X = topspace Y*
using *assms* **by** (*force simp*: *retraction_map_def retraction_maps_def continuous_map_def Pi_iff*)
next
fix *U*
assume *U*: *U ⊆ topspace Y*
have *openin Y U*
if *∀ x∈topspace Y. g x ∈ topspace X ∀ x∈topspace Y. f (g x) = x*
openin Y {x ∈ topspace Y. g x ∈ {x ∈ topspace X. f x ∈ U}} **for** *g*
using *openin_subopen U that* **by** *fastforce*
then show *openin X {x ∈ topspace X. f x ∈ U} = openin Y U*
using *assms* **by** (*auto simp*: *retraction_map_def retraction_maps_def continuous_map_def Pi_iff*)
qed

lemma *retraction_maps_compose*:
⟦retraction_maps X Y f f'; retraction_maps Y Z g g'⟧ ⟹ retraction_maps X Z (g ∘ f) (f' ∘ g')
unfolding *retraction_maps_def*
by (*smt* (*verit*, *ccfv_threshold*) *comp_apply continuous_map_compose continuous_map_image_subset_topspace image_subset_iff*)

lemma *retraction_map_compose*:
⟦retraction_map X Y f; retraction_map Y Z g⟧ ⟹ retraction_map X Z (g ∘ f)
by (*meson retraction_map_def retraction_maps_compose*)

lemma *section_map_compose*:
⟦section_map X Y f; section_map Y Z g⟧ ⟹ section_map X Z (g ∘ f)
by (*meson retraction_maps_compose section_map_def*)

lemma *surjective_section_eq_homeomorphic_map*:
section_map X Y f ∧ f ' (topspace X) = topspace Y ⟷ homeomorphic_map X Y f
by (*meson section_and_retraction_eq_homeomorphic_map section_imp_embedding_map surjective_embedding_map*)

lemma *surjective_retraction_or_section_map*:
f ' (topspace X) = topspace Y ⟹ retraction_map X Y f ∨ section_map X Y f ⟷ retraction_map X Y f
using *section_and_retraction_eq_homeomorphic_map surjective_section_eq_homeomorphic_map*
by *fastforce*

lemma *retraction_imp_surjective_map*:
retraction_map X Y f ⟹ f ' (topspace X) = topspace Y
by (*simp add*: *retraction_imp_quotient_map quotient_imp_surjective_map*)

lemma *section_imp_injective_map*:
  ⟦*section_map X Y f*; *x* ∈ *topspace X*; *y* ∈ *topspace X*⟧ ⟹ *f x* = *f y* ⟷ *x* = *y*
 by (*metis* (*mono_tags*, *opaque_lifting*) *retraction_maps_def section_map_def*)

lemma *retraction_maps_to_retract_maps*:
  retraction_maps X Y r s
    ⟹ *retraction_maps X* (*subtopology X* (*s* ' (*topspace Y*))) (*s* ∘ *r*) *id*
 unfolding *retraction_maps_def*
  by (*auto simp*: *continuous_map_compose continuous_map_into_subtopology continuous_map_from_subtopology*)

1.11.20 Continuity

lemma *continuous_on_open*:
  continuous_on S f ⟷
    (∀ *T*. *openin* (*top_of_set* (*f* ' *S*)) *T* ⟶
      openin (*top_of_set S*) (*S* ∩ *f* −' *T*))
 unfolding *continuous_on_open_invariant openin_open Int_def vimage_def Int_commute*
  by (*simp add*: *imp_ex imageI conj_commute eq_commute cong*: *conj_cong*)

lemma *continuous_on_closed*:
  continuous_on S f ⟷
    (∀ *T*. *closedin* (*top_of_set* (*f* ' *S*)) *T* ⟶
      closedin (*top_of_set S*) (*S* ∩ *f* −' *T*))
 unfolding *continuous_on_closed_invariant closedin_closed Int_def vimage_def Int_commute*
  by (*simp add*: *imp_ex imageI conj_commute eq_commute cong*: *conj_cong*)

lemma *continuous_on_imp_closedin*:
  assumes *continuous_on S f closedin* (*top_of_set* (*f* ' *S*)) *T*
  shows *closedin* (*top_of_set S*) (*S* ∩ *f* −' *T*)
  using *assms continuous_on_closed* **by** *blast*

lemma *continuous_map_subtopology_eu* [*simp*]:
  continuous_map (*top_of_set S*) (*subtopology euclidean T*) *h* ⟷ *continuous_on S h* ∧ *h* ' *S* ⊆ *T*
  by (*simp add*: *continuous_map_in_subtopology*)

lemma *continuous_map_euclidean_top_of_set*:
  assumes *eq*: *f* −' *S* = *UNIV* **and** *cont*: *continuous_on UNIV f*
  shows *continuous_map euclidean* (*top_of_set S*) *f*
 by (*simp add*: *cont continuous_map_in_subtopology eq image_subset_iff_subset_vimage*)

1.11.21 Half-global and completely global cases

lemma *continuous_openin_preimage_gen*:
  assumes *continuous_on S f open T*
  shows *openin* (*top_of_set S*) (*S* ∩ *f* −' *T*)
proof −

```
  have *: (S ∩ f -` T) = (S ∩ f -` (T ∩ f ` S))
    by auto
  have openin (top_of_set (f ` S)) (T ∩ f ` S)
    using openin_open_Int[of T f ` S, OF assms(2)] unfolding openin_open by
auto
  then show ?thesis
    using assms(1)[unfolded continuous_on_open, THEN spec[where x=T ∩ f `
S]]
    using * by auto
qed

lemma continuous_closedin_preimage:
  assumes continuous_on S f and closed T
  shows closedin (top_of_set S) (S ∩ f -` T)
proof -
  have *: (S ∩ f -` T) = (S ∩ f -` (T ∩ f ` S))
    by auto
  have closedin (top_of_set (f ` S)) (T ∩ f ` S)
    using closedin_closed_Int[of T f ` S, OF assms(2)]
    by (simp add: Int_commute)
  then show ?thesis
    using assms(1)[unfolded continuous_on_closed, THEN spec[where x=T ∩ f
` S]]
    using * by auto
qed

lemma continuous_openin_preimage_eq:
   continuous_on S f ⟷ (∀ T. open T ⟶ openin (top_of_set S) (S ∩ f -` T))
 by (metis Int_commute continuous_on_open_invariant open_openin openin_subtopology)

lemma continuous_closedin_preimage_eq:
   continuous_on S f ⟷
    (∀ T. closed T ⟶ closedin (top_of_set S) (S ∩ f -` T))
  by (metis Int_commute closedin_closed continuous_on_closed_invariant)

lemma continuous_open_preimage:
  assumes contf: continuous_on S f and open S open T
  shows open (S ∩ f -` T)
proof-
  obtain U where open U (S ∩ f -` T) = S ∩ U
    using continuous_openin_preimage_gen[OF contf ‹open T›]
    unfolding openin_open by auto
  then show ?thesis
    using open_Int[of S U, OF ‹open S›] by auto
qed

lemma continuous_closed_preimage:
  assumes contf: continuous_on S f and closed S closed T
  shows closed (S ∩ f -` T)
```

proof−
obtain *U* **where** *closed U* (*S* ∩ *f* −‘ *T*) = *S* ∩ *U*
using *continuous_closedin_preimage*[*OF contf* ‹*closed T*›]
unfolding *closedin_closed* **by** *auto*
then show *?thesis* **using** *closed_Int*[*of S U*, *OF* ‹*closed S*›] **by** *auto*
qed

lemma *continuous_open_vimage*: *open S* ⟹ (⋀*x*. *continuous* (*at x*) *f*) ⟹ *open* (*f* −‘ *S*)
by (*metis continuous_on_eq_continuous_within open_vimage*)

lemma *continuous_closed_vimage*: *closed S* ⟹ (⋀*x*. *continuous* (*at x*) *f*) ⟹ *closed* (*f* −‘ *S*)
by (*simp add*: *closed_vimage continuous_on_eq_continuous_within*)

lemma *Times_in_interior_subtopology*:
assumes (*x*, *y*) ∈ *U openin* (*top_of_set* (*S* × *T*)) *U*
obtains *V W* **where** *openin* (*top_of_set S*) *V x* ∈ *V*
openin (*top_of_set T*) *W y* ∈ *W* (*V* × *W*) ⊆ *U*
proof −
from *assms* **obtain** *E* **where** *open E U* = *S* × *T* ∩ *E* (*x*, *y*) ∈ *E x* ∈ *S y* ∈ *T*
by (*auto simp*: *openin_open*)
from *open_prod_elim*[*OF* ‹*open E*› ‹(*x*, *y*) ∈ *E*›]
obtain *E1 E2* **where** *open E1 open E2* (*x*, *y*) ∈ *E1* × *E2 E1* × *E2* ⊆ *E*
by *blast*
show *?thesis*
proof
show *openin* (*top_of_set S*) (*E1* ∩ *S*) *openin* (*top_of_set T*) (*E2* ∩ *T*)
using ‹*open E1*› ‹*open E2*› **by** (*auto simp*: *openin_open*)
show *x* ∈ *E1* ∩ *S y* ∈ *E2* ∩ *T*
using ‹(*x*, *y*) ∈ *E1* × *E2*› ‹*x* ∈ *S*› ‹*y* ∈ *T*› **by** *auto*
show (*E1* ∩ *S*) × (*E2* ∩ *T*) ⊆ *U*
using ‹*E1* × *E2* ⊆ *E*› ‹*U* = _› **by** *auto*
qed
qed

lemma *closedin_Times*:
closedin (*top_of_set S*) *S*′ ⟹ *closedin* (*top_of_set T*) *T*′ ⟹
closedin (*top_of_set* (*S* × *T*)) (*S*′ × *T*′)
unfolding *closedin_closed* **using** *closed_Times* **by** *blast*

lemma *openin_Times*:
openin (*top_of_set S*) *S*′ ⟹ *openin* (*top_of_set T*) *T*′ ⟹
openin (*top_of_set* (*S* × *T*)) (*S*′ × *T*′)
unfolding *openin_open* **using** *open_Times* **by** *blast*

lemma *openin_Times_eq*:
fixes *S* :: ′*a*::*topological_space set* **and** *T* :: ′*b*::*topological_space set*
shows

```
    openin (top_of_set (S × T)) (S' × T') ⟷
      S' = {} ∨ T' = {} ∨ openin (top_of_set S) S' ∧ openin (top_of_set T) T'
    (is ?lhs = ?rhs)
proof (cases S' = {} ∨ T' = {})
  case True
  then show ?thesis by auto
next
  case False
  then obtain x y where x ∈ S' y ∈ T'
    by blast
  show ?thesis
  proof
    assume ?lhs
    have openin (top_of_set S) S'
    proof (subst openin_subopen, clarify)
      show ∃ U. openin (top_of_set S) U ∧ x ∈ U ∧ U ⊆ S' if x ∈ S' for x
        using that ‹y ∈ T'› Times_in_interior_subtopology [OF _ ‹?lhs›, of x y]
        by simp (metis mem_Sigma_iff subsetD subsetI)
    qed
    moreover have openin (top_of_set T) T'
    proof (subst openin_subopen, clarify)
      show ∃ U. openin (top_of_set T) U ∧ y ∈ U ∧ U ⊆ T' if y ∈ T' for y
        using that ‹x ∈ S'› Times_in_interior_subtopology [OF _ ‹?lhs›, of x y]
        by simp (metis mem_Sigma_iff subsetD subsetI)
    qed
    ultimately show ?rhs
      by simp
  next
    assume ?rhs
    with False show ?lhs
      by (simp add: openin_Times)
  qed
qed

lemma Lim_transform_within_openin:
  assumes f: (f ⟶ l) (at a within T)
    and openin (top_of_set T) S a ∈ S
    and eq: ⋀x. ⟦x ∈ S; x ≠ a⟧ ⟹ f x = g x
  shows (g ⟶ l) (at a within T)
proof −
  have ∀F x in at a within T. x ∈ T ∧ x ≠ a
    by (simp add: eventually_at_filter)
  moreover
  from ‹openin _ _› obtain U where open U S = T ∩ U
    by (auto simp: openin_open)
  then have a ∈ U using ‹a ∈ S› by auto
  from topological_tendstoD[OF tendsto_ident_at ‹open U› ‹a ∈ U›]
  have ∀F x in at a within T. x ∈ U by auto
  ultimately
```

```
  have ∀F x in at a within T. f x = g x
    by eventually_elim (auto simp: ‹S = _› eq)
  with f show ?thesis
    by (rule Lim_transform_eventually)
qed

lemma continuous_on_open_gen:
  assumes f ∈ S → T
    shows continuous_on S f ⟷
          (∀ U. openin (top_of_set T) U
              ⟶ openin (top_of_set S) (S ∩ f -‘ U))
     (is ?lhs = ?rhs)
proof
  assume ?lhs
  with assms show ?rhs
    apply (simp add: Pi_iff continuous_openin_preimage_eq openin_open)
    by (metis inf.orderE inf_assoc subsetI vimageI vimage_Int)
next
  assume R [rule_format]: ?rhs
  show ?lhs
  proof (clarsimp simp add: continuous_openin_preimage_eq)
    show ⋀T. open T ⟹ openin (top_of_set S) (S ∩ f -‘ T)
    by (metis (no_types) R assms image_subset_iff_funcset image_subset_iff_subset_vimage
inf.orderE inf_assoc openin_open_Int vimage_Int)
  qed
qed

lemma continuous_openin_preimage:
  ⟦continuous_on S f; f ∈ S → T; openin (top_of_set T) U⟧
       ⟹ openin (top_of_set S) (S ∩ f -‘ U)
  by (simp add: continuous_on_open_gen)

lemma continuous_on_closed_gen:
  assumes f ∈ S → T
  shows continuous_on S f ⟷
          (∀ U. closedin (top_of_set T) U
              ⟶ closedin (top_of_set S) (S ∩ f -‘ U))
proof -
  have ∗: U ⊆ T ⟹ S ∩ f -‘ (T - U) = S - (S ∩ f -‘ U) for U
    using assms by blast
  then show ?thesis
      unfolding continuous_on_open_gen [OF assms]
    by (metis closedin_def inf.cobounded1 openin_closedin_eq topspace_euclidean_subtopology)
qed

lemma continuous_closedin_preimage_gen:
  assumes continuous_on S f f ∈ S → T closedin (top_of_set T) U
    shows closedin (top_of_set S) (S ∩ f -‘ U)
using assms continuous_on_closed_gen by blast
```

lemma *continuous_transform_within_openin*:
assumes *continuous* (*at a within T*) *f*
and *openin* (*top_of_set T*) *S a* ∈ *S*
and *eq*: ⋀*x*. *x* ∈ *S* ⟹ *f x* = *g x*
shows *continuous* (*at a within T*) *g*
using *assms* **by** (*simp add*: *Lim_transform_within_openin continuous_within*)

1.11.22 The topology generated by some (open) subsets

In the definition below of a generated topology, the *Empty* case is not necessary, as it follows from *UN* taking for *K* the empty set. However, it is convenient to have, and is never a problem in proofs, so I prefer to write it down explicitly.

We do not require *UNIV* to be an open set, as this will not be the case in applications. (We are thinking of a topology on a subset of *UNIV*, the remaining part of *UNIV* being irrelevant.)

inductive *generate_topology_on* **for** *S* **where**
Empty: *generate_topology_on S* {}
| *Int*: *generate_topology_on S a* ⟹ *generate_topology_on S b* ⟹ *generate_topology_on S* (*a* ∩ *b*)
| *UN*: (⋀*k*. *k* ∈ *K* ⟹ *generate_topology_on S k*) ⟹ *generate_topology_on S* (⋃ *K*)
| *Basis*: *s* ∈ *S* ⟹ *generate_topology_on S s*

lemma *istopology_generate_topology_on*:
istopology (*generate_topology_on S*)
unfolding *istopology_def* **by** (*auto intro*: *generate_topology_on.intros*)

The basic property of the topology generated by a set *S* is that it is the smallest topology containing all the elements of *S*:

lemma *generate_topology_on_coarsest*:
assumes *T*: *istopology T* ⋀*s*. *s* ∈ *S* ⟹ *T s*
and *gen*: *generate_topology_on S s0*
shows *T s0*
using *gen*
by (*induct rule*: *generate_topology_on.induct*) (*use T* **in** ‹*auto simp*: *istopology_def*›)

abbreviation *topology_generated_by*::('*a set set*) ⇒ ('*a topology*)
where *topology_generated_by S* ≡ *topology* (*generate_topology_on S*)

lemma *openin_topology_generated_by_iff*:
openin (*topology_generated_by S*) *s* ⟷ *generate_topology_on S s*
using *topology_inverse'*[*OF istopology_generate_topology_on*[*of S*]] **by** *simp*

lemma *openin_topology_generated_by*:
openin (*topology_generated_by S*) *s* ⟹ *generate_topology_on S s*

using *openin_topology_generated_by_iff* **by** *auto*

lemma *topology_generated_by_topspace* [*simp*]:
topspace (*topology_generated_by S*) = ($\bigcup S$)
proof
{
fix *s* **assume** *openin* (*topology_generated_by S*) *s*
then have *generate_topology_on S s* **by** (*rule openin_topology_generated_by*)
then have *s* ⊆ ($\bigcup S$) **by** (*induct*, *auto*)
}
then show *topspace* (*topology_generated_by S*) ⊆ ($\bigcup S$)
unfolding *topspace_def* **by** *auto*
next
have *generate_topology_on S* ($\bigcup S$)
using *generate_topology_on.UN*[*OF generate_topology_on.Basis*, *of S S*] **by** *simp*
then show ($\bigcup S$) ⊆ *topspace* (*topology_generated_by S*)
unfolding *topspace_def* **using** *openin_topology_generated_by_iff* **by** *auto*
qed

lemma *topology_generated_by_Basis*:
s ∈ *S* ⟹ *openin* (*topology_generated_by S*) *s*
by (*simp add*: *Basis openin_topology_generated_by_iff*)

lemma *generate_topology_on_Inter*:
⟦*finite* $\mathcal{F}$; ⋀*K*. *K* ∈ $\mathcal{F}$ ⟹ *generate_topology_on* $\mathcal{S}$ *K*; $\mathcal{F}$ ≠ {}⟧ ⟹ *generate_topology_on* $\mathcal{S}$ ($\bigcap \mathcal{F}$)
by (*induction* $\mathcal{F}$ *rule*: *finite_induct*; *force intro*: *generate_topology_on.intros*)

1.11.23 Topology bases and sub-bases

lemma *istopology_base_alt*:
istopology (*arbitrary union_of P*) ⟷
(∀ *S T*. (*arbitrary union_of P*) *S* ∧ (*arbitrary union_of P*) *T*
⟶ (*arbitrary union_of P*) (*S* ∩ *T*))
by (*simp add*: *istopology_def*) (*blast intro*: *arbitrary_union_of_Union*)

lemma *istopology_base_eq*:
istopology (*arbitrary union_of P*) ⟷
(∀ *S T*. *P S* ∧ *P T* ⟶ (*arbitrary union_of P*) (*S* ∩ *T*))
by (*simp add*: *istopology_base_alt arbitrary_union_of_Int_eq*)

lemma *istopology_base*:
(⋀*S T*. ⟦*P S*; *P T*⟧ ⟹ *P*(*S* ∩ *T*)) ⟹ *istopology* (*arbitrary union_of P*)
by (*simp add*: *arbitrary_def istopology_base_eq union_of_inc*)

lemma *openin_topology_base_unique*:
openin X = *arbitrary union_of P* ⟷
(∀ *V*. *P V* ⟶ *openin X V*) ∧ (∀ *U x*. *openin X U* ∧ *x* ∈ *U* ⟶ (∃ *V*. *P V*

$\wedge\ x \in V \wedge V \subseteq U))$
  (**is** *?lhs* = *?rhs*)
proof
  assume *?lhs*
  then show *?rhs*
    by (*auto simp*: *union_of_def arbitrary_def*)
next
  assume *R*: *?rhs*
  then have $*$: $\exists \mathcal{U} \subseteq$ *Collect P*. $\bigcup \mathcal{U} = S$ **if** *openin X S* **for** *S*
    using *that* **by** (*rule_tac x*=$\{V.\ P\ V \wedge V \subseteq S\}$ **in** *exI*) *fastforce*
  from *R* **show** *?lhs*
    by (*fastforce simp add*: *union_of_def arbitrary_def intro*: $*$)
qed

lemma *topology_base_unique*:
  assumes $\bigwedge S.\ P\ S \Longrightarrow$ *openin X S*
        $\bigwedge U\ x.\ [\![$*openin X U*; $x \in U]\!] \Longrightarrow \exists B.\ P\ B \wedge x \in B \wedge B \subseteq U$
  shows *topology* (*arbitrary union_of P*) = *X*
proof −
  have *X* = *topology* (*openin X*)
    by (*simp add*: *openin_inverse*)
  also from *assms* **have** *openin X* = *arbitrary union_of P*
    by (*subst openin_topology_base_unique*) *auto*
  finally show *?thesis* **..**
qed

lemma *topology_bases_eq_aux*:
  $[\![$(*arbitrary union_of P*) *S*;
    $\bigwedge U\ x.\ [\![P\ U; x \in U]\!] \Longrightarrow \exists V.\ Q\ V \wedge x \in V \wedge V \subseteq U]\!]$
      $\Longrightarrow$ (*arbitrary union_of Q*) *S*
  by (*metis arbitrary_union_of_alt arbitrary_union_of_idempot*)

lemma *topology_bases_eq*:
  $[\![\bigwedge U\ x.\ [\![P\ U; x \in U]\!] \Longrightarrow \exists V.\ Q\ V \wedge x \in V \wedge V \subseteq U;$
    $\bigwedge V\ x.\ [\![Q\ V; x \in V]\!] \Longrightarrow \exists U.\ P\ U \wedge x \in U \wedge U \subseteq V]\!]$
      $\Longrightarrow$ *topology* (*arbitrary union_of P*) =
          topology (*arbitrary union_of Q*)
 by (*fastforce intro*: *arg_cong* [**where** *f*=*topology*] *elim*: *topology_bases_eq_aux*)

lemma *istopology_subbase*:
  istopology (*arbitrary union_of* (*finite intersection_of P relative_to S*))
  by (*simp add*: *finite_intersection_of_Int istopology_base relative_to_Int*)

lemma *openin_subbase*:
  openin (*topology* (*arbitrary union_of* (*finite intersection_of B relative_to U*)))
S
  $\longleftrightarrow$ (*arbitrary union_of* (*finite intersection_of B relative_to U*)) *S*
  by (*simp add*: *istopology_subbase topology_inverse′*)

lemma *topspace_subbase* [*simp*]:
topspace(*topology* (*arbitrary union_of* (*finite intersection_of B relative_to U*)))
= *U* (**is** *?lhs* = _)
proof
show *?lhs* ⊆ *U*
by (*metis arbitrary_union_of_relative_to openin_subbase openin_topspace relative_to_imp_subset*)
show *U* ⊆ *?lhs*
by (*metis arbitrary_union_of_inc finite_intersection_of_empty inf.orderE istopology_subbase*
openin_subset relative_to_inc subset_UNIV topology_inverse')
qed

lemma *minimal_topology_subbase*:
assumes *X*: ⋀*S*. *P S* ⟹ *openin X S* **and** *openin X U*
and *S*: *openin*(*topology*(*arbitrary union_of* (*finite intersection_of P relative_to U*))) *S*
shows *openin X S*
proof −
have (*arbitrary union_of* (*finite intersection_of P relative_to U*)) *S*
using *S openin_subbase* **by** *blast*
with *X* ‹*openin X U*› **show** *?thesis*
by (*force simp add*: *union_of_def intersection_of_def relative_to_def intro*: *openin_Int_Inter*)
qed

lemma *istopology_subbase_UNIV*:
istopology (*arbitrary union_of* (*finite intersection_of P*))
by (*simp add*: *istopology_base finite_intersection_of_Int*)

lemma *generate_topology_on_eq*:
generate_topology_on S = *arbitrary union_of finite' intersection_of* (λ*x*. *x* ∈ *S*) (**is** *?lhs* = *?rhs*)
proof (*intro ext iffI*)
fix *A*
assume *?lhs A*
then show *?rhs A*
proof *induction*
case (*Int a b*)
then show *?case*
by (*metis* (*mono_tags*, *lifting*) *istopology_base_alt finite'_intersection_of_Int istopology_base*)
next
case (*UN K*)
then show *?case*
by (*simp add*: *arbitrary_union_of_Union*)
next
case (*Basis s*)

```
    then show ?case
     by (simp add: Sup_upper arbitrary_union_of_inc finite'_intersection_of_inc
relative_to_subset_inc)
  qed auto
next
  fix A
  assume ?rhs A
  then obtain 𝒰 where 𝒰: ⋀T. T ∈ 𝒰 ⟹ ∃ℱ. finite' ℱ ∧ ℱ ⊆ S ∧ ⋂ℱ = T
and eq: A = ⋃𝒰
    unfolding union_of_def intersection_of_def by auto
  show ?lhs A
    unfolding eq
  proof (rule generate_topology_on.UN)
    fix T
    assume T ∈ 𝒰
    with 𝒰 obtain ℱ where finite' ℱ ℱ ⊆ S ⋂ℱ = T
      by blast
    have generate_topology_on S (⋂ℱ)
    proof (rule generate_topology_on_Inter)
      show finite ℱ ℱ ≠ {}
        by (auto simp: ‹finite' ℱ›)
      show ⋀K. K ∈ ℱ ⟹ generate_topology_on S K
        by (metis ‹ℱ ⊆ S› generate_topology_on.simps subset_iff)
    qed
    then show generate_topology_on S T
      using ‹⋂ℱ = T› by blast
  qed
qed

lemma continuous_on_generated_topo_iff:
  continuous_map T1 (topology_generated_by S) f ⟷
      ((∀U. U ∈ S ⟶ openin T1 (f−‘U ∩ topspace(T1))) ∧ (f‘(topspace T1) ⊆
(⋃ S)))
unfolding continuous_map_alt topology_generated_by_topspace
proof (auto simp add: topology_generated_by_Basis)
  assume H: ∀U. U ∈ S ⟶ openin T1 (f −‘ U ∩ topspace T1)
  fix U assume openin (topology_generated_by S) U
  then have generate_topology_on S U by (rule openin_topology_generated_by)
  then show openin T1 (f −‘ U ∩ topspace T1)
  proof (induct)
    fix a b
    assume H: openin T1 (f −‘ a ∩ topspace T1) openin T1 (f −‘ b ∩ topspace
T1)
    have f −‘ (a ∩ b) ∩ topspace T1 = (f−‘a ∩ topspace T1) ∩ (f−‘b ∩ topspace
T1)
      by auto
    then show openin T1 (f −‘ (a ∩ b) ∩ topspace T1) using H by auto
  next
    fix K
```

assume *H*: *openin T1* (*f* −‘ *k* ∩ *topspace T1*) **if** *k*∈ *K* **for** *k*
define *L* **where** *L* = {*f* −‘ *k* ∩ *topspace T1*|*k*. *k* ∈ *K*}
have ∗: *openin T1 l* **if** *l* ∈*L* **for** *l* **using** *that H* **unfolding** *L_def* **by** *auto*
have *openin T1* (⋃ *L*) **using** *openin_Union*[*OF* ∗] **by** *simp*
moreover have (⋃ *L*) = (*f* −‘ ⋃ *K* ∩ *topspace T1*) **unfolding** *L_def* **by** *auto*
ultimately show *openin T1* (*f* −‘ ⋃ *K* ∩ *topspace T1*) **by** *simp*
qed (*auto simp add*: *H*)
qed

lemma *continuous_on_generated_topo*:
assumes ⋀*U*. *U* ∈*S* ⟹ *openin T1* (*f*−‘*U* ∩ *topspace*(*T1*))
f‘(*topspace T1*) ⊆ (⋃ *S*)
shows *continuous_map T1* (*topology_generated_by S*) *f*
using *assms continuous_on_generated_topo_iff* **by** *blast*

1.11.24 Continuity via bases/subbases, hence upper and lower semicontinuity

lemma *continuous_map_into_topology_base*:
assumes *P*: *openin Y* = *arbitrary union_of P*
and *f*: ⋀*x*. *x* ∈ *topspace X* ⟹ *f x* ∈ *topspace Y*
and *ope*: ⋀*U*. *P U* ⟹ *openin X* {*x* ∈ *topspace X*. *f x* ∈ *U*}
shows *continuous_map X Y f*
proof −
have ∗: ⋀*U*. (⋀*t*. *t* ∈ *U* ⟹ *P t*) ⟹ *openin X* {*x* ∈ *topspace X*. ∃ *U*∈*U*. *f x* ∈ *U*}
by (*smt* (*verit*) *Ball_Collect ope mem_Collect_eq openin_subopen*)
show *?thesis*
using *P* **by** (*auto simp*: *continuous_map_def arbitrary_def union_of_def intro*!: *f* ∗)
qed

lemma *continuous_map_into_topology_base_eq*:
assumes *P*: *openin Y* = *arbitrary union_of P*
shows
continuous_map X Y f ⟷
f ∈ *topspace X* → *topspace Y* ∧ (∀ *U*. *P U* ⟶ *openin X* {*x* ∈ *topspace X*. *f x* ∈ *U*})
(**is** *?lhs*=*?rhs*)
proof
assume *L*: *?lhs*
then have *f* ∈ *topspace X* → *topspace Y*
by (*simp add*: *continuous_map_funspace*)
moreover have ⋀*U*. *P U* ⟹ *openin X* {*x* ∈ *topspace X*. *f x* ∈ *U*}
using *L assms continuous_map openin_topology_base_unique* **by** *fastforce*
ultimately show *?rhs* **by** *auto*
qed (*simp add*: *assms Pi_iff continuous_map_into_topology_base*)

lemma *continuous_map_into_topology_subbase*:

fixes U P
defines $Y \equiv$ *topology*(*arbitrary union_of* (*finite intersection_of* P *relative_to* U))
assumes f: $\bigwedge x.\ x \in$ *topspace* $X \Longrightarrow f\ x \in$ *topspace* Y
and *ope*: $\bigwedge U.\ P\ U \Longrightarrow$ *openin* X $\{x \in$ *topspace* $X.\ f\ x \in U\}$
shows *continuous_map* X Y f
proof (*intro continuous_map_into_topology_base*)
show *openin* $Y =$ *arbitrary union_of* (*finite intersection_of* P *relative_to* U)
unfolding *Y_def* **using** *istopology_subbase topology_inverse'* **by** *blast*
show *openin* X $\{x \in$ *topspace* $X.\ f\ x \in V\}$
if §: (*finite intersection_of* P *relative_to* U) V **for** V
proof −
define *finv* **where** *finv* $\equiv \lambda V.\ \{x \in$ *topspace* $X.\ f\ x \in V\}$
obtain $\mathcal{U}$ **where** $\mathcal{U}$: *finite* $\mathcal{U}$ $\bigwedge V.\ V \in \mathcal{U} \Longrightarrow P\ V$
$\{x \in$ *topspace* $X.\ f\ x \in V\} = (\bigcap V \in$ *insert* U $\mathcal{U}.$ *finv* $V)$
using § **by** (*fastforce simp*: *finv_def intersection_of_def relative_to_def*)
show *?thesis*
unfolding $\mathcal{U}$
proof (*intro openin_Inter ope*)
have U: $U =$ *topspace* Y
unfolding *Y_def* **using** *topspace_subbase* **by** *fastforce*
fix V
assume V: $V \in$ *finv* ‘ *insert* U $\mathcal{U}$
with U f **have** *openin* X $\{x \in$ *topspace* $X.\ f\ x \in U\}$
by (*auto simp*: *openin_subopen* [*of* X *Collect* _])
then show *openin* X V
using V $\mathcal{U}(2)$ *ope* **by** (*fastforce simp*: *finv_def*)
qed (*use* ‹*finite* $\mathcal{U}$› **in** *auto*)
qed
qed (*use* f **in** *auto*)

lemma *continuous_map_into_topology_subbase_eq*:
assumes $Y =$ *topology*(*arbitrary union_of* (*finite intersection_of* P *relative_to* U))
shows
continuous_map X Y $f \longleftrightarrow$
$f \in$ *topspace* $X \to$ *topspace* $Y \wedge (\forall U.\ P\ U \longrightarrow$ *openin* X $\{x \in$ *topspace* $X.\ f\ x \in U\})$
(**is** *?lhs=?rhs*)
proof
assume L: *?lhs*
show *?rhs*
proof (*intro conjI strip*)
show $f \in$ *topspace* $X \to$ *topspace* Y
using L *continuous_map_def* **by** *fastforce*
fix V
assume P V
have $U =$ *topspace* Y
unfolding *assms* **using** *topspace_subbase* **by** *fastforce*

```
    then have eq: {x ∈ topspace X. f x ∈ V} = {x ∈ topspace X. f x ∈ U ∩ V}
      using L by (auto simp: continuous_map)
    have openin Y (U ∩ V)
      unfolding assms openin_subbase
       by (meson ‹P V› arbitrary_union_of_inc finite_intersection_of_inc rela-
tive_to_inc)
    show openin X {x ∈ topspace X. f x ∈ V}
      using L ‹openin Y (U ∩ V)› continuous_map eq by fastforce
  qed
next
  show ?rhs ⟹ ?lhs
    unfolding assms
    by (intro continuous_map_into_topology_subbase) auto
qed

lemma subbase_subtopology_euclidean:
  fixes U :: 'a::order_topology set
  shows
  topology
    (arbitrary union_of
       (finite intersection_of (λx. x ∈ range greaterThan ∪ range lessThan) rela-
tive_to U))
 = subtopology euclidean U
proof −
   have ∃ V. (finite intersection_of (λx. x ∈ range greaterThan ∨ x ∈ range
lessThan)) V ∧ x ∈ V ∧ V ⊆ W
    if open W x ∈ W for W and x::'a
    using ‹open W› [unfolded open_generated_order] ‹x ∈ W›
  proof (induct rule: generate_topology.induct)
    case UNIV
    then show ?case
      using finite_intersection_of_empty by blast
  next
    case (Int a b)
    then show ?case
       by (meson Int_iff finite_intersection_of_Int inf_mono)
  next
    case (UN K)
    then show ?case
      by (meson Union_iff subset_iff)
  next
    case (Basis s)
    then show ?case
      by (metis (no_types, lifting) Un_iff finite_intersection_of_inc order_refl)
  qed
  moreover
  have ⋀V::'a set. (finite intersection_of (λx. x ∈ range greaterThan ∨ x ∈ range
lessThan)) V ⟹ open V
    by (force simp: intersection_of_def subset_iff)
```

ultimately have $*$: *openin* (*euclidean*::'*a topology*) =
(*arbitrary union_of* (*finite intersection_of* ($\lambda x.\ x \in$ *range greaterThan* $\vee$ $x \in$ *range lessThan*)))
by (*smt* (*verit*, *best*) *openin_topology_base_unique open_openin*)
show *?thesis*
unfolding *subtopology_def arbitrary_union_of_relative_to* [*symmetric*]
apply (*simp add*: *relative_to_def flip*: $*$)
by (*metis Int_commute*)
qed

lemma *continuous_map_upper_lower_semicontinuous_lt_gen*:
fixes U :: '*a*::*order_topology set*
shows *continuous_map* X (*subtopology euclidean* U) $f \longleftrightarrow$
$(\forall x \in$ *topspace* $X.\ f\ x \in U) \wedge$
$(\forall a.$ *openin* X $\{x \in$ *topspace* $X.\ f\ x > a\}) \wedge$
$(\forall a.$ *openin* X $\{x \in$ *topspace* $X.\ f\ x < a\})$
by (*auto simp*: *continuous_map_into_topology_subbase_eq* [*OF subbase_subtopology_euclidean* [*symmetric*, *of U*]]
greaterThan_def lessThan_def image_iff simp flip: *all_simps*)

lemma *continuous_map_upper_lower_semicontinuous_lt*:
fixes f :: '*a* $\Rightarrow$ '*b*::*order_topology*
shows *continuous_map* X *euclidean* $f \longleftrightarrow$
$(\forall a.$ *openin* X $\{x \in$ *topspace* $X.\ f\ x > a\}) \wedge$
$(\forall a.$ *openin* X $\{x \in$ *topspace* $X.\ f\ x < a\})$
using *continuous_map_upper_lower_semicontinuous_lt_gen* [**where** U=*UNIV*]
by *auto*

lemma *Int_Collect_imp_eq*: $A \cap \{x.\ x{\in}A \longrightarrow P\ x\} = \{x{\in}A.\ P\ x\}$
by *blast*

lemma *continuous_map_upper_lower_semicontinuous_le_gen*:
shows
continuous_map X (*subtopology euclideanreal* U) $f \longleftrightarrow$
$(\forall x \in$ *topspace* $X.\ f\ x \in U) \wedge$
$(\forall a.$ *closedin* X $\{x \in$ *topspace* $X.\ f\ x \geq a\}) \wedge$
$(\forall a.$ *closedin* X $\{x \in$ *topspace* $X.\ f\ x \leq a\})$
unfolding *continuous_map_upper_lower_semicontinuous_lt_gen*
by (*auto simp*: *closedin_def Diff_eq Compl_eq not_le Int_Collect_imp_eq*)

lemma *continuous_map_upper_lower_semicontinuous_le*:
continuous_map X *euclideanreal* $f \longleftrightarrow$
$(\forall a.$ *closedin* X $\{x \in$ *topspace* $X.\ f\ x \geq a\}) \wedge$
$(\forall a.$ *closedin* X $\{x \in$ *topspace* $X.\ f\ x \leq a\})$
using *continuous_map_upper_lower_semicontinuous_le_gen* [**where** U=*UNIV*]
by *auto*

lemma *continuous_map_upper_lower_semicontinuous_lte_gen*:
continuous_map X (*subtopology euclideanreal* U) $f \longleftrightarrow$

$(\forall x \in topspace\ X.\ f\ x \in U) \wedge$
$(\forall a.\ openin\ X\ \{x \in topspace\ X.\ f\ x < a\}) \wedge$
$(\forall a.\ closedin\ X\ \{x \in topspace\ X.\ f\ x \leq a\})$
unfolding *continuous_map_upper_lower_semicontinuous_lt_gen*
by (*auto simp*: *closedin_def Diff_eq Compl_eq not_le Int_Collect_imp_eq*)

lemma *continuous_map_upper_lower_semicontinuous_lte*:
continuous_map X euclideanreal f $\longleftrightarrow$
$(\forall a.\ openin\ X\ \{x \in topspace\ X.\ f\ x < a\}) \wedge$
$(\forall a.\ closedin\ X\ \{x \in topspace\ X.\ f\ x \leq a\})$
using *continuous_map_upper_lower_semicontinuous_lte_gen* [**where** *U*=*UNIV*]
by *auto*

1.11.25 Pullback topology

Pulling back a topology by map gives again a topology. *subtopology* is a special case of this notion, pulling back by the identity. We introduce the general notion as we will need it to define the strong operator topology on the space of continuous linear operators, by pulling back the product topology on the space of all functions.

pullback_topology A f T is the pullback of the topology T by the map f on the set A.

definition *pullback_topology*::('*a set*) ⇒ ('*a* ⇒ '*b*) ⇒ ('*b topology*) ⇒ ('*a topology*)
where *pullback_topology A f T* = *topology* (λ*S*. ∃ *U*. *openin T U* ∧ *S* = *f*−‘*U* ∩ *A*)

lemma *istopology_pullback_topology*:
istopology (λ*S*. ∃ *U*. *openin T U* ∧ *S* = *f*−‘*U* ∩ *A*)
unfolding *istopology_def* **proof** (*auto*)
fix *K* **assume** ∀ *S*∈*K*. ∃ *U*. *openin T U* ∧ *S* = *f* −‘ *U* ∩ *A*
then have ∃ *U*. ∀ *S*∈*K*. *openin T* (*U S*) ∧ *S* = *f*−‘(*U S*) ∩ *A*
by (*rule bchoice*)
then obtain *U* **where** *U*: ∀ *S*∈*K*. *openin T* (*U S*) ∧ *S* = *f*−‘(*U S*) ∩ *A*
by *blast*
define *V* **where** *V* = (⋃ *S*∈*K*. *U S*)
have *openin T V* ⋃ *K* = *f* −‘ *V* ∩ *A* **unfolding** *V_def* **using** *U* **by** *auto*
then show ∃ *V*. *openin T V* ∧ ⋃ *K* = *f* −‘ *V* ∩ *A* **by** *auto*
qed

lemma *openin_pullback_topology*:
openin (*pullback_topology A f T*) *S* $\longleftrightarrow$ (∃ *U*. *openin T U* ∧ *S* = *f*−‘*U* ∩ *A*)
unfolding *pullback_topology_def topology_inverse'*[*OF istopology_pullback_topology*]
by *auto*

lemma *topspace_pullback_topology*:
topspace (*pullback_topology A f T*) = *f*−‘(*topspace T*) ∩ *A*
by (*auto simp add*: *topspace_def openin_pullback_topology*)

proposition *continuous_map_pullback* [*intro*]:
assumes *continuous_map T1 T2 g*
shows *continuous_map* (*pullback_topology A f T1*) *T2* (*g o f*)
unfolding *continuous_map_alt*
proof (*auto*)
fix *U*::'*b set* **assume** *openin T2 U*
then have *openin T1* (*g*−'*U* ∩ *topspace T1*)
using *assms* **unfolding** *continuous_map_alt* **by** *auto*
have (*g o f*)−'*U* ∩ *topspace* (*pullback_topology A f T1*) = (*g o f*)−'*U* ∩ *A* ∩ *f*−'(*topspace T1*)
unfolding *topspace_pullback_topology* **by** *auto*
also have ... = *f*−'(*g*−'*U* ∩ *topspace T1*) ∩ *A*
by *auto*
also have *openin* (*pullback_topology A f T1*) (...)
unfolding *openin_pullback_topology* **using** ‹*openin T1* (*g*−'*U* ∩ *topspace T1*)›
by *auto*
finally show *openin* (*pullback_topology A f T1*) ((*g* ∘ *f*) −' *U* ∩ *topspace* (*pullback_topology A f T1*))
by *auto*
next
fix *x* **assume** *x* ∈ *topspace* (*pullback_topology A f T1*)
then have *f x* ∈ *topspace T1*
unfolding *topspace_pullback_topology* **by** *auto*
then show *g* (*f x*) ∈ *topspace T2*
using *assms* **unfolding** *continuous_map_def* **by** *auto*
qed

proposition *continuous_map_pullback'* [*intro*]:
assumes *continuous_map T1 T2* (*f o g*) *topspace T1* ⊆ *g*−'*A*
shows *continuous_map T1* (*pullback_topology A f T2*) *g*
unfolding *continuous_map_alt*
proof (*auto*)
fix *U* **assume** *openin* (*pullback_topology A f T2*) *U*
then have ∃ *V*. *openin T2 V* ∧ *U* = *f*−'*V* ∩ *A*
unfolding *openin_pullback_topology* **by** *auto*
then obtain *V* **where** *openin T2 V U* = *f*−'*V* ∩ *A*
by *blast*
then have *g* −' *U* ∩ *topspace T1* = *g*−'(*f*−'*V* ∩ *A*) ∩ *topspace T1*
by *blast*
also have ... = (*f o g*)−'*V* ∩ (*g*−'*A* ∩ *topspace T1*)
by *auto*
also have ... = (*f o g*)−'*V* ∩ *topspace T1*
using *assms*(*2*) **by** *auto*
also have *openin T1* (...)
using *assms*(*1*) ‹*openin T2 V*› **by** *auto*
finally show *openin T1* (*g* −' *U* ∩ *topspace T1*) **by** *simp*
next
fix *x* **assume** *x* ∈ *topspace T1*

have $(f\ o\ g)\ x \in$ *topspace T2*
using *assms*(1) ‹$x \in$ *topspace T1*› **unfolding** *continuous_map_def* **by** *auto*
then have $g\ x \in f-`(\mathit{topspace}\ T2)$
unfolding *comp_def* **by** *blast*
moreover have $g\ x \in A$ **using** *assms*(2) ‹$x \in$ *topspace T1*› **by** *blast*
ultimately show $g\ x \in$ *topspace* (*pullback_topology A f T2*)
unfolding *topspace_pullback_topology* **by** *blast*
qed

1.11.26 Proper maps (not a priori assumed continuous)

definition *proper_map*
where
proper_map X Y f ≡
closed_map X Y f ∧ (∀ $y \in$ *topspace Y*. *compactin X* {$x \in$ *topspace X*. $f\ x = y$})

lemma *proper_imp_closed_map*:
proper_map X Y f ⟹ *closed_map X Y f*
by (*simp add*: *proper_map_def*)

lemma *proper_map_imp_subset_topspace*:
proper_map X Y f ⟹ $f \in (\mathit{topspace}\ X) \to \mathit{topspace}\ Y$
by (*simp add*: *closed_map_imp_subset_topspace proper_map_def*)

lemma *proper_map_restriction*:
assumes *proper_map X Y f* {$x \in$ *topspace X*. $f\ x \in V$} = U
shows *proper_map* (*subtopology X U*) (*subtopology Y V*) f
proof −
have [*simp*]: {$x \in$ *topspace X*. $f\ x \in V \wedge f\ x = y$} = {$x \in$ *topspace X*. $f\ x = y$}
if $y \in V$ **for** y
using *that* **by** *auto*
show *?thesis*
using *assms* **unfolding** *proper_map_def* **using** *closed_map_restriction*
by (*force simp*: *compactin_subtopology*)
qed

lemma *closed_injective_imp_proper_map*:
assumes *f*: *closed_map X Y f* **and** *inj*: *inj_on f* (*topspace X*)
shows *proper_map X Y f*
unfolding *proper_map_def*
proof (*clarsimp simp*: *f*)
show *compactin X* {$x \in$ *topspace X*. $f\ x = y$}
if $y \in$ *topspace Y* **for** y
using *inj_on_eq_iff* [*OF inj*] *that*
proof −
have {$x \in$ *topspace X*. $f\ x = y$} = {} ∨ (∃ $a \in$ *topspace X*. {$x \in$ *topspace X*. $f\ x = y$} = {a})
using *inj_on_eq_iff* [*OF inj*] **by** *auto*

```
    then show ?thesis
      using that by (metis (no_types, lifting) compactin_empty compactin_sing)
  qed
qed

lemma injective_imp_proper_eq_closed_map:
   inj_on f (topspace X) ⟹ (proper_map X Y f ⟷ closed_map X Y f)
  using closed_injective_imp_proper_map proper_imp_closed_map by blast

lemma homeomorphic_imp_proper_map:
   homeomorphic_map X Y f ⟹ proper_map X Y f
 by (simp add: closed_injective_imp_proper_map homeomorphic_eq_everything_map)

lemma compactin_proper_map_preimage:
  assumes f: proper_map X Y f and compactin Y K
  shows compactin X {x. x ∈ topspace X ∧ f x ∈ K}
proof −
  have f ∈ (topspace X) → topspace Y
    by (simp add: f proper_map_imp_subset_topspace)
  have ∗: ⋀y. y ∈ topspace Y ⟹ compactin X {x ∈ topspace X. f x = y}
    using f by (auto simp: proper_map_def)
  show ?thesis
    unfolding compactin_def
  proof clarsimp
    show ∃ℱ. finite ℱ ∧ ℱ ⊆ 𝒰 ∧ {x ∈ topspace X. f x ∈ K} ⊆ ⋃ℱ
      if 𝒰: ∀U∈𝒰. openin X U and sub: {x ∈ topspace X. f x ∈ K} ⊆ ⋃𝒰
      for 𝒰
    proof −
      have ∀y ∈ K. ∃𝒱. finite 𝒱 ∧ 𝒱 ⊆ 𝒰  ∧ {x ∈ topspace X. f x = y} ⊆ ⋃𝒱
      proof
        fix y
        assume y ∈ K
        then have compactin X {x ∈ topspace X. f x = y}
          by (metis ∗ ‹compactin Y K› compactin_subspace subsetD)
        with ‹y ∈ K› show ∃𝒱. finite 𝒱 ∧ 𝒱 ⊆ 𝒰  ∧ {x ∈ topspace X. f x = y} ⊆
⋃𝒱
          unfolding compactin_def using 𝒰 sub by fastforce
      qed
      then obtain 𝒱 where 𝒱: ⋀y. y ∈ K ⟹ finite (𝒱 y) ∧ 𝒱 y ⊆ 𝒰  ∧ {x ∈
topspace X. f x = y} ⊆ ⋃(𝒱 y)
        by (metis (full_types))
      define F where F ≡ λy. topspace Y − f ‘ (topspace X − ⋃(𝒱 y))
      have ∃ℱ. finite ℱ ∧ ℱ ⊆ F ‘ K ∧ K ⊆ ⋃ℱ
      proof (rule compactinD [OF ‹compactin Y K›])
        have ⋀x. x ∈ K ⟹ closedin Y (f ‘ (topspace X − ⋃(𝒱 x)))
          using f unfolding proper_map_def closed_map_def
          by (meson 𝒰 𝒱 openin_Union openin_closedin_eq subsetD)
        then show openin Y U if U ∈ F ‘ K for U
          using that by (auto simp: F_def)
```

```
      show K ⊆ ⋃(F ‘ K)
        using 𝒱 ‹compactin Y K› unfolding F_def compactin_def by fastforce
      qed
      then obtain J where finite J J ⊆ K and J: K ⊆ ⋃(F ‘ J)
        by (auto simp: ex_finite_subset_image)
      show ?thesis
        unfolding F_def
      proof (intro exI conjI)
        show finite (⋃(𝒱 ‘ J))
          using 𝒱 ‹J ⊆ K› ‹finite J› by blast
        show ⋃(𝒱 ‘ J) ⊆ 𝒰
          using 𝒱 ‹J ⊆ K› by blast
        show {x ∈ topspace X. f x ∈ K} ⊆ ⋃(⋃(𝒱 ‘ J))
          using J ‹J ⊆ K› unfolding F_def by auto
      qed
    qed
  qed
qed


lemma compact_space_proper_map_preimage:
  assumes f: proper_map X Y f and fim: f ‘ (topspace X) = topspace Y and
compact_space Y
  shows compact_space X
proof −
  have eq: topspace X = {x ∈ topspace X. f x ∈ topspace Y}
    using fim by blast
  moreover have compactin Y (topspace Y)
    using ‹compact_space Y› compact_space_def by auto
  ultimately show ?thesis
    unfolding compact_space_def
    using eq f compactin_proper_map_preimage by fastforce
qed

lemma proper_map_alt:
   proper_map X Y f ⟷
    closed_map X Y f ∧ (∀K. compactin Y K ⟶ compactin X {x. x ∈ topspace
X ∧ f x ∈ K})
  proof (intro iffI conjI allI impI)
  show compactin X {x ∈ topspace X. f x ∈ K}
    if proper_map X Y f and compactin Y K for K
    using that by (simp add: compactin_proper_map_preimage)
  show proper_map X Y f
    if f: closed_map X Y f ∧ (∀K. compactin Y K ⟶ compactin X {x ∈ topspace
X. f x ∈ K})
  proof −
    have compactin X {x ∈ topspace X. f x = y} if y ∈ topspace Y for y
    proof −
      have compactin X {x ∈ topspace X. f x ∈ {y}}
```

```
      using f compactin_sing that by fastforce
    then show ?thesis
      by auto
  qed
  with f show ?thesis
    by (auto simp: proper_map_def)
 qed
qed (simp add: proper_imp_closed_map)

lemma proper_map_on_empty [simp]: proper_map trivial_topology Y f
 by (auto simp: proper_map_def closed_map_on_empty)

lemma proper_map_id [simp]:
  proper_map X X id
proof (clarsimp simp: proper_map_alt closed_map_id)
 fix K
 assume K: compactin X K
 then have {a ∈ topspace X. a ∈ K} = K
   by (simp add: compactin_subspace subset_antisym subset_iff)
 then show compactin X {a ∈ topspace X. a ∈ K}
   using K by auto
qed

lemma proper_map_compose:
 assumes proper_map X Y f proper_map Y Z g
 shows proper_map X Z (g ∘ f)
proof −
  have closed_map X Y f and f: ⋀K. compactin Y K ⟹ compactin X {x ∈
topspace X. f x ∈ K}
   and closed_map Y Z g and g: ⋀K. compactin Z K ⟹ compactin Y {x ∈
topspace Y. g x ∈ K}
   using assms by (auto simp: proper_map_alt)
 show ?thesis
   unfolding proper_map_alt
 proof (intro conjI allI impI)
   show closed_map X Z (g ∘ f)
     using ‹closed_map X Y f› ‹closed_map Y Z g› closed_map_compose by
blast
   have {x ∈ topspace X. g (f x) ∈ K} = {x ∈ topspace X. f x ∈ {b ∈ topspace
Y. g b ∈ K}} for K
     using ‹closed_map X Y f› closed_map_imp_subset_topspace by blast
   then show compactin X {x ∈ topspace X. (g ∘ f) x ∈ K}
     if compactin Z K for K
     using f [OF g [OF that]] by auto
 qed
qed

lemma proper_map_const:
   proper_map X Y (λx. c) ⟷ compact_space X ∧ (X = trivial_topology ∨
```

closedin Y {c})
proof (*cases X = trivial_topology*)
case *True*
then show *?thesis*
by *simp*
next
case *False*
have ∗: *compactin X {x ∈ topspace X. c = y}* **if** *compact_space X* **for** *y*
using *that* **unfolding** *compact_space_def*
by (*metis* (*mono_tags*, *lifting*) *compactin_empty empty_subsetI mem_Collect_eq subsetI subset_antisym*)
then show *?thesis*
using *closed_compactin closedin_subset*
by (*force simp*: *False proper_map_def closed_map_const compact_space_def*)
qed

lemma *proper_map_inclusion*:
S ⊆ topspace X ⟹ proper_map (subtopology X S) X id ⟷ closedin X S ∧ (∀ k. compactin X k ⟶ compactin X (S ∩ k))
by (*metis closed_Int_compactin closed_map_inclusion_eq inf.absorb_iff2 inj_on_id injective_imp_proper_eq_closed_map*)

lemma *proper_map_into_subtopology*:
⟦proper_map X Y f; f ∈ topspace X → C⟧ ⟹ proper_map X (subtopology Y C) f
by (*simp add*: *closed_map_into_subtopology compactin_subtopology proper_map_alt*)

lemma *proper_map_from_composition_left*:
assumes *gf*: *proper_map X Z (g ∘ f)* **and** *contf*: *continuous_map X Y f* **and** *fim*: *f ‘ topspace X = topspace Y*
shows *proper_map Y Z g*
unfolding *proper_map_def*
proof (*intro strip conjI*)
show *closed_map Y Z g*
by (*meson closed_map_from_composition_left gf contf fim proper_imp_closed_map*)
fix *z* **assume** *z ∈ topspace Z*
have *eq*: *{y ∈ topspace Y. g y = z} = f ‘ {x ∈ topspace X. (g ∘ f) x = z}*
using *fim* **by** *force*
show *compactin Y {x ∈ topspace Y. g x = z}*
unfolding *eq*
proof (*rule image_compactin* [*OF _ contf*])
show *compactin X {x ∈ topspace X. (g ∘ f) x = z}*
using ‹*z ∈ topspace Z*› *gf proper_map_def* **by** *fastforce*
qed
qed

lemma *proper_map_from_composition_right_inj*:
assumes *gf*: *proper_map X Z (g ∘ f)* **and** *fim*: *f ∈ topspace X → topspace Y*
and *contf*: *continuous_map Y Z g* **and** *inj*: *inj_on g (topspace Y)*

shows *proper_map X Y f*
unfolding *proper_map_def*
proof (*intro strip conjI*)
show *closed_map X Y f*
by (*meson closed_map_from_composition_right assms proper_imp_closed_map*)
fix *y*
assume *y ∈ topspace Y*
with *fim inj* **have** *eq*: {*x ∈ topspace X. f x = y*} = {*x ∈ topspace X. (g ∘ f) x = g y*}
by (*auto simp*: *Pi_iff inj_onD*)
show *compactin X* {*x ∈ topspace X. f x = y*}
using *contf gf* ‹*y ∈ topspace Y*›
unfolding *eq continuous_map_def proper_map_def*
by *blast*
qed

1.11.27 Perfect maps (proper, continuous and surjective)

definition *perfect_map*
where *perfect_map X Y f ≡ continuous_map X Y f ∧ proper_map X Y f ∧ f ‘ (topspace X) = topspace Y*

lemma *homeomorphic_imp_perfect_map*:
homeomorphic_map X Y f ⟹ perfect_map X Y f
by (*simp add*: *homeomorphic_eq_everything_map homeomorphic_imp_proper_map perfect_map_def*)

lemma *perfect_imp_quotient_map*:
perfect_map X Y f ⟹ quotient_map X Y f
by (*simp add*: *continuous_closed_imp_quotient_map perfect_map_def proper_map_def*)

lemma *homeomorphic_eq_injective_perfect_map*:
homeomorphic_map X Y f ⟷ perfect_map X Y f ∧ inj_on f (topspace X)
using *homeomorphic_imp_perfect_map homeomorphic_map_def perfect_imp_quotient_map*
by *blast*

lemma *perfect_injective_eq_homeomorphic_map*:
perfect_map X Y f ∧ inj_on f (topspace X) ⟷ homeomorphic_map X Y f
by (*simp add*: *homeomorphic_eq_injective_perfect_map*)

lemma *perfect_map_id* [*simp*]: *perfect_map X X id*
by (*simp add*: *homeomorphic_imp_perfect_map*)

lemma *perfect_map_compose*:
⟦*perfect_map X Y f*; *perfect_map Y Z g*⟧ *⟹ perfect_map X Z (g ∘ f)*
by (*meson continuous_map_compose perfect_imp_quotient_map perfect_map_def proper_map_compose quotient_map_compose_eq quotient_map_def*)

lemma *perfect_imp_continuous_map*:

perfect_map X Y f ⟹ *continuous_map X Y f*
using *perfect_map_def* **by** *blast*

lemma *perfect_imp_closed_map*:
perfect_map X Y f ⟹ *closed_map X Y f*
by (*simp add*: *perfect_map_def proper_map_def*)

lemma *perfect_imp_proper_map*:
perfect_map X Y f ⟹ *proper_map X Y f*
by (*simp add*: *perfect_map_def*)

lemma *perfect_imp_surjective_map*:
perfect_map X Y f ⟹ *f* ' (*topspace X*) = *topspace Y*
by (*simp add*: *perfect_map_def*)

lemma *perfect_map_from_composition_left*:
assumes *perfect_map X Z* (*g* ∘ *f*) **and** *continuous_map X Y f*
and *continuous_map Y Z g* **and** *f* ' *topspace X* = *topspace Y*
shows *perfect_map Y Z g*
using *assms* **unfolding** *perfect_map_def*
by (*metis image_comp proper_map_from_composition_left*)

end

1.12 *F*-Sigma and *G*-Delta sets in a Topological Space

An *F*-sigma set is a countable union of closed sets; a *G*-delta set is the dual.

theory *FSigma*
imports *Abstract_Topology*
begin

definition *fsigma_in*
where *fsigma_in X* ≡ *countable union_of closedin X*

definition *gdelta_in*
where *gdelta_in X* ≡ (*countable intersection_of openin X*) *relative_to topspace X*

lemma *fsigma_in_ascending*:
fsigma_in X S ⟷ (∃ *C*. (∀ *n*. *closedin X* (*C n*)) ∧ (∀ *n*. *C n* ⊆ *C*(*Suc n*)) ∧ ⋃ (*range C*) = *S*)
unfolding *fsigma_in_def*
by (*metis closedin_Un countable_union_of_ascending closedin_empty*)

lemma *gdelta_in_alt*:
gdelta_in X S ⟷ *S* ⊆ *topspace X* ∧ (*countable intersection_of openin X*) *S*

proof −
have (*countable intersection_of openin X*) (*topspace X*)
by (*simp add*: *countable_intersection_of_inc*)
then show *?thesis*
unfolding *gdelta_in_def*
by (*metis countable_intersection_of_inter relative_to_def relative_to_imp_subset relative_to_subset_inc*)
qed

lemma *fsigma_in_subset*: *fsigma_in X S* ⟹ *S* ⊆ *topspace X*
using *closedin_subset* **by** (*fastforce simp*: *fsigma_in_def union_of_def subset_iff*)

lemma *gdelta_in_subset*: *gdelta_in X S* ⟹ *S* ⊆ *topspace X*
by (*simp add*: *gdelta_in_alt*)

lemma *closed_imp_fsigma_in*: *closedin X S* ⟹ *fsigma_in X S*
by (*simp add*: *countable_union_of_inc fsigma_in_def*)

lemma *open_imp_gdelta_in*: *openin X S* ⟹ *gdelta_in X S*
by (*simp add*: *countable_intersection_of_inc gdelta_in_alt openin_subset*)

lemma *fsigma_in_empty* [*iff*]: *fsigma_in X* {}
by (*simp add*: *closed_imp_fsigma_in*)

lemma *gdelta_in_empty* [*iff*]: *gdelta_in X* {}
by (*simp add*: *open_imp_gdelta_in*)

lemma *fsigma_in_topspace* [*iff*]: *fsigma_in X* (*topspace X*)
by (*simp add*: *closed_imp_fsigma_in*)

lemma *gdelta_in_topspace* [*iff*]: *gdelta_in X* (*topspace X*)
by (*simp add*: *open_imp_gdelta_in*)

lemma *fsigma_in_Union*:
⟦*countable T*; ⋀*S*. *S* ∈ *T* ⟹ *fsigma_in X S*⟧ ⟹ *fsigma_in X* (⋃ *T*)
by (*simp add*: *countable_union_of_Union fsigma_in_def*)

lemma *fsigma_in_Un*:
⟦*fsigma_in X S*; *fsigma_in X T*⟧ ⟹ *fsigma_in X* (*S* ∪ *T*)
by (*simp add*: *countable_union_of_Un fsigma_in_def*)

lemma *fsigma_in_Int*:
⟦*fsigma_in X S*; *fsigma_in X T*⟧ ⟹ *fsigma_in X* (*S* ∩ *T*)
by (*simp add*: *closedin_Int countable_union_of_Int fsigma_in_def*)

lemma *gdelta_in_Inter*:
⟦*countable T*; *T*≠{}; ⋀*S*. *S* ∈ *T* ⟹ *gdelta_in X S*⟧ ⟹ *gdelta_in X* (⋂ *T*)
by (*simp add*: *Inf_less_eq countable_intersection_of_Inter gdelta_in_alt*)

lemma *gdelta_in_Int*:
  ⟦*gdelta_in X S*; *gdelta_in X T*⟧ ⟹ *gdelta_in X* (*S* ∩ *T*)
  by (*simp add*: *countable_intersection_of_inter gdelta_in_alt le_infI2*)

lemma *gdelta_in_Un*:
  ⟦*gdelta_in X S*; *gdelta_in X T*⟧ ⟹ *gdelta_in X* (*S* ∪ *T*)
  by (*simp add*: *countable_intersection_of_union gdelta_in_alt openin_Un*)

lemma *fsigma_in_diff*:
  assumes *fsigma_in X S gdelta_in X T*
  shows *fsigma_in X* (*S* − *T*)
proof −
  have [*simp*]: *S* − (*S* ∩ *T*) = *S* − *T* **for** *S T* :: *'a set*
    by *auto*
  have [*simp*]: *topspace X* − ⋂*𝒯* = (⋃ *T*∈*𝒯*. *topspace X* − *T*) **for** *𝒯*
    by *auto*
  have ⋀*𝒯*. ⟦*countable 𝒯*; *𝒯* ⊆ *Collect* (*openin X*)⟧ ⟹
          (*countable union_of closedin X*) (⋃ ((−) (*topspace X*) ‘ *𝒯*))
   by (*metis Ball_Collect countable_union_of_UN countable_union_of_inc openin_closedin_eq*)
  then have ∀ *S*. *gdelta_in X S* ⟶ *fsigma_in X* (*topspace X* − *S*)
   by (*simp add*: *fsigma_in_def gdelta_in_def all_relative_to all_intersection_of del*: *UN_simps*)
  then show *?thesis*
   by (*metis Diff_Int2 Diff_Int_distrib2 assms fsigma_in_Int fsigma_in_subset inf.absorb_iff2*)
qed

lemma *gdelta_in_diff*:
  assumes *gdelta_in X S fsigma_in X T*
  shows *gdelta_in X* (*S* − *T*)
proof −
  have [*simp*]: *topspace X* − ⋃*𝒯* = *topspace X* ∩ (⋂ *T*∈*𝒯*. *topspace X* − *T*) **for** *𝒯*
    by *auto*
  have ⋀*𝒯*. ⟦*countable 𝒯*; *𝒯* ⊆ *Collect* (*closedin X*)⟧
        ⟹ (*countable intersection_of openin X relative_to topspace X*)
           (*topspace X* ∩ ⋂ ((−) (*topspace X*) ‘ *𝒯*))
    by (*metis Ball_Collect closedin_def countable_intersection_of_INT countable_intersection_of_inc relative_to_inc*)
  then have ∀ *S*. *fsigma_in X S* ⟶ *gdelta_in X* (*topspace X* − *S*)
    by (*simp add*: *fsigma_in_def gdelta_in_def all_union_of del*: *INT_simps*)
  then show *?thesis*
     by (*metis Diff_Int2 Diff_Int_distrib2 assms gdelta_in_Int gdelta_in_alt inf.orderE inf_commute*)
qed

lemma *gdelta_in_fsigma_in*:
  gdelta_in X S ⟷ *S* ⊆ *topspace X* ∧ *fsigma_in X* (*topspace X* − *S*)

by (*metis double_diff fsigma_in_diff fsigma_in_topspace gdelta_in_alt gdelta_in_diff gdelta_in_topspace*)

lemma *fsigma_in_gdelta_in*:
fsigma_in X S $\longleftrightarrow$ *S* $\subseteq$ *topspace X* $\wedge$ *gdelta_in X* (*topspace X* $-$ *S*)
by (*metis Diff_Diff_Int fsigma_in_subset gdelta_in_fsigma_in inf.absorb_iff2*)

lemma *gdelta_in_descending*:
gdelta_in X S $\longleftrightarrow$ ($\exists$ *C*. ($\forall$ *n*. *openin X* (*C n*)) $\wedge$ ($\forall$ *n*. *C*(*Suc n*) $\subseteq$ *C n*) $\wedge$ $\bigcap$(*range C*) = *S*) (**is** *?lhs=?rhs*)
proof
assume *?lhs*
then obtain *C* **where** *C*: *S* $\subseteq$ *topspace X* $\bigwedge$*n*. *closedin X* (*C n*)
$\bigwedge$*n*. *C n* $\subseteq$ *C*(*Suc n*) $\bigcup$(*range C*) = *topspace X* $-$ *S*
by (*meson fsigma_in_ascending gdelta_in_fsigma_in*)
define *D* **where** *D* $\equiv$ λ*n*. *topspace X* $-$ *C n*
have $\bigwedge$*n*. *openin X* (*D n*) $\wedge$ *D* (*Suc n*) $\subseteq$ *D n*
by (*simp add*: *Diff_mono C D_def openin_diff*)
moreover have $\bigcap$(*range D*) = *S*
by (*simp add*: *Diff_Diff_Int Int_absorb1 C D_def*)
ultimately show *?rhs*
by *metis*
next
assume *?rhs*
then obtain *C* **where** *S* $\subseteq$ *topspace X*
and *C*: $\bigwedge$*n*. *openin X* (*C n*) $\bigwedge$*n*. *C*(*Suc n*) $\subseteq$ *C n* $\bigcap$(*range C*) = *S*
using *openin_subset* **by** *fastforce*
define *D* **where** *D* $\equiv$ λ*n*. *topspace X* $-$ *C n*
have $\bigwedge$*n*. *closedin X* (*D n*) $\wedge$ *D n* $\subseteq$ *D*(*Suc n*)
by (*simp add*: *Diff_mono C closedin_diff D_def*)
moreover have $\bigcup$(*range D*) = *topspace X* $-$ *S*
using *C D_def* **by** *blast*
ultimately show *?lhs*
by (*metis* ‹*S* $\subseteq$ *topspace X*› *fsigma_in_ascending gdelta_in_fsigma_in*)
qed

lemma *homeomorphic_map_fsigmaness*:
assumes *f*: *homeomorphic_map X Y f* **and** *U* $\subseteq$ *topspace X*
shows *fsigma_in Y* (*f* ‘ *U*) $\longleftrightarrow$ *fsigma_in X U* (**is** *?lhs=?rhs*)
proof $-$
obtain *g* **where** *g*: *homeomorphic_maps X Y f g* **and** *g*: *homeomorphic_map Y X g*
and *1*: ($\forall$ *x* $\in$ *topspace X*. *g*(*f x*) = *x*) **and** *2*: ($\forall$ *y* $\in$ *topspace Y*. *f*(*g y*) = *y*)
using *assms homeomorphic_map_maps* **by** (*metis homeomorphic_maps_map*)
show *?thesis*
proof
assume *?lhs*
then obtain $\mathcal{V}$ **where** *countable* $\mathcal{V}$ **and** $\mathcal{V}$: $\mathcal{V}$ $\subseteq$ *Collect* (*closedin Y*) $\bigcup\mathcal{V}$ = *f*‘*U*

by (*force simp*: *fsigma_in_def union_of_def*)
define $\mathcal{U}$ **where** $\mathcal{U} \equiv$ *image* (*image g*) $\mathcal{V}$
have *countable* $\mathcal{U}$
using $\mathcal{U}$*_def* ‹*countable* $\mathcal{V}$› **by** *blast*
moreover have $\mathcal{U} \subseteq$ *Collect* (*closedin X*)
using *f g homeomorphic_map_closedness_eq* $\mathcal{V}$ **unfolding** $\mathcal{U}$*_def* **by** *blast*
moreover have $\bigcup\mathcal{U} \subseteq U$
unfolding $\mathcal{U}$*_def* **by** (*smt* (*verit*) *assms 1* $\mathcal{V}$ *image_Union image_iff in_mono subsetI*)
moreover have $U \subseteq \bigcup\mathcal{U}$
unfolding $\mathcal{U}$*_def* **using** *assms* $\mathcal{V}$
by (*smt* (*verit*, *del_insts*) *1 imageI image_Union subset_iff*)
ultimately show *?rhs*
by (*metis fsigma_in_def subset_antisym union_of_def*)
next
assume *?rhs*
then obtain $\mathcal{V}$ **where** *countable* $\mathcal{V}$ **and** $\mathcal{V}$: $\mathcal{V} \subseteq$ *Collect* (*closedin X*) $\bigcup\mathcal{V} = U$
by (*auto simp*: *fsigma_in_def union_of_def*)
define $\mathcal{U}$ **where** $\mathcal{U} \equiv$ *image* (*image f*) $\mathcal{V}$
have *countable* $\mathcal{U}$
using $\mathcal{U}$*_def* ‹*countable* $\mathcal{V}$› **by** *blast*
moreover have $\mathcal{U} \subseteq$ *Collect* (*closedin Y*)
using *f g homeomorphic_map_closedness_eq* $\mathcal{V}$ **unfolding** $\mathcal{U}$*_def* **by** *blast*
moreover have $\bigcup\mathcal{U} = f`U$
unfolding $\mathcal{U}$*_def* **using** $\mathcal{V}$ **by** *blast*
ultimately show *?lhs*
by (*metis fsigma_in_def union_of_def*)
qed
qed

lemma *homeomorphic_map_fsigmaness_eq*:
homeomorphic_map X Y f
$\Longrightarrow$ (*fsigma_in X U* $\longleftrightarrow$ $U \subseteq$ *topspace X* $\wedge$ *fsigma_in Y* (*f ‘ U*))
by (*metis fsigma_in_subset homeomorphic_map_fsigmaness*)

lemma *homeomorphic_map_gdeltaness*:
assumes *homeomorphic_map X Y f* $U \subseteq$ *topspace X*
shows *gdelta_in Y* (*f ‘ U*) $\longleftrightarrow$ *gdelta_in X U*
proof −
have *topspace Y* − *f ‘ U* = *f ‘* (*topspace X* − *U*)
by (*metis* (*no_types*, *lifting*) *Diff_subset assms homeomorphic_eq_everything_map inj_on_image_set_diff*)
then show *?thesis*
using *assms homeomorphic_imp_surjective_map*
by (*fastforce simp*: *gdelta_in_fsigma_in homeomorphic_map_fsigmaness_eq*)
qed

lemma *homeomorphic_map_gdeltaness_eq*:
homeomorphic_map X Y f

$\Longrightarrow$ *gdelta_in X U* $\longleftrightarrow$ *U* $\subseteq$ *topspace X* $\wedge$ *gdelta_in Y* (*f* ' *U*)
by (*meson gdelta_in_subset homeomorphic_map_gdeltaness*)

lemma *fsigma_in_relative_to*:
(*fsigma_in X relative_to S*) = *fsigma_in* (*subtopology X S*)
by (*simp add*: *fsigma_in_def countable_union_of_relative_to closedin_relative_to*)

lemma *fsigma_in_subtopology*:
fsigma_in (*subtopology X U*) *S* $\longleftrightarrow$ ($\exists$ *T*. *fsigma_in X T* $\wedge$ *S* = *T* $\cap$ *U*)
by (*metis fsigma_in_relative_to inf_commute relative_to_def*)

lemma *gdelta_in_relative_to*:
(*gdelta_in X relative_to S*) = *gdelta_in* (*subtopology X S*)
apply (*simp add*: *gdelta_in_def*)
by (*metis countable_intersection_of_relative_to openin_relative_to subtopology_restrict*)

lemma *gdelta_in_subtopology*:
gdelta_in (*subtopology X U*) *S* $\longleftrightarrow$ ($\exists$ *T*. *gdelta_in X T* $\wedge$ *S* = *T* $\cap$ *U*)
by (*metis gdelta_in_relative_to inf_commute relative_to_def*)

lemma *fsigma_in_fsigma_subtopology*:
fsigma_in X S $\Longrightarrow$ (*fsigma_in* (*subtopology X S*) *T* $\longleftrightarrow$ *fsigma_in X T* $\wedge$ *T* $\subseteq$ *S*)
by (*metis fsigma_in_Int fsigma_in_gdelta_in fsigma_in_subtopology inf.orderE topspace_subtopology_subset*)

lemma *gdelta_in_gdelta_subtopology*:
gdelta_in X S $\Longrightarrow$ (*gdelta_in* (*subtopology X S*) *T* $\longleftrightarrow$ *gdelta_in X T* $\wedge$ *T* $\subseteq$ *S*)
by (*metis gdelta_in_Int gdelta_in_subset gdelta_in_subtopology inf.orderE topspace_subtopology_subs*

end

1.13 Disjoint sum of arbitarily many spaces

theory *Sum_Topology*
imports *Abstract_Topology*
begin

definition *sum_topology* :: ('*a* $\Rightarrow$ '*b topology*) $\Rightarrow$ '*a set* $\Rightarrow$ ('*a* $\times$ '*b*) *topology* **where**
sum_topology X I $\equiv$
topology (λ*U*. *U* $\subseteq$ *Sigma I* (*topspace* $\circ$ *X*) $\wedge$ ($\forall$ *i* $\in$ *I*. *openin* (*X i*) {*x*. (*i*,*x*) $\in$ *U*}))

lemma *is_sum_topology*: *istopology* (λ*U*. *U* $\subseteq$ *Sigma I* (*topspace* $\circ$ *X*) $\wedge$ ($\forall$ *i*$\in$*I*. *openin* (*X i*) {*x*. (*i*, *x*) $\in$ *U*}))
proof −

have *1*: $\{x.\ (i,\ x) \in S \cap T\} = \{x.\ (i,\ x) \in S\} \cap \{x::'b.\ (i,\ x) \in T\}$ **for** $S\ T$ **and** $i::'a$
by *auto*
have *2*: $\{x.\ (i,\ x) \in \bigcup\ \mathcal{K}\} = (\bigcup K{\in}\mathcal{K}.\ \{x::'b.\ (i,\ x) \in K\})$ **for** $\mathcal{K}$ **and** $i::'a$
by *auto*
show *?thesis*
unfolding *istopology_def 1 2* **by** *blast*
qed

lemma *openin_sum_topology*:
openin (*sum_topology X I*) *U* $\longleftrightarrow$
$U \subseteq$ *Sigma I* (*topspace* $\circ$ *X*) $\wedge$ ($\forall\, i \in I.$ *openin* (*X i*) $\{x.\ (i,x) \in U\}$)
by (*auto simp*: *sum_topology_def is_sum_topology*)

lemma *openin_disjoint_union*:
openin (*sum_topology X I*) (*Sigma I U*) $\longleftrightarrow$ ($\forall\, i \in I.$ *openin* (*X i*) (*U i*))
using *openin_subset* **by** (*force simp*: *openin_sum_topology*)

lemma *topspace_sum_topology* [*simp*]:
topspace(*sum_topology X I*) = *Sigma I* (*topspace* $\circ$ *X*)
by (*metis comp_apply openin_disjoint_union openin_subset openin_sum_topology openin_topspace subset_antisym*)

lemma *openin_sum_topology_alt*:
openin (*sum_topology X I*) *U* $\longleftrightarrow$ ($\exists\, T.$ *U* = *Sigma I T* $\wedge$ ($\forall\, i \in I.$ *openin* (*X i*) (*T i*)))
by (*bestsimp simp*: *openin_sum_topology dest*: *openin_subset*)

lemma *forall_openin_sum_topology*:
($\forall\, U.$ *openin* (*sum_topology X I*) *U* $\longrightarrow$ *P U*) $\longleftrightarrow$ ($\forall\, T.$ ($\forall\, i \in I.$ *openin* (*X i*) (*T i*)) $\longrightarrow$ *P*(*Sigma I T*))
by (*auto simp*: *openin_sum_topology_alt*)

lemma *exists_openin_sum_topology*:
($\exists\, U.$ *openin* (*sum_topology X I*) *U* $\wedge$ *P U*) $\longleftrightarrow$
($\exists\, T.$ ($\forall\, i \in I.$ *openin* (*X i*) (*T i*)) $\wedge$ *P*(*Sigma I T*))
by (*auto simp*: *openin_sum_topology_alt*)

lemma *closedin_sum_topology*:
closedin (*sum_topology X I*) *U* $\longleftrightarrow$ *U* $\subseteq$ *Sigma I* (*topspace* $\circ$ *X*) $\wedge$ ($\forall\, i \in I.$ *closedin* (*X i*) $\{x.\ (i,x) \in U\}$)
(**is** *?lhs* = *?rhs*)
proof
assume *L*: *?lhs*
then have *U*: *U* $\subseteq$ *Sigma I* (*topspace* $\circ$ *X*)
using *closedin_subset* **by** *fastforce*
then have $\forall\, i{\in}I.\ \{x.\ (i,\ x) \in U\} \subseteq$ *topspace* (*X i*)
by *fastforce*
moreover have *openin* (*X i*) (*topspace* (*X i*) $-$ $\{x.\ (i,\ x) \in U\}$) **if** $i{\in}I$ **for** *i*

apply (*subst openin_subopen*)
using *L that* **unfolding** *closedin_def openin_sum_topology*
by (*bestsimp simp*: *o_def subset_iff*)
ultimately show *?rhs*
by (*simp add*: *U closedin_def*)
next
assume *R*: *?rhs*
then have *openin* (*X i*) {*x*. (*i*, *x*) ∈ *topspace* (*sum_topology X I*) − *U*} **if** *i*∈*I* **for** *i*
apply (*subst openin_subopen*)
using *that* **unfolding** *closedin_def openin_sum_topology*
by (*bestsimp simp*: *o_def subset_iff*)
then show *?lhs*
by (*simp add*: *R closedin_def openin_sum_topology*)
qed

lemma *closedin_disjoint_union*:
closedin (*sum_topology X I*) (*Sigma I U*) ⟷ (∀ *i* ∈ *I*. *closedin* (*X i*) (*U i*))
using *closedin_subset* **by** (*force simp*: *closedin_sum_topology*)

lemma *closedin_sum_topology_alt*:
closedin (*sum_topology X I*) *U* ⟷ (∃ *T*. *U* = *Sigma I T* ∧ (∀ *i* ∈ *I*. *closedin* (*X i*) (*T i*)))
using *closedin_subset* **unfolding** *closedin_sum_topology* **by** *auto blast*

lemma *forall_closedin_sum_topology*:
(∀ *U*. *closedin* (*sum_topology X I*) *U* ⟶ *P U*) ⟷
(∀ *t*. (∀ *i* ∈ *I*. *closedin* (*X i*) (*t i*)) ⟶ *P*(*Sigma I t*))
by (*metis closedin_sum_topology_alt*)

lemma *exists_closedin_sum_topology*:
(∃ *U*. *closedin* (*sum_topology X I*) *U* ∧ *P U*) ⟷
(∃ *T*. (∀ *i* ∈ *I*. *closedin* (*X i*) (*T i*)) ∧ *P*(*Sigma I T*))
by (*simp add*: *closedin_sum_topology_alt*) *blast*

lemma *open_map_component_injection*:
i ∈ *I* ⟹ *open_map* (*X i*) (*sum_topology X I*) (λ*x*. (*i*,*x*))
unfolding *open_map_def openin_sum_topology*
using *openin_subset openin_subopen* **by** (*force simp*: *image_iff*)

lemma *closed_map_component_injection*:
assumes *i* ∈ *I*
shows *closed_map*(*X i*) (*sum_topology X I*) (λ*x*. (*i*,*x*))
proof −
have *closedin* (*X j*) {*x*. *j* = *i* ∧ *x* ∈ *U*}
if ⋀*U S*. *closedin U S* ⟹ *S* ⊆ *topspace U* **and** *closedin* (*X i*) *U* **and** *j* ∈ *I* **for** *U j*
using *that* **by** (*cases j*=*i*) *auto*
then show *?thesis*

using *closedin_subset assms* **by** (*force simp*: *closed_map_def closedin_sum_topology image_iff*)
qed

lemma *continuous_map_component_injection*:
i ∈ *I* ⟹ *continuous_map*(*X i*) (*sum_topology X I*) (λ*x*. (*i*,*x*))
apply (*clarsimp simp*: *continuous_map_def openin_sum_topology*)
by (*smt* (*verit*, *best*) *Collect_cong mem_Collect_eq openin_subset subsetD*)

lemma *subtopology_sum_topology*:
subtopology (*sum_topology X I*) (*Sigma I S*) =
sum_topology (λ*i*. *subtopology* (*X i*) (*S i*)) *I*
unfolding *topology_eq*
proof (*intro strip iffI*)
fix *T*
assume ∗: *openin* (*subtopology* (*sum_topology X I*) (*Sigma I S*)) *T*
then obtain *U* **where** *U*: ∀ *i*∈*I*. *openin* (*X i*) (*U i*) ∧ *T* = *Sigma I U* ∩ *Sigma I S*
by (*auto simp*: *openin_subtopology openin_sum_topology_alt*)
have *T* = (*SIGMA i*:*I*. *U i* ∩ *S i*)
proof
show *T* ⊆ (*SIGMA i*:*I*. *U i* ∩ *S i*)
by (*metis* ∗ *SigmaE Sigma_Int_distrib2 U openin_imp_subset subset_iff*)
show (*SIGMA i*:*I*. *U i* ∩ *S i*) ⊆ *T*
using *U* **by** *fastforce*
qed
then show *openin* (*sum_topology* (λ*i*. *subtopology* (*X i*) (*S i*)) *I*) *T*
by (*simp add*: *U openin_disjoint_union openin_subtopology_Int*)
next
fix *T*
assume ∗: *openin* (*sum_topology* (λ*i*. *subtopology* (*X i*) (*S i*)) *I*) *T*
then obtain *U* **where** ∀ *i*∈*I*. ∃ *T*. *openin* (*X i*) *T* ∧ *U i* = *T* ∩ *S i* **and** *Teq*: *T* = *Sigma I U*
by (*auto simp*: *openin_subtopology openin_sum_topology_alt*)
then obtain *B* **where** ⋀*i*. *i*∈*I* ⟹ *openin* (*X i*) (*B i*) ∧ *U i* = *B i* ∩ *S i*
by *metis*
then show *openin* (*subtopology* (*sum_topology X I*) (*Sigma I S*)) *T*
by (*auto simp*: *openin_subtopology openin_sum_topology_alt Teq*)
qed

lemma *embedding_map_component_injection*:
i ∈ *I* ⟹ *embedding_map* (*X i*) (*sum_topology X I*) (λ*x*. (*i*,*x*))
by (*metis injective_open_imp_embedding_map continuous_map_component_injection open_map_component_injection inj_onI prod.inject*)

lemma *topological_property_of_sum_component*:
assumes *major*: *P* (*sum_topology X I*)
and *minor*: ⋀*X S*. ⟦*P X*; *closedin X S*; *openin X S*⟧ ⟹ *P*(*subtopology X S*)
and *PQ*: ⋀*X Y*. *X homeomorphic_space Y* ⟹ (*P X* ⟷ *Q Y*)

shows $(\forall\, i \in I.\ Q(X\ i))$
proof −
have $Q(X\ i)$ **if** $i \in I$ **for** i
proof −
have $P(subtopology\ (sum\_topology\ X\ I)\ (Pair\ i\ `\ topspace\ (X\ i)))$
by (*meson closed_map_component_injection closed_map_def closedin_topspace major minor open_map_component_injection open_map_def openin_topspace that*)
then show *?thesis*
by (*metis PQ embedding_map_component_injection embedding_map_imp_homeomorphic_space homeomorphic_space_sym that*)
qed
then show *?thesis* **by** *metis*
qed

end

Chapter 2

Topology

theory *Elementary_Topology*
imports
HOL−Library.Set_Idioms
HOL−Library.Disjoint_Sets
Product_Vector
begin

2.1 Elementary Topology

Affine transformations of intervals

lemma *real_affinity_le*: $0 < m \Longrightarrow m * x + c \leq y \longleftrightarrow x \leq inverse\ m * y + -(c\ /\ m)$
for $m :: 'a$::*linordered_field*
by (*simp add*: *field_simps*)

lemma *real_le_affinity*: $0 < m \Longrightarrow y \leq m * x + c \longleftrightarrow inverse\ m * y + -(c\ /\ m) \leq x$
for $m :: 'a$::*linordered_field*
by (*simp add*: *field_simps*)

lemma *real_affinity_lt*: $0 < m \Longrightarrow m * x + c < y \longleftrightarrow x < inverse\ m * y + -(c\ /\ m)$
for $m :: 'a$::*linordered_field*
by (*simp add*: *field_simps*)

lemma *real_lt_affinity*: $0 < m \Longrightarrow y < m * x + c \longleftrightarrow inverse\ m * y + -(c\ /\ m) < x$
for $m :: 'a$::*linordered_field*
by (*simp add*: *field_simps*)

lemma *real_affinity_eq*: $m \neq 0 \Longrightarrow m * x + c = y \longleftrightarrow x = inverse\ m * y + -(c\ /\ m)$
for $m :: 'a$::*linordered_field*

by (*simp add*: *field_simps*)

lemma *real_eq_affinity*: $m \neq 0 \Longrightarrow y = m * x + c \longleftrightarrow inverse\ m * y + - (c / m) = x$
for $m :: 'a$::*linordered_field*
by (*simp add*: *field_simps*)

2.1.1 Topological Basis

context *topological_space*
begin

definition *topological_basis* $B \longleftrightarrow$
$(\forall b{\in}B.\ open\ b) \wedge (\forall x.\ open\ x \longrightarrow (\exists B'.\ B' \subseteq B \wedge \bigcup B' = x))$

lemma *topological_basis*:
topological_basis $B \longleftrightarrow (\forall x.\ open\ x \longleftrightarrow (\exists B'.\ B' \subseteq B \wedge \bigcup B' = x))$
(**is** *?lhs* = *?rhs*)
proof
show *?lhs* $\Longrightarrow$ *?rhs*
by (*meson local.open_Union subsetD topological_basis_def*)
show *?rhs* $\Longrightarrow$ *?lhs*
unfolding *topological_basis_def*
by (*metis cSup_singleton empty_subsetI insert_subset*)
qed

lemma *topological_basis_iff*:
assumes $\bigwedge B'.\ B' \in B \Longrightarrow open\ B'$
shows *topological_basis* $B \longleftrightarrow (\forall O'.\ open\ O' \longrightarrow (\forall x{\in}O'.\ \exists B'{\in}B.\ x \in B' \wedge B' \subseteq O'))$
(**is** _ $\longleftrightarrow$ *?rhs*)
proof *safe*
fix O' **and** $x::'a$
assume H: *topological_basis* B *open* O' $x \in O'$
then have $(\exists B'{\subseteq}B.\ \bigcup B' = O')$ **by** (*simp add*: *topological_basis_def*)
then obtain B' **where** $B' \subseteq B$ $O' = \bigcup B'$ **by** *auto*
then show $\exists B'{\in}B.\ x \in B' \wedge B' \subseteq O'$ **using** H **by** *auto*
next
assume H: *?rhs*
show *topological_basis* B
using *assms* **unfolding** *topological_basis_def*
proof *safe*
fix $O' :: 'a\ set$
assume *open* O'
with H **obtain** f **where** $\forall x{\in}O'.\ f\ x \in B \wedge x \in f\ x \wedge f\ x \subseteq O'$
by (*force intro*: *bchoice simp*: *Bex_def*)
then show $\exists B'{\subseteq}B.\ \bigcup B' = O'$
by (*auto intro*: *exI*[**where** $x{=}\{f\ x\ |x.\ x \in O'\}$])
qed

qed

lemma *topological_basisI*:
assumes $\bigwedge B'.\ B' \in B \Longrightarrow \mathit{open}\ B'$
and $\bigwedge O'\ x.\ \mathit{open}\ O' \Longrightarrow x \in O' \Longrightarrow \exists\, B' \in B.\ x \in B' \wedge B' \subseteq O'$
shows *topological_basis B*
by (*simp add*: *assms topological_basis_iff*)

lemma *topological_basisE*:
fixes O'
assumes *topological_basis B*
and *open* O'
and $x \in O'$
obtains B' **where** $B' \in B\ x \in B'\ B' \subseteq O'$
by (*metis assms topological_basis_def topological_basis_iff*)

lemma *topological_basis_open*:
assumes *topological_basis B* **and** $X \in B$
shows *open X*
using *assms* **by** (*simp add*: *topological_basis_def*)

lemma *topological_basis_imp_subbasis*:
assumes *B*: *topological_basis B*
shows *open* = *generate_topology B*
proof (*intro ext iffI*)
fix $S :: 'a\ \mathit{set}$
assume *open S*
with *B* **obtain** B' **where** $B' \subseteq B\ S = \bigcup B'$
unfolding *topological_basis_def* **by** *blast*
then show *generate_topology B S*
by (*auto intro*: *generate_topology.intros dest*: *topological_basis_open*)
next
fix $S :: 'a\ \mathit{set}$
assume *generate_topology B S*
then show *open S*
by *induct* (*auto dest*: *topological_basis_open*[*OF B*])
qed

lemma *basis_dense*:
fixes $B :: 'a\ \mathit{set}\ \mathit{set}$
and $f :: 'a\ \mathit{set} \Rightarrow 'a$
assumes *topological_basis B* **and** $\bigwedge B'.\ B' \neq \{\} \Longrightarrow f\ B' \in B'$
shows $\forall X.\ \mathit{open}\ X \longrightarrow X \neq \{\} \longrightarrow (\exists\, B' \in B.\ f\ B' \in X)$
by (*metis assms equals0D in_mono topological_basisE*)

end

lemma *topological_basis_prod*:
assumes *A*: *topological_basis A*

and B: *topological_basis B*
shows *topological_basis* $((\lambda(a,\ b).\ a \times b)\ `\ (A \times B))$
proof −
have $\exists\, X{\subseteq}A \times B.\ (\bigcup(a,b){\in}X.\ a \times b) = S$ **if** *open S* **for** S
proof −
have $(x,\ y) \in (\bigcup(a,\ b){\in}\{X \in A \times B.\ fst\ X \times snd\ X \subseteq S\}.\ a \times b)$
if *xy*: $(x,\ y) \in S$ **for** $x\ y$
proof −
obtain $a\ b$ **where** a: *open ax* $\in a$ **and** b: *open b y* $\in b$ **and** $a \times b \subseteq S$
by (*metis open_prod_elim*[*OF* ‹*open S*›] *xy mem_Sigma_iff*)
moreover obtain *A0* **where** *A0* $\in A$ $x \in A0$ $A0 \subseteq a$
using A a b *topological_basisE* **by** *blast*
moreover
from B b **obtain** *B0* **where** *B0* $\in B$ $y \in B0$ $B0 \subseteq b$
by (*rule topological_basisE*)
ultimately show *?thesis*
by (*intro UN_I*[*of* (*A0*, *B0*)]) *auto*
qed
then have $(\bigcup(a,\ b){\in}\{x \in A \times B.\ fst\ x \times snd\ x \subseteq S\}.\ a \times b) = S$
by *force*
then show *?thesis*
using *subset_eq* **by** *force*
qed
with A B **show** *?thesis*
unfolding *topological_basis_def*
by (*smt* (*verit*) *SigmaE imageE image_mono open_Times case_prod_conv*)
qed

2.1.2 Countable Basis

locale *countable_basis* = *topological_space p* **for** $p{::}'a\ set \Rightarrow bool$ +
fixes $B :: 'a\ set\ set$
assumes *is_basis*: *topological_basis B*
and *countable_basis*: *countable B*
begin

lemma *open_countable_basis_ex*:
assumes $p\ X$
shows $\exists\, B' \subseteq B.\ X = \bigcup B'$
using *assms countable_basis is_basis*
unfolding *topological_basis_def* **by** *blast*

lemma *open_countable_basisE*:
assumes $p\ X$
obtains B' **where** $B' \subseteq B\ X = \bigcup B'$
using *assms open_countable_basis_ex* **by** *auto*

lemma *countable_dense_exists*:
$\exists\, D{::}'a\ set.\ countable\ D \wedge (\forall\, X.\ p\ X \longrightarrow X \neq \{\} \longrightarrow (\exists\, d \in D.\ d \in X))$

proof −
let *?f* = (λ*B'*. *SOME x*. *x* ∈ *B'*)
have *countable* (*?f* ‘ *B*) **using** *countable_basis* **by** *simp*
with *basis_dense*[*OF is_basis*, *of ?f*] **show** *?thesis*
by (*intro exI*[**where** *x*=*?f* ‘ *B*]) (*metis* (*mono_tags*) *all_not_in_conv imageI someI*)
qed

lemma *countable_dense_setE*:
obtains *D* :: *'a set*
where *countable D* ⋀*X*. *p X* ⟹ *X* ≠ {} ⟹ ∃ *d* ∈ *D*. *d* ∈ *X*
using *countable_dense_exists* **by** *blast*

end

lemma *countable_basis_openI*: *countable_basis open B*
if *countable B topological_basis B*
using *that*
by *unfold_locales*
(*simp_all add*: *topological_basis topological_space.topological_basis topological_space_axioms*)

lemma (**in** *first_countable_topology*) *first_countable_basisE*:
fixes *x* :: *'a*
obtains 𝒜 **where** *countable* 𝒜 ⋀*A*. *A* ∈ 𝒜 ⟹ *x* ∈ *A* ⋀*A*. *A* ∈ 𝒜 ⟹ *open A*
⋀*S*. *open S* ⟹ *x* ∈ *S* ⟹ (∃ *A*∈𝒜. *A* ⊆ *S*)
proof −
obtain 𝒜 **where** 𝒜: (∀ *i*::*nat*. *x* ∈ 𝒜 *i* ∧ *open* (𝒜 *i*)) (∀ *S*. *open S* ∧ *x* ∈ *S* ⟶ (∃ *i*. 𝒜 *i* ⊆ *S*))
using *first_countable_basis*[*of x*] **by** *metis*
moreover have *countable* (*range* 𝒜)
by *simp*
ultimately show *thesis*
by (*smt* (*verit*, *best*) *imageE rangeI that*)
qed

lemma (**in** *first_countable_topology*) *first_countable_basis_Int_stableE*:
obtains 𝒜 **where** *countable* 𝒜 ⋀*A*. *A* ∈ 𝒜 ⟹ *x* ∈ *A* ⋀*A*. *A* ∈ 𝒜 ⟹ *open A*
⋀*S*. *open S* ⟹ *x* ∈ *S* ⟹ (∃ *A*∈𝒜. *A* ⊆ *S*)
⋀*A B*. *A* ∈ 𝒜 ⟹ *B* ∈ 𝒜 ⟹ *A* ∩ *B* ∈ 𝒜
proof *atomize_elim*
obtain ℬ **where** ℬ:
countable ℬ
⋀*B*. *B* ∈ ℬ ⟹ *x* ∈ *B*
⋀*B*. *B* ∈ ℬ ⟹ *open B*
⋀*S*. *open S* ⟹ *x* ∈ *S* ⟹ ∃ *B*∈ℬ. *B* ⊆ *S*
by (*rule first_countable_basisE*) *blast*
define 𝒜 **where** [*abs_def*]:
𝒜 = (λ*N*. ⋂((λ*n*. *from_nat_into* ℬ *n*) ‘ *N*)) ‘ (*Collect finite*::*nat set set*)

```
  then show ∃𝒜. countable 𝒜 ∧ (∀A. A ∈ 𝒜 ⟶ x ∈ A) ∧ (∀A. A ∈ 𝒜 ⟶ open
A) ∧
       (∀S. open S ⟶ x ∈ S ⟶ (∃A∈𝒜. A ⊆ S)) ∧ (∀A B. A ∈ 𝒜 ⟶ B ∈ 𝒜
⟶ A ∩ B ∈ 𝒜)
  proof (safe intro!: exI[where x=𝒜])
    show countable 𝒜
      unfolding 𝒜_def by (intro countable_image countable_Collect_finite)
    fix A
    assume A ∈ 𝒜
    then show x ∈ A open A
      using ℬ(4)[OF open_UNIV] by (auto simp: 𝒜_def intro: ℬ from_nat_into)
  next
    let ?int = λN. ⋂(from_nat_into ℬ ` N)
    fix A B
    assume A ∈ 𝒜 B ∈ 𝒜
    then obtain N M where A = ?int N B = ?int M finite (N ∪ M)
      by (auto simp: 𝒜_def)
    then show A ∩ B ∈ 𝒜
      by (auto simp: 𝒜_def intro!: image_eqI[where x=N ∪ M])
  next
    fix S
    assume open S x ∈ S
    then obtain a where a: a∈ℬ a ⊆ S using ℬ by blast
    moreover have a∈𝒜
      unfolding 𝒜_def
    proof (rule image_eqI)
      show a = ⋂ (from_nat_into ℬ ` {to_nat_on ℬ a})
        by (simp add: ℬ a)
    qed auto
    ultimately show ∃a∈𝒜. a ⊆ S
      by blast
  qed
qed

lemma (in topological_space) first_countableI:
  assumes countable 𝒜
    and 1: ⋀A. A ∈ 𝒜 ⟹ x ∈ A ⋀A. A ∈ 𝒜 ⟹ open A
    and 2: ⋀S. open S ⟹ x ∈ S ⟹ ∃A∈𝒜. A ⊆ S
  shows ∃𝒜::nat ⇒ 'a set. (∀i. x ∈ 𝒜 i ∧ open (𝒜 i)) ∧ (∀S. open S ∧ x ∈ S
⟶ (∃i. 𝒜 i ⊆ S))
proof (safe intro!: exI[of _ from_nat_into 𝒜])
  fix i
  have 𝒜 ≠ {} using 2[of UNIV] by auto
  show x ∈ from_nat_into 𝒜 i open (from_nat_into 𝒜 i)
    using range_from_nat_into_subset[OF ‹𝒜 ≠ {}›] 1 by auto
next
  fix S
  assume open S x∈S
  then show ∃i. from_nat_into 𝒜 i ⊆ S
```

by (*metis 2* ‹*countable* $\mathcal{A}$› *from_nat_into_surj*)
qed

instance *prod* :: (*first_countable_topology*, *first_countable_topology*) *first_countable_topology*
proof
fix x :: $'a \times 'b$
obtain $\mathcal{A}$ **where** $\mathcal{A}$:
countable $\mathcal{A}$
$\bigwedge a.\ a \in \mathcal{A} \Longrightarrow fst\ x \in a$
$\bigwedge a.\ a \in \mathcal{A} \Longrightarrow open\ a$
$\bigwedge S.\ open\ S \Longrightarrow fst\ x \in S \Longrightarrow \exists\, a{\in}\mathcal{A}.\ a \subseteq S$
by (*rule first_countable_basisE*[*of fst x*]) *blast*
obtain B **where** B:
countable B
$\bigwedge a.\ a \in B \Longrightarrow snd\ x \in a$
$\bigwedge a.\ a \in B \Longrightarrow open\ a$
$\bigwedge S.\ open\ S \Longrightarrow snd\ x \in S \Longrightarrow \exists\, a{\in}B.\ a \subseteq S$
by (*rule first_countable_basisE*[*of snd x*]) *blast*
show $\exists\, \mathcal{A}$::*nat* $\Rightarrow$ ($'a \times 'b$) *set*.
$(\forall\, i.\ x \in \mathcal{A}\ i \wedge open\ (\mathcal{A}\ i)) \wedge (\forall\, S.\ open\ S \wedge x \in S \longrightarrow (\exists\, i.\ \mathcal{A}\ i \subseteq S))$
proof (*rule first_countableI*[*of* $(\lambda(a,\ b).\ a \times b)$ ‘ $(\mathcal{A} \times B)$], *safe*)
fix $a\ b$
assume x: $a \in \mathcal{A}\ b \in B$
show $x \in a \times b$
by (*simp add*: $\mathcal{A}(2)\ B(2)$ *mem_Times_iff x*)
show *open* $(a \times b)$
by (*simp add*: $\mathcal{A}(3)\ B(3)$ *open_Times x*)
next
fix S
assume *open* $S\ x \in S$
then obtain $a'\ b'$ **where** $a'b'$: *open* a' *open* $b'\ x \in a' \times b'\ a' \times b' \subseteq S$
by (*rule open_prod_elim*)
moreover
obtain $a\ b$ **where** $a \in \mathcal{A}\ a \subseteq a'\ b \in B\ b \subseteq b'$
by (*meson* $B(4)$ $\mathcal{A}(4)$ $a'b'$ *mem_Times_iff*)
ultimately
show $\exists\, a{\in}(\lambda(a,\ b).\ a \times b)$ ‘ $(\mathcal{A} \times B).\ a \subseteq S$
by (*auto intro*!: *bexI*[*of* _ $a \times b$] *bexI*[*of* _ a] *bexI*[*of* _ b])
qed (*simp add*: $\mathcal{A}\ B$)
qed

class *second_countable_topology* = *topological_space* +
assumes *ex_countable_subbasis*:
$\exists\, B$::$'a$ *set set*. *countable* $B \wedge$ *open* = *generate_topology* B
begin

lemma *ex_countable_basis*: $\exists\, B$::$'a$ *set set*. *countable* $B \wedge$ *topological_basis* B
proof −
from *ex_countable_subbasis* **obtain** B **where** B: *countable* B *open* = *gener-*

```
ate_topology B
    by blast
  let ?B = Inter ` {b. finite b ∧ b ⊆ B }

  show ?thesis
  proof (intro exI conjI)
    show countable ?B
      by (intro countable_image countable_Collect_finite_subset B)
    {
      fix S
      assume open S
      then have ∃B'⊆{b. finite b ∧ b ⊆ B}. (⋃b∈B'. ⋂ b) = S
        unfolding B
      proof induct
        case UNIV
        show ?case by (intro exI[of _ {{}}]) simp
      next
        case (Int a b)
       then obtain x y where x: a = ⋃(Inter ` x) ⋀i. i ∈ x ⟹ finite i ∧ i ⊆ B
          and y: b = ⋃(Inter ` y) ⋀i. i ∈ y ⟹ finite i ∧ i ⊆ B
          by blast
        show ?case
          unfolding x y Int_UN_distrib2
          by (intro exI[of _ {i ∪ j| i j.  i ∈ x ∧ j ∈ y}]) (auto dest: x(2) y(2))
      next
        case (UN K)
       then have ∀k∈K. ∃B'⊆{b. finite b ∧ b ⊆ B}. ⋃ (Inter ` B') = k by auto
       then obtain k where
            ∀ka∈K. k ka ⊆ {b. finite b ∧ b ⊆ B} ∧ ⋃(Inter ` (k ka)) = ka
          unfolding bchoice_iff ..
       then show ∃B'⊆{b. finite b ∧ b ⊆ B}. ⋃ (Inter ` B') = ⋃K
          by (intro exI[of _ ⋃(k ` K)]) auto
      next
        case (Basis S)
        then show ?case
          by (intro exI[of _ {{S}}]) auto
      qed
      then have (∃B'⊆Inter ` {b. finite b ∧ b ⊆ B}. ⋃B' = S)
        unfolding subset_image_iff by blast }
    then show topological_basis ?B
      unfolding topological_basis_def
      by (safe intro!: open_Inter)
         (simp_all add: B generate_topology.Basis subset_eq)
  qed
qed


end
```

lemma *univ_second_countable*:
obtains $\mathcal{B}$ :: *'a*::*second_countable_topology set set*
where *countable* $\mathcal{B}$ $\bigwedge C.\ C \in \mathcal{B} \Longrightarrow$ *open* C
$\bigwedge S.$ *open* $S \Longrightarrow \exists U.\ U \subseteq \mathcal{B} \wedge S = \bigcup U$
by (*metis ex_countable_basis topological_basis_def*)

proposition *Lindelof*:
fixes $\mathcal{F}$ :: *'a*::*second_countable_topology set set*
assumes $\mathcal{F}$: $\bigwedge S.\ S \in \mathcal{F} \Longrightarrow$ *open* S
obtains $\mathcal{F}'$ **where** $\mathcal{F}' \subseteq \mathcal{F}$ *countable* $\mathcal{F}'$ $\bigcup \mathcal{F}' = \bigcup \mathcal{F}$
proof −
obtain $\mathcal{B}$ :: *'a set set*
where *countable* $\mathcal{B}$ $\bigwedge C.\ C \in \mathcal{B} \Longrightarrow$ *open* C
and $\mathcal{B}$: $\bigwedge S.$ *open* $S \Longrightarrow \exists U.\ U \subseteq \mathcal{B} \wedge S = \bigcup U$
using *univ_second_countable* **by** *blast*
define $\mathcal{D}$ **where** $\mathcal{D} \equiv \{S.\ S \in \mathcal{B} \wedge (\exists U.\ U \in \mathcal{F} \wedge S \subseteq U)\}$
have *countable* $\mathcal{D}$
by (*simp add*: $\mathcal{D}$_*def* ‹*countable* $\mathcal{B}$›)
have $\bigwedge S.\ \exists U.\ S \in \mathcal{D} \longrightarrow U \in \mathcal{F} \wedge S \subseteq U$
by (*simp add*: $\mathcal{D}$_*def*)
then obtain G **where** G: $\bigwedge S.\ S \in \mathcal{D} \longrightarrow G\ S \in \mathcal{F} \wedge S \subseteq G\ S$
by *metis*
have $\bigcup \mathcal{F} \subseteq \bigcup \mathcal{D}$
unfolding $\mathcal{D}$_*def* **by** (*blast dest*: $\mathcal{F}$ $\mathcal{B}$)
moreover have $\bigcup \mathcal{D} \subseteq \bigcup \mathcal{F}$
using $\mathcal{D}$_*def* **by** *blast*
ultimately have *eq1*: $\bigcup \mathcal{F} = \bigcup \mathcal{D}$ **..**
moreover have $\bigcup \mathcal{D} = \bigcup\ (G\ `\ \mathcal{D})$
using G *eq1* **by** *auto*
ultimately show *?thesis*
by (*metis* G ‹*countable* $\mathcal{D}$› *countable_image image_subset_iff that*)
qed

lemma *countable_disjoint_open_subsets*:
fixes $\mathcal{F}$ :: *'a*::*second_countable_topology set set*
assumes $\bigwedge S.\ S \in \mathcal{F} \Longrightarrow$ *open* S **and** *pw*: *pairwise disjnt* $\mathcal{F}$
shows *countable* $\mathcal{F}$
proof −
obtain $\mathcal{F}'$ **where** $\mathcal{F}' \subseteq \mathcal{F}$ *countable* $\mathcal{F}'$ $\bigcup \mathcal{F}' = \bigcup \mathcal{F}$
by (*meson assms Lindelof*)
with *pw* **have** $\mathcal{F} \subseteq$ *insert* {} $\mathcal{F}'$
by (*fastforce simp add*: *pairwise_def disjnt_iff*)
then show *?thesis*
by (*simp add*: ‹*countable* $\mathcal{F}'$› *countable_subset*)
qed

sublocale *second_countable_topology* <
countable_basis open SOME B. *countable B* ∧ *topological_basis B*
using *someI_ex*[*OF ex_countable_basis*]

by *unfold_locales safe*

instance *prod* :: (*second_countable_topology*, *second_countable_topology*) *second_countable_topology*
proof
obtain *A* :: *'a set set* **where** *countable A topological_basis A*
using *ex_countable_basis* **by** *auto*
moreover
obtain *B* :: *'b set set* **where** *countable B topological_basis B*
using *ex_countable_basis* **by** *auto*
ultimately show ∃ *B*::(*'a* × *'b*) *set set*. *countable B* ∧ *open* = *generate_topology B*
by (*auto intro*!: *exI*[*of* _ (λ(*a*, *b*). *a* × *b*) ‘ (*A* × *B*)] *topological_basis_prod topological_basis_imp_subbasis*)
qed

instance *second_countable_topology* ⊆ *first_countable_topology*
proof
fix *x* :: *'a*
define *B* :: *'a set set* **where** *B* = (*SOME B*. *countable B* ∧ *topological_basis B*)
then have *B*: *countable B topological_basis B*
using *countable_basis is_basis*
by (*auto simp*: *countable_basis is_basis*)
then show ∃ *A*::*nat* ⇒ *'a set*.
(∀ *i*. *x* ∈ *A i* ∧ *open* (*A i*)) ∧ (∀ *S*. *open S* ∧ *x* ∈ *S* ⟶ (∃ *i*. *A i* ⊆ *S*))
by (*intro first_countableI*[*of* {*b*∈*B*. *x* ∈ *b*}])
(*fastforce simp*: *topological_space_class.topological_basis_def*)+
qed

instance *nat* :: *second_countable_topology*
proof
show ∃ *B*::*nat set set*. *countable B* ∧ *open* = *generate_topology B*
by (*intro exI*[*of* _ *range lessThan* ∪ *range greaterThan*]) (*auto simp*: *open_nat_def*)
qed

lemma *countable_separating_set_linorder1*:
shows ∃ *B*::(*'a*::{*linorder_topology*, *second_countable_topology*} *set*). *countable B* ∧ (∀ *x y*. *x* < *y* ⟶ (∃ *b* ∈ *B*. *x* < *b* ∧ *b* ≤ *y*))
proof −
obtain *A*::*'a set set* **where** *countable A topological_basis A* **using** *ex_countable_basis* **by** *auto*
define *B1* **where** *B1* = {(*LEAST x*. *x* ∈ *U*)| *U*. *U* ∈ *A*}
then have *countable B1* **using** ‹*countable A*› **by** (*simp add*: *Setcompr_eq_image*)
define *B2* **where** *B2* = {(*SOME x*. *x* ∈ *U*)| *U*. *U* ∈ *A*}
then have *countable B2* **using** ‹*countable A*› **by** (*simp add*: *Setcompr_eq_image*)
have ∃ *b* ∈ *B1* ∪ *B2*. *x* < *b* ∧ *b* ≤ *y* **if** *x* < *y* **for** *x y*
proof (*cases*)
assume ∃ *z*. *x* < *z* ∧ *z* < *y*
then obtain *z* **where** *z*: *x* < *z* ∧ *z* < *y* **by** *auto*

```
    define U where U = {x<..<y}
    then have open U by simp
    moreover have z ∈ U using z U_def by simp
    ultimately obtain V where V ∈ A z ∈ V V ⊆ U
      using topological_basisE[OF ‹topological_basis A›] by auto
    define w where w = (SOME x. x ∈ V)
    then have w ∈ V using ‹z ∈ V› by (metis someI2)
    then have x < w ∧ w ≤ y using ‹w ∈ V› ‹V ⊆ U› U_def by fastforce
    moreover have w ∈ B1 ∪ B2 using w_def B2_def ‹V ∈ A› by auto
    ultimately show ?thesis by auto
  next
    assume ¬(∃ z. x < z ∧ z < y)
    then have ∗: ⋀z. z > x ⟹ z ≥ y by auto
    define U where U = {x<..}
    then have open U by simp
    moreover have y ∈ U using ‹x < y› U_def by simp
    ultimately obtain V where V ∈ A y ∈ V V ⊆ U
      using topological_basisE[OF ‹topological_basis A›] by auto
    have U = {y..} unfolding U_def using ∗ ‹x < y› by auto
    then have V ⊆ {y..} using ‹V ⊆ U› by simp
    then have (LEAST w. w ∈ V) = y using ‹y ∈ V› by (meson Least_equality
atLeast_iff subsetCE)
    then have y ∈ B1 ∪ B2 using ‹V ∈ A› B1_def by auto
    moreover have x < y ∧ y ≤ y using ‹x < y› by simp
    ultimately show ?thesis by auto
  qed
  moreover have countable (B1 ∪ B2) using ‹countable B1› ‹countable B2› by
simp
  ultimately show ?thesis by auto
qed

lemma countable_separating_set_linorder2:
  shows ∃ B::('a::{linorder_topology, second_countable_topology} set). countable
B ∧ (∀ x y. x < y ⟶ (∃ b ∈ B. x ≤ b ∧ b < y))
proof −
 obtain A::'a set set where countable A topological_basis A using ex_countable_basis
by auto
  define B1 where B1 = {(GREATEST x. x ∈ U) | U. U ∈ A}
 then have countable B1 using ‹countable A› by (simp add: Setcompr_eq_image)
  define B2 where B2 = {(SOME x. x ∈ U)| U. U ∈ A}
 then have countable B2 using ‹countable A› by (simp add: Setcompr_eq_image)
  have ∃ b ∈ B1 ∪ B2. x ≤ b ∧ b < y if x < y for x y
  proof (cases)
    assume ∃ z. x < z ∧ z < y
    then obtain z where z: x < z ∧ z < y by auto
    define U where U = {x<..<y}
    then have open U by simp
    moreover have z ∈ U using z U_def by simp
    ultimately obtain V where V ∈ A z ∈ V V ⊆ U
```

using *topological_basisE*[*OF* ‹*topological_basis A*›] **by** *auto*
define w **where** $w = (SOME\ x.\ x \in V)$
then have $w \in V$ **using** ‹$z \in V$› **by** (*metis someI2*)
then have $x \leq w \wedge w < y$ **using** ‹$w \in V$› ‹$V \subseteq U$› *U_def* **by** *fastforce*
moreover have $w \in B1 \cup B2$ **using** *w_def B2_def* ‹$V \in A$› **by** *auto*
ultimately show *?thesis* **by** *auto*
next
assume $\neg(\exists z.\ x < z \wedge z < y)$
then have $*$: $\bigwedge z.\ z < y \Longrightarrow z \leq x$ **using** *leI* **by** *blast*
define U **where** $U = \{..<y\}$
then have *open U* **by** *simp*
moreover have $x \in U$ **using** ‹$x < y$› *U_def* **by** *simp*
ultimately obtain V **where** $V \in A\ x \in V\ V \subseteq U$
using *topological_basisE*[*OF* ‹*topological_basis A*›] **by** *auto*
have $U = \{..x\}$ **unfolding** *U_def* **using** $*$ ‹$x < y$› **by** *auto*
then have $V \subseteq \{..x\}$ **using** ‹$V \subseteq U$› **by** *simp*
then have (*GREATEST x. x ∈ V*) = x **using** ‹$x \in V$› **by** (*meson Greatest_equality atMost_iff subsetCE*)
then have $x \in B1 \cup B2$ **using** ‹$V \in A$› *B1_def* **by** *auto*
moreover have $x \leq x \wedge x < y$ **using** ‹$x < y$› **by** *simp*
ultimately show *?thesis* **by** *auto*
qed
moreover have *countable* (*B1 ∪ B2*) **using** ‹*countable B1*› ‹*countable B2*› **by** *simp*
ultimately show *?thesis* **by** *auto*
qed

lemma *countable_separating_set_dense_linorder*:
shows ∃ *B*::('*a*::{*linorder_topology*, *dense_linorder*, *second_countable_topology*} *set*). *countable B* ∧ (∀ *x y*. *x* < *y* ⟶ (∃ *b* ∈ *B*. *x* < *b* ∧ *b* < *y*))
proof −
obtain *B*::'*a set* **where** *B*: *countable B* $\bigwedge x\ y.\ x < y \Longrightarrow (\exists\, b \in B.\ x < b \wedge b \leq y)$
using *countable_separating_set_linorder1* **by** *auto*
have ∃ *b* ∈ *B*. *x* < *b* ∧ *b* < *y* **if** *x* < *y* **for** *x y*
proof −
obtain *z* **where** *x* < *z z* < *y* **using** ‹*x* < *y*› *dense* **by** *blast*
then obtain *b* **where** *b* ∈ *B x* < *b* ∧ *b* ≤ *z* **using** *B*(*2*) **by** *auto*
then have *x* < *b* ∧ *b* < *y* **using** ‹*z* < *y*› **by** *auto*
then show *?thesis* **using** ‹*b* ∈ *B*› **by** *auto*
qed
then show *?thesis* **using** *B*(*1*) **by** *auto*
qed

2.1.3 Polish spaces

Textbooks define Polish spaces as completely metrizable. We assume the topology to be complete for a given metric.

class *polish_space* = *complete_space* + *second_countable_topology*

2.1.4 Limit Points

definition (**in** *topological_space*) *islimpt*:: *'a* ⇒ *'a set* ⇒ *bool* (**infixr** *islimpt 60*)
where *x islimpt S* ⟷ (∀ *T*. *x*∈*T* ⟶ *open T* ⟶ (∃ *y*∈*S*. *y*∈*T* ∧ *y*≠*x*))

lemma *islimptI*:
assumes ⋀*T*. *x* ∈ *T* ⟹ *open T* ⟹ ∃ *y*∈*S*. *y* ∈ *T* ∧ *y* ≠ *x*
shows *x islimpt S*
using *assms* **unfolding** *islimpt_def* **by** *auto*

lemma *islimptE*:
assumes *x islimpt S* **and** *x* ∈ *T* **and** *open T*
obtains *y* **where** *y* ∈ *S* **and** *y* ∈ *T* **and** *y* ≠ *x*
using *assms* **unfolding** *islimpt_def* **by** *auto*

lemma *islimpt_iff_eventually*: *x islimpt S* ⟷ ¬ *eventually* (λ*y*. *y* ∉ *S*) (*at x*)
unfolding *islimpt_def eventually_at_topological* **by** *auto*

lemma *islimpt_subset*: *x islimpt S* ⟹ *S* ⊆ *T* ⟹ *x islimpt T*
unfolding *islimpt_def* **by** *fast*

lemma *islimpt_UNIV_iff*: *x islimpt UNIV* ⟷ ¬ *open* {*x*}
unfolding *islimpt_def* **by** (*safe*, *fast*, *case_tac T* = {*x*}, *fast*, *fast*)

lemma *islimpt_punctured*: *x islimpt S* = *x islimpt* (*S*−{*x*})
unfolding *islimpt_def* **by** *blast*

A perfect space has no isolated points.

lemma *islimpt_UNIV* [*simp*, *intro*]: *x islimpt UNIV*
for *x* :: *'a*::*perfect_space*
unfolding *islimpt_UNIV_iff* **by** (*rule not_open_singleton*)

lemma *closed_limpt*: *closed S* ⟷ (∀ *x*. *x islimpt S* ⟶ *x* ∈ *S*)
unfolding *closed_def open_subopen* [*of* −*S*]
by (*metis Compl_iff islimptE islimptI open_subopen subsetI*)

lemma *islimpt_EMPTY*[*simp*]: ¬ *x islimpt* {}
by (*auto simp*: *islimpt_def*)

lemma *islimpt_Un*: *x islimpt* (*S* ∪ *T*) ⟷ *x islimpt S* ∨ *x islimpt T*
by (*simp add*: *islimpt_iff_eventually eventually_conj_iff*)

lemma *islimpt_finite_union_iff*:
assumes *finite A*
shows *z islimpt* (⋃*x*∈*A*. *B x*) ⟷ (∃ *x*∈*A*. *z islimpt B x*)
using *assms* **by** (*induction rule*: *finite_induct*) (*simp_all add*: *islimpt_Un*)

lemma *islimpt_insert*:
fixes *x* :: *'a*::*t1_space*
shows *x islimpt* (*insert a S*) ⟷ *x islimpt S*

proof
assume *x islimpt (insert a S)*
then show *x islimpt S*
by (*metis closed_limpt closed_singleton empty_iff insert_iff insert_is_Un islimpt_Un islimpt_def*)
next
assume *x islimpt S*
then show *x islimpt (insert a S)*
by (*rule islimpt_subset*) *auto*
qed

lemma *islimpt_finite*:
fixes *x* :: *'a*::*t1_space*
shows *finite S* ⟹ ¬ *x islimpt S*
by (*induct set*: *finite*) (*simp_all add*: *islimpt_insert*)

lemma *islimpt_Un_finite*:
fixes *x* :: *'a*::*t1_space*
shows *finite S* ⟹ *x islimpt* (*S* ∪ *T*) ⟷ *x islimpt T*
by (*simp add*: *islimpt_Un islimpt_finite*)

lemma *islimpt_eq_acc_point*:
fixes *l* :: *'a* :: *t1_space*
shows *l islimpt S* ⟷ (∀ *U*. *l*∈*U* ⟶ *open U* ⟶ *infinite* (*U* ∩ *S*))
proof (*safe intro*!: *islimptI*)
fix *U*
assume *l islimpt S l* ∈ *U open U finite* (*U* ∩ *S*)
then have *l islimpt S l* ∈ (*U* − (*U* ∩ *S* − {*l*})) *open* (*U* − (*U* ∩ *S* − {*l*}))
by (*auto intro*: *finite_imp_closed*)
then show *False*
by (*rule islimptE*) *auto*
next
fix *T*
assume ∗: ∀ *U*. *l*∈*U* ⟶ *open U* ⟶ *infinite* (*U* ∩ *S*) *l* ∈ *T open T*
then have ∃ *x*. *x* ∈ (*T* ∩ *S* − {*l*})
by (*metis ex_in_conv finite.emptyI infinite_remove*)
then show ∃ *y*∈*S*. *y* ∈ *T* ∧ *y* ≠ *l*
by *auto*
qed

lemma *acc_point_range_imp_convergent_subsequence*:
fixes *l* :: *'a* :: *first_countable_topology*
assumes *l*: ∀ *U*. *l*∈*U* ⟶ *open U* ⟶ *infinite* (*U* ∩ *range f*)
shows ∃ *r*::*nat*⇒*nat*. *strict_mono r* ∧ (*f* ∘ *r*) ⟶ *l*
proof −
from *countable_basis_at_decseq*[*of l*]
obtain *A* **where** *A*:
⋀*i*. *open* (*A i*)
⋀*i*. *l* ∈ *A i*

```
    ⋀S. open S ⟹ l ∈ S ⟹ eventually (λi. A i ⊆ S) sequentially
   by blast
 define s where s n i = (SOME j. i < j ∧ f j ∈ A (Suc n)) for n i
 {
   fix n i
   have infinite (A (Suc n) ∩ range f − f‘{.. i})
     using l A by auto
   then have ∃x. x ∈ A (Suc n) ∩ range f − f‘{.. i}
     by (metis all_not_in_conv finite.emptyI)
   then have ∃a. i < a ∧ f a ∈ A (Suc n)
     by (force simp: linorder_not_le)
   then have i < s n i f (s n i) ∈ A (Suc n)
     unfolding s_def by (auto intro: someI2_ex)
 }
 note s = this
 define r where r = rec_nat (s 0 0) s
 have strict_mono r
   by (auto simp: r_def s strict_mono_Suc_iff)
 moreover
 have (λn. f (r n)) ⟶ l
 proof (rule topological_tendstoI)
   fix S
   assume open S l ∈ S
   with A(3) have eventually (λi. A i ⊆ S) sequentially
     by auto
   moreover
   {
     fix i
     assume Suc 0 ≤ i
     then have f (r i) ∈ A i
       by (cases i) (simp_all add: r_def s)
   }
   then have eventually (λi. f (r i) ∈ A i) sequentially
     by (auto simp: eventually_sequentially)
   ultimately show eventually (λi. f (r i) ∈ S) sequentially
     by eventually_elim auto
 qed
 ultimately show ∃r::nat⇒nat. strict_mono r ∧ (f ∘ r) ⟶ l
   by (auto simp: convergent_def comp_def)
qed

lemma islimpt_range_imp_convergent_subsequence:
 fixes l :: 'a :: {t1_space, first_countable_topology}
 assumes l: l islimpt (range f)
 shows ∃r::nat⇒nat. strict_mono r ∧ (f ∘ r) ⟶ l
 using l unfolding islimpt_eq_acc_point
 by (rule acc_point_range_imp_convergent_subsequence)

lemma sequence_unique_limpt:
```

```
  fixes f :: nat ⇒ 'a::t2_space
  assumes f: (f ⟶⟶ l) sequentially
    and l' islimpt (range f)
  shows l' = l
proof (rule ccontr)
  assume l' ≠ l
  obtain s t where open s open t l' ∈ s l ∈ t s ∩ t = {}
    using hausdorff [OF ‹l' ≠ l›] by auto
  then obtain N where ∀ n≥N. f n ∈ t
    by (meson f lim_explicit)

  have UNIV = {..<N} ∪ {N..}
    by auto
  then have l' islimpt (f ‘ ({..<N} ∪ {N..}))
    using assms(2) by simp
  then have l' islimpt (f ‘ {..<N} ∪ f ‘ {N..})
    by (simp add: image_Un)
  then have l' islimpt (f ‘ {N..})
    by (simp add: islimpt_Un_finite)
  then obtain y where y ∈ f ‘ {N..} y ∈ s y ≠ l'
    using ‹l' ∈ s› ‹open s› by (rule islimptE)
  then obtain n where N ≤ n f n ∈ s f n ≠ l'
    by auto
  with ‹∀ n≥N. f n ∈ t› ‹s ∩ t = {}› show False
    by blast
qed


lemma islimpt_sequential:
  fixes x :: 'a::first_countable_topology
  shows x islimpt S ⟷ (∃ f. (∀ n::nat. f n ∈ S - {x}) ∧ (f ⟶⟶ x) sequentially)
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  from countable_basis_at_decseq[of x] obtain A where A:
      ⋀i. open (A i)
      ⋀i. x ∈ A i
      ⋀S. open S ⟹ x ∈ S ⟹ eventually (λi. A i ⊆ S) sequentially
    by blast
  define f where f n = (SOME y. y ∈ S ∧ y ∈ A n ∧ x ≠ y) for n
  {
    fix n
    from ‹?lhs› have ∃ y. y ∈ S ∧ y ∈ A n ∧ x ≠ y
      unfolding islimpt_def using A(1,2)[of n] by auto
    then have f n ∈ S ∧ f n ∈ A n ∧ x ≠ f n
      unfolding f_def by (rule someI_ex)
    then have f n ∈ S f n ∈ A n x ≠ f n by auto
  }
  then have ∀ n. f n ∈ S - {x} by auto
```

```
  moreover have (λn. f n) ⟶ x
  proof (rule topological_tendstoI)
    fix S
    assume open S x ∈ S
    from A(3)[OF this] ‹⋀n. f n ∈ A n›
    show eventually (λx. f x ∈ S) sequentially
      by (auto elim!: eventually_mono)
  qed
  ultimately show ?rhs by fast
next
  assume ?rhs
  then obtain f :: nat ⇒ 'a where f: ⋀n. f n ∈ S − {x} and lim: f ⟶ x
    by auto
  show ?lhs
    unfolding islimpt_def
  proof safe
    fix T
    assume open T x ∈ T
    from lim[THEN topological_tendstoD, OF this] f
    show ∃y∈S. y ∈ T ∧ y ≠ x
      unfolding eventually_sequentially by auto
  qed
qed

lemma islimpt_isCont_image:
 fixes f :: 'a :: {first_countable_topology, t2_space} ⇒ 'b :: {first_countable_topology,
t2_space}
  assumes x islimpt A and isCont f x and ev: eventually (λy. f y ≠ f x) (at x)
  shows   f x islimpt f ‘ A
proof −
  from assms(1) obtain g where g: g ⟶ x range g ⊆ A − {x}
    unfolding islimpt_sequential by blast
  have filterlim g (at x) sequentially
    using g by (auto simp: filterlim_at intro!: always_eventually)
  then obtain N where N: ⋀n. n ≥ N ⟹ f (g n) ≠ f x
    by (metis (mono_tags, lifting) ev eventually_at_top_linorder filterlim_iff)
  have (λx. g (x + N)) ⟶ x
    using g(1) by (rule LIMSEQ_ignore_initial_segment)
  hence (λx. f (g (x + N))) ⟶ f x
    using assms(2) isCont_tendsto_compose by blast
  moreover have range (λx. f (g (x + N))) ⊆ f ‘ A − {f x}
    using g(2) N by auto
  ultimately show ?thesis
    unfolding islimpt_sequential by (intro exI[of _ λx. f (g (x + N))]) auto
qed

lemma islimpt_image:
  assumes z islimpt g −‘ A ∩ B g z ∉ A z ∈ B continuous_on B g
  shows   g z islimpt A
```

by (*smt* (*verit*, *best*) *IntD1 assms continuous_on_topological inf_le2 islimpt_def subset_eq vimageE*)

2.1.5 Interior of a Set

definition *interior* :: ('*a*::*topological_space*) *set* ⇒ '*a set* **where**
interior S = ⋃{*T*. *open T* ∧ *T* ⊆ *S*}

lemma *interiorI* [*intro?*]:
assumes *open T* **and** *x* ∈ *T* **and** *T* ⊆ *S*
shows *x* ∈ *interior S*
using *assms* **unfolding** *interior_def* **by** *fast*

lemma *interiorE* [*elim?*]:
assumes *x* ∈ *interior S*
obtains *T* **where** *open T* **and** *x* ∈ *T* **and** *T* ⊆ *S*
using *assms* **unfolding** *interior_def* **by** *fast*

lemma *open_interior* [*simp*, *intro*]: *open* (*interior S*)
by (*simp add*: *interior_def open_Union*)

lemma *interior_subset*: *interior S* ⊆ *S*
by (*auto simp*: *interior_def*)

lemma *interior_maximal*: *T* ⊆ *S* ⟹ *open T* ⟹ *T* ⊆ *interior S*
by (*auto simp*: *interior_def*)

lemma *interior_open*: *open S* ⟹ *interior S* = *S*
by (*intro equalityI interior_subset interior_maximal subset_refl*)

lemma *interior_eq*: *interior S* = *S* ⟷ *open S*
by (*metis open_interior interior_open*)

lemma *open_subset_interior*: *open S* ⟹ *S* ⊆ *interior T* ⟷ *S* ⊆ *T*
by (*metis interior_maximal interior_subset subset_trans*)

lemma *interior_empty* [*simp*]: *interior* {} = {}
using *open_empty* **by** (*rule interior_open*)

lemma *interior_UNIV* [*simp*]: *interior UNIV* = *UNIV*
using *open_UNIV* **by** (*rule interior_open*)

lemma *interior_interior* [*simp*]: *interior* (*interior S*) = *interior S*
using *open_interior* **by** (*rule interior_open*)

lemma *interior_mono*: *S* ⊆ *T* ⟹ *interior S* ⊆ *interior T*
by (*auto simp*: *interior_def*)

lemma *interior_unique*:

assumes $T \subseteq S$ **and** *open* T
assumes $\bigwedge T'.\ T' \subseteq S \Longrightarrow$ *open* $T' \Longrightarrow T' \subseteq T$
shows *interior* $S = T$
by (*intro equalityI assms interior_subset open_interior interior_maximal*)

lemma *interior_singleton* [*simp*]: *interior* $\{a\} = \{\}$
for $a :: 'a$::*perfect_space*
by (*meson interior_eq interior_subset not_open_singleton subset_singletonD*)

lemma *interior_Int* [*simp*]: *interior* $(S \cap T) =$ *interior* $S \cap$ *interior* T
by (*meson Int_mono Int_subset_iff antisym_conv interior_maximal interior_subset open_Int open_interior*)

lemma *eventually_nhds_in_nhd*: $x \in$ *interior* $s \Longrightarrow$ *eventually* $(\lambda y.\ y \in s)$ (*nhds* x)
using *interior_subset*[*of s*] **by** (*subst eventually_nhds*) *blast*

lemma *interior_limit_point* [*intro*]:
fixes $x :: 'a$::*perfect_space*
assumes x: $x \in$ *interior* S
shows x *islimpt* S
proof −
obtain T **where** $x \in T$ $T \subseteq S$ *open* T
using *interior_subset* x **by** *blast*
with x *islimpt_UNIV* [*of x*] **show** *?thesis*
unfolding *islimpt_def* **by** (*metis* (*full_types*) *Int_iff open_Int subsetD*)
qed

lemma *open_imp_islimpt*:
fixes x::$'a$:: *perfect_space*
assumes *open* S $x \in S$
shows x *islimpt* S
using *assms interior_eq interior_limit_point* **by** *auto*

lemma *islimpt_Int_eventually*:
assumes x *islimpt* A *eventually* $(\lambda y.\ y \in B)$ (*at* x)
shows x *islimpt* $A \cap B$
using *assms* **unfolding** *islimpt_def eventually_at_filter eventually_nhds*
by (*metis Int_iff UNIV_I open_Int*)

lemma *islimpt_conv_frequently_at*:
x *islimpt* $A \longleftrightarrow$ *frequently* $(\lambda y.\ y \in A)$ (*at* x)
by (*simp add*: *frequently_def islimpt_iff_eventually*)

lemma *frequently_at_imp_islimpt*:
assumes *frequently* $(\lambda y.\ y \in A)$ (*at* x)
shows x *islimpt* A
by (*simp add*: *assms islimpt_conv_frequently_at*)

lemma *interior_closed_Un_empty_interior*:
assumes *cS*: *closed S*
and *iT*: *interior T* = {}
shows *interior* (*S* ∪ *T*) = *interior S*
proof
show *interior S* ⊆ *interior* (*S* ∪ *T*)
by (*rule interior_mono*) (*rule Un_upper1*)
show *interior* (*S* ∪ *T*) ⊆ *interior S*
proof
fix *x*
assume *x* ∈ *interior* (*S* ∪ *T*)
then obtain *R* **where** *open R x* ∈ *R R* ⊆ *S* ∪ *T* **..**
show *x* ∈ *interior S*
proof (*rule ccontr*)
assume *x* ∉ *interior S*
with ‹*x* ∈ *R*› ‹*open R*› **obtain** *y* **where** *y* ∈ *R* − *S*
unfolding *interior_def* **by** *fast*
then show *False*
by (*metis Diff_subset_conv* ‹*R* ⊆ *S* ∪ *T*› ‹*open R*› *cS empty_iff iT interiorI open_Diff*)
qed
qed
qed

lemma *interior_Times*: *interior* (*A* × *B*) = *interior A* × *interior B*
proof (*rule interior_unique*)
show *interior A* × *interior B* ⊆ *A* × *B*
by (*intro Sigma_mono interior_subset*)
show *open* (*interior A* × *interior B*)
by (*intro open_Times open_interior*)
fix *T*
assume *T* ⊆ *A* × *B* **and** *open T*
then show *T* ⊆ *interior A* × *interior B*
proof *safe*
fix *x y*
assume (*x*, *y*) ∈ *T*
then obtain *C D* **where** *open C open D C* × *D* ⊆ *T x* ∈ *C y* ∈ *D*
using ‹*open T*› **unfolding** *open_prod_def* **by** *fast*
then have *open C open D C* ⊆ *A D* ⊆ *B x* ∈ *C y* ∈ *D*
using ‹*T* ⊆ *A* × *B*› **by** *auto*
then show *x* ∈ *interior A* **and** *y* ∈ *interior B*
by (*auto intro*: *interiorI*)
qed
qed

lemma *interior_Ici*:
fixes *x* :: *'a* :: {*dense_linorder*,*linorder_topology*}
assumes *b* < *x*
shows *interior* {*x* ..} = {*x* <..}

proof (*rule interior_unique*)
fix T
assume $T \subseteq \{x ..\}$ *open* T
moreover have $x \notin T$
proof
assume $x \in T$
obtain y **where** $y < x$ $\{y <.. x\} \subseteq T$
using *open_left*[*OF* ‹*open* T› ‹$x \in T$› ‹$b < x$›] **by** *auto*
with *dense*[*OF* ‹$y < x$›] **obtain** z **where** $z \in T$ $z < x$
by (*auto simp*: *subset_eq Ball_def*)
with ‹$T \subseteq \{x ..\}$› **show** *False* **by** *auto*
qed
ultimately show $T \subseteq \{x <..\}$
by (*auto simp*: *subset_eq less_le*)
qed *auto*

lemma *interior_Iic*:
fixes x :: $'a$ ::{*dense_linorder*,*linorder_topology*}
assumes $x < b$
shows *interior* $\{.. x\} = \{..< x\}$
proof (*rule interior_unique*)
fix T
assume $T \subseteq \{.. x\}$ *open* T
moreover have $x \notin T$
proof
assume $x \in T$
obtain y **where** $x < y$ $\{x ..< y\} \subseteq T$
using *open_right*[*OF* ‹*open* T› ‹$x \in T$› ‹$x < b$›] **by** *auto*
with *dense*[*OF* ‹$x < y$›] **obtain** z **where** $z \in T$ $x < z$
by (*auto simp*: *subset_eq Ball_def less_le*)
with ‹$T \subseteq \{.. x\}$› **show** *False* **by** *auto*
qed
ultimately show $T \subseteq \{..< x\}$
by (*auto simp*: *subset_eq less_le*)
qed *auto*

lemma *countable_disjoint_nonempty_interior_subsets*:
fixes $\mathcal{F}$ :: $'a$::*second_countable_topology set set*
assumes *pw*: *pairwise disjnt* $\mathcal{F}$ **and** *int*: $\bigwedge S.\ [\![S \in \mathcal{F}; \textit{interior } S = \{\}]\!] \Longrightarrow S = \{\}$
shows *countable* $\mathcal{F}$
proof (*rule countable_image_inj_on*)
have *disjoint* (*interior* ‘ $\mathcal{F}$)
using *pw* **by** (*simp add*: *disjoint_image_subset interior_subset*)
then show *countable* (*interior* ‘ $\mathcal{F}$)
by (*auto intro*: *countable_disjoint_open_subsets*)
show *inj_on interior* $\mathcal{F}$
using *pw* **apply** (*clarsimp simp*: *inj_on_def pairwise_def*)
apply (*metis disjnt_def disjnt_subset1 inf.orderE int interior_subset*)

done
qed

2.1.6 Closure of a Set

definition *closure* :: ('*a*::*topological_space*) *set* ⇒ '*a set* **where**
closure S = *S* ∪ {*x* . *x islimpt S*}

lemma *interior_closure*: *interior S* = − (*closure* (− *S*))
by (*auto simp*: *interior_def closure_def islimpt_def*)

lemma *closure_interior*: *closure S* = − *interior* (− *S*)
by (*simp add*: *interior_closure*)

lemma *closed_closure*[*simp*, *intro*]: *closed* (*closure S*)
by (*simp add*: *closure_interior closed_Compl*)

lemma *closure_subset*: *S* ⊆ *closure S*
by (*simp add*: *closure_def*)

lemma *closure_hull*: *closure S* = *closed hull S*
by (*auto simp*: *hull_def closure_interior interior_def*)

lemma *closure_eq*: *closure S* = *S* ⟷ *closed S*
unfolding *closure_hull* **using** *closed_Inter* **by** (*rule hull_eq*)

lemma *closure_closed* [*simp*]: *closed S* ⟹ *closure S* = *S*
by (*simp only*: *closure_eq*)

lemma *closure_closure* [*simp*]: *closure* (*closure S*) = *closure S*
unfolding *closure_hull* **by** (*rule hull_hull*)

lemma *closure_mono*: *S* ⊆ *T* ⟹ *closure S* ⊆ *closure T*
unfolding *closure_hull* **by** (*rule hull_mono*)

lemma *closure_minimal*: *S* ⊆ *T* ⟹ *closed T* ⟹ *closure S* ⊆ *T*
unfolding *closure_hull* **by** (*rule hull_minimal*)

lemma *closure_unique*:
assumes *S* ⊆ *T*
and *closed T*
and ⋀*T'*. *S* ⊆ *T'* ⟹ *closed T'* ⟹ *T* ⊆ *T'*
shows *closure S* = *T*
using *assms* **unfolding** *closure_hull* **by** (*rule hull_unique*)

lemma *closure_empty* [*simp*]: *closure* {} = {}
using *closed_empty* **by** (*rule closure_closed*)

lemma *closure_UNIV* [*simp*]: *closure UNIV* = *UNIV*

using *closed_UNIV* **by** (*rule closure_closed*)

lemma *closure_Un* [*simp*]: *closure* $(S \cup T)$ = *closure* $S \cup$ *closure* T
by (*simp add*: *closure_interior*)

lemma *closure_eq_empty* [*iff*]: *closure* $S = \{\} \longleftrightarrow S = \{\}$
using *closure_empty closure_subset*[*of* S] **by** *blast*

lemma *closure_subset_eq*: *closure* $S \subseteq S \longleftrightarrow$ *closed* S
using *closure_eq*[*of* S] *closure_subset*[*of* S] **by** *simp*

lemma *open_Int_closure_eq_empty*: *open* $S \Longrightarrow (S \cap$ *closure* $T) = \{\} \longleftrightarrow S \cap T = \{\}$
using *open_subset_interior*[*of* $S - T$]
using *interior_subset*[*of* $- T$]
by (*auto simp*: *closure_interior*)

lemma *open_Int_closure_subset*: *open* $S \Longrightarrow S \cap$ *closure* $T \subseteq$ *closure* $(S \cap T)$
proof
fix x
assume $*$: *open* S $x \in S \cap$ *closure* T
then have x *islimpt* $(S \cap T)$ **if** x *islimpt* T
by (*metis IntD1 eventually_at_in_open' inf_commute islimpt_Int_eventually that*)
with $*$ **show** $x \in$ *closure* $(S \cap T)$
unfolding *closure_def* **by** *blast*
qed

lemma *closure_complement*: *closure* $(- S) = -$ *interior* S
by (*simp add*: *closure_interior*)

lemma *interior_complement*: *interior* $(- S) = -$ *closure* S
by (*simp add*: *closure_interior*)

lemma *interior_diff*: *interior*$(S - T)$ = *interior* $S -$ *closure* T
by (*simp add*: *Diff_eq interior_complement*)

lemma *closure_Times*: *closure* $(A \times B)$ = *closure* $A \times$ *closure* B
proof (*rule closure_unique*)
show $A \times B \subseteq$ *closure* $A \times$ *closure* B
by (*intro Sigma_mono closure_subset*)
show *closed* (*closure* $A \times$ *closure* B)
by (*intro closed_Times closed_closure*)
fix T
assume $A \times B \subseteq T$ **and** *closed* T
then show *closure* $A \times$ *closure* $B \subseteq T$
apply (*simp add*: *closed_def open_prod_def*, *clarify*)
apply (*rule ccontr*)
apply (*drule_tac* x=(a, b) **in** *bspec*, *simp*, *clarify*, *rename_tac C D*)

apply (*simp add*: *closure_interior interior_def*)
apply (*drule_tac x*=*C* **in** *spec*)
apply (*drule_tac x*=*D* **in** *spec*, *auto*)
done
qed

lemma *closure_open_Int_superset*:
assumes *open S S* ⊆ *closure T*
shows *closure*(*S* ∩ *T*) = *closure S*
by (*metis Int_Un_distrib Un_Int_eq*(4) *assms closure_Un closure_closure open_Int_closure_subset sup.orderE*)

lemma *closure_Int*: *closure* (⋂ *I*) ⊆ ⋂ {*closure S* |*S*. *S* ∈ *I*}
by (*simp add*: *INF_greatest Inter_lower Setcompr_eq_image closure_mono*)

lemma *islimpt_in_closure*: (*x islimpt S*) = (*x*∈*closure*(*S*−{*x*}))
unfolding *closure_def* **using** *islimpt_punctured* **by** *blast*

lemma *connected_imp_connected_closure*: *connected S* ⟹ *connected* (*closure S*)
by (*rule connectedI*) (*meson closure_subset open_Int open_Int_closure_eq_empty subset_trans connectedD*)

lemma *bdd_below_closure*:
fixes *A* :: *real set*
assumes *bdd_below A*
shows *bdd_below* (*closure A*)
proof −
from *assms* **obtain** *m* **where** ⋀*x*. *x* ∈ *A* ⟹ *m* ≤ *x*
by (*auto simp*: *bdd_below_def*)
then have *A* ⊆ {*m*..} **by** *auto*
then have *closure A* ⊆ {*m*..}
using *closed_real_atLeast* **by** (*rule closure_minimal*)
then show *?thesis*
by (*auto simp*: *bdd_below_def*)
qed

2.1.7 Frontier (also known as boundary)

definition *frontier* :: ('*a*::*topological_space*) *set* ⇒ '*a set* **where**
frontier S = *closure S* − *interior S*

lemma *frontier_closed* [*iff*]: *closed* (*frontier S*)
by (*simp add*: *frontier_def closed_Diff*)

lemma *frontier_closures*: *frontier S* = *closure S* ∩ *closure* (− *S*)
by (*auto simp*: *frontier_def interior_closure*)

lemma *frontier_Int*: *frontier*(*S* ∩ *T*) = *closure*(*S* ∩ *T*) ∩ (*frontier S* ∪ *frontier T*)

proof −
have *closure* $(S \cap T) \subseteq$ *closure* S *closure* $(S \cap T) \subseteq$ *closure* T
by (*simp_all add*: *closure_mono*)
then show *?thesis*
by (*auto simp*: *frontier_closures*)
qed

lemma *frontier_Int_subset*: *frontier*$(S \cap T) \subseteq$ *frontier* $S \cup$ *frontier* T
by (*auto simp*: *frontier_Int*)

lemma *frontier_Int_closed*:
assumes *closed* S *closed* T
shows *frontier*$(S \cap T) =$ (*frontier* $S \cap T) \cup (S \cap$ *frontier* $T)$
by (*smt* (*verit*, *best*) *Diff_Int Int_Diff assms closed_Int closure_eq frontier_def inf_commute interior_Int*)

lemma *frontier_subset_closed*: *closed* $S \Longrightarrow$ *frontier* $S \subseteq S$
by (*metis frontier_def closure_closed Diff_subset*)

lemma *frontier_empty* [*simp*]: *frontier* $\{\} = \{\}$
by (*simp add*: *frontier_def*)

lemma *frontier_subset_eq*: *frontier* $S \subseteq S \longleftrightarrow$ *closed* S
by (*metis Diff_subset_conv closure_subset_eq frontier_def interior_subset subset_Un_eq*)

lemma *frontier_complement* [*simp*]: *frontier* $(-S) =$ *frontier* S
by (*auto simp*: *frontier_def closure_complement interior_complement*)

lemma *frontier_Un_subset*: *frontier*$(S \cup T) \subseteq$ *frontier* $S \cup$ *frontier* T
by (*metis compl_sup frontier_Int_subset frontier_complement*)

lemma *frontier_disjoint_eq*: *frontier* $S \cap S = \{\} \longleftrightarrow$ *open* S
using *frontier_complement frontier_subset_eq*[*of* $-S$]
unfolding *open_closed* **by** *auto*

lemma *frontier_UNIV* [*simp*]: *frontier UNIV* $= \{\}$
using *frontier_complement frontier_empty* **by** *fastforce*

lemma *frontier_interiors*: *frontier* $s = -$ *interior*$(s) -$ *interior*$(-s)$
by (*simp add*: *Int_commute frontier_def interior_closure*)

lemma *frontier_interior_subset*: *frontier*(*interior* $S) \subseteq$ *frontier* S
by (*simp add*: *Diff_mono frontier_interiors interior_mono interior_subset*)

lemma *closure_Un_frontier*: *closure* $S = S \cup$ *frontier* S
by (*simp add*: *closure_def frontier_closures sup_inf_distrib1*)

2.1.8 Filters and the "eventually true" quantifier

Identify Trivial limits, where we can't approach arbitrarily closely.

lemma *trivial_limit_within*: *trivial_limit* (*at a within S*) ⟷ ¬ *a islimpt S*
unfolding *trivial_limit_def eventually_at_topological islimpt_def*
by *blast*

lemma *trivial_limit_at_iff*: *trivial_limit* (*at a*) ⟷ ¬ *a islimpt UNIV*
using *trivial_limit_within* [*of a UNIV*] **by** *simp*

lemma *trivial_limit_at*: ¬ *trivial_limit* (*at a*)
for *a* :: *'a*::*perfect_space*
by (*rule at_neq_bot*)

lemma *not_trivial_limit_within*: ¬ *trivial_limit* (*at x within S*) = (*x* ∈ *closure* (*S* − {*x*}))
using *islimpt_in_closure* **by** (*metis trivial_limit_within*)

lemma *not_in_closure_trivial_limitI*:
x ∉ *closure S* ⟹ *trivial_limit* (*at x within S*)
using *not_trivial_limit_within*[*of x S*]
by *safe* (*metis Diff_empty Diff_insert0 closure_subset contra_subsetD*)

lemma *filterlim_at_within_closure_implies_filterlim*: *filterlim f l* (*at x within S*)
if *x* ∈ *closure S* ⟹ *filterlim f l* (*at x within S*)
by (*metis bot.extremum filterlim_iff_le_filtercomap not_in_closure_trivial_limitI that*)

lemma *at_within_eq_bot_iff*: *at c within A* = *bot* ⟷ *c* ∉ *closure* (*A* − {*c*})
using *not_trivial_limit_within*[*of c A*] **by** *blast*

2.1.9 Limits

The expected monotonicity property.

lemma *Lim_Un*:
assumes (*f* ⟶ *l*) (*at x within S*) (*f* ⟶ *l*) (*at x within T*)
shows (*f* ⟶ *l*) (*at x within* (*S* ∪ *T*))
using *assms* **unfolding** *at_within_union* **by** (*rule filterlim_sup*)

lemma *Lim_Un_univ*:
(*f* ⟶ *l*) (*at x within S*) ⟹ (*f* ⟶ *l*) (*at x within T*) ⟹
S ∪ *T* = *UNIV* ⟹ (*f* ⟶ *l*) (*at x*)
by (*metis Lim_Un*)

Interrelations between restricted and unrestricted limits.

lemma *Lim_at_imp_Lim_at_within*: (*f* ⟶ *l*) (*at x*) ⟹ (*f* ⟶ *l*) (*at x within S*)
by (*metis order_refl filterlim_mono subset_UNIV at_le*)

lemma *eventually_within_interior*:
assumes $x \in$ *interior* S
shows *eventually* P (*at* x *within* S) $\longleftrightarrow$ *eventually* P (*at* x)
by (*metis assms at_within_open_subset interior_subset open_interior*)

lemma *at_within_interior*: $x \in$ *interior* $S \Longrightarrow$ *at* x *within* $S =$ *at* x
unfolding *filter_eq_iff* **by** (*intro allI eventually_within_interior*)

lemma *Lim_within_LIMSEQ*:
fixes a :: $'a$::*first_countable_topology*
assumes $\forall S.\ (\forall n.\ S\ n \neq a \wedge S\ n \in T) \wedge S \longrightarrow a \longrightarrow (\lambda n.\ X\ (S\ n)) \longrightarrow L$
shows ($X \longrightarrow L$) (*at* a *within* T)
using *assms* **unfolding** *tendsto_def* [**where** l=L]
by (*simp add*: *sequentially_imp_eventually_within*)

lemma *Lim_right_bound*:
fixes f :: $'a$ :: {*linorder_topology*, *conditionally_complete_linorder*, *no_top*} $\Rightarrow$
$'b$::{*linorder_topology*, *conditionally_complete_linorder*}
assumes *mono*: $\bigwedge a\ b.\ a \in I \Longrightarrow b \in I \Longrightarrow x < a \Longrightarrow a \leq b \Longrightarrow f\ a \leq f\ b$
and *bnd*: $\bigwedge a.\ a \in I \Longrightarrow x < a \Longrightarrow K \leq f\ a$
shows ($f \longrightarrow$ *Inf* (f ‘ ({x<..} $\cap$ I))) (*at* x *within* ({x<..} $\cap$ I))
proof (*cases* {x<..} $\cap$ I = {})
case *True*
then show *?thesis* **by** *simp*
next
case *False*
show *?thesis*
proof (*rule order_tendstoI*)
fix a
assume a: $a <$ *Inf* (f ‘ ({x<..} $\cap$ I))
{
fix y
assume $y \in$ {x<..} $\cap$ I
with *False bnd* **have** *Inf* (f ‘ ({x<..} $\cap$ I)) $\leq f\ y$
by (*auto intro*!: *cInf_lower bdd_belowI2*)
with a **have** $a < f\ y$
by (*blast intro*: *less_le_trans*)
}
then show *eventually* ($\lambda x.\ a < f\ x$) (*at* x *within* ({x<..} $\cap$ I))
by (*auto simp*: *eventually_at_filter intro*: *exI*[*of* _ *1*] *zero_less_one*)
next
fix a
assume *Inf* (f ‘ ({x<..} $\cap$ I)) $< a$
from *cInf_lessD*[*OF* _ *this*] *False* **obtain** y **where** y: $x < y$ $y \in I$ $f\ y < a$
by *auto*
then have *eventually* ($\lambda x.\ x \in I \longrightarrow f\ x < a$) (*at_right* x)
unfolding *eventually_at_right*[*OF* ‹$x < y$›] **by** (*metis less_imp_le le_less_trans*

mono)
  then show *eventually* (λ*x*. *f x* < *a*) (*at x within* ({*x*<..} ∩ *I*))
   unfolding *eventually_at_filter* **by** *eventually_elim simp*
 qed
qed

These are special for limits out of the same topological space.

lemma *Lim_within_id*: (*id* ⟶ *a*) (*at a within s*)
 unfolding *id_def* **by** (*rule tendsto_ident_at*)

lemma *Lim_at_id*: (*id* ⟶ *a*) (*at a*)
 unfolding *id_def* **by** (*rule tendsto_ident_at*)

It's also sometimes useful to extract the limit point from the filter.

abbreviation *netlimit* :: *'a*::*t2_space filter* ⇒ *'a*
 where *netlimit F* ≡ *Lim F* (λ*x*. *x*)

lemma *netlimit_at* [*simp*]:
 fixes *a* :: *'a*::{*perfect_space*,*t2_space*}
 shows *netlimit* (*at a*) = *a*
 using *Lim_ident_at* [*of a UNIV*] **by** *simp*

lemma *lim_within_interior*:
 x ∈ *interior S* ⟹ (*f* ⟶ *l*) (*at x within S*) ⟷ (*f* ⟶ *l*) (*at x*)
 by (*metis at_within_interior*)

lemma *netlimit_within_interior*:
 fixes *x* :: *'a*::{*t2_space*,*perfect_space*}
 assumes *x* ∈ *interior S*
 shows *netlimit* (*at x within S*) = *x*
 using *assms* **by** (*metis at_within_interior netlimit_at*)

Useful lemmas on closure and set of possible sequential limits.

lemma *closure_sequential*:
 fixes *l* :: *'a*::*first_countable_topology*
 shows *l* ∈ *closure S* ⟷ (∃ *x*. (∀ *n*. *x n* ∈ *S*) ∧ (*x* ⟶ *l*) *sequentially*)
 by (*metis Diff_empty Diff_insert0 Un_iff closure_def islimpt_sequential mem_Collect_eq tendsto_const*)

lemma *closed_sequential_limits*:
 fixes *S* :: *'a*::*first_countable_topology set*
 shows *closed S* ⟷ (∀ *x l*. (∀ *n*. *x n* ∈ *S*) ∧ (*x* ⟶ *l*) *sequentially* ⟶ *l* ∈ *S*)
by (*metis closure_sequential closure_subset_eq subset_iff*)

lemma *tendsto_If_within_closures*:
 assumes *f*: *x* ∈ *S* ∪ (*closure S* ∩ *closure T*) ⟹
    (*f* ⟶ *l x*) (*at x within S* ∪ (*closure S* ∩ *closure T*))
 assumes *g*: *x* ∈ *T* ∪ (*closure S* ∩ *closure T*) ⟹
    (*g* ⟶ *l x*) (*at x within T* ∪ (*closure S* ∩ *closure T*))

assumes $x \in S \cup T$
shows $((\lambda x.\ \mathit{if}\ x \in S\ \mathit{then}\ f\ x\ \mathit{else}\ g\ x) \longrightarrow l\ x)\ (\mathit{at}\ x\ \mathit{within}\ S \cup T)$
proof −
have ∗: $(S \cup T) \cap \{x.\ x \in S\} = S\ (S \cup T) \cap \{x.\ x \notin S\} = T - S$
by *auto*
have $(f \longrightarrow l\ x)\ (\mathit{at}\ x\ \mathit{within}\ S)$
by (*rule filterlim_at_within_closure_implies_filterlim*)
(*use* ‹$x \in \_$› **in** ‹*auto simp*: *inf_commute closure_def intro*: *tendsto_within_subset*[*OF f*]›)
moreover
have $(g \longrightarrow l\ x)\ (\mathit{at}\ x\ \mathit{within}\ T - S)$
by (*rule filterlim_at_within_closure_implies_filterlim*)
(*use* ‹$x \in \_$› **in**
‹*auto intro*!: *tendsto_within_subset*[*OF g*] *simp*: *closure_def intro*: *islimpt_subset*›)
ultimately show *?thesis*
by (*intro filterlim_at_within_If*) (*simp_all only*: ∗)
qed

2.1.10 Compactness

lemma *brouwer_compactness_lemma*:
fixes f :: $'a$::*topological_space* ⇒ $'b$::*real_normed_vector*
assumes *compact S*
and *continuous_on S f*
and ¬ $(\exists x{\in}S.\ f\ x = 0)$
obtains d **where** $0 < d$ **and** $\forall x{\in}S.\ d \le \mathit{norm}\ (f\ x)$
proof (*cases* $S = \{\}$)
case *True*
show *thesis*
by (*rule that* [*of 1*]) (*auto simp*: *True*)
next
case *False*
have *continuous_on S* (*norm* ∘ *f*)
by (*rule continuous_intros continuous_on_norm assms*(*2*))+
with *False* **obtain** x **where** x: $x \in S\ \forall y{\in}S.\ (\mathit{norm} \circ f)\ x \le (\mathit{norm} \circ f)\ y$
using *continuous_attains_inf*[*OF assms*(*1*), *of norm* ∘ *f*]
unfolding *o_def*
by *auto*
then show *?thesis*
by (*metis assms*(*3*) *that comp_apply zero_less_norm_iff*)
qed

Bolzano-Weierstrass property

proposition *Heine_Borel_imp_Bolzano_Weierstrass*:
assumes *compact S*
and *infinite T*
and $T \subseteq S$
shows $\exists x \in S.\ x\ \mathit{islimpt}\ T$

proof (*rule ccontr*)
assume ¬ (∃ *x* ∈ *S*. *x islimpt T*)
then obtain *f* **where** *f*: ∀ *x*∈*S*. *x* ∈ *f x* ∧ *open* (*f x*) ∧ (∀ *y*∈*T*. *y* ∈ *f x* ⟶ *y* = *x*)
unfolding *islimpt_def* **by** *metis*
obtain *g* **where** *g*: *g* ⊆ {*T*. ∃ *x*. *x* ∈ *S* ∧ *T* = *f x*} *finite g S* ⊆ ⋃ *g*
using *assms*(*1*)[*unfolded compact_eq_Heine_Borel*, *THEN spec*[**where** *x*={*T*. ∃ *x*. *x*∈*S* ∧ *T* = *f x*}]]
using *f* **by** *auto*
then have *g'*: ∀ *x*∈*g*. ∃ *y* ∈ *S*. *x* = *f y*
by *auto*
have *inj_on f T*
by (*smt* (*verit*, *best*) *assms*(*3*) *f inj_onI subsetD*)
then have *infinite* (*f* ‘ *T*)
using *assms*(*2*) **using** *finite_imageD* **by** *auto*
moreover
have *False* **if** *x* ∈ *T f x* ∉ *g* **for** *x*
by (*smt* (*verit*) *UnionE assms*(*3*) *f g' g*(*3*) *subsetD that*)
then have *f* ‘ *T* ⊆ *g* **by** *auto*
ultimately show *False*
using *g*(*2*) **using** *finite_subset* **by** *auto*
qed

lemma *sequence_infinite_lemma*:
fixes *f* :: *nat* ⇒ *'a*::*t1_space*
assumes ∀ *n*. *f n* ≠ *l*
and (*f* ⟶⟶ *l*) *sequentially*
shows *infinite* (*range f*)
proof
assume *finite* (*range f*)
then have *l* ∉ *range f* ∧ *closed* (*range f*)
using ‹*finite* (*range f*)› *assms*(*1*) *finite_imp_closed* **by** *blast*
then have *eventually* (λ*n*. *f n* ∈ − *range f*) *sequentially*
by (*metis Compl_iff assms*(*2*) *open_Compl topological_tendstoD*)
then show *False*
unfolding *eventually_sequentially* **by** *auto*
qed

lemma *Bolzano_Weierstrass_imp_closed*:
fixes *S* :: *'a*::{*first_countable_topology*,*t2_space*} *set*
assumes ∀ *T*. *infinite T* ∧ *T* ⊆ *S* −−> (∃ *x* ∈ *S*. *x islimpt T*)
shows *closed S*
proof −
{
fix *x l*
assume *as*: ∀ *n*::*nat*. *x n* ∈ *S* (*x* ⟶⟶ *l*) *sequentially*
have *l* ∈ *S*
proof (*cases* ∀ *n*. *x n* ≠ *l*)
case *False*

```
    then show l∈S using as(1) by auto
   next
    case True
    with as(2) have infinite (range x)
     using sequence_infinite_lemma[of x l] by auto
    then obtain l' where l'∈S l' islimpt (range x)
     using as(1) assms by auto
    then show l∈S using sequence_unique_limpt as True by auto
   qed
  }
  then show ?thesis
   unfolding closed_sequential_limits by fast
qed

lemma closure_insert:
  fixes x :: 'a::t1_space
  shows closure (insert x S) = insert x (closure S)
  by (metis closed_singleton closure_Un closure_closed insert_is_Un)

lemma finite_not_islimpt_in_compact:
  assumes compact A ⋀z. z ∈ A ⟹ ¬z islimpt B
  shows   finite (A ∩ B)
   by (meson Heine_Borel_imp_Bolzano_Weierstrass assms inf_le1 inf_le2 is-
limpt_subset)
```

In particular, some common special cases.

```
lemma compact_Un [intro]:
  assumes compact S
    and compact T
  shows  compact (S ∪ T)
proof (rule compactI)
  fix f
  assume ∗: Ball f open S ∪ T ⊆ ⋃f
  from ∗ ‹compact S› obtain s' where s' ⊆ f ∧ finite s' ∧ S ⊆ ⋃s'
   unfolding compact_eq_Heine_Borel by (auto elim!: allE[of _ f])
  moreover
  from ∗ ‹compact T› obtain t' where t' ⊆ f ∧ finite t' ∧ T ⊆ ⋃t'
   unfolding compact_eq_Heine_Borel by (auto elim!: allE[of _ f])
  ultimately show ∃f'⊆f. finite f' ∧ S ∪ T ⊆ ⋃f'
   by (auto intro!: exI[of _ s' ∪ t'])
qed

lemma compact_Union [intro]: finite S ⟹ (⋀T. T ∈ S ⟹ compact T) ⟹
compact (⋃S)
  by (induct set: finite) auto

lemma compact_UN [intro]:
  finite A ⟹ (⋀x. x ∈ A ⟹ compact (B x)) ⟹ compact (⋃x∈A. B x)
  by blast
```

lemma *closed_Int_compact* [*intro*]:
assumes *closed S* **and** *compact T*
shows *compact* (*S* ∩ *T*)
using *compact_Int_closed* [*of T S*] *assms*
by (*simp add*: *Int_commute*)

lemma *compact_Int* [*intro*]:
fixes *S T* :: *'a* :: *t2_space set*
assumes *compact S* **and** *compact T*
shows *compact* (*S* ∩ *T*)
using *assms* **by** (*intro compact_Int_closed compact_imp_closed*)

lemma *compact_sing* [*simp*]: *compact* {*a*}
unfolding *compact_eq_Heine_Borel* **by** *auto*

lemma *compact_insert* [*simp*]:
assumes *compact S*
shows *compact* (*insert x S*)
by (*metis assms compact_Un compact_sing insert_is_Un*)

lemma *finite_imp_compact*: *finite S* ⟹ *compact S*
by (*induct set*: *finite*) *simp_all*

lemma *open_delete*:
fixes *S* :: *'a*::*t1_space set*
shows *open S* ⟹ *open* (*S* − {*x*})
by (*simp add*: *open_Diff*)

Compactness expressed with filters

lemma *closure_iff_nhds_not_empty*:
x ∈ *closure X* ⟷ (∀ *A*. ∀ *S*⊆*A*. *open S* ⟶ *x* ∈ *S* ⟶ *X* ∩ *A* ≠ {})
proof *safe*
assume *x*: *x* ∈ *closure X*
fix *S A*
assume §: *open S x* ∈ *S X* ∩ *A* = {} *S* ⊆ *A*
then have *x* ∉ *closure* (−*S*)
by (*simp add*: *closed_open*)
with *x* **have** *x* ∈ *closure X* − *closure* (−*S*)
by *auto*
with § **show** *False*
by (*metis Compl_iff Diff_iff closure_mono disjoint_iff subsetD subsetI*)
next
assume ∀ *A S*. *S* ⊆ *A* ⟶ *open S* ⟶ *x* ∈ *S* ⟶ *X* ∩ *A* ≠ {}
then show *x* ∈ *closure X*
by (*metis ComplI Compl_disjoint closure_interior interior_subset open_interior*)
qed

lemma *compact_filter*:

compact U ⟷ (∀ *F*. *F* ≠ *bot* ⟶ *eventually* (λ*x*. *x* ∈ *U*) *F* ⟶ (∃ *x*∈*U*. *inf* (*nhds x*) *F* ≠ *bot*))
proof (*intro allI iffI impI compact_fip*[*THEN iffD2*] *notI*)
fix *F*
assume *compact U*
assume *F*: *F* ≠ *bot eventually* (λ*x*. *x* ∈ *U*) *F*
then have *U* ≠ {}
by (*auto simp*: *eventually_False*)

define *Z* **where** *Z* = *closure* ‘ {*A*. *eventually* (λ*x*. *x* ∈ *A*) *F*}
then have ∀ *z*∈*Z*. *closed z*
by *auto*
moreover
have *ev_Z*: ⋀*z*. *z* ∈ *Z* ⟹ *eventually* (λ*x*. *x* ∈ *z*) *F*
unfolding *Z_def* **by** (*auto elim*: *eventually_mono intro*: *subsetD*[*OF closure_subset*])
have (∀ *B* ⊆ *Z*. *finite B* ⟶ *U* ∩ ⋂ *B* ≠ {})
proof (*intro allI impI*)
fix *B* **assume** *finite B B* ⊆ *Z*
with ‹*finite B*› *ev_Z F*(*2*) **have** *eventually* (λ*x*. *x* ∈ *U* ∩ (⋂ *B*)) *F*
by (*auto simp*: *eventually_ball_finite_distrib eventually_conj_iff*)
with *F* **show** *U* ∩ ⋂ *B* ≠ {}
by (*intro notI*) (*simp add*: *eventually_False*)
qed
ultimately have *U* ∩ ⋂ *Z* ≠ {}
using ‹*compact U*› **unfolding** *compact_fip* **by** *blast*
then obtain *x* **where** *x* ∈ *U* **and** *x*: ⋀*z*. *z* ∈ *Z* ⟹ *x* ∈ *z*
by *auto*

have ⋀*P*. *eventually P* (*inf* (*nhds x*) *F*) ⟹ *P* ≠ *bot*
unfolding *eventually_inf eventually_nhds*
proof *safe*
fix *P Q R S*
assume *eventually R F open S x* ∈ *S*
with *open_Int_closure_eq_empty*[*of S* {*x*. *R x*}] *x*
have *S* ∩ {*x*. *R x*} ≠ {} **by** (*auto simp*: *Z_def*)
moreover assume *Ball S Q* ∀ *x*. *Q x* ∧ *R x* ⟶ *bot x*
ultimately show *False* **by** (*auto simp*: *set_eq_iff*)
qed
with ‹*x* ∈ *U*› **show** ∃ *x*∈*U*. *inf* (*nhds x*) *F* ≠ *bot*
by (*metis eventually_bot*)
next
fix *A*
assume *A*: ∀ *a*∈*A*. *closed a* ∀ *B*⊆*A*. *finite B* ⟶ *U* ∩ ⋂ *B* ≠ {} *U* ∩ ⋂ *A* = {}
define *F* **where** *F* = (*INF a*∈*insert U A*. *principal a*)
have *F* ≠ *bot*
unfolding *F_def*
proof (*rule INF_filter_not_bot*)
fix *X*

```
    assume X: X ⊆ insert U A finite X
    with A(2)[THEN spec, of X − {U}] have U ∩ ⋂(X − {U}) ≠ {}
      by auto
    with X show (INF a∈X. principal a) ≠ bot
      by (auto simp: INF_principal_finite principal_eq_bot_iff)
  qed
  moreover
  have F ≤ principal U
    unfolding F_def by auto
  then have eventually (λx. x ∈ U) F
    by (auto simp: le_filter_def eventually_principal)
  moreover
  assume ∀ F. F ≠ bot ⟶ eventually (λx. x ∈ U) F ⟶ (∃ x∈U. inf (nhds x)
F ≠ bot)
  ultimately obtain x where x ∈ U and x: inf (nhds x) F ≠ bot
    by auto

  { fix V assume V ∈ A
    then have F ≤ principal V
      unfolding F_def by (intro INF_lower2[of V]) auto
    then have V: eventually (λx. x ∈ V) F
      by (auto simp: le_filter_def eventually_principal)
    have x ∈ closure V
      unfolding closure_iff_nhds_not_empty
    proof (intro impI allI)
      fix S A
      assume open S x ∈ S S ⊆ A
      then have eventually (λx. x ∈ A) (nhds x)
        by (auto simp: eventually_nhds)
      with V have eventually (λx. x ∈ V ∩ A) (inf (nhds x) F)
        by (auto simp: eventually_inf)
      with x show V ∩ A ≠ {}
        by (auto simp del: Int_iff simp add: trivial_limit_def)
    qed
    then have x ∈ V
      using ‹V ∈ A› A(1) by simp
  }
  with ‹x∈U› have x ∈ U ∩ ⋂A by auto
  with ‹U ∩ ⋂A = {}› show False by auto
qed

definition countably_compact :: ('a::topological_space) set ⇒ bool where
countably_compact U ⟷
  (∀ A. countable A ⟶ (∀ a∈A. open a) ⟶ U ⊆ ⋃A
    ⟶ (∃ T⊆A. finite T ∧ U ⊆ ⋃T))

lemma countably_compactE:
  assumes countably_compact s and ∀ t∈C. open t and s ⊆ ⋃C countable C
  obtains C' where C' ⊆ C and finite C' and s ⊆ ⋃C'
```

using *assms* **unfolding** *countably_compact_def* **by** *metis*

lemma *countably_compactI*:
assumes $\bigwedge C.\ \forall\, t{\in}C.\ \mathit{open}\ t \Longrightarrow s \subseteq \bigcup C \Longrightarrow \mathit{countable}\ C \Longrightarrow (\exists\, C'{\subseteq}C.\ \mathit{finite}\ C' \wedge s \subseteq \bigcup C')$
shows *countably_compact s*
using *assms* **unfolding** *countably_compact_def* **by** *metis*

lemma *compact_imp_countably_compact*: *compact U* $\Longrightarrow$ *countably_compact U*
by (*auto simp*: *compact_eq_Heine_Borel countably_compact_def*)

lemma *countably_compact_imp_compact*:
assumes *countably_compact U*
and *ccover*: *countable B* $\forall\, b{\in}B.\ \mathit{open}\ b$
and *basis*: $\bigwedge T\ x.\ \mathit{open}\ T \Longrightarrow x \in T \Longrightarrow x \in U \Longrightarrow \exists\, b{\in}B.\ x \in b \wedge b \cap U \subseteq T$
shows *compact U*
using ‹*countably_compact U*›
unfolding *compact_eq_Heine_Borel countably_compact_def*
proof *safe*
fix *A*
assume *A*: $\forall\, a{\in}A.\ \mathit{open}\ a\ U \subseteq \bigcup A$
assume $*$: $\forall\, A.\ \mathit{countable}\ A \longrightarrow (\forall\, a{\in}A.\ \mathit{open}\ a) \longrightarrow U \subseteq \bigcup A \longrightarrow (\exists\, T{\subseteq}A.\ \mathit{finite}\ T \wedge U \subseteq \bigcup T)$
moreover define *C* **where** $C = \{b{\in}B.\ \exists\, a{\in}A.\ b \cap U \subseteq a\}$
ultimately have *countable C* $\forall\, a{\in}C.\ \mathit{open}\ a$
unfolding *C_def* **using** *ccover* **by** *auto*
moreover
have $\bigcup A \cap U \subseteq \bigcup C$
proof *safe*
fix *x a*
assume $x \in U\ x \in a\ a \in A$
with *basis*[*of a x*] *A* **obtain** *b* **where** $b \in B\ x \in b\ b \cap U \subseteq a$
by *blast*
with ‹$a \in A$› **show** $x \in \bigcup C$
unfolding *C_def* **by** *auto*
qed
then have $U \subseteq \bigcup C$ **using** ‹$U \subseteq \bigcup A$› **by** *auto*
ultimately obtain *T* **where** *T*: $T{\subseteq}C$ *finite T* $U \subseteq \bigcup T$
using $*$ **by** *metis*
then have $\forall\, t{\in}T.\ \exists\, a{\in}A.\ t \cap U \subseteq a$
by (*auto simp*: *C_def*)
then obtain *f* **where** $\forall\, t{\in}T.\ f\ t \in A \wedge t \cap U \subseteq f\ t$
unfolding *bchoice_iff Bex_def* **..**
with *T* **show** $\exists\, T{\subseteq}A.\ \mathit{finite}\ T \wedge U \subseteq \bigcup T$
unfolding *C_def* **by** (*intro exI*[*of* _ *f*‘*T*]) *fastforce*
qed

proposition *countably_compact_imp_compact_second_countable*:

countably_compact U ⟹ *compact* (*U* :: *'a* :: *second_countable_topology set*)
proof (*rule countably_compact_imp_compact*)
fix *T* **and** *x* :: *'a*
assume *open T x* ∈ *T*
from *topological_basisE*[*OF is_basis this*] **obtain** *b* **where**
b ∈ (*SOME B. countable B* ∧ *topological_basis B*) *x* ∈ *b b* ⊆ *T* **.**
then show ∃ *b*∈*SOME B. countable B* ∧ *topological_basis B. x* ∈ *b* ∧ *b* ∩ *U* ⊆ *T*
by *blast*
qed (*insert countable_basis topological_basis_open*[*OF is_basis*], *auto*)

lemma *countably_compact_eq_compact*:
countably_compact U ⟷ *compact* (*U* :: *'a* :: *second_countable_topology set*)
using *countably_compact_imp_compact_second_countable compact_imp_countably_compact*
by *blast*

Sequential compactness

definition *seq_compact* :: *'a*::*topological_space set* ⇒ *bool* **where**
seq_compact S ⟷
(∀ *f*. (∀ *n*. *f n* ∈ *S*) ⟶ (∃ *l*∈*S*. ∃ *r*::*nat*⇒*nat. strict_mono r* ∧ (*f* ∘ *r*) ⟶ *l*))

lemma *seq_compactI*:
assumes ⋀*f*. ∀ *n*. *f n* ∈ *S* ⟹ ∃ *l*∈*S*. ∃ *r*::*nat*⇒*nat. strict_mono r* ∧ ((*f* ∘ *r*) ⟶ *l*) *sequentially*
shows *seq_compact S*
unfolding *seq_compact_def* **using** *assms* **by** *fast*

lemma *seq_compactE*:
assumes *seq_compact S* ∀ *n*. *f n* ∈ *S*
obtains *l r* **where** *l* ∈ *S strict_mono* (*r* :: *nat* ⇒ *nat*) ((*f* ∘ *r*) ⟶ *l*) *sequentially*
using *assms* **unfolding** *seq_compact_def* **by** *fast*

lemma *closed_sequentially*:
assumes *closed S* **and** ⋀*n*. *f n* ∈ *S* **and** *f* ⟶ *l*
shows *l* ∈ *S*
by (*metis Lim_in_closed_set assms eventually_sequentially trivial_limit_sequentially*)

lemma *seq_compact_Int_closed*:
assumes *seq_compact S* **and** *closed T*
shows *seq_compact* (*S* ∩ *T*)
proof (*rule seq_compactI*)
fix *f* **assume** ∀ *n*::*nat*. *f n* ∈ *S* ∩ *T*
hence ∀ *n*. *f n* ∈ *S* **and** ∀ *n*. *f n* ∈ *T*
by *simp_all*
from ‹*seq_compact S*› **and** ‹∀ *n*. *f n* ∈ *S*›
obtain *l r* **where** *l* ∈ *S* **and** *r*: *strict_mono r* **and** *l*: (*f* ∘ *r*) ⟶ *l*

```
    by (rule seq_compactE)
  from ‹∀n. f n ∈ T› have ∀n. (f ∘ r) n ∈ T
    by simp
  with ‹l ∈ S› and r and l show ∃l∈S ∩ T. ∃r. strict_mono r ∧ (f ∘ r) ⟶
l
    by (metis Int_iff ‹closed T› closed_sequentially)
qed

lemma seq_compact_closed_subset:
  assumes closed S and S ⊆ T and seq_compact T
  shows seq_compact S
  using assms seq_compact_Int_closed [of T S] by (simp add: Int_absorb1)

lemma seq_compact_imp_countably_compact:
  fixes U :: 'a :: first_countable_topology set
  assumes seq_compact U
  shows countably_compact U
proof (safe intro!: countably_compactI)
  fix A
  assume A: ∀a∈A. open a U ⊆ ⋃A countable A
  have subseq: ⋀X. range X ⊆ U ⟹ ∃r x. x ∈ U ∧ strict_mono (r :: nat ⇒
nat) ∧ (X ∘ r) ⟶ x
    using ‹seq_compact U› by (fastforce simp: seq_compact_def subset_eq)
  show ∃T⊆A. finite T ∧ U ⊆ ⋃T
  proof cases
    assume finite A
    with A show ?thesis by auto
  next
    assume infinite A
    then have A ≠ {} by auto
    show ?thesis
    proof (rule ccontr)
      assume ¬ (∃T⊆A. finite T ∧ U ⊆ ⋃T)
      then have ∀T. ∃x. T ⊆ A ∧ finite T ⟶ (x ∈ U − ⋃T)
        by auto
      then obtain X' where T: ⋀T. T ⊆ A ⟹ finite T ⟹ X' T ∈ U − ⋃T
        by metis
      define X where X n = X' (from_nat_into A ` {.. n}) for n
      have X: ⋀n. X n ∈ U − (⋃i≤n. from_nat_into A i)
       using ‹A ≠ {}› unfolding X_def by (intro T) (auto intro: from_nat_into)
      then have range X ⊆ U
        by auto
      with subseq[of X] obtain r x where x ∈ U and r: strict_mono r (X ∘ r)
⟶ x
        by auto
      from ‹x∈U› ‹U ⊆ ⋃A› from_nat_into_surj[OF ‹countable A›]
      obtain n where x ∈ from_nat_into A n by auto
      with r(2) A(1) from_nat_into[OF ‹A ≠ {}›]
      have eventually (λi. X (r i) ∈ from_nat_into A n) sequentially
```

```
      unfolding tendsto_def by fastforce
    then obtain N where ⋀i. N ≤ i ⟹ X (r i) ∈ from_nat_into A n
      by (auto simp: eventually_sequentially)
    moreover from X have ⋀i. n ≤ r i ⟹ X (r i) ∉ from_nat_into A n
      by auto
    moreover from ‹strict_mono r›[THEN seq_suble, of max n N] have ∃i. n
≤ r i ∧ N ≤ i
      by (auto intro!: exI[of _ max n N])
    ultimately show False
      by auto
  qed
 qed
qed

lemma compact_imp_seq_compact:
 fixes U :: 'a :: first_countable_topology set
 assumes compact U
 shows seq_compact U
 unfolding seq_compact_def
proof safe
 fix X :: nat ⇒ 'a
 assume ∀n. X n ∈ U
 then have eventually (λx. x ∈ U) (filtermap X sequentially)
  by (auto simp: eventually_filtermap)
 moreover
 have filtermap X sequentially ≠ bot
  by (simp add: trivial_limit_def eventually_filtermap)
 ultimately
 obtain x where x ∈ U and x: inf (nhds x) (filtermap X sequentially) ≠ bot (is
?F ≠ _)
  using ‹compact U› by (auto simp: compact_filter)

 from countable_basis_at_decseq[of x]
 obtain A where A:
    ⋀i. open (A i)
    ⋀i. x ∈ A i
    ⋀S. open S ⟹ x ∈ S ⟹ eventually (λi. A i ⊆ S) sequentially
  by blast
 define s where s n i = (SOME j. i < j ∧ X j ∈ A (Suc n)) for n i
 {
  fix n i
  have ∃a. i < a ∧ X a ∈ A (Suc n)
  proof (rule ccontr)
   assume ¬ (∃a>i. X a ∈ A (Suc n))
   then have ⋀a. Suc i ≤ a ⟹ X a ∉ A (Suc n)
    by auto
   then have eventually (λx. x ∉ A (Suc n)) (filtermap X sequentially)
    by (auto simp: eventually_filtermap eventually_sequentially)
   moreover have eventually (λx. x ∈ A (Suc n)) (nhds x)
```

```
      using A(1,2)[of Suc n] by (auto simp: eventually_nhds)
    ultimately have eventually (λx. False) ?F
      by (auto simp: eventually_inf)
    with x show False
      by (simp add: eventually_False)
  qed
  then have i < s n i X (s n i) ∈ A (Suc n)
    unfolding s_def by (auto intro: someI2_ex)
 }
 note s = this
 define r where r = rec_nat (s 0 0) s
 have strict_mono r
  by (auto simp: r_def s strict_mono_Suc_iff)
 moreover
 have (λn. X (r n)) ⟶ x
 proof (rule topological_tendstoI)
  fix S
  assume open S x ∈ S
  with A(3) have eventually (λi. A i ⊆ S) sequentially
    by auto
  moreover
  {
    fix i
    assume Suc 0 ≤ i
    then have X (r i) ∈ A i
      by (cases i) (simp_all add: r_def s)
  }
  then have eventually (λi. X (r i) ∈ A i) sequentially
    by (auto simp: eventually_sequentially)
  ultimately show eventually (λi. X (r i) ∈ S) sequentially
    by eventually_elim auto
 qed
 ultimately show ∃x∈U. ∃r. strict_mono r ∧ (X ∘ r) ⟶ x
  using ‹x ∈ U› by (auto simp: convergent_def comp_def)
qed

lemma countably_compact_imp_acc_point:
 assumes countably_compact S
  and countable T
  and infinite T
  and T ⊆ S
 shows ∃x∈S. ∀U. x∈U ∧ open U ⟶ infinite (U ∩ T)
proof (rule ccontr)
 define C where C = (λF. interior (F ∪ (- T))) ` {F. finite F ∧ F ⊆ T }
 note ‹countably_compact S›
 moreover have ∀T∈C. open T
  by (auto simp: C_def)
 moreover
 assume ¬ (∃x∈S. ∀U. x∈U ∧ open U ⟶ infinite (U ∩ T))
```

```
  then have S: ⋀x. x ∈ S ⟹ ∃ U. x∈U ∧ open U ∧ finite (U ∩ T) by metis
  have S ⊆ ⋃ C
    using ‹T ⊆ S›
    unfolding C_def
    apply (safe dest!: S)
    apply (rule_tac a=U ∩ T in UN_I)
    apply (auto intro!: interiorI simp add: finite_subset)
    done
  moreover
  from ‹countable T› have countable C
    unfolding C_def by (auto intro: countable_Collect_finite_subset)
  ultimately
  obtain D where D ⊆ C finite D S ⊆ ⋃ D
    by (rule countably_compactE)
  then obtain E where E: E ⊆ {F. finite F ∧ F ⊆ T } finite E
    and S: S ⊆ (⋃ F∈E. interior (F ∪ (- T)))
    by (metis (lifting) finite_subset_image C_def)
  from S ‹T ⊆ S› have T ⊆ ⋃ E
    using interior_subset by blast
  moreover have finite (⋃ E)
    using E by auto
  ultimately show False using ‹infinite T›
    by (auto simp: finite_subset)
qed

lemma countable_acc_point_imp_seq_compact:
  fixes S :: 'a::first_countable_topology set
  assumes ⋀T. ⟦infinite T; countable T; T ⊆ S⟧ ⟹ ∃ x∈S. ∀ U. x∈U ∧ open U
⟶ infinite (U ∩ T)
  shows seq_compact S
  unfolding seq_compact_def
proof (intro strip)
  fix f :: nat ⇒ 'a
  assume f: ∀ n. f n ∈ S
  show ∃ l∈S. ∃ r. strict_mono r ∧ ((f ∘ r) ⟶⟶ l) sequentially
  proof (cases finite (range f))
    case True
    obtain l where infinite {n. f n = f l}
      using pigeonhole_infinite[OF _ True] by auto
    then obtain r :: nat ⇒ nat where strict_mono  r and fr: ∀ n. f (r n) = f l
      using infinite_enumerate by blast
    then have strict_mono r ∧ (f ∘ r) ⟶⟶ f l
      by (simp add: fr o_def)
    with f show ∃ l∈S. ∃ r. strict_mono  r ∧ (f ∘ r) ⟶⟶ l
      by auto
  next
    case False
    with f assms obtain l where l ∈ S ∀ U. l∈U ∧ open U ⟶ infinite (U ∩
range f)
```

by (*metis image_subset_iff uncountable_def*)
with ‹$l \in S$› **show** $\exists\, l{\in}S.\ \exists\, r.$ *strict_mono* $r \wedge ((f \circ r) \longrightarrow l)$ *sequentially*
by (*meson acc_point_range_imp_convergent_subsequence*)
qed
qed

lemma *seq_compact_eq_countably_compact*:
fixes U :: $'a$ :: *first_countable_topology set*
shows *seq_compact* $U \longleftrightarrow$ *countably_compact* U
by (*metis countable_acc_point_imp_seq_compact countably_compact_imp_acc_point seq_compact_imp_countably_compact*)

lemma *seq_compact_eq_acc_point*:
fixes S :: $'a$ :: *first_countable_topology set*
shows *seq_compact* $S \longleftrightarrow$
$(\forall\, T.$ *infinite* $T \wedge$ *countable* $T \wedge T \subseteq S \longrightarrow (\exists\, x{\in}S.\ \forall\, U.\ x{\in}U \wedge$ *open* $U \longrightarrow$ *infinite* $(U \cap T)))$
by (*metis countable_acc_point_imp_seq_compact countably_compact_imp_acc_point seq_compact_imp_countably_compact*)

lemma *seq_compact_eq_compact*:
fixes U :: $'a$ :: *second_countable_topology set*
shows *seq_compact* $U \longleftrightarrow$ *compact* U
using *seq_compact_eq_countably_compact countably_compact_eq_compact* **by** *blast*

proposition *Bolzano_Weierstrass_imp_seq_compact*:
fixes S :: $'a$::{*t1_space*, *first_countable_topology*} *set*
shows $(\bigwedge T.$ ⟦*infinite* T; $T \subseteq S$⟧ $\Longrightarrow \exists\, x \in S.$ x *islimpt* $T) \Longrightarrow$ *seq_compact* S
by (*rule countable_acc_point_imp_seq_compact*) (*metis islimpt_eq_acc_point*)

2.1.11 Cartesian products

lemma *seq_compact_Times*:
assumes *seq_compact* S *seq_compact* T
shows *seq_compact* $(S \times T)$
unfolding *seq_compact_def*
proof *clarify*
fix h :: *nat* $\Rightarrow$ $'a \times 'b$
assume $\forall\, n.$ $h\ n \in S \times T$
then have $*$: $\bigwedge n.$ $(fst \circ h)\ n \in S$ $\bigwedge n.$ $(snd \circ h)\ n \in T$
by (*simp_all add*: *mem_Times_iff*)
then obtain lS **and** rS :: *nat*⇒*nat*
where $lS{\in}S$ *strict_mono* rS **and** lS: $(fst \circ h \circ rS) \longrightarrow lS$
using *assms seq_compact_def* **by** *metis*
then obtain lT **and** rT :: *nat*⇒*nat*
where $lT{\in}T$ *strict_mono* rT **and** lT: $(snd \circ h \circ rS \circ rT) \longrightarrow lT$
using *assms seq_compact_def* $*$
by (*metis* (*mono_tags*, *lifting*) *comp_apply*)

```
  have strict_mono (rS ∘ rT)
    by (simp add: ‹strict_mono rS› ‹strict_mono rT› strict_mono_o)
  moreover have (h ∘ (rS ∘ rT)) ⟶ (lS,lT)
    using tendsto_Pair [OF LIMSEQ_subseq_LIMSEQ [OF lS ‹strict_mono rT›]
lT]
    by (simp add: o_def)
  ultimately show ∃ l∈S × T. ∃ r. strict_mono r ∧ (h ∘ r) ⟶ l
    using ‹lS ∈ S› ‹lT ∈ T› by blast
qed

lemma compact_Times:
  assumes compact S compact T
  shows compact (S × T)
proof (rule compactI)
  fix 𝒞
  assume C: ∀ T∈𝒞. open T S × T ⊆ ⋃𝒞
  have ∀ x∈S. ∃ A. open A ∧ x ∈ A ∧ (∃ D⊆𝒞. finite D ∧ A × T ⊆ ⋃ D)
  proof
    fix x
    assume x ∈ S
    have ∀ y∈T. ∃ A B C. C ∈ 𝒞 ∧ open A ∧ open B ∧ x ∈ A ∧ y ∈ B ∧ A × B
⊆ C
    by (smt (verit, ccfv_threshold) C UnionE ‹x ∈ S› mem_Sigma_iff open_prod_def
subsetD)
    then obtain a b c where b: ⋀y. y ∈ T ⟹ open (b y)
      and c: ⋀y. y ∈ T ⟹ c y ∈ 𝒞 ∧ open (a y) ∧ open (b y) ∧ x ∈ a y ∧ y ∈ b
y ∧ a y × b y ⊆ c y
      by metis
    then have ∀ y∈T. open (b y) T ⊆ (⋃ y∈T. b y) by auto
    with compactE_image[OF ‹compact T›] obtain D where D: D ⊆ T finite D
T ⊆ (⋃ y∈D. b y)
      by metis
    moreover from D c have (⋂ y∈D. a y) × T ⊆ (⋃ y∈D. c y)
      by (fastforce simp: subset_eq)
    ultimately show ∃ a. open a ∧ x ∈ a ∧ (∃ d⊆𝒞. finite d ∧ a × T ⊆ ⋃ d)
        using c by (intro exI[of _ c‘D] exI[of _ ⋂(a‘D)] conjI) (auto intro!:
open_INT)
  qed
  then obtain a d where a: ⋀x. x∈S ⟹ open (a x) S ⊆ (⋃ x∈S. a x)
    and d: ⋀x. x ∈ S ⟹ d x ⊆ 𝒞 ∧ finite (d x) ∧ a x × T ⊆ ⋃(d x)
    unfolding subset_eq UN_iff by metis
  moreover
  from compactE_image[OF ‹compact S› a]
  obtain e where e: e ⊆ S finite e and S: S ⊆ (⋃ x∈e. a x)
    by auto
  moreover
  have S × T ⊆ (⋃ x∈e. ⋃(d x))
    by (smt (verit, del_insts) S SigmaE UN_iff d e(1) mem_Sigma_iff subset_eq)
  ultimately show ∃ C'⊆𝒞. finite C' ∧ S × T ⊆ ⋃ C'
```

```
    by (intro exI[of _ (⋃x∈e. d x)]) (auto simp: subset_eq)
qed


lemma tube_lemma:
  assumes compact K
  assumes open W
  assumes {x0} × K ⊆ W
  shows ∃X0. x0 ∈ X0 ∧ open X0 ∧ X0 × K ⊆ W
proof −
  {
    fix y assume y ∈ K
    then have (x0, y) ∈ W using assms by auto
    with ‹open W› 
    have ∃X0 Y. open X0 ∧ open Y ∧ x0 ∈ X0 ∧ y ∈ Y ∧ X0 × Y ⊆ W
      by (rule open_prod_elim) blast
  }
  then obtain X0 Y where
    ∗: ∀y ∈ K. open (X0 y) ∧ open (Y y) ∧ x0 ∈ X0 y ∧ y ∈ Y y ∧ X0 y × Y y
⊆ W
    by metis
  from ∗ have ∀t∈Y ‘ K. open t K ⊆ ⋃(Y ‘ K) by auto
  with ‹compact K› obtain CC where CC: CC ⊆ Y ‘ K finite CC K ⊆ ⋃CC
    by (meson compactE)
  then obtain c where c: ⋀C. C ∈ CC ⟹ c C ∈ K ∧ C = Y (c C)
    by (force intro!: choice)
  with ∗ CC show ?thesis
    by (force intro!: exI[where x=⋂C∈CC. X0 (c C)])
qed

lemma continuous_on_prod_compactE:
  fixes fx::'a::topological_space × 'b::topological_space ⇒ 'c::metric_space
    and e::real
  assumes cont_fx: continuous_on (U × C) fx
  assumes compact C
  assumes [intro]: x0 ∈ U
  notes [continuous_intros] = continuous_on_compose2[OF cont_fx]
  assumes e > 0
  obtains X0 where x0 ∈ X0 open X0
    ∀x∈X0 ∩ U. ∀t ∈ C. dist (fx (x, t)) (fx (x0, t)) ≤ e
proof −
  define psi where psi = (λ(x, t). dist (fx (x, t)) (fx (x0, t)))
  define W0 where W0 = {(x, t) ∈ U × C. psi (x, t) < e}
  have W0_eq: W0 = psi −‘ {..<e} ∩ U × C
    by (auto simp: vimage_def W0_def)
  have open {..<e} by simp
  have continuous_on (U × C) psi
    by (auto intro!: continuous_intros simp: psi_def split_beta')
  then obtain W where W: open W W ∩ U × C = W0 ∩ U × C
```

unfolding *W0_eq*
by (*metis* ‹*open* {..<*e*}› *continuous_on_open_invariant inf_right_idem*)
have {*x0*} × *C* ⊆ *W* ∩ *U* × *C*
unfolding *W*
by (*auto simp*: *W0_def psi_def* ‹*0* < *e*›)
then have {*x0*} × *C* ⊆ *W* **by** *blast*
from *tube_lemma*[*OF* ‹*compact C*› ‹*open W*› *this*]
obtain *X0* **where** *X0*: *x0* ∈ *X0 open X0 X0* × *C* ⊆ *W*
by *blast*

have ∀ *x*∈*X0* ∩ *U*. ∀ *t* ∈ *C*. *dist* (*fx* (*x*, *t*)) (*fx* (*x0*, *t*)) ≤ *e*
proof *safe*
fix *x* **assume** *x*: *x* ∈ *X0 x* ∈ *U*
fix *t* **assume** *t*: *t* ∈ *C*
have *dist* (*fx* (*x*, *t*)) (*fx* (*x0*, *t*)) = *psi* (*x*, *t*)
by (*auto simp*: *psi_def*)
also have *psi* (*x*, *t*) < *e*
using *W*(*2*) *W0_def X0*(*3*) *t x* **by** *fastforce*
finally show *dist* (*fx* (*x*,*t*)) (*fx* (*x0*,*t*)) ≤ *e* **by** *simp*
qed
from *X0*(*1*,*2*) *this* **show** *?thesis* **..**
qed

2.1.12 Continuity

lemma *continuous_at_imp_continuous_within*:
continuous (*at x*) *f* ⟹ *continuous* (*at x within s*) *f*
unfolding *continuous_within continuous_at* **using** *Lim_at_imp_Lim_at_within*
by *auto*

lemma *Lim_trivial_limit*: *trivial_limit net* ⟹ (*f* ⟶ *l*) *net*
by *simp*

lemmas *continuous_on* = *continuous_on_def* — legacy theorem name

lemma *continuous_within_subset*:
continuous (*at x within S*) *f* ⟹ *t* ⊆ *S* ⟹ *continuous* (*at x within t*) *f*
unfolding *continuous_within* **by**(*metis tendsto_within_subset*)

lemma *continuous_on_interior*:
continuous_on S f ⟹ *x* ∈ *interior S* ⟹ *continuous* (*at x*) *f*
by (*metis continuous_on_eq_continuous_at continuous_on_subset interiorE*)

lemma *continuous_on_eq*:
⟦*continuous_on S f*; ⋀*x*. *x* ∈ *S* ⟹ *f x* = *g x*⟧ ⟹ *continuous_on S g*
unfolding *continuous_on_def tendsto_def eventually_at_topological*
by *simp*

Characterization of various kinds of continuity in terms of sequences.

lemma *continuous_within_sequentiallyI*:
fixes f :: *'a*::{*first_countable_topology*, *t2_space*} ⇒ *'b*::*topological_space*
assumes ⋀*u*::*nat* ⇒ *'a*. *u* ⟶ *a* ⟹ (∀ *n*. *u n* ∈ *S*) ⟹ (λ*n*. *f* (*u n*)) ⟶ *f a*
shows *continuous* (*at a within S*) *f*
using *assms* **unfolding** *continuous_within tendsto_def*[**where** *l* = *f a*]
by (*auto intro*!: *sequentially_imp_eventually_within*)

lemma *continuous_within_tendsto_compose*:
fixes *f*::*'a*::*t2_space* ⇒ *'b*::*topological_space*
assumes *f*: *continuous* (*at a within S*) *f*
and *eventually* (λ*n*. *x n* ∈ *S*) *F* (*x* ⟶ *a*) *F*
shows ((λ*n*. *f* (*x n*)) ⟶ *f a*) *F*
proof −
have ∗: *filterlim x* (*inf* (*nhds a*) (*principal S*)) *F*
by (*simp add*: *assms filterlim_inf filterlim_principal*)
show *?thesis*
using ∗ *f continuous_within filterlim_compose tendsto_at_within_iff_tendsto_nhds*
by *blast*
qed

lemma *continuous_within_tendsto_compose'*:
fixes *f*::*'a*::*t2_space* ⇒ *'b*::*topological_space*
assumes *continuous* (*at a within S*) *f*
⋀*n*. *x n* ∈ *S*
(*x* ⟶ *a*) *F*
shows ((λ*n*. *f* (*x n*)) ⟶ *f a*) *F*
using *always_eventually assms continuous_within_tendsto_compose* **by** *blast*

lemma *continuous_within_sequentially*:
fixes *f* :: *'a*::{*first_countable_topology*, *t2_space*} ⇒ *'b*::*topological_space*
shows *continuous* (*at a within S*) *f* ⟷
(∀ *x*. (∀ *n*::*nat*. *x n* ∈ *S*) ∧ (*x* ⟶ *a*) *sequentially*
⟶ ((*f* ∘ *x*) ⟶ *f a*) *sequentially*)
using *continuous_within_tendsto_compose'*[*of a S f _ sequentially*]
continuous_within_sequentiallyI[*of a S f*]
by (*auto simp*: *o_def*)

lemma *continuous_at_sequentiallyI*:
fixes *f* :: *'a*::{*first_countable_topology*, *t2_space*} ⇒ *'b*::*topological_space*
assumes ⋀*u*. *u* ⟶ *a* ⟹ (λ*n*. *f* (*u n*)) ⟶ *f a*
shows *continuous* (*at a*) *f*
using *continuous_within_sequentiallyI*[*of a UNIV f*] *assms* **by** *auto*

lemma *continuous_at_sequentially*:
fixes *f* :: *'a*::*metric_space* ⇒ *'b*::*topological_space*
shows *continuous* (*at a*) *f* ⟷
(∀ *x*. (*x* ⟶ *a*) *sequentially* −−> ((*f* ∘ *x*) ⟶ *f a*) *sequentially*)
using *continuous_within_sequentially*[*of a UNIV f*] **by** *simp*

lemma *continuous_on_sequentiallyI*:
fixes *f* :: *'a*::{*first_countable_topology*, *t2_space*} ⇒ *'b*::*topological_space*
assumes ⋀*u a*. (∀ *n*. *u n* ∈ *S*) ⟹ *a* ∈ *S* ⟹ *u* ⟶ *a* ⟹ (λ*n*. *f* (*u n*)) ⟶ *f a*
shows *continuous_on S f*
using *assms* **unfolding** *continuous_on_eq_continuous_within*
using *continuous_within_sequentiallyI*[*of* _ *S f*] **by** *auto*

lemma *continuous_on_sequentially*:
fixes *f* :: *'a*::{*first_countable_topology*, *t2_space*} ⇒ *'b*::*topological_space*
shows *continuous_on S f* ⟷
(∀ *x*. ∀ *a* ∈ *S*. (∀ *n*. *x*(*n*) ∈ *S*) ∧ (*x* ⟶ *a*) *sequentially*
⟶ ((*f* ∘ *x*) ⟶ *f a*) *sequentially*)
by (*meson continuous_on_eq_continuous_within continuous_within_sequentially*)

Continuity in terms of open preimages.

lemma *continuous_at_open*:
continuous (*at x*) *f* ⟷ (∀ *t*. *open t* ∧ *f x* ∈ *t* ⟶ (∃ *S*. *open S* ∧ *x* ∈ *S* ∧ (∀ *x'* ∈ *S*. (*f x'*) ∈ *t*)))
by (*metis UNIV_I continuous_within_topological*)

lemma *continuous_imp_tendsto*:
assumes *continuous* (*at x0*) *f* **and** *x* ⟶ *x0*
shows (*f* ∘ *x*) ⟶ (*f x0*)
proof (*rule topological_tendstoI*)
fix *S*
assume *open S f x0* ∈ *S*
then obtain *T* **where** *T_def*: *open T x0* ∈ *T* ∀ *x*∈*T*. *f x* ∈ *S*
using *assms continuous_at_open* **by** *metis*
then have *eventually* (λ*n*. *x n* ∈ *T*) *sequentially*
using *assms T_def* **by** (*auto simp*: *tendsto_def*)
then show *eventually* (λ*n*. (*f* ∘ *x*) *n* ∈ *S*) *sequentially*
using *T_def* **by** (*auto elim*!: *eventually_mono*)
qed

2.1.13 Homeomorphisms

definition *homeomorphism S T f g* ⟷
(∀ *x*∈*S*. (*g*(*f x*) = *x*)) ∧ (*f* ‘ *S* = *T*) ∧ *continuous_on S f* ∧
(∀ *y*∈*T*. (*f*(*g y*) = *y*)) ∧ (*g* ‘ *T* = *S*) ∧ *continuous_on T g*

lemma *homeomorphismI* [*intro?*]:
assumes *continuous_on S f continuous_on T g*
f ‘ *S* ⊆ *T g* ‘ *T* ⊆ *S* ⋀*x*. *x* ∈ *S* ⟹ *g*(*f x*) = *x* ⋀*y*. *y* ∈ *T* ⟹ *f*(*g y*) = *y*
shows *homeomorphism S T f g*
using *assms* **by** (*force simp*: *homeomorphism_def*)

lemma *homeomorphism_translation*:

fixes $a :: 'a :: real\_normed\_vector$
shows *homeomorphism* $((+)\ a\ `\ S)\ S\ ((+)\ (-\ a))\ ((+)\ a)$
unfolding *homeomorphism_def* **by** (*auto simp*: *algebra_simps continuous_intros*)

lemma *homeomorphism_ident*: *homeomorphism* $T\ T\ (\lambda a.\ a)\ (\lambda a.\ a)$
by (*rule homeomorphismI*) *auto*

lemma *homeomorphism_compose*:
assumes *homeomorphism* $S\ T\ f\ g$ *homeomorphism* $T\ U\ h\ k$
shows *homeomorphism* $S\ U\ (h\ o\ f)\ (g\ o\ k)$
using *assms*
unfolding *homeomorphism_def*
by (*intro conjI ballI continuous_on_compose*) (*auto simp*: *image_iff*)

lemma *homeomorphism_cong*:
homeomorphism $X'\ Y'\ f'\ g'$
if *homeomorphism* $X\ Y\ f\ g\ X' = X\ Y' = Y\ \bigwedge x.\ x \in X \Longrightarrow f'\ x = f\ x\ \bigwedge y.\ y \in Y \Longrightarrow g'\ y = g\ y$
using *that* **by** (*auto simp add*: *homeomorphism_def*)

lemma *homeomorphism_empty* [*simp*]:
homeomorphism $\{\}\ \{\}\ f\ g$
unfolding *homeomorphism_def* **by** *auto*

lemma *homeomorphism_symD*: *homeomorphism* $S\ t\ f\ g \Longrightarrow$ *homeomorphism* $t\ S\ g\ f$
by (*simp add*: *homeomorphism_def*)

lemma *homeomorphism_sym*: *homeomorphism* $S\ t\ f\ g =$ *homeomorphism* $t\ S\ g\ f$
by (*force simp*: *homeomorphism_def*)

lemma *continuous_on_translation_eq*:
fixes $g :: 'a :: real\_normed\_vector \Rightarrow 'b :: real\_normed\_vector$
shows *continuous_on* $A\ ((+)\ a \circ g) =$ *continuous_on* $A\ g$
proof −
have *g*: $g = (\lambda x.\ -a + x) \circ ((\lambda x.\ a + x) \circ g)$
by (*rule ext*) *simp*
show *?thesis*
by (*metis* (*no_types*, *opaque_lifting*) *g continuous_on_compose homeomorphism_def homeomorphism_translation*)
qed

definition *homeomorphic* :: *'a::topological_space set* ⇒ *'b::topological_space set* ⇒ *bool*
(**infixr** *homeomorphic 60*)
where *s homeomorphic t* ≡ $(\exists f\ g.$ *homeomorphism s t f g*$)$

lemma *homeomorphic_empty* [*iff*]:
S homeomorphic $\{\} \longleftrightarrow S = \{\}$ $\{\}$ *homeomorphic* $S \longleftrightarrow S = \{\}$

by (*auto simp*: *homeomorphic_def homeomorphism_def*)

lemma *homeomorphic_refl*: *S homeomorphic S*
using *homeomorphic_def homeomorphism_ident* **by** *fastforce*

lemma *homeomorphic_sym*: *S homeomorphic T* $\longleftrightarrow$ *T homeomorphic S*
unfolding *homeomorphic_def homeomorphism_def*
by *blast*

lemma *homeomorphic_trans* [*trans*]:
assumes *S homeomorphic T* **and** *T homeomorphic U*
shows *S homeomorphic U*
using *assms* **unfolding** *homeomorphic_def*
by (*metis homeomorphism_compose*)

lemma *homeomorphic_minimal*:
S homeomorphic T $\longleftrightarrow$
$(\exists f\ g.\ (\forall\, x{\in}S.\ f(x) \in T \wedge (g(f(x)) = x)) \wedge$
$(\forall\, y{\in}T.\ g(y) \in S \wedge (f(g(y)) = y)) \wedge$
continuous_on S f $\wedge$ *continuous_on T g*)
by (*smt* (*verit*, *ccfv_threshold*) *homeomorphic_def homeomorphismI homeomorphism_def image_eqI image_subset_iff*)

lemma *homeomorphicI* [*intro?*]:
$[\![f\ `\ S = T;\ g\ `\ T = S;$
continuous_on S f; *continuous_on T g*;
$\bigwedge x.\ x \in S \Longrightarrow g(f(x)) = x;$
$\bigwedge y.\ y \in T \Longrightarrow f(g(y)) = y]\!] \Longrightarrow$ *S homeomorphic T*
unfolding *homeomorphic_def homeomorphism_def* **by** *metis*

lemma *homeomorphism_of_subsets*:
$[\![$*homeomorphism S T f g*; $S' \subseteq S$; $T'' \subseteq T$; $f\ `\ S' = T']\!]$
$\Longrightarrow$ *homeomorphism S' T' f g*
by (*smt* (*verit*, *del_insts*) *continuous_on_subset homeomorphismI homeomorphism_def imageE subset_eq*)

lemma *homeomorphism_apply1*: $[\![$*homeomorphism S T f g*; $x \in S]\!] \Longrightarrow g(f\ x) = x$
by (*simp add*: *homeomorphism_def*)

lemma *homeomorphism_apply2*: $[\![$*homeomorphism S T f g*; $x \in T]\!] \Longrightarrow f(g\ x) = x$
by (*simp add*: *homeomorphism_def*)

lemma *homeomorphism_image1*: *homeomorphism S T f g* $\Longrightarrow$ $f\ `\ S = T$
by (*simp add*: *homeomorphism_def*)

lemma *homeomorphism_image2*: *homeomorphism S T f g* $\Longrightarrow$ $g\ `\ T = S$
by (*simp add*: *homeomorphism_def*)

lemma *homeomorphism_cont1*: *homeomorphism S T f g* ⟹ *continuous_on S f*
by (*simp add*: *homeomorphism_def*)

lemma *homeomorphism_cont2*: *homeomorphism S T f g* ⟹ *continuous_on T g*
by (*simp add*: *homeomorphism_def*)

lemma *continuous_on_no_limpt*:
(⋀*x*. ¬ *x islimpt S*) ⟹ *continuous_on S f*
unfolding *continuous_on_def*
by (*metis UNIV_I empty_iff eventually_at_topological islimptE open_UNIV tendsto_def trivial_limit_within*)

lemma *continuous_on_finite*:
fixes *S* :: *'a*::*t1_space set*
shows *finite S* ⟹ *continuous_on S f*
by (*metis continuous_on_no_limpt islimpt_finite*)

lemma *homeomorphic_finite*:
fixes *S* :: *'a*::*t1_space set* **and** *T* :: *'b*::*t1_space set*
assumes *finite T*
shows *S homeomorphic T* ⟷ *finite S* ∧ *finite T* ∧ *card S* = *card T* (**is** *?lhs* = *?rhs*)
proof
assume *S homeomorphic T*
with *assms* **show** *?rhs*
by (*metis* (*full_types*) *card_image_le finite_imageI homeomorphic_def homeomorphism_def le_antisym*)
next
assume *R*: *?rhs*
with *finite_same_card_bij* **obtain** *h* **where** *bij_betw h S T*
by *auto*
with *R* **show** *?lhs*
apply (*simp only*: *homeomorphic_def homeomorphism_def continuous_on_finite*)
by (*smt* (*verit*, *ccfv_SIG*) *bij_betw_imp_surj_on bij_betw_inv_into bij_betw_inv_into_left bij_betw_inv_into_right*)
qed

Relatively weak hypotheses if a set is compact.

lemma *homeomorphism_compact*:
fixes *f* :: *'a*::*topological_space* ⇒ *'b*::*t2_space*
assumes *compact S continuous_on S f f* ‘ *S* = *T inj_on f S*
shows ∃ *g*. *homeomorphism S T f g*
proof −
obtain *g* **where** *g*: ∀ *x*∈*S*. *g* (*f x*) = *x* ∀ *x*∈*T*. *f* (*g x*) = *x g* ‘ *T* = *S*
using *assms the_inv_into_f_f* **by** *fastforce*
with *assms* **show** *?thesis*
unfolding *homeomorphism_def homeomorphic_def* **by** (*metis continuous_on_inv*)
qed

lemma *homeomorphic_compact*:
fixes *f* :: *'a*::*topological_space* ⇒ *'b*::*t2_space*
shows *compact S* ⟹ *continuous_on S f* ⟹ (*f* ‘ *S* = *T*) ⟹ *inj_on f S* ⟹ *S homeomorphic T*
unfolding *homeomorphic_def* **by** (*metis homeomorphism_compact*)

Preservation of topological properties.

lemma *homeomorphic_compactness*: *S homeomorphic T* ⟹ (*compact S* ⟷ *compact T*)
unfolding *homeomorphic_def homeomorphism_def*
by (*metis compact_continuous_image*)

2.1.14 On Linorder Topologies

lemma *islimpt_greaterThanLessThan1*:
fixes *a b*::*'a*::{*linorder_topology*, *dense_order*}
assumes *a* < *b*
shows *a islimpt* {*a*<..<*b*}
proof (*rule islimptI*)
fix *T*
assume *open T a* ∈ *T*
then obtain *c* **where** *c*: *a* < *c* {*a*..<*c*} ⊆ *T*
by (*meson assms open_right*)
with *assms dense*[*of a min c b*]
show ∃ *y*∈{*a*<..<*b*}. *y* ∈ *T* ∧ *y* ≠ *a*
by (*metis atLeastLessThan_iff greaterThanLessThan_iff min_less_iff_conj not_le order.strict_implies_order subset_eq*)
qed

lemma *islimpt_greaterThanLessThan2*:
fixes *a b*::*'a*::{*linorder_topology*, *dense_order*}
assumes *a* < *b*
shows *b islimpt* {*a*<..<*b*}
proof (*rule islimptI*)
fix *T*
assume *open T b* ∈ *T*
from *open_left*[*OF this* ‹*a* < *b*›]
obtain *c* **where** *c*: *c* < *b* {*c*<..*b*} ⊆ *T* **by** *auto*
with *assms dense*[*of max a c b*]
show ∃ *y*∈{*a*<..<*b*}. *y* ∈ *T* ∧ *y* ≠ *b*
by (*metis greaterThanAtMost_iff greaterThanLessThan_iff max_less_iff_conj not_le order.strict_implies_order subset_eq*)
qed

lemma *closure_greaterThanLessThan*[*simp*]:
fixes *a b*::*'a*::{*linorder_topology*, *dense_order*}
shows *a* < *b* ⟹ *closure* {*a* <..< *b*} = {*a* .. *b*} (**is** _ ⟹ *?l* = *?r*)
proof
have *?l* ⊆ *closure ?r*

```
    by (rule closure_mono) auto
  thus closure {a<..<b} ⊆ {a..b} by simp
qed (auto simp: closure_def order.order_iff_strict islimpt_greaterThanLessThan1
  islimpt_greaterThanLessThan2)

lemma closure_greaterThan[simp]:
  fixes a b::'a::{no_top, linorder_topology, dense_order}
  shows closure {a<..} = {a..}
proof −
  from gt_ex obtain b where a < b by auto
  hence {a<..} = {a<..<b} ∪ {b..} by auto
  also have closure ... = {a..} using ‹a < b› unfolding closure_Un
    by auto
  finally show ?thesis .
qed

lemma closure_lessThan[simp]:
  fixes b::'a::{no_bot, linorder_topology, dense_order}
  shows closure {..<b} = {..b}
proof −
  from lt_ex obtain a where a < b by auto
  hence {..<b} = {a<..<b} ∪ {..a} by auto
  also have closure ... = {..b} using ‹a < b› unfolding closure_Un
    by auto
  finally show ?thesis .
qed

lemma closure_atLeastLessThan[simp]:
  fixes a b::'a::{linorder_topology, dense_order}
  assumes a < b
  shows closure {a ..< b} = {a .. b}
proof −
  from assms have {a ..< b} = {a} ∪ {a <..< b} by auto
  also have closure ... = {a .. b} unfolding closure_Un
    by (auto simp: assms less_imp_le)
  finally show ?thesis .
qed

lemma closure_greaterThanAtMost[simp]:
  fixes a b::'a::{linorder_topology, dense_order}
  assumes a < b
  shows closure {a <.. b} = {a .. b}
proof −
  from assms have {a <.. b} = {b} ∪ {a <..< b} by auto
  also have closure ... = {a .. b} unfolding closure_Un
    by (auto simp: assms less_imp_le)
  finally show ?thesis .
qed
```

end
theory *Abstract_Limits*
imports
Abstract_Topology
begin

2.1.15 nhdsin and atin

definition *nhdsin* :: *'a topology* ⇒ *'a* ⇒ *'a filter*
where *nhdsin X a* =
(*if a* ∈ *topspace X then* (*INF S*∈{*S*. *openin X S* ∧ *a* ∈ *S*}. *principal S*)
else bot)

definition *atin* :: *'a topology* ⇒ *'a* ⇒ *'a filter*
where *atin X a* ≡ *inf* (*nhdsin X a*) (*principal* (*topspace X* − {*a*}))

lemma *nhdsin_degenerate* [*simp*]: *a* ∉ *topspace X* ⟹ *nhdsin X a* = *bot*
and *atin_degenerate* [*simp*]: *a* ∉ *topspace X* ⟹ *atin X a* = *bot*
by (*simp_all add*: *nhdsin_def atin_def*)

lemma *eventually_nhdsin*:
eventually P (*nhdsin X a*) ⟷ *a* ∉ *topspace X* ∨ (∃ *S*. *openin X S* ∧ *a* ∈ *S* ∧
(∀ *x*∈*S*. *P x*))
proof (*cases a* ∈ *topspace X*)
case *True*
hence *nhdsin X a* = (*INF S*∈{*S*. *openin X S* ∧ *a* ∈ *S*}. *principal S*)
by (*simp add*: *nhdsin_def*)
also have *eventually P* ... ⟷ (∃ *S*. *openin X S* ∧ *a* ∈ *S* ∧ (∀ *x*∈*S*. *P x*))
using *True* **by** (*subst eventually_INF_base*) (*auto simp*: *eventually_principal*)
finally show *?thesis* **using** *True* **by** *simp*
qed *auto*

lemma *eventually_atin*:
eventually P (*atin X a*) ⟷ *a* ∉ *topspace X* ∨
(∃ *U*. *openin X U* ∧ *a* ∈ *U* ∧ (∀ *x* ∈ *U* − {*a*}. *P x*))
proof (*cases a* ∈ *topspace X*)
case *True*
hence *eventually P* (*atin X a*) ⟷ (∃ *S*. *openin X S* ∧
a ∈ *S* ∧ (∀ *x*∈*S*. *x* ∈ *topspace X* ∧ *x* ≠ *a* ⟶ *P x*))
by (*simp add*: *atin_def eventually_inf_principal eventually_nhdsin*)
also have ... ⟷ (∃ *U*. *openin X U* ∧ *a* ∈ *U* ∧ (∀ *x* ∈ *U* − {*a*}. *P x*))
using *openin_subset* **by** (*intro ex_cong*) *auto*
finally show *?thesis* **by** (*simp add*: *True*)
qed *auto*

lemma *nontrivial_limit_atin*:
atin X a ≠ *bot* ⟷ *a* ∈ *X derived_set_of topspace X*
proof

assume *L*: *atin X a* ≠ *bot*
then have *a* ∈ *topspace X*
by (*meson atin_degenerate*)
moreover have ¬ *openin X* {*a*}
using *L* **by** (*auto simp*: *eventually_atin trivial_limit_eq*)
ultimately
show *a* ∈ *X derived_set_of topspace X*
by (*auto simp*: *derived_set_of_topspace*)
next
assume *a*: *a* ∈ *X derived_set_of topspace X*
show *atin X a* ≠ *bot*
proof
assume *atin X a* = *bot*
then have *eventually* (λ_. *False*) (*atin X a*)
by *simp*
then show *False*
by (*smt* (*verit*, *best*) *a eventually_atin in_derived_set_of insertE insert_Diff*)
qed
qed

2.1.16 Limits in a topological space

definition *limitin* :: *'a topology* ⇒ (*'b* ⇒ *'a*) ⇒ *'a* ⇒ *'b filter* ⇒ *bool* **where**
limitin X f l F ≡ *l* ∈ *topspace X* ∧ (∀ *U*. *openin X U* ∧ *l* ∈ *U* ⟶ *eventually* (λ*x*. *f x* ∈ *U*) *F*)

lemma *limitinD*: ⟦*limitin X f l F*; *openin X U*; *l* ∈ *U*⟧ ⟹ *eventually* (λ*x*. *f x* ∈ *U*) *F*
by (*simp add*: *limitin_def*)

lemma *limitin_canonical_iff* [*simp*]: *limitin euclidean f l F* ⟷ (*f* ⟶ *l*) *F*
by (*auto simp*: *limitin_def tendsto_def*)

lemma *limitin_topspace*: *limitin X f l F* ⟹ *l* ∈ *topspace X*
by (*simp add*: *limitin_def*)

lemma *limitin_const_iff* [*simp*]: *limitin X* (λ*a*. *l*) *l F* ⟷ *l* ∈ *topspace X*
by (*simp add*: *limitin_def*)

lemma *limitin_const*: *limitin euclidean* (λ*a*. *l*) *l F*
by *simp*

lemma *limitin_eventually*:
⟦*l* ∈ *topspace X*; *eventually* (λ*x*. *f x* = *l*) *F*⟧ ⟹ *limitin X f l F*
by (*auto simp*: *limitin_def eventually_mono*)

lemma *limitin_subsequence*:
⟦*strict_mono r*; *limitin X f l sequentially*⟧ ⟹ *limitin X* (*f* ∘ *r*) *l sequentially*
unfolding *limitin_def* **using** *eventually_subseq* **by** *fastforce*

lemma *limitin_subtopology*:
  limitin (*subtopology X S*) *f l F*
   ⟷ *l* ∈ *S* ∧ *eventually* (λ*a*. *f a* ∈ *S*) *F* ∧ *limitin X f l F* (**is** *?lhs* = *?rhs*)
proof (*cases l* ∈ *S* ∩ *topspace X*)
  case *True*
  show *?thesis*
  proof
    assume *L*: *?lhs*
    with *True*
    have ∀ *F b in F*. *f b* ∈ *topspace X* ∩ *S*
      by (*metis* (*no_types*) *limitin_def openin_topspace topspace_subtopology*)
    with *L* **show** *?rhs*
    apply (*clarsimp simp add*: *limitin_def eventually_mono openin_subtopology_alt*)
      apply (*drule_tac x*=*S* ∩ *U* **in** *spec*, *force simp*: *elim*: *eventually_mono*)
      done
  next
    assume *?rhs*
    then show *?lhs*
      using *eventually_elim2*
      by (*fastforce simp add*: *limitin_def openin_subtopology_alt*)
  qed
qed (*auto simp*: *limitin_def*)

lemma *limitin_canonical_iff_gen* [*simp*]:
  assumes *open S*
  shows *limitin* (*top_of_set S*) *f l F* ⟷ (*f* ⟶ *l*) *F* ∧ *l* ∈ *S*
  using *assms* **by** (*auto simp*: *limitin_subtopology tendsto_def*)

lemma *limitin_sequentially*:
   limitin X S l sequentially ⟷
    l ∈ *topspace X* ∧ (∀ *U*. *openin X U* ∧ *l* ∈ *U* ⟶ (∃ *N*. ∀ *n*. *N* ≤ *n* ⟶ *S n* ∈
U))
  by (*simp add*: *limitin_def eventually_sequentially*)

lemma *limitin_sequentially_offset*:
   limitin X f l sequentially ⟹ *limitin X* (λ*i*. *f* (*i* + *k*)) *l sequentially*
  unfolding *limitin_sequentially*
  by (*metis add.commute le_add2 order_trans*)

lemma *limitin_sequentially_offset_rev*:
  assumes *limitin X* (λ*i*. *f* (*i* + *k*)) *l sequentially*
  shows *limitin X f l sequentially*
proof −
  have ∃ *N*. ∀ *n*≥*N*. *f n* ∈ *U* **if** *U*: *openin X U l* ∈ *U* **for** *U*
  proof −
    obtain *N* **where** ⋀*n*. *n*≥*N* ⟹ *f* (*n* + *k*) ∈ *U*
      using *assms U* **unfolding** *limitin_sequentially* **by** *blast*

```
    then have ∀ n≥N+k. f n ∈ U
    by (metis add_leD2 le_add_diff_inverse ordered_cancel_comm_monoid_diff_class.le_diff_conv2
add.commute)
    then show ?thesis ..
  qed
  with assms show ?thesis
    unfolding limitin_sequentially
    by simp
qed

lemma limitin_atin:
   limitin Y f y (atin X x) ⟷
        y ∈ topspace Y ∧
        (x ∈ topspace X
        ⟶ (∀ V. openin Y V ∧ y ∈ V
                ⟶ (∃ U. openin X U ∧ x ∈ U ∧ f ‘ (U − {x}) ⊆ V)))
  by (auto simp: limitin_def eventually_atin image_subset_iff)

lemma limitin_atin_self:
   limitin Y f (f a) (atin X a) ⟷
        f a ∈ topspace Y ∧
        (a ∈ topspace X
         ⟶ (∀ V. openin Y V ∧ f a ∈ V
                 ⟶ (∃ U. openin X U ∧ a ∈ U ∧ f ‘ U ⊆ V)))
  unfolding limitin_atin by fastforce

lemma limitin_trivial:
   ⟦trivial_limit F; y ∈ topspace X⟧ ⟹ limitin X f y F
  by (simp add: limitin_def)

lemma limitin_transform_eventually:
   ⟦eventually (λx. f x = g x) F; limitin X f l F⟧ ⟹ limitin X g l F
  unfolding limitin_def using eventually_elim2 by fastforce

lemma continuous_map_limit:
  assumes continuous_map X Y g and f: limitin X f l F
  shows limitin Y (g ∘ f) (g l) F
proof −
  have g l ∈ topspace Y
    by (meson assms continuous_map f image_eqI in_mono limitin_def)
  moreover
  have ⋀U. ⟦∀ V. openin X V ∧ l ∈ V ⟶ (∀F x in F. f x ∈ V); openin Y U; g
l ∈ U⟧
           ⟹ ∀F x in F. g (f x) ∈ U
    using assms eventually_mono
    by (fastforce simp: limitin_def dest!: openin_continuous_map_preimage)
  ultimately show ?thesis
    using f by (fastforce simp add: limitin_def)
qed
```

2.1.17 Pointwise continuity in topological spaces

definition *topcontinuous_at* **where**
topcontinuous_at X Y f x ⟷
x ∈ *topspace X* ∧
f ∈ *topspace X* → *topspace Y* ∧
(∀ *V*. *openin Y V* ∧ *f x* ∈ *V*
⟶ (∃ *U*. *openin X U* ∧ *x* ∈ *U* ∧ (∀ *y* ∈ *U*. *f y* ∈ *V*)))

lemma *topcontinuous_at_atin*:
topcontinuous_at X Y f x ⟷
x ∈ *topspace X* ∧
f ∈ *topspace X* → *topspace Y* ∧
limitin Y f (*f x*) (*atin X x*)
unfolding *topcontinuous_at_def*
by (*fastforce simp add*: *limitin_atin*)+

lemma *continuous_map_eq_topcontinuous_at*:
continuous_map X Y f ⟷ (∀ *x* ∈ *topspace X*. *topcontinuous_at X Y f x*)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (*auto simp*: *continuous_map_def topcontinuous_at_def*)
next
assume *R*: *?rhs*
then show *?lhs*
apply (*auto simp*: *continuous_map_def topcontinuous_at_def*)
apply (*subst openin_subopen*, *safe*)
apply (*drule bspec*, *assumption*)
using *openin_subset*[*of X*] **apply** (*auto simp*: *subset_iff dest*!: *spec*)
done
qed

lemma *continuous_map_atin*:
continuous_map X Y f ⟷ (∀ *x* ∈ *topspace X*. *limitin Y f* (*f x*) (*atin X x*))
by (*auto simp*: *limitin_def topcontinuous_at_atin continuous_map_eq_topcontinuous_at*)

lemma *limitin_continuous_map*:
⟦*continuous_map X Y f*; *a* ∈ *topspace X*; *f a* = *b*⟧ ⟹ *limitin Y f b* (*atin X a*)
by (*auto simp*: *continuous_map_atin*)

2.1.18 Combining theorems for continuous functions into the reals

lemma *continuous_map_canonical_const* [*continuous_intros*]:
continuous_map X euclidean (λ*x*. *c*)
by *simp*

lemma *continuous_map_real_mult* [*continuous_intros*]:

⟦*continuous_map X euclideanreal f*; *continuous_map X euclideanreal g*⟧
⟹ *continuous_map X euclideanreal* (λ*x*. *f x* ∗ *g x*)
by (*simp add*: *continuous_map_atin tendsto_mult*)

lemma *continuous_map_real_pow* [*continuous_intros*]:
continuous_map X euclideanreal f ⟹ *continuous_map X euclideanreal* (λ*x*. *f x* ^ *n*)
by (*induction n*) (*auto simp*: *continuous_map_real_mult*)

lemma *continuous_map_real_mult_left*:
continuous_map X euclideanreal f ⟹ *continuous_map X euclideanreal* (λ*x*. *c* ∗ *f x*)
by (*simp add*: *continuous_map_atin tendsto_mult*)

lemma *continuous_map_real_mult_left_eq*:
continuous_map X euclideanreal (λ*x*. *c* ∗ *f x*) ⟷ *c* = *0* ∨ *continuous_map X euclideanreal f*
proof (*cases c* = *0*)
case *False*
have *continuous_map X euclideanreal* (λ*x*. *c* ∗ *f x*) ⟹ *continuous_map X euclideanreal f*
apply (*frule continuous_map_real_mult_left* [**where** *c*=*inverse c*])
apply (*simp add*: *field_simps False*)
done
with *False* **show** *?thesis*
using *continuous_map_real_mult_left* **by** *blast*
qed *simp*

lemma *continuous_map_real_mult_right*:
continuous_map X euclideanreal f ⟹ *continuous_map X euclideanreal* (λ*x*. *f x* ∗ *c*)
by (*simp add*: *continuous_map_atin tendsto_mult*)

lemma *continuous_map_real_mult_right_eq*:
continuous_map X euclideanreal (λ*x*. *f x* ∗ *c*) ⟷ *c* = *0* ∨ *continuous_map X euclideanreal f*
by (*simp add*: *mult.commute flip*: *continuous_map_real_mult_left_eq*)

lemma *continuous_map_minus* [*continuous_intros*]:
fixes *f* :: *'a*⇒*'b*::*real_normed_vector*
shows *continuous_map X euclidean f* ⟹ *continuous_map X euclidean* (λ*x*. − *f x*)
by (*simp add*: *continuous_map_atin tendsto_minus*)

lemma *continuous_map_minus_eq* [*simp*]:
fixes *f* :: *'a*⇒*'b*::*real_normed_vector*
shows *continuous_map X euclidean* (λ*x*. − *f x*) ⟷ *continuous_map X euclidean f*
using *continuous_map_minus add.inverse_inverse continuous_map_eq* **by** *fast-*

force

lemma *continuous_map_add* [*continuous_intros*]:
fixes $f :: 'a \Rightarrow 'b::real\_normed\_vector$
shows $[\![continuous\_map\ X\ euclidean\ f;\ continuous\_map\ X\ euclidean\ g]\!] \Longrightarrow continuous\_map\ X\ euclidean\ (\lambda x.\ f\ x + g\ x)$
by (*simp add*: *continuous_map_atin tendsto_add*)

lemma *continuous_map_diff* [*continuous_intros*]:
fixes $f :: 'a \Rightarrow 'b::real\_normed\_vector$
shows $[\![continuous\_map\ X\ euclidean\ f;\ continuous\_map\ X\ euclidean\ g]\!] \Longrightarrow continuous\_map\ X\ euclidean\ (\lambda x.\ f\ x - g\ x)$
by (*simp add*: *continuous_map_atin tendsto_diff*)

lemma *continuous_map_norm* [*continuous_intros*]:
fixes $f :: 'a \Rightarrow 'b::real\_normed\_vector$
shows $continuous\_map\ X\ euclidean\ f \Longrightarrow continuous\_map\ X\ euclidean\ (\lambda x.\ norm(f\ x))$
by (*simp add*: *continuous_map_atin tendsto_norm*)

lemma *continuous_map_real_abs* [*continuous_intros*]:
$continuous\_map\ X\ euclideanreal\ f \Longrightarrow continuous\_map\ X\ euclideanreal\ (\lambda x.\ abs(f\ x))$
by (*simp add*: *continuous_map_atin tendsto_rabs*)

lemma *continuous_map_real_max* [*continuous_intros*]:
$[\![continuous\_map\ X\ euclideanreal\ f;\ continuous\_map\ X\ euclideanreal\ g]\!]$
$\Longrightarrow continuous\_map\ X\ euclideanreal\ (\lambda x.\ max\ (f\ x)\ (g\ x))$
by (*simp add*: *continuous_map_atin tendsto_max*)

lemma *continuous_map_real_min* [*continuous_intros*]:
$[\![continuous\_map\ X\ euclideanreal\ f;\ continuous\_map\ X\ euclideanreal\ g]\!]$
$\Longrightarrow continuous\_map\ X\ euclideanreal\ (\lambda x.\ min\ (f\ x)\ (g\ x))$
by (*simp add*: *continuous_map_atin tendsto_min*)

lemma *continuous_map_sum* [*continuous_intros*]:
fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c::real\_normed\_vector$
shows $[\![finite\ I;\ \bigwedge i.\ i \in I \Longrightarrow continuous\_map\ X\ euclidean\ (\lambda x.\ f\ x\ i)]\!]$
$\Longrightarrow continuous\_map\ X\ euclidean\ (\lambda x.\ sum\ (f\ x)\ I)$
by (*simp add*: *continuous_map_atin tendsto_sum*)

lemma *continuous_map_prod* [*continuous_intros*]:
$[\![finite\ I;$
$\bigwedge i.\ i \in I \Longrightarrow continuous\_map\ X\ euclideanreal\ (\lambda x.\ f\ x\ i)]\!]$
$\Longrightarrow continuous\_map\ X\ euclideanreal\ (\lambda x.\ prod\ (f\ x)\ I)$
by (*simp add*: *continuous_map_atin tendsto_prod*)

lemma *continuous_map_real_inverse* [*continuous_intros*]:
$[\![continuous\_map\ X\ euclideanreal\ f;\ \bigwedge x.\ x \in topspace\ X \Longrightarrow f\ x \neq 0]\!]$

$\Longrightarrow$ *continuous_map X euclideanreal* (λx. *inverse*(*f x*))
by (*simp add*: *continuous_map_atin tendsto_inverse*)

lemma *continuous_map_real_divide* [*continuous_intros*]:
⟦*continuous_map X euclideanreal f*; *continuous_map X euclideanreal g*; $\bigwedge x$. *x* $\in$ *topspace X* $\Longrightarrow$ *g x* $\neq$ *0*⟧
$\Longrightarrow$ *continuous_map X euclideanreal* (λx. *f x* / *g x*)
by (*simp add*: *continuous_map_atin tendsto_divide*)

end

2.2 Non-Denumerability of the Continuum

theory *Continuum_Not_Denumerable*
imports
Complex_Main
HOL−Library.Countable_Set
begin

2.2.1 Abstract

The following document presents a proof that the Continuum is uncountable. It is formalised in the Isabelle/Isar theorem proving system.

Theorem: The Continuum $\mathbb{R}$ is not denumerable. In other words, there does not exist a function f: $\mathbb{N} \Rightarrow \mathbb{R}$ such that f is surjective.

Outline: An elegant informal proof of this result uses Cantor's Diagonalisation argument. The proof presented here is not this one.

First we formalise some properties of closed intervals, then we prove the Nested Interval Property. This property relies on the completeness of the Real numbers and is the foundation for our argument. Informally it states that an intersection of countable closed intervals (where each successive interval is a subset of the last) is non-empty. We then assume a surjective function f: $\mathbb{N} \Rightarrow \mathbb{R}$ exists and find a real x such that x is not in the range of f by generating a sequence of closed intervals then using the Nested Interval Property.

theorem *real_non_denum*: $\nexists f$ :: *nat* $\Rightarrow$ *real*. *surj f*
proof
assume $\exists f$::*nat* $\Rightarrow$ *real*. *surj f*
then obtain *f* :: *nat* $\Rightarrow$ *real* **where** *surj f* **..**

First we construct a sequence of nested intervals, ignoring *range f*.

have *a* < *b* $\Longrightarrow$ $\exists$ *ka kb*. *ka* < *kb* $\wedge$ {*ka*..*kb*} $\subseteq$ {*a*..*b*} $\wedge$ *c* $\notin$ {*ka*..*kb*} **for** *a b c* :: *real*
by (*auto simp add*: *not_le cong*: *conj_cong*)
(*metis dense le_less_linear less_linear less_trans order_refl*)

```
  then obtain i j where ij:
    a < b ⟹ i a b c < j a b c
      a < b ⟹ {i a b c .. j a b c} ⊆ {a .. b}
      a < b ⟹ c ∉ {i a b c .. j a b c}
    for a b c :: real
    by metis

  define ivl where ivl =
    rec_nat (f 0 + 1, f 0 + 2) (λn x. (i (fst x) (snd x) (f n), j (fst x) (snd x) (f
n)))
  define I where I n = {fst (ivl n) .. snd (ivl n)} for n

  have ivl [simp]:
    ivl 0 = (f 0 + 1, f 0 + 2)
    ⋀n. ivl (Suc n) = (i (fst (ivl n)) (snd (ivl n)) (f n), j (fst (ivl n)) (snd (ivl n))
(f n))
    unfolding ivl_def by simp_all
```

This is a decreasing sequence of non-empty intervals.

```
  have less: fst (ivl n) < snd (ivl n) for n
    by (induct n) (auto intro!: ij)

  have decseq I
    unfolding I_def decseq_Suc_iff ivl fst_conv snd_conv
    by (intro ij allI less)
```

Now we apply the finite intersection property of compact sets.

```
  have I 0 ∩ (⋂ i. I i) ≠ {}
  proof (rule compact_imp_fip_image)
    fix S :: nat set
    assume fin: finite S
    have {} ⊂ I (Max (insert 0 S))
      unfolding I_def using less[of Max (insert 0 S)] by auto
    also have I (Max (insert 0 S)) ⊆ (⋂ i∈insert 0 S. I i)
      using fin decseqD[OF ‹decseq I›, of _ Max (insert 0 S)]
      by (auto simp: Max_ge_iff)
    also have (⋂ i∈insert 0 S. I i) = I 0 ∩ (⋂ i∈S. I i)
      by auto
    finally show I 0 ∩ (⋂ i∈S. I i) ≠ {}
      by auto
  qed (auto simp: I_def)
  then obtain x where x ∈ I n for n
    by blast
  moreover from ‹surj f› obtain j where x = f j
    by blast
  ultimately have f j ∈ I (Suc j)
    by blast
  with ij(3)[OF less] show False
    unfolding I_def ivl fst_conv snd_conv by auto
```

qed

lemma *uncountable_UNIV_real*: *uncountable* (*UNIV* :: *real set*)
using *real_non_denum* **unfolding** *uncountable_def* **by** *auto*

corollary *complex_non_denum*: $\nexists f$ :: *nat* ⇒ *complex*. *surj f*
by (*metis* (*full_types*) *Re_complex_of_real comp_surj real_non_denum surj_def*)

lemma *uncountable_UNIV_complex*: *uncountable* (*UNIV* :: *complex set*)
using *complex_non_denum* **unfolding** *uncountable_def* **by** *auto*

lemma *bij_betw_open_intervals*:
fixes *a b c d* :: *real*
assumes *a* < *b c* < *d*
shows ∃*f*. *bij_betw f* {*a*<..<*b*} {*c*<..<*d*}
proof −
define *f* **where** *f a b c d x* = (*d* − *c*)/(*b* − *a*) ∗ (*x* − *a*) + *c* **for** *a b c d x* :: *real*
{
fix *a b c d x* :: *real*
assume ∗: *a* < *b c* < *d a* < *x x* < *b*
moreover from ∗ **have** (*d* − *c*) ∗ (*x* − *a*) < (*d* − *c*) ∗ (*b* − *a*)
by (*intro mult_strict_left_mono*) *simp_all*
moreover from ∗ **have** *0* < (*d* − *c*) ∗ (*x* − *a*) / (*b* − *a*)
by *simp*
ultimately have *f a b c d x* < *d c* < *f a b c d x*
by (*simp_all add*: *f_def field_simps*)
}
with *assms* **have** *bij_betw* (*f a b c d*) {*a*<..<*b*} {*c*<..<*d*}
by (*intro bij_betw_byWitness*[**where** *f'*=*f c d a b*]) (*auto simp*: *f_def*)
then show *?thesis* **by** *auto*
qed

lemma *bij_betw_tan*: *bij_betw tan* {−*pi*/*2*<..<*pi*/*2*} *UNIV*
using *arctan_ubound* **by** (*intro bij_betw_byWitness*[**where** *f'*=*arctan*]) (*auto simp*: *arctan arctan_tan*)

lemma *uncountable_open_interval*: *uncountable* {*a*<..<*b*} ⟷ *a* < *b* **for** *a b* :: *real*
proof
show *a* < *b* **if** *uncountable* {*a*<..<*b*}
using *uncountable_def that* **by** *force*
show *uncountable* {*a*<..<*b*} **if** *a* < *b*
proof −
obtain *f* **where** *bij_betw f* {*a* <..< *b*} {−*pi*/*2*<..<*pi*/*2*}
using *bij_betw_open_intervals*[*OF* ‹*a* < *b*›, *of* −*pi*/*2 pi*/*2*] **by** *auto*
then show *?thesis*
by (*metis bij_betw_tan uncountable_bij_betw uncountable_UNIV_real*)
qed
qed

lemma *uncountable_half_open_interval_1*: *uncountable* $\{a..<b\} \longleftrightarrow a < b$ **for** $a\ b$ :: *real*
apply *auto*
using *atLeastLessThan_empty_iff*
apply *fastforce*
using *uncountable_open_interval* [*of a b*]
apply (*metis countable_Un_iff ivl_disj_un_singleton*(3))
done

lemma *uncountable_half_open_interval_2*: *uncountable* $\{a<..b\} \longleftrightarrow a < b$ **for** $a\ b$ :: *real*
apply *auto*
using *atLeastLessThan_empty_iff*
apply *fastforce*
using *uncountable_open_interval* [*of a b*]
apply (*metis countable_Un_iff ivl_disj_un_singleton*(4))
done

lemma *real_interval_avoid_countable_set*:
fixes $a\ b$ :: *real* **and** A :: *real set*
assumes $a < b$ **and** *countable* A
shows $\exists x{\in}\{a<..<b\}.\ x \notin A$
proof −
from ‹*countable* A› **have** $*$: *countable* $(A \cap \{a<..<b\})$
by *auto*
with ‹$a < b$› **have** ¬ *countable* $\{a<..<b\}$
by (*simp add*: *uncountable_open_interval*)
with $*$ **have** $A \cap \{a<..<b\} \neq \{a<..<b\}$
by *auto*
then have $A \cap \{a<..<b\} \subset \{a<..<b\}$
by (*intro psubsetI*) *auto*
then have $\exists x.\ x \in \{a<..<b\} - A \cap \{a<..<b\}$
by (*rule psubset_imp_ex_mem*)
then show *?thesis*
by *auto*
qed

lemma *uncountable_closed_interval*: *uncountable* $\{a..b\} \longleftrightarrow a < b$ **for** $a\ b$ :: *real*
using *infinite_Icc_iff* **by** (*fastforce dest*: *countable_finite real_interval_avoid_countable_set*)

lemma *open_minus_countable*:
fixes $S\ A$ :: *real set*
assumes *countable* A $S \neq \{\}$ *open* S
shows $\exists x{\in}S.\ x \notin A$
proof −
obtain x **where** $x \in S$
using ‹$S \neq \{\}$› **by** *auto*
then obtain e **where** $0 < e$ $\{y.\ dist\ y\ x < e\} \subseteq S$

```
    using ‹open S› by (auto simp: open_dist subset_eq)
  moreover have {y. dist y x < e} = {x - e <..< x + e}
    by (auto simp: dist_real_def)
  ultimately have uncountable (S - A)
    using uncountable_open_interval[of x - e x + e] ‹countable A›
    by (intro uncountable_minus_countable) (auto dest: countable_subset)
  then show ?thesis
    unfolding uncountable_def by auto
qed

end
```

2.3 Abstract Topology 2

```
theory Abstract_Topology_2
  imports
    Elementary_Topology Abstract_Topology Continuum_Not_Denumerable
    HOL−Library.Indicator_Function
    HOL−Library.Equipollence
begin
```

Combination of Elementary and Abstract Topology

```
lemma approachable_lt_le2:
    (∃(d::real) > 0. ∀x. Q x ⟶ f x < d ⟶ P x) ⟷ (∃d>0. ∀x. f x ≤ d ⟶
Q x ⟶ P x)
by (meson dense less_eq_real_def order_le_less_trans)

lemma triangle_lemma:
  fixes x y z :: real
  assumes x: 0 ≤ x
    and y: 0 ≤ y
    and z: 0 ≤ z
    and xy: x^2 ≤ y^2 + z^2
  shows x ≤ y + z
proof −
  have y^2 + z^2 ≤ y^2 + 2 * y * z + z^2
    using z y by simp
  with xy have th: x^2 ≤ (y + z)^2
    by (simp add: power2_eq_square field_simps)
  from y z have yz: y + z ≥ 0
    by arith
  from power2_le_imp_le[OF th yz] show ?thesis .
qed

lemma isCont_indicator:
  fixes x :: 'a::t2_space
  shows isCont (indicator A :: 'a ⇒ real) x = (x ∉ frontier A)
proof auto
  fix x
```

assume *cts_at*: *isCont* (*indicator A* :: *'a* ⇒ *real*) *x* **and** *fr*: *x* ∈ *frontier A*
with *continuous_at_open* **have** *1*: ∀ *V*::*real set*. *open V* ∧ *indicator A x* ∈ *V*
⟶
(∃ *U*::*'a set*. *open U* ∧ *x* ∈ *U* ∧ (∀ *y*∈*U*. *indicator A y* ∈ *V*)) **by** *auto*
show *False*
proof (*cases x* ∈ *A*)
assume *x*: *x* ∈ *A*
hence *indicator A x* ∈ ({*0*<..<*2*} :: *real set*) **by** *simp*
with *1* **obtain** *U* **where** *U*: *open U x* ∈ *U* ∀ *y*∈*U*. *indicator A y* ∈ ({*0*<..<*2*}
:: *real set*)
using *open_greaterThanLessThan* **by** *metis*
hence ∀ *y*∈*U*. *indicator A y* > (*0*::*real*)
unfolding *greaterThanLessThan_def* **by** *auto*
hence *U* ⊆ *A* **using** *indicator_eq_0_iff* **by** *force*
hence *x* ∈ *interior A* **using** *U interiorI* **by** *auto*
thus *?thesis* **using** *fr* **unfolding** *frontier_def* **by** *simp*
next
assume *x*: *x* ∉ *A*
hence *indicator A x* ∈ ({*−1*<..<*1*} :: *real set*) **by** *simp*
with *1* **obtain** *U* **where** *U*: *open U x* ∈ *U* ∀ *y*∈*U*. *indicator A y* ∈ ({*−1*<..<*1*}
:: *real set*)
using *1 open_greaterThanLessThan* **by** *metis*
hence ∀ *y*∈*U*. *indicator A y* < (*1*::*real*)
unfolding *greaterThanLessThan_def* **by** *auto*
hence *U* ⊆ *−A* **by** *auto*
hence *x* ∈ *interior* (*−A*) **using** *U interiorI* **by** *auto*
thus *?thesis* **using** *fr interior_complement* **unfolding** *frontier_def* **by** *auto*
qed
next
assume *nfr*: *x* ∉ *frontier A*
hence *x* ∈ *interior A* ∨ *x* ∈ *interior* (*−A*)
by (*auto simp*: *frontier_def closure_interior*)
thus *isCont* ((*indicator A*)::*'a* ⇒ *real*) *x*
proof
assume *int*: *x* ∈ *interior A*
then obtain *U* **where** *U*: *open U x* ∈ *U U* ⊆ *A* **unfolding** *interior_def* **by**
auto
hence ∀ *y*∈*U*. *indicator A y* = (*1*::*real*) **unfolding** *indicator_def* **by** *auto*
hence *continuous_on U* (*indicator A*) **by** (*simp add*: *indicator_eq_1_iff*)
thus *?thesis* **using** *U continuous_on_eq_continuous_at* **by** *auto*
next
assume *ext*: *x* ∈ *interior* (*−A*)
then obtain *U* **where** *U*: *open U x* ∈ *U U* ⊆ *−A* **unfolding** *interior_def* **by**
auto
then have *continuous_on U* (*indicator A*)
using *continuous_on_topological* **by** (*auto simp*: *subset_iff*)
thus *?thesis* **using** *U continuous_on_eq_continuous_at* **by** *auto*
qed
qed

lemma *islimpt_closure*:
$[\![S \subseteq T; \bigwedge x.\ [\![x\ \mathit{islimpt}\ S; x \in T]\!] \Longrightarrow x \in S]\!] \Longrightarrow S = T \cap \mathit{closure}\ S$
using *closure_def* **by** *fastforce*

lemma *closedin_limpt*:
$\mathit{closedin}\ (\mathit{top\_of\_set}\ T)\ S \longleftrightarrow S \subseteq T \wedge (\forall x.\ x\ \mathit{islimpt}\ S \wedge x \in T \longrightarrow x \in S)$
proof −
have $\bigwedge U\ x.\ [\![\mathit{closed}\ U; S = T \cap U; x\ \mathit{islimpt}\ S; x \in T]\!] \Longrightarrow x \in S$
by (*meson IntI closed_limpt inf_le2 islimpt_subset*)
then show *?thesis*
by (*metis closed_closure closedin_closed closedin_imp_subset islimpt_closure*)
qed

lemma *closedin_closed_eq*: $\mathit{closed}\ S \Longrightarrow \mathit{closedin}\ (\mathit{top\_of\_set}\ S)\ T \longleftrightarrow \mathit{closed}\ T \wedge T \subseteq S$
by (*meson closedin_limpt closed_subset closedin_closed_trans*)

lemma *connected_closed_set*:
$\mathit{closed}\ S \Longrightarrow \mathit{connected}\ S \longleftrightarrow (\nexists A\ B.\ \mathit{closed}\ A \wedge \mathit{closed}\ B \wedge A \neq \{\} \wedge B \neq \{\} \wedge A \cup B = S \wedge A \cap B = \{\})$
unfolding *connected_closedin_eq closedin_closed_eq connected_closedin_eq* **by** *blast*

If a connnected set is written as the union of two nonempty closed sets, then these sets have to intersect.

lemma *connected_as_closed_union*:
assumes $\mathit{connected}\ C\ C = A \cup B\ \mathit{closed}\ A\ \mathit{closed}\ B\ A \neq \{\}\ B \neq \{\}$
shows $A \cap B \neq \{\}$
by (*metis assms closed_Un connected_closed_set*)

lemma *closedin_subset_trans*:
$\mathit{closedin}\ (\mathit{top\_of\_set}\ U)\ S \Longrightarrow S \subseteq T \Longrightarrow T \subseteq U \Longrightarrow \mathit{closedin}\ (\mathit{top\_of\_set}\ T)\ S$
by (*meson closedin_limpt subset_iff*)

lemma *openin_subset_trans*:
$\mathit{openin}\ (\mathit{top\_of\_set}\ U)\ S \Longrightarrow S \subseteq T \Longrightarrow T \subseteq U \Longrightarrow \mathit{openin}\ (\mathit{top\_of\_set}\ T)\ S$
by (*auto simp*: *openin_open*)

lemma *closedin_compact*:
$[\![\mathit{compact}\ S; \mathit{closedin}\ (\mathit{top\_of\_set}\ S)\ T]\!] \Longrightarrow \mathit{compact}\ T$
by (*metis closedin_closed compact_Int_closed*)

lemma *closedin_compact_eq*:
fixes $S :: 'a{::}t2\_space\ \mathit{set}$
shows $\mathit{compact}\ S \Longrightarrow (\mathit{closedin}\ (\mathit{top\_of\_set}\ S)\ T \longleftrightarrow \mathit{compact}\ T \wedge T \subseteq S)$

by (*metis closedin_imp_subset closedin_compact closed_subset compact_imp_closed*)

2.3.1 Closure

lemma *euclidean_closure_of* [*simp*]: *euclidean closure_of S = closure S*
by (*auto simp*: *closure_of_def closure_def islimpt_def*)

lemma *closure_openin_Int_closure*:
assumes *ope*: *openin* (*top_of_set U*) *S* **and** *T* ⊆ *U*
shows *closure*(*S* ∩ *closure T*) = *closure*(*S* ∩ *T*)
proof
obtain *V* **where** *open V* **and** *S*: *S* = *U* ∩ *V*
using *ope* **using** *openin_open* **by** *metis*
show *closure* (*S* ∩ *closure T*) ⊆ *closure* (*S* ∩ *T*)
proof (*clarsimp simp*: *S*)
fix *x*
assume *x* ∈ *closure* (*U* ∩ *V* ∩ *closure T*)
then have *V* ∩ *closure T* ⊆ *A* ⟹ *x* ∈ *closure A* **for** *A*
by (*metis closure_mono subsetD inf.coboundedI2 inf_assoc*)
then have *x* ∈ *closure* (*T* ∩ *V*)
by (*metis* ‹*open V*› *closure_closure inf_commute open_Int_closure_subset*)
then show *x* ∈ *closure* (*U* ∩ *V* ∩ *T*)
by (*metis* ‹*T* ⊆ *U*› *inf.absorb_iff2 inf_assoc inf_commute*)
qed
next
show *closure* (*S* ∩ *T*) ⊆ *closure* (*S* ∩ *closure T*)
by (*meson Int_mono closure_mono closure_subset order_refl*)
qed

corollary *infinite_openin*:
fixes *S* :: *'a* :: *t1_space set*
shows ⟦*openin* (*top_of_set U*) *S*; *x* ∈ *S*; *x islimpt U*⟧ ⟹ *infinite S*
by (*clarsimp simp add*: *openin_open islimpt_eq_acc_point inf_commute*)

lemma *closure_Int_ballI*:
assumes ⋀*U*. ⟦*openin* (*top_of_set S*) *U*; *U* ≠ {}⟧ ⟹ *T* ∩ *U* ≠ {}
shows *S* ⊆ *closure T*
proof (*clarsimp simp*: *closure_iff_nhds_not_empty*)
fix *x* **and** *A* **and** *V*
assume *x* ∈ *S V* ⊆ *A open V x* ∈ *V T* ∩ *A* = {}
then have *openin* (*top_of_set S*) (*A* ∩ *V* ∩ *S*)
by (*simp add*: *inf_absorb2 openin_subtopology_Int*)
moreover have *A* ∩ *V* ∩ *S* ≠ {} **using** ‹*x* ∈ *V*› ‹*V* ⊆ *A*› ‹*x* ∈ *S*›
by *auto*
ultimately show *False*
using ‹*T* ∩ *A* = {}› *assms* **by** *fastforce*
qed

2.3.2 Frontier

lemma *euclidean_interior_of* [*simp*]: *euclidean interior_of S = interior S*
by (*auto simp*: *interior_of_def interior_def*)

lemma *euclidean_frontier_of* [*simp*]: *euclidean frontier_of S = frontier S*
by (*auto simp*: *frontier_of_def frontier_def*)

lemma *connected_Int_frontier*:
⟦*connected S*; *S* ∩ *T* ≠ {}; *S* − *T* ≠ {}⟧ ⟹ *S* ∩ *frontier T* ≠ {}
apply (*simp add*: *frontier_interiors connected_openin*, *safe*)
apply (*drule_tac x*=*S* ∩ *interior T* **in** *spec*, *safe*)
apply (*drule_tac* [2] *x*=*S* ∩ *interior* (−*T*) **in** *spec*)
apply (*auto simp*: *disjoint_eq_subset_Compl dest*: *interior_subset* [*THEN subsetD*])
done

2.3.3 Compactness

lemma *openin_delete*:
fixes *a* :: *'a* :: *t1_space*
shows *openin* (*top_of_set u*) *S* ⟹ *openin* (*top_of_set u*) (*S* − {*a*})
by (*metis Int_Diff open_delete openin_open*)

lemma *compact_eq_openin_cover*:
compact S ⟷
(∀ *C*. (∀ *c*∈*C*. *openin* (*top_of_set S*) *c*) ∧ *S* ⊆ ⋃ *C* ⟶
(∃ *D*⊆*C*. *finite D* ∧ *S* ⊆ ⋃ *D*))
proof *safe*
fix *C*
assume *compact S* **and** ∀ *c*∈*C*. *openin* (*top_of_set S*) *c* **and** *S* ⊆ ⋃ *C*
then have ∀ *c*∈{*T*. *open T* ∧ *S* ∩ *T* ∈ *C*}. *open c* **and** *S* ⊆ ⋃ {*T*. *open T* ∧ *S* ∩ *T* ∈ *C*}
unfolding *openin_open* **by** *force*+
with ‹*compact S*› **obtain** *D* **where** *D* ⊆ {*T*. *open T* ∧ *S* ∩ *T* ∈ *C*} **and** *finite D* **and** *S* ⊆ ⋃ *D*
by (*meson compactE*)
then have *image* (λ*T*. *S* ∩ *T*) *D* ⊆ *C* ∧ *finite* (*image* (λ*T*. *S* ∩ *T*) *D*) ∧ *S* ⊆ ⋃ (*image* (λ*T*. *S* ∩ *T*) *D*)
by *auto*
then show ∃ *D*⊆*C*. *finite D* ∧ *S* ⊆ ⋃ *D* **..**
next
assume *1*: ∀ *C*. (∀ *c*∈*C*. *openin* (*top_of_set S*) *c*) ∧ *S* ⊆ ⋃ *C* ⟶
(∃ *D*⊆*C*. *finite D* ∧ *S* ⊆ ⋃ *D*)
show *compact S*
proof (*rule compactI*)
fix *C*
let *?C* = *image* (λ*T*. *S* ∩ *T*) *C*
assume ∀ *t*∈*C*. *open t* **and** *S* ⊆ ⋃ *C*
then have (∀ *c*∈*?C*. *openin* (*top_of_set S*) *c*) ∧ *S* ⊆ ⋃ *?C*

unfolding *openin_open* **by** *auto*
with *1* **obtain** D **where** $D \subseteq ?C$ **and** *finite* D **and** $S \subseteq \bigcup D$
by *metis*
let $?D = inv\_into\ C\ (\lambda T.\ S \cap T)\ `\ D$
have $?D \subseteq C \wedge finite\ ?D \wedge S \subseteq \bigcup ?D$
proof (*intro conjI*)
from ‹$D \subseteq ?C$› **show** $?D \subseteq C$
by (*fast intro*: *inv_into_into*)
from ‹*finite* D› **show** *finite* $?D$
by (*rule finite_imageI*)
from ‹$S \subseteq \bigcup D$› **show** $S \subseteq \bigcup ?D$
by (*metis* ‹$D \subseteq (\cap)\ S\ `\ C$› *image_inv_into_cancel inf_Sup le_infE*)
qed
then show $\exists D{\subseteq}C.\ finite\ D \wedge S \subseteq \bigcup D$ **..**
qed
qed

2.3.4 Continuity

lemma *interior_image_subset*:
assumes *inj* f $\bigwedge x.\ continuous\ (at\ x)\ f$
shows $interior\ (f\ `\ S) \subseteq f\ `\ (interior\ S)$
proof
fix x **assume** $x \in interior\ (f\ `\ S)$
then obtain T **where** *as*: *open* T $x \in T$ $T \subseteq f\ `\ S$ **..**
then have $x \in f\ `\ S$ **by** *auto*
then obtain y **where** y: $y \in S$ $x = f\ y$ **by** *auto*
have *open* $(f\ -`\ T)$
using *assms* ‹*open* T› **by** (*simp add*: *continuous_at_imp_continuous_on open_vimage*)
moreover have $y \in vimage\ f\ T$
using ‹$x = f\ y$› ‹$x \in T$› **by** *simp*
moreover have $vimage\ f\ T \subseteq S$
using ‹$T \subseteq image\ f\ S$› ‹*inj* f› **unfolding** *inj_on_def subset_eq* **by** *auto*
ultimately have $y \in interior\ S$ **..**
with ‹$x = f\ y$› **show** $x \in f\ `\ interior\ S$ **..**
qed

2.3.5 Equality of continuous functions on closure and related results

lemma *continuous_closedin_preimage_constant*:
fixes $f :: \_ \Rightarrow 'b{::}t1\_space$
shows *continuous_on* S $f \Longrightarrow closedin\ (top\_of\_set\ S)\ \{x \in S.\ f\ x = a\}$
using *continuous_closedin_preimage*[*of* S f $\{a\}$] **by** (*simp add*: *vimage_def Collect_conj_eq*)

lemma *continuous_closed_preimage_constant*:
fixes $f :: \_ \Rightarrow 'b{::}t1\_space$

shows *continuous_on S f* $\Longrightarrow$ *closed S* $\Longrightarrow$ *closed* $\{x \in S.\ f\ x = a\}$
using *continuous_closed_preimage*[*of S f* $\{a\}$] **by** (*simp add*: *vimage_def Collect_conj_eq*)

lemma *continuous_constant_on_closure*:
fixes *f* :: _ $\Rightarrow$ *'b*::*t1_space*
assumes *continuous_on* (*closure S*) *f*
and $\bigwedge x.\ x \in S \Longrightarrow f\ x = a$
and $x \in$ *closure S*
shows $f\ x = a$
using *continuous_closed_preimage_constant*[*of closure S f a*]
assms closure_minimal[*of S* $\{x \in$ *closure S*. $f\ x = a\}$] *closure_subset*
unfolding *subset_eq*
by *auto*

lemma *image_closure_subset*:
assumes *contf*: *continuous_on* (*closure S*) *f*
and *closed T*
and $(f\ `\ S) \subseteq T$
shows $f\ `$ (*closure S*) $\subseteq T$
proof −
have $S \subseteq \{x \in$ *closure S*. $f\ x \in T\}$
using *assms*(*3*) *closure_subset* **by** *auto*
moreover have *closed* (*closure S* $\cap\ f\ -`\ T$)
using *continuous_closed_preimage*[*OF contf*] ‹*closed T*› **by** *auto*
ultimately have *closure S* = (*closure S* $\cap\ f\ -`\ T$)
using *closure_minimal*[*of S* (*closure S* $\cap\ f\ -`\ T$)] **by** *auto*
then show *?thesis* **by** *auto*
qed

2.3.6 A function constant on a set

definition *constant_on* (**infixl** (*constant'_on*) *50*)
where *f constant_on A* $\equiv \exists y.\ \forall x{\in}A.\ f\ x = y$

lemma *constant_on_subset*: $[\![f$ *constant_on A*; $B \subseteq A]\!] \Longrightarrow f$ *constant_on B*
unfolding *constant_on_def* **by** *blast*

lemma *injective_not_constant*:
fixes *S* :: *'a*::{*perfect_space*} *set*
shows $[\![$*open S*; *inj_on f S*; *f constant_on S*$]\!] \Longrightarrow S = \{\}$
unfolding *constant_on_def*
by (*metis equals0I inj_on_contraD islimpt_UNIV islimpt_def*)

lemma *constant_on_compose*:
assumes *f constant_on A*
shows *g* $\circ$ *f constant_on A*
using *assms* **by** (*auto simp*: *constant_on_def*)

lemma *not_constant_onI*:
f x ≠ *f y* ⟹ *x* ∈ *A* ⟹ *y* ∈ *A* ⟹ ¬*f constant_on A*
unfolding *constant_on_def* **by** *metis*

lemma *constant_onE*:
assumes *f constant_on S* **and** ⋀*x*. *x*∈*S* ⟹ *f x* = *g x*
shows *g constant_on S*
using *assms* **unfolding** *constant_on_def* **by** *simp*

lemma *constant_on_closureI*:
fixes *f* :: _ ⇒ *'b*::*t1_space*
assumes *cof*: *f constant_on S* **and** *contf*: *continuous_on* (*closure S*) *f*
shows *f constant_on* (*closure S*)
using *continuous_constant_on_closure* [*OF contf*] *cof* **unfolding** *constant_on_def*
by *metis*

2.3.7 Continuity relative to a union.

lemma *continuous_on_Un_local*:
⟦*closedin* (*top_of_set* (*S* ∪ *T*)) *S*; *closedin* (*top_of_set* (*S* ∪ *T*)) *T*;
continuous_on S f; *continuous_on T f*⟧
⟹ *continuous_on* (*S* ∪ *T*) *f*
unfolding *continuous_on closedin_limpt*
by (*metis Lim_trivial_limit Lim_within_Un Un_iff trivial_limit_within*)

lemma *continuous_on_cases_local*:
⟦*closedin* (*top_of_set* (*S* ∪ *T*)) *S*; *closedin* (*top_of_set* (*S* ∪ *T*)) *T*;
continuous_on S f; *continuous_on T g*;
⋀*x*. ⟦*x* ∈ *S* ∧ ¬*P x* ∨ *x* ∈ *T* ∧ *P x*⟧ ⟹ *f x* = *g x*⟧
⟹ *continuous_on* (*S* ∪ *T*) (λ*x*. *if P x then f x else g x*)
by (*rule continuous_on_Un_local*) (*auto intro*: *continuous_on_eq*)

lemma *continuous_on_cases_le*:
fixes *h* :: *'a* :: *topological_space* ⇒ *real*
assumes *continuous_on* {*x* ∈ *S*. *h x* ≤ *a*} *f*
and *continuous_on* {*x* ∈ *S*. *a* ≤ *h x*} *g*
and *h*: *continuous_on S h*
and ⋀*x*. ⟦*x* ∈ *S*; *h x* = *a*⟧ ⟹ *f x* = *g x*
shows *continuous_on S* (λ*x*. *if h x* ≤ *a then f*(*x*) *else g*(*x*))
proof −
have *S*: *S* = (*S* ∩ *h* −‘ *atMost a*) ∪ (*S* ∩ *h* −‘ *atLeast a*)
by *force*
have *1*: *closedin* (*top_of_set S*) (*S* ∩ *h* −‘ *atMost a*)
by (*rule continuous_closedin_preimage* [*OF h closed_atMost*])
have *2*: *closedin* (*top_of_set S*) (*S* ∩ *h* −‘ *atLeast a*)
by (*rule continuous_closedin_preimage* [*OF h closed_atLeast*])
have [*simp*]: *S* ∩ *h* −‘ {..*a*} = {*x* ∈ *S*. *h x* ≤ *a*} *S* ∩ *h* −‘ {*a*..} = {*x* ∈ *S*. *a* ≤ *h x*}
by *auto*

have *continuous_on* $(S \cap h -` \{..a\} \cup S \cap h -` \{a..\})$ $(\lambda x.$ *if* $h\ x \leq a$ *then* $f\ x$ *else* $g\ x)$
by (*intro continuous_on_cases_local*) (*use 1 2 S assms* **in** *auto*)
then show *?thesis*
using *S* **by** *force*
qed

lemma *continuous_on_cases_1*:
fixes *S* :: *real set*
assumes *continuous_on* $\{t \in S.\ t \leq a\}$ *f*
and *continuous_on* $\{t \in S.\ a \leq t\}$ *g*
and $a \in S \Longrightarrow f\ a = g\ a$
shows *continuous_on S* $(\lambda t.$ *if* $t \leq a$ *then* $f(t)$ *else* $g(t))$
using *assms*
by (*auto intro*: *continuous_on_cases_le* [**where** $h = id$, *simplified*])

2.3.8 Inverse function property for open/closed maps

lemma *continuous_on_inverse_open_map*:
assumes *contf*: *continuous_on S f*
and *imf*: $f\ `\ S = T$
and *injf*: $\bigwedge x.\ x \in S \Longrightarrow g\ (f\ x) = x$
and *oo*: $\bigwedge U.$ *openin* (*top_of_set S*) $U \Longrightarrow$ *openin* (*top_of_set T*) $(f\ `\ U)$
shows *continuous_on T g*
proof −
from *imf injf* **have** *gTS*: $g\ `\ T = S$
by *force*
from *imf injf* **have** *fU*: $U \subseteq S \Longrightarrow (f\ `\ U) = T \cap g -` U$ **for** *U*
by *force*
show *?thesis*
by (*simp add*: *continuous_on_open* [*of T g*] *gTS*) (*metis openin_imp_subset fU oo*)
qed

lemma *continuous_on_inverse_closed_map*:
assumes *contf*: *continuous_on S f*
and *imf*: $f\ `\ S = T$
and *injf*: $\bigwedge x.\ x \in S \Longrightarrow g(f\ x) = x$
and *oo*: $\bigwedge U.$ *closedin* (*top_of_set S*) $U \Longrightarrow$ *closedin* (*top_of_set T*) $(f\ `\ U)$
shows *continuous_on T g*
proof −
from *imf injf* **have** *gTS*: $g\ `\ T = S$
by *force*
from *imf injf* **have** *fU*: $U \subseteq S \Longrightarrow (f\ `\ U) = T \cap g -` U$ **for** *U*
by *force*
show *?thesis*
by (*simp add*: *continuous_on_closed* [*of T g*] *gTS*) (*metis closedin_imp_subset fU oo*)
qed

lemma *homeomorphism_injective_open_map*:
assumes *contf*: *continuous_on S f*
and *imf*: *f ‘ S = T*
and *injf*: *inj_on f S*
and *oo*: $\bigwedge U$. *openin* (*top_of_set S*) *U* $\Longrightarrow$ *openin* (*top_of_set T*) (*f ‘ U*)
obtains *g* **where** *homeomorphism S T f g*
proof
have *continuous_on T* (*inv_into S f*)
by (*metis contf continuous_on_inverse_open_map imf injf inv_into_f_f oo*)
with *imf injf contf* **show** *homeomorphism S T f* (*inv_into S f*)
by (*auto simp*: *homeomorphism_def*)
qed

lemma *homeomorphism_injective_closed_map*:
assumes *contf*: *continuous_on S f*
and *imf*: *f ‘ S = T*
and *injf*: *inj_on f S*
and *oo*: $\bigwedge U$. *closedin* (*top_of_set S*) *U* $\Longrightarrow$ *closedin* (*top_of_set T*) (*f ‘ U*)
obtains *g* **where** *homeomorphism S T f g*
proof
have *continuous_on T* (*inv_into S f*)
by (*metis contf continuous_on_inverse_closed_map imf injf inv_into_f_f oo*)
with *imf injf contf* **show** *homeomorphism S T f* (*inv_into S f*)
by (*auto simp*: *homeomorphism_def*)
qed

lemma *homeomorphism_imp_open_map*:
assumes *hom*: *homeomorphism S T f g*
and *oo*: *openin* (*top_of_set S*) *U*
shows *openin* (*top_of_set T*) (*f ‘ U*)
proof −
from *hom oo* **have** [*simp*]: *f ‘ U = T* ∩ *g* −‘ *U*
using *openin_subset* **by** (*fastforce simp*: *homeomorphism_def rev_image_eqI*)
from *hom* **have** *continuous_on T g*
unfolding *homeomorphism_def* **by** *blast*
moreover have *g ‘ T = S*
by (*metis hom homeomorphism_def*)
ultimately show *?thesis*
by (*simp add*: *continuous_on_open oo*)
qed

lemma *homeomorphism_imp_closed_map*:
assumes *hom*: *homeomorphism S T f g*
and *oo*: *closedin* (*top_of_set S*) *U*
shows *closedin* (*top_of_set T*) (*f ‘ U*)
proof −
from *hom oo* **have** [*simp*]: *f ‘ U = T* ∩ *g* −‘ *U*
using *closedin_subset* **by** (*fastforce simp*: *homeomorphism_def rev_image_eqI*)

```
  from hom have continuous_on T g
    unfolding homeomorphism_def by blast
  moreover have g ‘ T = S
    by (metis hom homeomorphism_def)
  ultimately show ?thesis
    by (simp add: continuous_on_closed oo)
qed
```

2.3.9 Seperability

```
lemma subset_second_countable:
  obtains ℬ :: 'a:: second_countable_topology set set
    where countable ℬ
          {} ∉ ℬ
          ⋀C. C ∈ ℬ ⟹ openin(top_of_set S) C
          ⋀T. openin(top_of_set S) T ⟹ ∃𝒰. 𝒰 ⊆ ℬ ∧ T = ⋃𝒰
proof −
  obtain ℬ :: 'a set set
    where countable ℬ
      and opeB: ⋀C. C ∈ ℬ ⟹ openin(top_of_set S) C
      and ℬ:    ⋀T. openin(top_of_set S) T ⟹ ∃𝒰. 𝒰 ⊆ ℬ ∧ T = ⋃𝒰
  proof −
    obtain 𝒞 :: 'a set set
      where countable 𝒞 and ope: ⋀C. C ∈ 𝒞 ⟹ open C
        and 𝒞: ⋀S. open S ⟹ ∃ U. U ⊆ 𝒞 ∧ S = ⋃ U
      by (metis univ_second_countable that)
    show ?thesis
    proof
      show countable ((λC. S ∩ C) ‘ 𝒞)
        by (simp add: ‹countable 𝒞›)
      show ⋀C. C ∈ (∩) S ‘ 𝒞 ⟹ openin (top_of_set S) C
        using ope by auto
      show ⋀T. openin (top_of_set S) T ⟹ ∃𝒰⊆(∩) S ‘ 𝒞. T = ⋃𝒰
        by (metis 𝒞 image_mono inf_Sup openin_open)
    qed
  qed
  show ?thesis
  proof
    show countable (ℬ − {{}})
      using ‹countable ℬ› by blast
    show ⋀C. ⟦C ∈ ℬ − {{}}⟧ ⟹ openin (top_of_set S) C
      by (simp add: ‹⋀C. C ∈ ℬ ⟹ openin (top_of_set S) C›)
    show ∃𝒰⊆ℬ − {{}}. T = ⋃𝒰 if openin (top_of_set S) T for T
      using ℬ [OF that]
      apply clarify
      by (rule_tac x=𝒰 − {{}} in exI, auto)
  qed auto
qed
```

lemma *Lindelof_openin*:
fixes $\mathcal{F}$ :: *'a::second_countable_topology set set*
assumes $\bigwedge S.\ S \in \mathcal{F} \Longrightarrow$ *openin* (*top_of_set U*) *S*
obtains $\mathcal{F}'$ **where** $\mathcal{F}' \subseteq \mathcal{F}$ *countable* $\mathcal{F}'$ $\bigcup \mathcal{F}' = \bigcup \mathcal{F}$
proof −
have $\bigwedge S.\ S \in \mathcal{F} \Longrightarrow \exists\, T.$ *open* $T \wedge S = U \cap T$
using *assms* **by** (*simp add*: *openin_open*)
then obtain *tf* **where** *tf*: $\bigwedge S.\ S \in \mathcal{F} \Longrightarrow$ *open* (*tf S*) $\wedge$ ($S = U \cap$ *tf S*)
by *metis*
have [*simp*]: $\bigwedge \mathcal{F}'.\ \mathcal{F}' \subseteq \mathcal{F} \Longrightarrow \bigcup \mathcal{F}' = U \cap \bigcup$ (*tf* ‘ $\mathcal{F}'$)
using *tf* **by** *fastforce*
obtain $\mathcal{G}$ **where** *countable* $\mathcal{G} \wedge \mathcal{G} \subseteq$ *tf* ‘ $\mathcal{F}$ $\bigcup \mathcal{G} = \bigcup$ (*tf* ‘ $\mathcal{F}$)
using *tf* **by** (*force intro*: *Lindelof* [*of tf* ‘ $\mathcal{F}$])
then obtain $\mathcal{F}'$ **where** $\mathcal{F}'$: $\mathcal{F}' \subseteq \mathcal{F}$ *countable* $\mathcal{F}'$ $\bigcup \mathcal{F}' = \bigcup \mathcal{F}$
by (*clarsimp simp add*: *countable_subset_image*)
then show *?thesis* **..**
qed

2.3.10 Closed Maps

lemma *continuous_imp_closed_map*:
fixes *f* :: *'a::t2_space* ⇒ *'b::t2_space*
assumes *closedin* (*top_of_set S*) *U*
continuous_on S f f ‘ *S* = *T compact S*
shows *closedin* (*top_of_set T*) (*f* ‘ *U*)
by (*metis assms closedin_compact_eq compact_continuous_image continuous_on_subset subset_image_iff*)

lemma *closed_map_restrict*:
assumes *cloU*: *closedin* (*top_of_set* (*S* ∩ *f* −‘ *T'*)) *U*
and *cc*: $\bigwedge U.$ *closedin* (*top_of_set S*) *U* $\Longrightarrow$ *closedin* (*top_of_set T*) (*f* ‘ *U*)
and $T' \subseteq T$
shows *closedin* (*top_of_set T'*) (*f* ‘ *U*)
proof −
obtain *V* **where** *closed V U* = *S* ∩ *f* −‘ *T'* ∩ *V*
using *cloU* **by** (*auto simp*: *closedin_closed*)
with *cc* [*of S* ∩ *V*] ‹*T'* ⊆ *T*› **show** *?thesis*
by (*fastforce simp add*: *closedin_closed*)
qed

2.3.11 Open Maps

lemma *open_map_restrict*:
assumes *opeU*: *openin* (*top_of_set* (*S* ∩ *f* −‘ *T'*)) *U*
and *oo*: $\bigwedge U.$ *openin* (*top_of_set S*) *U* $\Longrightarrow$ *openin* (*top_of_set T*) (*f* ‘ *U*)
and $T' \subseteq T$
shows *openin* (*top_of_set T'*) (*f* ‘ *U*)
proof −
obtain *V* **where** *open V U* = *S* ∩ *f* −‘ *T'* ∩ *V*
using *opeU* **by** (*auto simp*: *openin_open*)

with *oo* [*of* $S \cap V$] ‹$T' \subseteq T$› **show** *?thesis*
by (*fastforce simp add*: *openin_open*)
qed

2.3.12 Quotient maps

lemma *quotient_map_imp_continuous_open*:
assumes T: $f \in S \to T$
and *ope*: $\bigwedge U.\ U \subseteq T$
$\Longrightarrow$ (*openin* (*top_of_set* S) ($S \cap f -` U$) $\longleftrightarrow$
openin (*top_of_set* T) U)
shows *continuous_on* S f
proof −
have [*simp*]: $S \cap f -` f ` S = S$ **by** *auto*
show *?thesis*
by (*meson* T *continuous_on_open_gen ope openin_imp_subset*)
qed

lemma *quotient_map_imp_continuous_closed*:
assumes T: $f \in S \to T$
and *ope*: $\bigwedge U.\ U \subseteq T$
$\Longrightarrow$ (*closedin* (*top_of_set* S) ($S \cap f -` U$) $\longleftrightarrow$
closedin (*top_of_set* T) U)
shows *continuous_on* S f
proof −
have [*simp*]: $S \cap f -` f ` S = S$ **by** *auto*
show *?thesis*
by (*meson* T *closedin_imp_subset continuous_on_closed_gen ope*)
qed

lemma *open_map_imp_quotient_map*:
assumes *contf*: *continuous_on* S f
and T: $T \subseteq f ` S$
and *ope*: $\bigwedge T.$ *openin* (*top_of_set* S) T
$\Longrightarrow$ *openin* (*top_of_set* ($f ` S$)) ($f ` T$)
shows *openin* (*top_of_set* S) ($S \cap f -` T$) =
openin (*top_of_set* ($f ` S$)) T
proof −
have $T = f ` (S \cap f -` T)$
using T **by** *blast*
then show *?thesis*
using *ope contf continuous_on_open* **by** *metis*
qed

lemma *closed_map_imp_quotient_map*:
assumes *contf*: *continuous_on* S f
and T: $T \subseteq f ` S$
and *ope*: $\bigwedge T.$ *closedin* (*top_of_set* S) T
$\Longrightarrow$ *closedin* (*top_of_set* ($f ` S$)) ($f ` T$)

```
    shows openin (top_of_set S) (S ∩ f −‘ T) ⟷
          openin (top_of_set (f ‘ S)) T
        (is ?lhs = ?rhs)
proof
  assume ?lhs
  then have ∗: closedin (top_of_set S) (S − (S ∩ f −‘ T))
    using closedin_diff by fastforce
  have [simp]: (f ‘ S − f ‘ (S − (S ∩ f −‘ T))) = T
    using T by blast
  show ?rhs
    using ope [OF ∗, unfolded closedin_def] by auto
next
  assume ?rhs
  with contf show ?lhs
    by (auto simp: continuous_on_open)
qed

lemma continuous_right_inverse_imp_quotient_map:
  assumes contf: continuous_on S f and imf: f ∈ S → T
      and contg: continuous_on T g and img: g ∈ T → S
      and fg [simp]: ⋀y. y ∈ T ⟹ f(g y) = y
      and U: U ⊆ T
    shows openin (top_of_set S) (S ∩ f −‘ U) ⟷
          openin (top_of_set T) U
        (is ?lhs = ?rhs)
proof −
  have f: ⋀Z. openin (top_of_set (f ‘ S)) Z ⟹
              openin (top_of_set S) (S ∩ f −‘ Z)
  and  g: ⋀Z. openin (top_of_set (g ‘ T)) Z ⟹
              openin (top_of_set T) (T ∩ g −‘ Z)
    using contf contg by (auto simp: continuous_on_open)
  show ?thesis
  proof
    have T ∩ g −‘ (g ‘ T ∩ (S ∩ f −‘ U)) = {x ∈ T. f (g x) ∈ U}
      using imf img by blast
    also have ... = U
      using U by auto
    finally have eq: T ∩ g −‘ (g ‘ T ∩ (S ∩ f −‘ U)) = U .
    assume ?lhs
    then have ∗: openin (top_of_set (g ‘ T)) (g ‘ T ∩ (S ∩ f −‘ U))
    by (meson img openin_Int openin_subtopology_Int_subset openin_subtopology_self
image_subset_iff_funcset)
    show ?rhs
      using g [OF ∗] eq by auto
  next
    assume rhs: ?rhs
    show ?lhs
      using assms continuous_openin_preimage rhs by blast
  qed
```

qed

lemma *continuous_left_inverse_imp_quotient_map*:
assumes *continuous_on S f*
and *continuous_on* (*f ' S*) *g*
and $\bigwedge x.\ x \in S \Longrightarrow g(f\ x) = x$
and $U \subseteq f\ `\ S$
shows *openin* (*top_of_set S*) ($S \cap f\ -`\ U$) $\longleftrightarrow$
openin (*top_of_set* (*f ' S*)) *U*
using *assms*
by (*intro continuous_right_inverse_imp_quotient_map*) *auto*

lemma *continuous_imp_quotient_map*:
fixes *f* :: *'a::t2_space* ⇒ *'b::t2_space*
assumes *continuous_on S f f ' S* = *T compact S U* ⊆ *T*
shows *openin* (*top_of_set S*) ($S \cap f\ -`\ U$) $\longleftrightarrow$
openin (*top_of_set T*) *U*
by (*simp add*: *assms closed_map_imp_quotient_map continuous_imp_closed_map*)

2.3.13 Pasting lemmas for functions, for of casewise definitions

on open sets

lemma *pasting_lemma*:
assumes *ope*: $\bigwedge i.\ i \in I \Longrightarrow$ *openin X* (*T i*)
and *cont*: $\bigwedge i.\ i \in I \Longrightarrow$ *continuous_map*(*subtopology X* (*T i*)) *Y* (*f i*)
and *f*: $\bigwedge i\ j\ x.\ [\![i \in I; j \in I; x \in$ *topspace X* $\cap$ *T i* $\cap$ *T j*$]\!] \Longrightarrow f\ i\ x = f\ j\ x$
and *g*: $\bigwedge x.\ x \in$ *topspace X* $\Longrightarrow \exists j.\ j \in I \wedge x \in T\ j \wedge g\ x = f\ j\ x$
shows *continuous_map X Y g*
unfolding *continuous_map_openin_preimage_eq*
proof (*intro conjI allI impI*)
show *g* ∈ *topspace X* → *topspace Y*
using *g cont continuous_map_image_subset_topspace* **by** *fastforce*
next
fix *U*
assume *Y*: *openin Y U*
have *T*: *T i* ⊆ *topspace X* **if** *i* ∈ *I* **for** *i*
using *ope* **by** (*simp add*: *openin_subset that*)
have ∗: *topspace X* ∩ *g* −` *U* = ($\bigcup i \in I.\ T\ i \cap f\ i\ -`\ U$)
using *f g T* **by** *fastforce*
have $\bigwedge i.\ i \in I \Longrightarrow$ *openin X* (*T i* ∩ *f i* −` *U*)
using *cont* **unfolding** *continuous_map_openin_preimage_eq*
by (*metis Y T inf.commute inf_absorb1 ope topspace_subtopology openin_trans_full*)
then show *openin X* (*topspace X* ∩ *g* −` *U*)
by (*auto simp*: ∗)
qed

lemma *pasting_lemma_exists*:
assumes *X*: *topspace X* ⊆ ($\bigcup i \in I.\ T\ i$)

and *ope*: $\bigwedge i.\ i \in I \Longrightarrow$ *openin* X $(T\ i)$
and *cont*: $\bigwedge i.\ i \in I \Longrightarrow$ *continuous_map* $(subtopology\ X\ (T\ i))\ Y\ (f\ i)$
and f: $\bigwedge i\ j\ x.\ [\![i \in I;\ j \in I;\ x \in topspace\ X \cap T\ i \cap T\ j]\!] \Longrightarrow f\ i\ x = f\ j\ x$
obtains g **where** *continuous_map* $X\ Y\ g\ \bigwedge x\ i.\ [\![i \in I;\ x \in topspace\ X \cap T\ i]\!] \Longrightarrow g\ x = f\ i\ x$
proof
let *?h* $= \lambda x.\ f\ (SOME\ i.\ i \in I \wedge x \in T\ i)\ x$
show *continuous_map* $X\ Y$ *?h*
apply (*rule pasting_lemma* [*OF ope cont*])
apply (*blast intro*: f)+
by (*metis* (*no_types*, *lifting*) *UN_E X subsetD someI_ex*)
show $f\ (SOME\ i.\ i \in I \wedge x \in T\ i)\ x = f\ i\ x$ **if** $i \in I\ x \in topspace\ X \cap T\ i$ **for** $i\ x$
by (*metis* (*no_types*, *lifting*) *IntD2 IntI f someI_ex that*)
qed

lemma *pasting_lemma_locally_finite*:
assumes *fin*: $\bigwedge x.\ x \in topspace\ X \Longrightarrow \exists V.\ openin\ X\ V \wedge x \in V \wedge finite\ \{i \in I.\ T\ i \cap V \neq \{\}\}$
and *clo*: $\bigwedge i.\ i \in I \Longrightarrow$ *closedin* $X\ (T\ i)$
and *cont*: $\bigwedge i.\ i \in I \Longrightarrow$ *continuous_map*$(subtopology\ X\ (T\ i))\ Y\ (f\ i)$
and f: $\bigwedge i\ j\ x.\ [\![i \in I;\ j \in I;\ x \in topspace\ X \cap T\ i \cap T\ j]\!] \Longrightarrow f\ i\ x = f\ j\ x$
and g: $\bigwedge x.\ x \in topspace\ X \Longrightarrow \exists j.\ j \in I \wedge x \in T\ j \wedge g\ x = f\ j\ x$
shows *continuous_map* $X\ Y\ g$
unfolding *continuous_map_closedin_preimage_eq*
proof (*intro conjI allI impI*)
show $g \in topspace\ X \to topspace\ Y$
using g *cont continuous_map_image_subset_topspace* **by** *fastforce*
next
fix U
assume Y: *closedin* $Y\ U$
have T: $T\ i \subseteq topspace\ X$ **if** $i \in I$ **for** i
using *clo* **by** (*simp add*: *closedin_subset that*)
have $*$: *topspace* $X \cap g\ -`\ U = (\bigcup i \in I.\ T\ i \cap f\ i\ -`\ U)$
using $f\ g\ T$ **by** *fastforce*
have *cTf*: $\bigwedge i.\ i \in I \Longrightarrow$ *closedin* $X\ (T\ i \cap f\ i\ -`\ U)$
using *cont* **unfolding** *continuous_map_closedin_preimage_eq topspace_subtopology*
by (*simp add*: *Int_absorb1 T Y clo closedin_closed_subtopology*)
have *sub*: $\{Z \in (\lambda i.\ T\ i \cap f\ i\ -`\ U)\ `\ I.\ Z \cap V \neq \{\}\}$
$\subseteq (\lambda i.\ T\ i \cap f\ i\ -`\ U)\ `\ \{i \in I.\ T\ i \cap V \neq \{\}\}$ **for** V
by *auto*
have *1*: $(\bigcup i \in I.\ T\ i \cap f\ i\ -`\ U) \subseteq topspace\ X$
using T **by** *blast*
then have *locally_finite_in* $X\ ((\lambda i.\ T\ i \cap f\ i\ -`\ U)\ `\ I)$
unfolding *locally_finite_in_def*
using *finite_subset* [*OF sub*] *fin* **by** *force*
then show *closedin* X $(topspace\ X \cap g\ -`\ U)$
by (*smt* (*verit*, *best*) $*$ *cTf closedin_locally_finite_Union image_iff*)
qed

Likewise on closed sets, with a finiteness assumption

lemma *pasting_lemma_closed*:
assumes *fin*: *finite I*
and *clo*: $\bigwedge i.\ i \in I \Longrightarrow$ *closedin X* (*T i*)
and *cont*: $\bigwedge i.\ i \in I \Longrightarrow$ *continuous_map*(*subtopology X* (*T i*)) *Y* (*f i*)
and *f*: $\bigwedge i\ j\ x.\ [\![i \in I; j \in I; x \in$ *topspace X* $\cap$ *T i* $\cap$ *T j*$]\!] \Longrightarrow f\ i\ x = f\ j\ x$
and *g*: $\bigwedge x.\ x \in$ *topspace X* $\Longrightarrow \exists j.\ j \in I \wedge x \in T\ j \wedge g\ x = f\ j\ x$
shows *continuous_map X Y g*
using *pasting_lemma_locally_finite* [*OF _ clo cont f g*] *fin* **by** *auto*

lemma *pasting_lemma_exists_locally_finite*:
assumes *fin*: $\bigwedge x.\ x \in$ *topspace X* $\Longrightarrow \exists V.$ *openin X V* $\wedge\ x \in V \wedge$ *finite* $\{i \in I.\ T\ i \cap V \neq \{\}\}$
and *X*: *topspace X* $\subseteq \bigcup (T\ `\ I)$
and *clo*: $\bigwedge i.\ i \in I \Longrightarrow$ *closedin X* (*T i*)
and *cont*: $\bigwedge i.\ i \in I \Longrightarrow$ *continuous_map*(*subtopology X* (*T i*)) *Y* (*f i*)
and *f*: $\bigwedge i\ j\ x.\ [\![i \in I; j \in I; x \in$ *topspace X* $\cap$ *T i* $\cap$ *T j*$]\!] \Longrightarrow f\ i\ x = f\ j\ x$
and *g*: $\bigwedge x.\ x \in$ *topspace X* $\Longrightarrow \exists j.\ j \in I \wedge x \in T\ j \wedge g\ x = f\ j\ x$
obtains *g* **where** *continuous_map X Y g* $\bigwedge x\ i.\ [\![i \in I;\ x \in$ *topspace X* $\cap$ *T i*$]\!] \Longrightarrow g\ x = f\ i\ x$
proof
show *continuous_map X Y* ($\lambda x.\ f$(@*i*. $i \in I \wedge x \in T\ i$) *x*)
apply (*rule pasting_lemma_locally_finite* [*OF fin*])
apply (*blast intro*: *assms*)+
by (*metis* (*no_types*, *lifting*) *UN_E X set_rev_mp someI_ex*)
next
fix *x i*
assume $i \in I$ **and** $x \in$ *topspace X* $\cap$ *T i*
then show *f* (*SOME i*. $i \in I \wedge x \in T\ i$) $x = f\ i\ x$
by (*metis* (*mono_tags*, *lifting*) *IntE IntI f someI2*)
qed

lemma *pasting_lemma_exists_closed*:
assumes *fin*: *finite I*
and *X*: *topspace X* $\subseteq \bigcup (T\ `\ I)$
and *clo*: $\bigwedge i.\ i \in I \Longrightarrow$ *closedin X* (*T i*)
and *cont*: $\bigwedge i.\ i \in I \Longrightarrow$ *continuous_map*(*subtopology X* (*T i*)) *Y* (*f i*)
and *f*: $\bigwedge i\ j\ x.\ [\![i \in I; j \in I; x \in$ *topspace X* $\cap$ *T i* $\cap$ *T j*$]\!] \Longrightarrow f\ i\ x = f\ j\ x$
obtains *g* **where** *continuous_map X Y g* $\bigwedge x\ i.\ [\![i \in I;\ x \in$ *topspace X* $\cap$ *T i*$]\!] \Longrightarrow g\ x = f\ i\ x$
proof
show *continuous_map X Y* ($\lambda x.\ f$ (*SOME i*. $i \in I \wedge x \in T\ i$) *x*)
apply (*rule pasting_lemma_closed* [*OF* ‹*finite I*› *clo cont*])
apply (*blast intro*: *f*)+
by (*metis* (*mono_tags*, *lifting*) *UN_iff X someI_ex subset_iff*)
next
fix *x i*
assume $i \in I\ x \in$ *topspace X* $\cap$ *T i*
then show *f* (*SOME i*. $i \in I \wedge x \in T\ i$) $x = f\ i\ x$

```
    by (metis (no_types, lifting) IntD2 IntI f someI_ex)
qed

lemma continuous_map_cases:
  assumes f: continuous_map (subtopology X (X closure_of {x. P x})) Y f
      and g: continuous_map (subtopology X (X closure_of {x. ¬ P x})) Y g
      and fg: ⋀x. x ∈ X frontier_of {x. P x} ⟹ f x = g x
  shows continuous_map X Y (λx. if P x then f x else g x)
proof (rule pasting_lemma_closed)
  let ?f = λb. if b then f else g
  let ?g = λx. if P x then f x else g x
  let ?T = λb. if b then X closure_of {x. P x} else X closure_of {x. ~P x}
  show finite {True,False} by auto
  have eq: topspace X - Collect P = topspace X ∩ {x. ¬ P x}
    by blast
  show ?f i x = ?f j x
    if i ∈ {True,False} j ∈ {True,False} and x: x ∈ topspace X ∩ ?T i ∩ ?T j for
i j x
  proof -
    have f x = g x if i ¬ j
     by (smt (verit, best) Diff_Diff_Int closure_of_interior_of closure_of_restrict
eq fg
          frontier_of_closures interior_of_complement that x)
    moreover
    have g x = f x
      if x ∈ X closure_of {x. ¬ P x} x ∈ X closure_of Collect P ¬ i j for x
      by (metis IntI closure_of_restrict eq fg frontier_of_closures that)
    ultimately show ?thesis
      using that by (auto simp flip: closure_of_restrict)
  qed
  show ∃j. j ∈ {True,False} ∧ x ∈ ?T j ∧ (if P x then f x else g x) = ?f j x
    if x ∈ topspace X for x
    by simp (metis in_closure_of mem_Collect_eq that)
qed (auto simp: f g)

lemma continuous_map_cases_alt:
  assumes f: continuous_map (subtopology X (X closure_of {x ∈ topspace X. P
x})) Y f
      and g: continuous_map (subtopology X (X closure_of {x ∈ topspace X. ~P
x})) Y g
      and fg: ⋀x. x ∈ X frontier_of {x ∈ topspace X. P x} ⟹ f x = g x
    shows continuous_map X Y (λx. if P x then f x else g x)
  apply (rule continuous_map_cases)
  using assms
    apply (simp_all add: Collect_conj_eq closure_of_restrict [symmetric] fron-
tier_of_restrict [symmetric])
  done

lemma continuous_map_cases_function:
```

assumes *contp*: *continuous_map X Z p*
and *contf*: *continuous_map* (*subtopology X* {*x* ∈ *topspace X*. *p x* ∈ *Z closure_of U*}) *Y f*
and *contg*: *continuous_map* (*subtopology X* {*x* ∈ *topspace X*. *p x* ∈ *Z closure_of* (*topspace Z* − *U*)}) *Y g*
and *fg*: ⋀*x*. ⟦*x* ∈ *topspace X*; *p x* ∈ *Z frontier_of U*⟧ ⟹ *f x* = *g x*
shows *continuous_map X Y* (λ*x*. *if p x* ∈ *U then f x else g x*)
proof (*rule continuous_map_cases_alt*)
show *continuous_map* (*subtopology X* (*X closure_of* {*x* ∈ *topspace X*. *p x* ∈ *U*})) *Y f*
proof (*rule continuous_map_from_subtopology_mono*)
let *?T* = {*x* ∈ *topspace X*. *p x* ∈ *Z closure_of U*}
show *continuous_map* (*subtopology X ?T*) *Y f*
by (*simp add*: *contf*)
show *X closure_of* {*x* ∈ *topspace X*. *p x* ∈ *U*} ⊆ *?T*
by (*rule continuous_map_closure_preimage_subset* [*OF contp*])
qed
show *continuous_map* (*subtopology X* (*X closure_of* {*x* ∈ *topspace X*. *p x* ∉ *U*})) *Y g*
proof (*rule continuous_map_from_subtopology_mono*)
let *?T* = {*x* ∈ *topspace X*. *p x* ∈ *Z closure_of* (*topspace Z* − *U*)}
show *continuous_map* (*subtopology X ?T*) *Y g*
by (*simp add*: *contg*)
have *X closure_of* {*x* ∈ *topspace X*. *p x* ∉ *U*} ⊆ *X closure_of* {*x* ∈ *topspace X*. *p x* ∈ *topspace Z* − *U*}
by (*smt* (*verit*) *Collect_mono_iff DiffI closure_of_mono continuous_map contp image_subset_iff*)
then show *X closure_of* {*x* ∈ *topspace X*. *p x* ∉ *U*} ⊆ *?T*
by (*rule order_trans* [*OF _ continuous_map_closure_preimage_subset* [*OF contp*]])
qed
next
show *f x* = *g x* **if** *x* ∈ *X frontier_of* {*x* ∈ *topspace X*. *p x* ∈ *U*} **for** *x*
using *that continuous_map_frontier_frontier_preimage_subset* [*OF contp*, *of U*] *fg* **by** *blast*
qed

2.3.14 Retractions

definition *retraction* :: ('*a*::*topological_space*) *set* ⇒ '*a set* ⇒ ('*a* ⇒ '*a*) ⇒ *bool*
where *retraction S T r* ⟷
T ⊆ *S* ∧ *continuous_on S r* ∧ *r* ∈ *S* → *T* ∧ (∀ *x*∈*T*. *r x* = *x*)

definition *retract_of* (**infixl** *retract'_of 50*) **where**
T retract_of S ⟷ (∃ *r*. *retraction S T r*)

lemma *retraction_idempotent*: *retraction S T r* ⟹ *x* ∈ *S* ⟹ *r* (*r x*) = *r x*
unfolding *retraction_def* **by** *auto*

Preservation of fixpoints under (more general notion of) retraction

lemma *invertible_fixpoint_property*:
fixes S :: *'a*::*topological_space set*
and T :: *'b*::*topological_space set*
assumes *contt*: *continuous_on* T i
and $i \in T \to S$
and *contr*: *continuous_on* S r
and $r \in S \to T$
and *ri*: $\bigwedge y.\ y \in T \Longrightarrow r\ (i\ y) = y$
and *FP*: $\bigwedge f.\ [\![\mathit{continuous\_on}\ S\ f; f \in S \to S]\!] \Longrightarrow \exists\, x{\in}S.\ f\ x = x$
and *contg*: *continuous_on* T g
and $g \in T \to T$
obtains y **where** $y \in T$ **and** $g\ y = y$
proof −
have $\exists\, x{\in}S.\ (i \circ g \circ r)\ x = x$
proof (*rule FP*)
show *continuous_on* S $(i \circ g \circ r)$
by (*metis assms*(*4*) *assms*(*8*) *contg continuous_on_compose continuous_on_subset contr contt funcset_image*)
show $(i \circ g \circ r) \in S \to S$
using *assms*(*2*,*4*,*8*) **by** *force*
qed
then obtain x **where** x: $x \in S$ $(i \circ g \circ r)\ x = x$ **..**
then have $*$: $g\ (r\ x) \in T$
using *assms*(*4*,*8*) **by** *auto*
have $r\ ((i \circ g \circ r)\ x) = r\ x$
using x **by** *auto*
then show *?thesis*
using $*$ *ri that* **by** *auto*
qed

lemma *homeomorphic_fixpoint_property*:
fixes S :: *'a*::*topological_space set*
and T :: *'b*::*topological_space set*
assumes S *homeomorphic* T
shows $(\forall f.\ \mathit{continuous\_on}\ S\ f \wedge f \in S \to S \longrightarrow (\exists\, x{\in}S.\ f\ x = x)) \longleftrightarrow$
$(\forall g.\ \mathit{continuous\_on}\ T\ g \wedge g \in T \to T \longrightarrow (\exists\, y{\in}T.\ g\ y = y))$
(**is** *?lhs* = *?rhs*)
proof −
obtain r i **where** r:
$\forall\, x{\in}S.\ i\ (r\ x) = x$ r ‘ $S = T$ *continuous_on* S r
$\forall\, y{\in}T.\ r\ (i\ y) = y$ i ‘ $T = S$ *continuous_on* T i
using *assms* **unfolding** *homeomorphic_def homeomorphism_def* **by** *blast*
show *?thesis*
proof
assume *?lhs*
with r **show** *?rhs*
by (*smt* (*verit*, *del_insts*) *Pi_iff image_eqI invertible_fixpoint_property*[*of* T i S r])
next

assume *?rhs*
with *r* **show** *?lhs*
by (*smt* (*verit*, *del_insts*) *Pi_iff image_eqI invertible_fixpoint_property*[*of S r T i*])
qed
qed

lemma *retract_fixpoint_property*:
fixes *f* :: *'a*::*topological_space* ⇒ *'b*::*topological_space*
and *S* :: *'a set*
assumes *T retract_of S*
and *FP*: ⋀*f*. ⟦*continuous_on S f*; *f* ∈ *S* → *S*⟧ ⟹ ∃ *x*∈*S*. *f x* = *x*
and *contg*: *continuous_on T g*
and *g* ∈ *T* → *T*
obtains *y* **where** *y* ∈ *T* **and** *g y* = *y*
proof −
obtain *h* **where** *retraction S T h*
using *assms*(*1*) **unfolding** *retract_of_def* **..**
then show *?thesis*
unfolding *retraction_def*
using *invertible_fixpoint_property*[*OF continuous_on_id _ _ _ _ FP*]
by (*smt* (*verit*, *del_insts*) *Pi_iff assms*(*4*) *contg subsetD that*)
qed

lemma *retraction*:
retraction S T r ⟷ *T* ⊆ *S* ∧ *continuous_on S r* ∧ *r* ‘ *S* = *T* ∧ (∀ *x* ∈ *T*. *r x* = *x*)
by (*force simp*: *retraction_def simp flip*: *image_subset_iff_funcset*)

lemma *retractionE*: — yields properties normalized wrt. simp – less likely to loop
assumes *retraction S T r*
obtains *T* = *r* ‘ *S r* ∈ *S* → *S continuous_on S r* ⋀*x*. *x* ∈ *S* ⟹ *r* (*r x*) = *r x*
proof (*rule that*)
from *retraction* [*of S T r*] *assms*
have *T* ⊆ *S continuous_on S r r* ‘ *S* = *T* **and** ∀ *x* ∈ *T*. *r x* = *x*
by *simp_all*
then show *r* ∈ *S* → *S continuous_on S r*
by *auto*
then show *T* = *r* ‘ *S*
using ‹*r* ‘ *S* = *T*› **by** *blast*
from ‹∀ *x* ∈ *T*. *r x* = *x*› **have** *r x* = *x* **if** *x* ∈ *T* **for** *x*
using *that* **by** *simp*
with ‹*r* ‘ *S* = *T*› **show** *r* (*r x*) = *r x* **if** *x* ∈ *S* **for** *x*
using *that* **by** *auto*
qed

lemma *retract_ofE*: — yields properties normalized wrt. simp – less likely to loop
assumes *T retract_of S*
obtains *r* **where** *T* = *r* ‘ *S r* ∈ *S* → *S continuous_on S r* ⋀*x*. *x* ∈ *S* ⟹ *r* (*r*

$x) = r\ x$
proof −
from *assms* **obtain** r **where** *retraction* $S\ T\ r$
by (*auto simp add*: *retract_of_def*)
with *that* **show** *thesis*
by (*auto elim*: *retractionE*)
qed

lemma *retract_of_imp_extensible*:
assumes S *retract_of* T **and** *continuous_on* $S\ f$ **and** $f \in S \to U$
obtains g **where** *continuous_on* $T\ g\ g \in T \to U\ \bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
proof −
from ‹S *retract_of* T› **obtain** r **where** *retraction* $T\ S\ r$
by (*auto simp add*: *retract_of_def*)
then have *continuous_on* $T\ (f \circ r)$
by (*metis assms*(*2*) *continuous_on_compose retraction*)
then show *thesis*
by (*smt* (*verit*, *best*) *Pi_iff* ‹*retraction* $T\ S\ r$› *assms*(*3*) *comp_apply retraction_def that*)
qed

lemma *idempotent_imp_retraction*:
assumes *continuous_on* $S\ f$ **and** $f \in S \to S$ **and** $\bigwedge x.\ x \in S \Longrightarrow f(f\ x) = f\ x$
shows *retraction* $S\ (f\ `\ S)\ f$
by (*simp add*: *assms funcset_image retraction*)

lemma *retraction_subset*:
assumes *retraction* $S\ T\ r$ **and** $T \subseteq S'$ **and** $S' \subseteq S$
shows *retraction* $S'\ T\ r$
unfolding *retraction_def*
by (*metis assms continuous_on_subset image_mono image_subset_iff_funcset retraction*)

lemma *retract_of_subset*:
assumes T *retract_of* S **and** $T \subseteq S'$ **and** $S' \subseteq S$
shows T *retract_of* S'
by (*meson assms retract_of_def retraction_subset*)

lemma *retraction_refl* [*simp*]: *retraction* $S\ S\ (\lambda x.\ x)$
by (*simp add*: *retraction*)

lemma *retract_of_refl* [*iff*]: S *retract_of* S
unfolding *retract_of_def retraction_def*
using *continuous_on_id* **by** *blast*

lemma *retract_of_imp_subset*:
S *retract_of* $T \Longrightarrow S \subseteq T$
by (*simp add*: *retract_of_def retraction_def*)

lemma *retract_of_empty* [*simp*]:
  ({} *retract_of S*) ⟷ *S* = {} (*S retract_of* {}) ⟷ *S* = {}
by (*auto simp*: *retract_of_def retraction_def*)

lemma *retract_of_singleton* [*iff*]: ({*x*} *retract_of S*) ⟷ *x* ∈ *S*
  unfolding *retract_of_def retraction_def* **by** *force*

lemma *retraction_comp*:
  ⟦*retraction S T f*; *retraction T U g*⟧ ⟹ *retraction S U* (*g* ∘ *f*)
  by (*smt* (*verit*, *best*) *comp_apply continuous_on_compose image_comp retraction subset_iff*)

lemma *retract_of_trans* [*trans*]:
  assumes *S retract_of T* **and** *T retract_of U*
    shows *S retract_of U*
using *assms* **by** (*auto simp*: *retract_of_def intro*: *retraction_comp*)

lemma *closedin_retract*:
  fixes *S* :: *'a* :: *t2_space set*
  assumes *S retract_of T*
    shows *closedin* (*top_of_set T*) *S*
proof −
  obtain *r* **where** *r*: *S* ⊆ *T continuous_on T r r* ∈ *T* → *S* ⋀*x*. *x* ∈ *S* ⟹ *r x* = *x*
    using *assms* **by** (*auto simp*: *retract_of_def retraction_def*)
  have *S* = {*x*∈*T*. *x* = *r x*}
    using *r* **by** *force*
  also have ... = *T* ∩ ((λ*x*. (*x*, *r x*)) −‘ ({*y*. ∃ *x*. *y* = (*x*, *x*)}))
    unfolding *vimage_def mem_Times_iff fst_conv snd_conv*
    using *r*
    by *auto*
  also have *closedin* (*top_of_set T*) ...
    by (*rule continuous_closedin_preimage*) (*auto intro*!: *closed_diagonal continuous_on_Pair r*)
  finally show *?thesis* **.**
qed

lemma *closedin_self* [*simp*]: *closedin* (*top_of_set S*) *S*
  by *simp*

lemma *retract_of_closed*:
    fixes *S* :: *'a* :: *t2_space set*
    shows ⟦*closed T*; *S retract_of T*⟧ ⟹ *closed S*
  by (*metis closedin_retract closedin_closed_eq*)

lemma *retract_of_compact*:
    ⟦*compact T*; *S retract_of T*⟧ ⟹ *compact S*
  by (*metis compact_continuous_image retract_of_def retraction*)

lemma *retract_of_connected*:
⟦*connected T*; *S retract_of T*⟧ ⟹ *connected S*
by (*metis Topological_Spaces.connected_continuous_image retract_of_def retraction*)

lemma *retraction_openin_vimage_iff*:
assumes *r*: *retraction S T r* **and** *U* ⊆ *T*
shows *openin* (*top_of_set S*) (*S* ∩ *r* −‘ *U*) ⟷ *openin* (*top_of_set T*) *U* (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⟹ *?rhs*
using *r*
by (*smt* (*verit*, *del_insts*) *assms*(*2*) *continuous_right_inverse_imp_quotient_map image_subset_iff_funcset r retractionE retraction_def retraction_subset*)
show *?rhs* ⟹ *?lhs*
by (*metis continuous_on_open r retraction*)
qed

lemma *retract_of_Times*:
⟦*S retract_of S'*; *T retract_of T'*⟧ ⟹ (*S* × *T*) *retract_of* (*S'* × *T'*)
apply (*simp add*: *retract_of_def retraction_def Sigma_mono*, *clarify*)
apply (*rename_tac f g*)
apply (*rule_tac x*=λ*z*. ((*f* ∘ *fst*) *z*, (*g* ∘ *snd*) *z*) **in** *exI*)
apply (*rule conjI continuous_intros* | *erule continuous_on_subset* | *force*)+
done

2.3.15 Retractions on a topological space

definition *retract_of_space* :: *'a set* ⇒ *'a topology* ⇒ *bool* (**infix** *retract'_of'_space* *50*)
where *S retract_of_space X*
≡ *S* ⊆ *topspace X* ∧ (∃ *r*. *continuous_map X* (*subtopology X S*) *r* ∧ (∀ *x* ∈ *S*. *r x* = *x*))

lemma *retract_of_space_retraction_maps*:
S retract_of_space X ⟷ *S* ⊆ *topspace X* ∧ (∃ *r*. *retraction_maps X* (*subtopology X S*) *r id*)
by (*auto simp*: *retract_of_space_def retraction_maps_def*)

lemma *retract_of_space_section_map*:
S retract_of_space X ⟷ *S* ⊆ *topspace X* ∧ *section_map* (*subtopology X S*) *X id*
unfolding *retract_of_space_def retraction_maps_def section_map_def*
by (*auto simp*: *continuous_map_from_subtopology*)

lemma *retract_of_space_imp_subset*:
S retract_of_space X ⟹ *S* ⊆ *topspace X*
by (*simp add*: *retract_of_space_def*)

lemma *retract_of_space_topspace*:
  topspace X retract_of_space X
 using *retract_of_space_def* **by** *force*

lemma *retract_of_space_empty* [*simp*]:
  {} *retract_of_space X* ⟷ *X* = *trivial_topology*
 by (*auto simp*: *retract_of_space_def*)

lemma *retract_of_space_singleton* [*simp*]:
 {*a*} *retract_of_space X* ⟷ *a* ∈ *topspace X*
proof −
 have *continuous_map X* (*subtopology X* {*a*}) (λ*x*. *a*) ∧ (λ*x*. *a*) *a* = *a* **if** *a* ∈ *topspace X*
  using *that* **by** *simp*
 then show *?thesis*
  by (*force simp*: *retract_of_space_def*)
qed

lemma *retract_of_space_trans*:
 assumes *S retract_of_space X T retract_of_space subtopology X S*
 shows *T retract_of_space X*
 using *assms*
 apply (*simp add*: *retract_of_space_retraction_maps*)
by (*metis id_comp inf.absorb_iff2 retraction_maps_compose subtopology_subtopology*)

lemma *retract_of_space_subtopology*:
 assumes *S retract_of_space X S* ⊆ *U*
 shows *S retract_of_space subtopology X U*
 using *assms*
 apply (*clarsimp simp*: *retract_of_space_def*)
by (*metis continuous_map_from_subtopology inf.absorb2 subtopology_subtopology*)

lemma *retract_of_space_clopen*:
 assumes *openin X S closedin X S S* = {} ⟹ *X* = *trivial_topology*
 shows *S retract_of_space X*
proof (*cases S* = {})
 case *False*
 then obtain *a* **where** *a* ∈ *S*
  by *blast*
 show *?thesis*
  unfolding *retract_of_space_def*
 proof (*intro exI conjI*)
  show *S* ⊆ *topspace X*
   by (*simp add*: *assms closedin_subset*)
  have *continuous_map X X* (λ*x*. *if x* ∈ *S then x else a*)
  proof (*rule continuous_map_cases*)
   show *continuous_map* (*subtopology X* (*X closure_of* {*x*. *x* ∈ *S*})) *X* (λ*x*. *x*)
    by (*simp add*: *continuous_map_from_subtopology*)
   show *continuous_map* (*subtopology X* (*X closure_of* {*x*. *x* ∉ *S*})) *X* (λ*x*. *a*)

```
      using ‹S ⊆ topspace X› ‹a ∈ S› by force
    show x = a if x ∈ X frontier_of {x. x ∈ S} for x
      using assms that clopenin_eq_frontier_of by fastforce
  qed
  then show continuous_map X (subtopology X S) (λx. if x ∈ S then x else a)
  using ‹S ⊆ topspace X› ‹a ∈ S› by (auto simp: continuous_map_in_subtopology)
 qed auto
qed (use assms in auto)

lemma retract_of_space_disjoint_union:
 assumes openin X S openin X T and ST: disjnt S T S ∪ T = topspace X and
S = {} ⟹ X = trivial_topology
 shows S retract_of_space X
proof (rule retract_of_space_clopen)
 have S ∩ T = {}
  by (meson ST disjnt_def)
 then have S = topspace X - T
  using ST by auto
 then show closedin X S
  using ‹openin X T› by blast
qed (auto simp: assms)

lemma retraction_maps_section_image1:
 assumes retraction_maps X Y r s
 shows s ‘ (topspace Y) retract_of_space X
 unfolding retract_of_space_section_map
proof
 show s ‘ topspace Y ⊆ topspace X
   using assms continuous_map_image_subset_topspace retraction_maps_def
by blast
 show section_map (subtopology X (s ‘ topspace Y)) X id
  unfolding section_map_def
  using assms retraction_maps_to_retract_maps by blast
qed

lemma retraction_maps_section_image2:
  retraction_maps X Y r s
      ⟹ subtopology X (s ‘ (topspace Y)) homeomorphic_space Y
  using embedding_map_imp_homeomorphic_space homeomorphic_space_sym
section_imp_embedding_map
      section_map_def by blast

lemma hereditary_imp_retractive_property:
 assumes ⋀X S. P X ⟹ P(subtopology X S)
       ⋀X X'. X homeomorphic_space X' ⟹ (P X ⟷ Q X')
 assumes retraction_map X X' r P X
 shows Q X'
 by (meson assms retraction_map_def retraction_maps_section_image2)
```

2.3.16 Paths and path-connectedness

definition *pathin* :: *'a topology* ⇒ (*real* ⇒ *'a*) ⇒ *bool* **where**
pathin X g ≡ *continuous_map* (*subtopology euclideanreal* {*0..1*}) *X g*

lemma *pathin_compose*:
⟦*pathin X g*; *continuous_map X Y f*⟧ ⟹ *pathin Y* (*f* ∘ *g*)
by (*simp add*: *continuous_map_compose pathin_def*)

lemma *pathin_subtopology*:
pathin (*subtopology X S*) *g* ⟷ *pathin X g* ∧ (∀ *x* ∈ {*0..1*}. *g x* ∈ *S*)
by (*auto simp*: *pathin_def continuous_map_in_subtopology*)

lemma *pathin_const* [*simp*]: *pathin X* (λ*x*. *a*) ⟷ *a* ∈ *topspace X*
using *pathin_subtopology* **by** (*fastforce simp add*: *pathin_def*)

lemma *path_start_in_topspace*: *pathin X g* ⟹ *g 0* ∈ *topspace X*
by (*force simp*: *pathin_def continuous_map*)

lemma *path_finish_in_topspace*: *pathin X g* ⟹ *g 1* ∈ *topspace X*
by (*force simp*: *pathin_def continuous_map*)

lemma *path_image_subset_topspace*: *pathin X g* ⟹ *g* ∈ ({*0..1*}) → *topspace X*
by (*force simp*: *pathin_def continuous_map*)

definition *path_connected_space* :: *'a topology* ⇒ *bool*
where *path_connected_space X* ≡ ∀ *x* ∈ *topspace X*. ∀ *y* ∈ *topspace X*. ∃ *g*. *pathin X g* ∧ *g 0* = *x* ∧ *g 1* = *y*

definition *path_connectedin* :: *'a topology* ⇒ *'a set* ⇒ *bool*
where *path_connectedin X S* ≡ *S* ⊆ *topspace X* ∧ *path_connected_space*(*subtopology X S*)

lemma *path_connectedin_absolute* [*simp*]:
path_connectedin (*subtopology X S*) *S* ⟷ *path_connectedin X S*
by (*simp add*: *path_connectedin_def subtopology_subtopology*)

lemma *path_connectedin_subset_topspace*:
path_connectedin X S ⟹ *S* ⊆ *topspace X*
by (*simp add*: *path_connectedin_def*)

lemma *path_connectedin_subtopology*:
path_connectedin (*subtopology X S*) *T* ⟷ *path_connectedin X T* ∧ *T* ⊆ *S*
by (*auto simp*: *path_connectedin_def subtopology_subtopology inf.absorb2*)

lemma *path_connectedin*:
path_connectedin X S ⟷
S ⊆ *topspace X* ∧
(∀ *x* ∈ *S*. ∀ *y* ∈ *S*. ∃ *g*. *pathin X g* ∧ *g* ∈ {*0..1*} → *S* ∧ *g 0* = *x* ∧ *g 1* = *y*)
unfolding *path_connectedin_def path_connected_space_def pathin_def contin-*

uous_map_in_subtopology
by (*intro conj_cong refl ball_cong*) (*simp_all add*: *inf.absorb_iff2 flip*: *image_subset_iff_funcset*)

lemma *path_connectedin_topspace*:
path_connectedin X (*topspace X*) ⟷ *path_connected_space X*
by (*simp add*: *path_connectedin_def*)

lemma *path_connected_imp_connected_space*:
assumes *path_connected_space X*
shows *connected_space X*
proof −
have ∗: ∃ *S*. *connectedin X S* ∧ *g 0* ∈ *S* ∧ *g 1* ∈ *S* **if** *pathin X g* **for** *g*
proof (*intro exI conjI*)
have *continuous_map* (*subtopology euclideanreal* {*0..1*}) *X g*
using *connectedin_absolute that* **by** (*simp add*: *pathin_def*)
then show *connectedin X* (*g* ‘ {*0..1*})
by (*rule connectedin_continuous_map_image*) *auto*
qed *auto*
show *?thesis*
using *assms*
by (*auto intro*: ∗ *simp add*: *path_connected_space_def connected_space_subconnected Ball_def*)
qed

lemma *path_connectedin_imp_connectedin*:
path_connectedin X S ⟹ *connectedin X S*
by (*simp add*: *connectedin_def path_connected_imp_connected_space path_connectedin_def*)

lemma *path_connected_space_topspace_empty*:
path_connected_space trivial_topology
by (*simp add*: *path_connected_space_def*)

lemma *path_connectedin_empty* [*simp*]: *path_connectedin X* {}
by (*simp add*: *path_connectedin*)

lemma *path_connectedin_singleton* [*simp*]: *path_connectedin X* {*a*} ⟷ *a* ∈ *topspace X*
proof
show *path_connectedin X* {*a*} ⟹ *a* ∈ *topspace X*
by (*simp add*: *path_connectedin*)
show *a* ∈ *topspace X* ⟹ *path_connectedin X* {*a*}
unfolding *path_connectedin*
using *pathin_const* **by** *fastforce*
qed

lemma *path_connectedin_continuous_map_image*:
assumes *f*: *continuous_map X Y f* **and** *S*: *path_connectedin X S*
shows *path_connectedin Y* (*f* ‘ *S*)

proof −
have *fX*: $f \in (\mathit{topspace}\ X) \to \mathit{topspace}\ Y$
using *continuous_map_def f* **by** *fastforce*
show *?thesis*
unfolding *path_connectedin*
proof (*intro conjI ballI*; *clarify?*)
fix x
assume $x \in S$
show $f\ x \in \mathit{topspace}\ Y$
using S ‹$x \in S$› *fX path_connectedin_subset_topspace* **by** *fastforce*
next
fix $x\ y$
assume $x \in S$ **and** $y \in S$
then obtain g **where** g: $\mathit{pathin}\ X\ g\ g \in \{0..1\} \to S\ g\ 0 = x\ g\ 1 = y$
using S **by** (*force simp*: *path_connectedin*)
show $\exists g.\ \mathit{pathin}\ Y\ g \wedge g \in \{0..1\} \to f\ `\ S \wedge g\ 0 = f\ x \wedge g\ 1 = f\ y$
proof (*intro exI conjI*)
show $\mathit{pathin}\ Y\ (f \circ g)$
using ‹$\mathit{pathin}\ X\ g$› *f pathin_compose* **by** *auto*
qed (*use g* **in** *auto*)
qed
qed

lemma *path_connectedin_discrete_topology*:
$\mathit{path\_connectedin}\ (\mathit{discrete\_topology}\ U)\ S \longleftrightarrow S \subseteq U \wedge (\exists a.\ S \subseteq \{a\})$ (**is** *?lhs* = *?rhs*)
proof
show *?lhs* $\Longrightarrow$ *?rhs*
by (*meson connectedin_discrete_topology path_connectedin_imp_connectedin*)
show *?rhs* $\Longrightarrow$ *?lhs*
using *subset_singletonD* **by** *fastforce*
qed

lemma *path_connected_space_discrete_topology*:
$\mathit{path\_connected\_space}\ (\mathit{discrete\_topology}\ U) \longleftrightarrow (\exists a.\ U \subseteq \{a\})$
by (*metis path_connectedin_discrete_topology path_connectedin_topspace path_connected_space_topspace_empt*
subset_singletonD topspace_discrete_topology)

lemma *homeomorphic_path_connected_space_imp*:
$[\![\mathit{path\_connected\_space}\ X; X\ \mathit{homeomorphic\_space}\ Y]\!] \Longrightarrow \mathit{path\_connected\_space}\ Y$
unfolding *homeomorphic_space_def homeomorphic_maps_def*
by (*smt* (*verit*, *ccfv_threshold*) *homeomorphic_imp_surjective_map homeomorphic_maps_def homeomorphic_maps_map path_connectedin_continuous_map_image path_connectedin_topspace*)

lemma *homeomorphic_path_connected_space*:
$X\ \mathit{homeomorphic\_space}\ Y \Longrightarrow \mathit{path\_connected\_space}\ X \longleftrightarrow \mathit{path\_connected\_space}$

Y
by (*meson homeomorphic_path_connected_space_imp homeomorphic_space_sym*)

lemma *homeomorphic_map_path_connectedness*:
assumes *homeomorphic_map X Y f U ⊆ topspace X*
shows *path_connectedin Y (f ‘ U) ⟷ path_connectedin X U*
unfolding *path_connectedin_def*
proof (*intro conj_cong homeomorphic_path_connected_space*)
show *f ‘ U ⊆ topspace Y ⟷ (U ⊆ topspace X)*
using *assms homeomorphic_imp_surjective_map* **by** *blast*
next
assume *U ⊆ topspace X*
show *subtopology Y (f ‘ U) homeomorphic_space subtopology X U*
using *assms* **unfolding** *homeomorphic_eq_everything_map*
by (*metis* (*no_types*, *opaque_lifting*) *assms homeomorphic_map_subtopologies homeomorphic_space homeomorphic_space_sym image_mono inf.absorb_iff2*)
qed

lemma *homeomorphic_map_path_connectedness_eq*:
homeomorphic_map X Y f ⟹ path_connectedin X U ⟷ U ⊆ topspace X ∧ path_connectedin Y (f ‘ U)
by (*meson homeomorphic_map_path_connectedness path_connectedin_def*)

2.3.17 Connected components

definition *connected_component_of* :: *'a topology ⇒ 'a ⇒ 'a ⇒ bool*
where *connected_component_of X x y ≡*
∃ T. connectedin X T ∧ x ∈ T ∧ y ∈ T

abbreviation *connected_component_of_set*
where *connected_component_of_set X x ≡ Collect (connected_component_of X x)*

definition *connected_components_of* :: *'a topology ⇒ ('a set) set*
where *connected_components_of X ≡ connected_component_of_set X ‘ topspace X*

lemma *connected_component_in_topspace*:
connected_component_of X x y ⟹ x ∈ topspace X ∧ y ∈ topspace X
by (*meson connected_component_of_def connectedin_subset_topspace in_mono*)

lemma *connected_component_of_refl*:
connected_component_of X x x ⟷ x ∈ topspace X
by (*meson connected_component_in_topspace connected_component_of_def connectedin_sing insertI1*)

lemma *connected_component_of_sym*:
connected_component_of X x y ⟷ connected_component_of X y x
by (*meson connected_component_of_def*)

lemma *connected_component_of_trans*:
  ⟦*connected_component_of X x y*; *connected_component_of X y z*⟧
    ⟹ *connected_component_of X x z*
 unfolding *connected_component_of_def*
 using *connectedin_Un* **by** *blast*

lemma *connected_component_of_mono*:
  ⟦*connected_component_of* (*subtopology X S*) *x y*; *S* ⊆ *T*⟧
    ⟹ *connected_component_of* (*subtopology X T*) *x y*
 by (*metis connected_component_of_def connectedin_subtopology inf.absorb_iff2 subtopology_subtopology*)

lemma *connected_component_of_set*:
  connected_component_of_set X x = {*y*. ∃ *T*. *connectedin X T* ∧ *x* ∈ *T* ∧ *y* ∈ *T*}
 by (*meson connected_component_of_def*)

lemma *connected_component_of_subset_topspace*:
  connected_component_of_set X x ⊆ *topspace X*
 using *connected_component_in_topspace* **by** *force*

lemma *connected_component_of_eq_empty*:
  connected_component_of_set X x = {} ⟷ (*x* ∉ *topspace X*)
 using *connected_component_in_topspace connected_component_of_refl* **by** *fastforce*

lemma *connected_space_iff_connected_component*:
  connected_space X ⟷ (∀ *x* ∈ *topspace X*. ∀ *y* ∈ *topspace X*. *connected_component_of X x y*)
 by (*simp add*: *connected_component_of_def connected_space_subconnected*)

lemma *connected_space_imp_connected_component_of*:
  ⟦*connected_space X*; *a* ∈ *topspace X*; *b* ∈ *topspace X*⟧
   ⟹ *connected_component_of X a b*
 by (*simp add*: *connected_space_iff_connected_component*)

lemma *connected_space_connected_component_set*:
  connected_space X ⟷ (∀ *x* ∈ *topspace X*. *connected_component_of_set X x* = *topspace X*)
 using *connected_component_of_subset_topspace connected_space_iff_connected_component* **by** *fastforce*

lemma *connected_component_of_maximal*:
  ⟦*connectedin X S*; *x* ∈ *S*⟧ ⟹ *S* ⊆ *connected_component_of_set X x*
 by (*meson Ball_Collect connected_component_of_def*)

lemma *connected_component_of_equiv*:
  connected_component_of X x y ⟷

$x \in \mathit{topspace}\ X \wedge y \in \mathit{topspace}\ X \wedge \mathit{connected\_component\_of}\ X\ x = \mathit{connected\_component\_of}\ X\ y$
apply (*simp add*: *connected_component_in_topspace fun_eq_iff*)
by (*meson connected_component_of_refl connected_component_of_sym connected_component_of_trans*)

lemma *connected_component_of_disjoint*:
$\mathit{disjnt}\ (\mathit{connected\_component\_of\_set}\ X\ x)\ (\mathit{connected\_component\_of\_set}\ X\ y) \longleftrightarrow {}^\sim(\mathit{connected\_component\_of}\ X\ x\ y)$
using *connected_component_of_equiv* **unfolding** *disjnt_iff* **by** *force*

lemma *connected_component_of_eq*:
$\mathit{connected\_component\_of}\ X\ x = \mathit{connected\_component\_of}\ X\ y \longleftrightarrow$
$(x \notin \mathit{topspace}\ X) \wedge (y \notin \mathit{topspace}\ X) \vee$
$x \in \mathit{topspace}\ X \wedge y \in \mathit{topspace}\ X \wedge$
$\mathit{connected\_component\_of}\ X\ x\ y$
by (*metis Collect_empty_eq_bot connected_component_of_eq_empty connected_component_of_equiv*

lemma *connectedin_connected_component_of*:
$\mathit{connectedin}\ X\ (\mathit{connected\_component\_of\_set}\ X\ x)$
proof −
have $\mathit{connected\_component\_of\_set}\ X\ x = \bigcup \{T.\ \mathit{connectedin}\ X\ T \wedge x \in T\}$
by (*auto simp*: *connected_component_of_def*)
then show *?thesis*
by (*metis* (*no_types*, *lifting*) *InterI connectedin_Union emptyE mem_Collect_eq*)
qed

lemma *Union_connected_components_of*:
$\bigcup(\mathit{connected\_components\_of}\ X) = \mathit{topspace}\ X$
unfolding *connected_components_of_def*
using *connected_component_in_topspace connected_component_of_refl* **by** *fastforce*

lemma *connected_components_of_maximal*:
$[\![C \in \mathit{connected\_components\_of}\ X;\ \mathit{connectedin}\ X\ S;\ {}^\sim\mathit{disjnt}\ C\ S]\!] \Longrightarrow S \subseteq C$
unfolding *connected_components_of_def disjnt_def*
apply *clarify*
by (*metis Int_emptyI connected_component_of_def connected_component_of_trans mem_Collect_eq*)

lemma *pairwise_disjoint_connected_components_of*:
$\mathit{pairwise}\ \mathit{disjnt}\ (\mathit{connected\_components\_of}\ X)$
unfolding *connected_components_of_def pairwise_def*
by (*smt* (*verit*, *best*) *connected_component_of_disjoint connected_component_of_eq imageE*)

lemma *complement_connected_components_of_Union*:
$C \in \mathit{connected\_components\_of}\ X$

$\Longrightarrow$ *topspace X* − *C* = $\bigcup$ (*connected_components_of X* − {*C*})
by (*metis Union_connected_components_of bot.extremum ccpo_Sup_singleton diff_Union_pairwise_disjoint insert_subset pairwise_disjoint_connected_components_of*)

lemma *nonempty_connected_components_of*:
C ∈ *connected_components_of X* $\Longrightarrow$ *C* ≠ {}
unfolding *connected_components_of_def*
by (*metis* (*no_types*, *lifting*) *connected_component_of_eq_empty imageE*)

lemma *connected_components_of_subset*:
C ∈ *connected_components_of X* $\Longrightarrow$ *C* ⊆ *topspace X*
using *Union_connected_components_of* **by** *fastforce*

lemma *connectedin_connected_components_of*:
assumes *C* ∈ *connected_components_of X*
shows *connectedin X C*
proof −
have *C* ∈ *connected_component_of_set X* ‘ *topspace X*
using *assms connected_components_of_def* **by** *blast*
then show *?thesis*
using *connectedin_connected_component_of* **by** *fastforce*
qed

lemma *connected_component_in_connected_components_of*:
connected_component_of_set X a ∈ *connected_components_of X* ⟷ *a* ∈ *topspace X*
by (*metis* (*no_types*, *lifting*) *connected_component_of_eq_empty connected_components_of_def image_iff*)

lemma *connected_space_iff_components_eq*:
connected_space X ⟷ (∀ *C* ∈ *connected_components_of X*. ∀ *C′* ∈ *connected_components_of X*. *C* = *C′*)
(**is** *?lhs* = *?rhs*)
proof
show *?lhs* $\Longrightarrow$ *?rhs*
by (*simp add*: *connected_components_of_def connected_space_connected_component_set*)
show *?rhs* $\Longrightarrow$ *?lhs*
by (*metis Union_connected_components_of Union_iff connected_space_subconnected connectedin_connected_components_of*)
qed

lemma *connected_components_of_eq_empty*:
connected_components_of X = {} ⟷ *X* = *trivial_topology*
by (*simp add*: *connected_components_of_def*)

lemma *connected_components_of_empty_space*:
connected_components_of trivial_topology = {}
by (*simp add*: *connected_components_of_eq_empty*)

lemma *connected_components_of_subset_sing*:
connected_components_of X ⊆ {S} ⟷ connected_space X ∧ (X = trivial_topology ∨ topspace X = S)
proof (*cases X = trivial_topology*)
case *True*
then show *?thesis*
by (*simp add*: *connected_components_of_empty_space*)
next
case *False*
then have ⟦*connected_components_of X ⊆ {S}*⟧ ⟹ *topspace X = S*
using *Union_connected_components_of connected_components_of_eq_empty*
by *fastforce*
with *False* **show** *?thesis*
unfolding *connected_components_of_def*
by (*metis connected_space_connected_component_set empty_iff image_subset_iff insert_iff*)
qed

lemma *connected_space_iff_components_subset_singleton*:
connected_space X ⟷ (∃ a. connected_components_of X ⊆ {a})
by (*simp add*: *connected_components_of_subset_sing*)

lemma *connected_components_of_eq_singleton*:
connected_components_of X = {S} ⟷ connected_space X ∧ X ≠ trivial_topology ∧ S = topspace X
by (*metis connected_components_of_eq_empty connected_components_of_subset_sing insert_not_empty subset_singleton_iff*)

lemma *connected_components_of_connected_space*:
connected_space X ⟹ connected_components_of X = (if X = trivial_topology then {} else {topspace X})
by (*simp add*: *connected_components_of_eq_empty connected_components_of_eq_singleton*)

lemma *exists_connected_component_of_superset*:
assumes *connectedin X S* **and** *ne*: *X ≠ trivial_topology*
shows ∃ *C*. *C ∈ connected_components_of X ∧ S ⊆ C*
proof (*cases S = {}*)
case *True*
then show *?thesis*
using *ne connected_components_of_eq_empty* **by** *fastforce*
next
case *False*
then show *?thesis*
by (*meson all_not_in_conv assms*(*1*) *connected_component_in_connected_components_of connected_component_of_maximal connectedin_subset_topspace in_mono*)
qed

lemma *closedin_connected_components_of*:

assumes $C \in$ *connected_components_of* X
shows *closedin* X C
proof −
obtain x **where** $x \in$ *topspace* X **and** x: $C =$ *connected_component_of_set* X x
using *assms* **by** (*auto simp*: *connected_components_of_def*)
have *connected_component_of_set* X $x \subseteq$ *topspace* X
by (*simp add*: *connected_component_of_subset_topspace*)
moreover have X *closure_of connected_component_of_set* X $x \subseteq$ *connected_component_of_set* X x
proof (*rule connected_component_of_maximal*)
show *connectedin* X (X *closure_of connected_component_of_set* X x)
by (*simp add*: *connectedin_closure_of connectedin_connected_component_of*)
show $x \in X$ *closure_of connected_component_of_set* X x
by (*simp add*: ‹$x \in$ *topspace* X› *closure_of connected_component_of_refl*)
qed
ultimately
show *?thesis*
using *closure_of_subset_eq* x **by** *auto*
qed

lemma *closedin_connected_component_of*:
closedin X (*connected_component_of_set* X x)
by (*metis closedin_connected_components_of closedin_empty connected_component_in_connected_components_ connected_component_of_eq_empty*)

lemma *connected_component_of_eq_overlap*:
connected_component_of_set X x = *connected_component_of_set* X y ⟷
($x \notin$ *topspace* X) ∧ ($y \notin$ *topspace* X) ∨
~(*connected_component_of_set* X x ∩ *connected_component_of_set* X y = {})
using *connected_component_of_equiv* **by** *fastforce*

lemma *connected_component_of_nonoverlap*:
connected_component_of_set X x ∩ *connected_component_of_set* X y = {} ⟷
($x \notin$ *topspace* X) ∨ ($y \notin$ *topspace* X) ∨
~(*connected_component_of_set* X x = *connected_component_of_set* X y)
by (*metis connected_component_of_eq_empty connected_component_of_eq_overlap inf.idem*)

lemma *connected_component_of_overlap*:
~(*connected_component_of_set* X x ∩ *connected_component_of_set* X y = {}) ⟷
$x \in$ *topspace* X ∧ $y \in$ *topspace* X ∧
connected_component_of_set X x = *connected_component_of_set* X y
by (*meson connected_component_of_nonoverlap*)

2.3.18 Combining theorems for continuous functions into the reals

The homeomorphism between the real line and the open interval $(-1, 1)$

lemma *continuous_map_real_shrink*:
continuous_map euclideanreal (*top_of_set* {*−1<..<1*}) (*λx. x / (1 + |x|)*)
proof −
have *continuous_on UNIV* (*λx*::*real. x / (1 + |x|)*)
by (*intro continuous_intros*) *auto*
then show *?thesis*
by (*auto simp*: *continuous_map_in_subtopology divide_simps*)
qed

lemma *continuous_on_real_grow*:
continuous_on {*−1<..<1*} (*λx*::*real. x / (1 − |x|)*)
by (*intro continuous_intros*) *auto*

lemma *real_grow_shrink*:
fixes *x*::*real*
shows *x / (1 + |x|) / (1 − |x / (1 + |x|)|) = x*
by (*force simp*: *divide_simps*)

lemma *homeomorphic_maps_real_shrink*:
homeomorphic_maps euclideanreal (*subtopology euclideanreal* {*−1<..<1*})
(*λx. x / (1 + |x|)*) (*λy. y / (1 − |y|)*)
by (*force simp*: *homeomorphic_maps_def continuous_map_real_shrink continuous_on_real_grow divide_simps*)

lemma *real_shrink_Galois*:
fixes *x*::*real*
shows (*x / (1 + |x|) = y*) ⟷ (*|y| < 1 ∧ y / (1 − |y|) = x*)
using *real_grow_shrink* **by** (*fastforce simp add*: *distrib_left*)

lemma *real_shrink_eq*:
fixes *x y*::*real*
shows (*x / (1 + |x|) = y / (1 + |y|)*) ⟷ *x = y*
by (*metis real_shrink_Galois*)

lemma *real_shrink_lt*:
fixes *x*::*real*
shows *x / (1 + |x|) < y / (1 + |y|)* ⟷ *x < y*
using *zero_less_mult_iff* [*of x y*] **by** (*auto simp*: *field_simps abs_if not_less*)

lemma *real_shrink_le*:
fixes *x*::*real*
shows *x / (1 + |x|) ≤ y / (1 + |y|)* ⟷ *x ≤ y*
by (*meson linorder_not_le real_shrink_lt*)

lemma *real_shrink_grow*:

fixes x::*real*
shows $|x| < 1 \Longrightarrow x \,/\, (1 - |x|) \,/\, (1 + |x \,/\, (1 - |x|)|) = x$
using *real_shrink_Galois* **by** *blast*

lemma *continuous_shrink*:
continuous_on UNIV (λx::*real*. $x \,/\, (1 + |x|)$)
by (*intro continuous_intros*) *auto*

lemma *strict_mono_shrink*:
strict_mono (λx::*real*. $x \,/\, (1 + |x|)$)
by (*simp add*: *monotoneI real_shrink_lt*)

lemma *shrink_range*: (λx::*real*. $x \,/\, (1 + |x|)$) ‘ $S \subseteq \{-1<..<1\}$
by (*auto simp*: *divide_simps*)

Note: connected sets of real numbers are the same thing as intervals

lemma *connected_shrink*:
fixes S :: *real set*
shows *connected* ((λx. $x \,/\, (1 + |x|)$) ‘ S) $\longleftrightarrow$ *connected* S (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then have *connected* ((λx. $x \,/\, (1 - |x|)$) ‘ (λx. $x \,/\, (1 + |x|)$) ‘ S)
by (*metis continuous_on_real_grow shrink_range connected_continuous_image*

continuous_on_subset)
then show *?rhs*
using *real_grow_shrink* **by** (*force simp add*: *image_comp*)
next
assume *?rhs*
then show *?lhs*
using *connected_continuous_image*
by (*metis continuous_on_subset continuous_shrink subset_UNIV*)
qed

2.3.19 A few cardinality results

lemma *eqpoll_real_subset*:
fixes a b::*real*
assumes $a < b$ **and** S: $\bigwedge x.\ [\![a < x;\ x < b]\!] \Longrightarrow x \in S$
shows $S \approx$ (*UNIV*::*real set*)
proof (*rule lepoll_antisym*)
show $S \lesssim$ (*UNIV*::*real set*)
by (*simp add*: *subset_imp_lepoll*)
define f **where** $f \equiv \lambda x.\ (a+b) \,/\, 2 + (b-a) \,/\, 2 * (x \,/\, (1 + |x|))$
show (*UNIV*::*real set*) $\lesssim S$
unfolding *lepoll_def*
proof (*intro exI conjI*)
show *inj f*
unfolding *inj_on_def f_def*

```
    by (smt (verit, ccfv_SIG) real_shrink_eq ‹a<b› divide_eq_0_iff mult_cancel_left
times_divide_eq_right)
    have pos: (b−a) / 2 > 0
      using ‹a<b› by auto
    have ∗: a < (a + b) / 2 + (b − a) / 2 ∗ x ⟷ (b − a) > (b − a) ∗ −x
          (a + b) / 2 + (b − a) / 2 ∗ x < b ⟷ (b − a) ∗ x < (b − a) for x
      by (auto simp: field_simps)
    show range f ⊆ S
      using shrink_range [of UNIV] ‹a < b›
      unfolding subset_iff f_def greaterThanLessThan_iff image_iff
      by (smt (verit, best) S ∗ mult_less_cancel_left2 mult_minus_right)
  qed
qed

lemma reals01_lepoll_nat_sets: {0..<1::real} ≲ (UNIV::nat set set)
proof −
  define nxt where nxt ≡ λx::real. if x < 1/2 then (True, 2∗x) else (False, 2∗x
− 1)
  have nxt_fun: nxt ∈ {0..<1} → UNIV × {0..<1}
    by (simp add: nxt_def Pi_iff)
  define σ where σ ≡ λx. rec_nat (True, x) (λn (b,y). nxt y)
  have σSuc [simp]: σ x (Suc k) = nxt (snd (σ x k)) for k x
    by (simp add: σ_def case_prod_beta′)
  have σ01: x ∈ {0..<1} ⟹ σ x n ∈ UNIV × {0..<1} for x n
  proof (induction n)
    case 0
    then show ?case
      by (simp add: σ_def)
  next
    case (Suc n)
    with nxt_fun show ?case
      by (force simp add: Pi_iff split: prod.split)
  qed
  define f where f ≡ λx. {n. fst (σ x (Suc n))}
  have snd_nxt: snd (nxt y) − snd (nxt x) = 2 ∗ (y−x)
    if fst (nxt x) = fst (nxt y) for x y
    using that by (simp add: nxt_def split: if_split_asm)
  have False if f x = f y x < y 0 ≤ x x < 1 0 ≤ y y < 1 for x y :: real
  proof −
    have ⋀k. fst (σ x (Suc k)) = fst (σ y (Suc k))
      using that by (force simp add: f_def)
    then have eq: ⋀k. fst (nxt (snd (σ x k))) = fst (nxt (snd (σ y k)))
      by (metis σ_def case_prod_beta′ rec_nat_Suc_imp)
    have ∗: snd (σ y k) − snd (σ x k) = 2 ^ k ∗ (y−x) for k
    proof (induction k)
      case 0
      then show ?case
        by (simp add: σ_def)
    next
```

```
      case (Suc k)
      then show ?case
        by (simp add: eq snd_nxt)
    qed
    define n where n ≡ nat (⌈log 2 (1 / (y − x))⌉)
    have 2^n ≥ 1 / (y − x)
      by (simp add: n_def power_of_nat_log_ge)
    then have 2^n * (y−x) ≥ 1
      using ‹x < y› by (simp add: n_def field_simps)
    with * have snd (σ y n) − snd (σ x n) ≥ 1
      by presburger
    moreover have snd (σ x n) ∈ {0..<1} snd (σ y n) ∈ {0..<1}
      using that by (meson σ01 atLeastLessThan_iff mem_Times_iff)+
    ultimately show False by simp
  qed
  then have inj_on f {0..<1}
    by (meson atLeastLessThan_iff linorder_inj_onI')
  then show ?thesis
    unfolding lepoll_def by blast
qed

lemma nat_sets_lepoll_reals01: (UNIV::nat set set) ≲ {0..<1::real}
proof −
  define F where F ≡ λS i. if i∈S then (inverse 3::real) ^ i else 0
  have Fge0: F S i ≥ 0 for S i
    by (simp add: F_def)
  have F: summable (F S) for S
    unfolding F_def by (force intro: summable_comparison_test_ev [where g =
power (inverse 3)])
  have sum (F S) {..<n} ≤ 3/2 for n S
  proof (cases n)
    case (Suc n')
    have sum (F S) {..<n} ≤ (∑ i<n. inverse 3 ^ i)
      by (simp add: F_def sum_mono)
    also have ... = (∑ i=0..n'. inverse 3 ^ i)
      using Suc atLeast0AtMost lessThan_Suc_atMost by presburger
    also have ... = (3/2) * (1 − inverse 3 ^ n)
      using sum_gp_multiplied [of 0 n' inverse (3::real)] by (simp add: Suc
field_simps)
    also have ... ≤ 3/2
      by (simp add: field_simps)
    finally show ?thesis .
  qed auto
  then have F32: suminf (F S) ≤ 3/2 for S
    using F suminf_le_const by blast
  define f where f ≡ λS. suminf (F S) / 2
  have monoF: F S n ≤ F T n if S ⊆ T for S T n
    using F_def that by auto
  then have monof: f S ≤ f T if S ⊆ T for S T
```

```
    using that F by (simp add: f_def suminf_le)
  have f S ∈ {0..<1::real} for S
  proof −
    have 0 ≤ suminf (F S)
      using F by (simp add: F_def suminf_nonneg)
    with F32[of S] show ?thesis
      by (auto simp: f_def)
  qed
  moreover have inj f
  proof
    fix S T
    assume f S = f T
    show S = T
    proof (rule ccontr)
      assume S ≠ T
      then have ST_ne: (S−T) ∪ (T−S) ≠ {}
        by blast
      define n where n ≡ LEAST n. n ∈ (S−T) ∪ (T−S)
      have sum_split: suminf (F U) = sum (F U) {..<Suc n} + (∑ k. F U (k +
Suc n)) for U
        by (metis F add.commute suminf_split_initial_segment)
      have yes: f U ≥ (sum (F U) {..<n} + (inverse 3::real) ^ n) / 2
        if n ∈ U for U
      proof −
        have 0 ≤ (∑ k. F U (k + Suc n))
          by (metis F Fge0 suminf_nonneg summable_iff_shift)
        moreover have F U n = (1/3) ^ n
          by (simp add: F_def that)
        ultimately show ?thesis
          by (simp add: sum_split f_def)
      qed
      have ∗: (∑ k. F UNIV (k + n)) = (∑ k. F UNIV k) ∗ (inverse 3::real) ^ n
for n
        by (simp add: F_def power_add suminf_mult2)
      have no: f U < (sum (F U) {..<n} + (inverse 3::real) ^ n) / 2
        if n ∉ U for U
      proof −
        have [simp]: F U n = 0
          by (simp add: F_def that)
        have (∑ k. F U (k + Suc n)) ≤ (∑ k. F UNIV (k + Suc n))
       by (metis F monoF subset_UNIV suminf_le summable_ignore_initial_segment)
       then have suminf (F U) ≤ (∑ k. F UNIV (k + Suc n)) + (∑ i<n. F U i)
          by (simp add: sum_split)
        also have ... < (inverse 3::real) ^ n + (∑ i<n. F U i)
          unfolding ∗ using F32[of UNIV] by simp
        finally have suminf (F U) < inverse 3 ^ n + sum (F U) {..<n} .
        then show ?thesis
          by (simp add: f_def)
      qed
```

```
    have S ∩ {..<n} = T ∩ {..<n}
      using not_less_Least by (fastforce simp add: n_def)
    then have sum (F S) {..<n} = sum (F T) {..<n}
      by (metis (no_types, lifting) F_def Int_iff sum.cong)
    moreover consider n ∈ S−T | n ∈ T−S
      by (metis LeastI_ex ST_ne UnE ex_in_conv n_def)
    ultimately show False
      by (smt (verit, best) Diff_iff ‹f S = f T› yes no)
  qed
 qed
 ultimately show ?thesis
  by (meson image_subsetI lepoll_def)
qed

lemma open_interval_eqpoll_reals:
 fixes a b::real
 shows {a<..<b} ≈ (UNIV::real set) ⟷ a<b
 using bij_betw_tan bij_betw_open_intervals eqpoll_def
 by (smt (verit, best) UNIV_I eqpoll_real_subset eqpoll_iff_bijections greaterThanLessThan_iff)

lemma closed_interval_eqpoll_reals:
 fixes a b::real
 shows {a..b} ≈ (UNIV::real set) ⟷ a < b
proof
 show {a..b} ≈ (UNIV::real set) ⟹ a < b
  using eqpoll_finite_iff infinite_Icc_iff infinite_UNIV_char_0 by blast
qed (auto simp: eqpoll_real_subset)


lemma reals_lepoll_reals01: (UNIV::real set) ≲ {0..<1::real}
proof −
 have (UNIV::real set) ≈ {0<..<1::real}
  by (simp add: open_interval_eqpoll_reals eqpoll_sym)
 also have ... ≲ {0..<1::real}
  by (simp add: greaterThanLessThan_subseteq_atLeastLessThan_iff subset_imp_lepoll)
 finally show ?thesis .
qed

lemma nat_sets_eqpoll_reals: (UNIV::nat set set) ≈ (UNIV::real set)
 by (metis (mono_tags, opaque_lifting) reals_lepoll_reals01 lepoll_antisym lepoll_trans
    nat_sets_lepoll_reals01 reals01_lepoll_nat_sets subset_UNIV subset_imp_lepoll)

end
```

2.4 Connected Components

```
theory Connected
 imports
```

Abstract_Topology_2
begin

2.4.1 Connectedness

lemma *connected_local*:
connected S ⟷
¬ (∃ *e1 e2*.
openin (*top_of_set S*) *e1* ∧
openin (*top_of_set S*) *e2* ∧
S ⊆ *e1* ∪ *e2* ∧
e1 ∩ *e2* = {} ∧
e1 ≠ {} ∧
e2 ≠ {})
using *connected_openin* **by** *blast*

lemma *exists_diff*:
fixes *P* :: *'a set* ⇒ *bool*
shows (∃ *S*. *P* (− *S*)) ⟷ (∃ *S*. *P S*)
by (*metis boolean_algebra_class.boolean_algebra.double_compl*)

lemma *connected_clopen*: *connected S* ⟷
(∀ *T*. *openin* (*top_of_set S*) *T* ∧
closedin (*top_of_set S*) *T* ⟶ *T* = {} ∨ *T* = *S*) (**is** *?lhs* ⟷ *?rhs*)
proof −
have ¬ *connected S* ⟷
(∃ *e1 e2*. *open e1* ∧ *open* (− *e2*) ∧ *S* ⊆ *e1* ∪ (− *e2*) ∧ *e1* ∩ (− *e2*) ∩ *S* = {}
∧ *e1* ∩ *S* ≠ {} ∧ (− *e2*) ∩ *S* ≠ {})
unfolding *connected_def openin_open closedin_closed*
by (*metis double_complement*)
then have *th0*: *connected S* ⟷
¬ (∃ *e2 e1*. *closed e2* ∧ *open e1* ∧ *S* ⊆ *e1* ∪ (− *e2*) ∧ *e1* ∩ (− *e2*) ∩ *S* = {}
∧ *e1* ∩ *S* ≠ {} ∧ (− *e2*) ∩ *S* ≠ {})
(**is** _ ⟷ ¬ (∃ *e2 e1*. *?P e2 e1*))
by (*simp add*: *closed_def*) *metis*
have *th1*: *?rhs* ⟷ ¬ (∃ *t' t*. *closed t'*∧*t* = *S*∩*t'* ∧ *t*≠{} ∧ *t*≠*S* ∧ (∃ *t'*. *open t'*
∧ *t* = *S* ∩ *t'*))
(**is** _ ⟷ ¬ (∃ *t' t*. *?Q t' t*))
unfolding *connected_def openin_open closedin_closed* **by** *auto*
have (∃ *e1*. *?P e2 e1*) ⟷ (∃ *t*. *?Q e2 t*) **for** *e2*
proof −
have *?P e2 e1* ⟷ (∃ *t*. *closed e2* ∧ *t* = *S*∩*e2* ∧ *open e1* ∧ *t* = *S*∩*e1* ∧ *t*≠{}
∧ *t* ≠ *S*) **for** *e1*
by *auto*
then show *?thesis*
by *metis*
qed
then have ∀ *e2*. (∃ *e1*. *?P e2 e1*) ⟷ (∃ *t*. *?Q e2 t*)
by *blast*

then show *?thesis*
by (*simp add*: *th0 th1*)
qed

2.4.2 Connected components, considered as a connectedness relation or a set

definition *connected_component S x y* ≡ ∃ *T*. *connected T* ∧ *T* ⊆ *S* ∧ *x* ∈ *T* ∧ *y* ∈ *T*

abbreviation *connected_component_set S x* ≡ *Collect* (*connected_component S x*)

lemma *connected_componentI*:
connected T ⟹ *T* ⊆ *S* ⟹ *x* ∈ *T* ⟹ *y* ∈ *T* ⟹ *connected_component S x y*
by (*auto simp*: *connected_component_def*)

lemma *connected_component_in*: *connected_component S x y* ⟹ *x* ∈ *S* ∧ *y* ∈ *S*
by (*auto simp*: *connected_component_def*)

lemma *connected_component_refl*: *x* ∈ *S* ⟹ *connected_component S x x*
using *connected_component_def connected_sing* **by** *blast*

lemma *connected_component_refl_eq* [*simp*]: *connected_component S x x* ⟷ *x* ∈ *S*
using *connected_component_in connected_component_refl* **by** *blast*

lemma *connected_component_sym*: *connected_component S x y* ⟹ *connected_component S y x*
by (*auto simp*: *connected_component_def*)

lemma *connected_component_trans*:
connected_component S x y ⟹ *connected_component S y z* ⟹ *connected_component S x z*
unfolding *connected_component_def*
by (*metis Int_iff Un_iff Un_subset_iff equals0D connected_Un*)

lemma *connected_component_of_subset*:
connected_component S x y ⟹ *S* ⊆ *T* ⟹ *connected_component T x y*
by (*auto simp*: *connected_component_def*)

lemma *connected_component_Union*: *connected_component_set S x* = ⋃ {*T*. *connected T* ∧ *x* ∈ *T* ∧ *T* ⊆ *S*}
by (*auto simp*: *connected_component_def*)

lemma *connected_connected_component* [*iff*]: *connected* (*connected_component_set S x*)
by (*auto simp*: *connected_component_Union intro*: *connected_Union*)

lemma *connected_iff_eq_connected_component_set*:
connected S $\longleftrightarrow$ $(\forall x \in S.$ *connected_component_set S x* $= S)$
by (*metis connected_component_def connected_component_in connected_connected_component mem_Collect_eq subsetI subset_antisym*)

lemma *connected_component_subset*: *connected_component_set S x* $\subseteq S$
using *connected_component_in* **by** *blast*

lemma *connected_component_eq_self*: *connected S* $\Longrightarrow x \in S \Longrightarrow$ *connected_component_set S x* $= S$
by (*simp add*: *connected_iff_eq_connected_component_set*)

lemma *connected_iff_connected_component*:
connected S $\longleftrightarrow$ $(\forall x \in S. \forall y \in S.$ *connected_component S x y*)
using *connected_component_in* **by** (*auto simp*: *connected_iff_eq_connected_component_set*)

lemma *connected_component_maximal*:
$x \in T \Longrightarrow$ *connected T* $\Longrightarrow T \subseteq S \Longrightarrow T \subseteq$ (*connected_component_set S x*)
using *connected_component_eq_self connected_component_of_subset* **by** *blast*

lemma *connected_component_mono*:
$S \subseteq T \Longrightarrow$ *connected_component_set S x* $\subseteq$ *connected_component_set T x*
by (*simp add*: *Collect_mono connected_component_of_subset*)

lemma *connected_component_eq_empty* [*simp*]: *connected_component_set S x* $=$ $\{\} \longleftrightarrow x \notin S$
using *connected_component_refl* **by** (*fastforce simp*: *connected_component_in*)

lemma *connected_component_set_empty* [*simp*]: *connected_component_set* $\{\}$ *x* $= \{\}$
using *connected_component_eq_empty* **by** *blast*

lemma *connected_component_eq*:
$y \in$ *connected_component_set S x* $\Longrightarrow$ (*connected_component_set S y* $=$ *connected_component_set S x*)
by (*metis* (*no_types*, *lifting*)
Collect_cong connected_component_sym connected_component_trans mem_Collect_eq)

lemma *closed_connected_component*:
assumes *S*: *closed S*
shows *closed* (*connected_component_set S x*)
proof (*cases x* $\in S$)
case *False*
then show *?thesis*
by (*metis connected_component_eq_empty closed_empty*)
next
case *True*
show *?thesis*

unfolding *closure_eq* [*symmetric*]
proof
show *closure* (*connected_component_set S x*) ⊆ *connected_component_set S x*
proof (*rule connected_component_maximal*)
show *x* ∈ *closure* (*connected_component_set S x*)
by (*simp add*: *closure_def True*)
show *connected* (*closure* (*connected_component_set S x*))
by (*simp add*: *connected_imp_connected_closure*)
show *closure* (*connected_component_set S x*) ⊆ *S*
by (*simp add*: *S closure_minimal connected_component_subset*)
qed
qed (*simp add*: *closure_subset*)
qed

lemma *connected_component_disjoint*:
connected_component_set S a ∩ *connected_component_set S b* = {} ⟷
a ∉ *connected_component_set S b*
by (*smt* (*verit*) *connected_component_eq connected_component_eq_empty connected_component_refl_eq*
disjoint_iff_not_equal mem_Collect_eq)

lemma *connected_component_nonoverlap*:
connected_component_set S a ∩ *connected_component_set S b* = {} ⟷
a ∉ *S* ∨ *b* ∉ *S* ∨ *connected_component_set S a* ≠ *connected_component_set S b*
by (*metis connected_component_disjoint connected_component_eq connected_component_eq_empty inf.idem*)

lemma *connected_component_overlap*:
connected_component_set S a ∩ *connected_component_set S b* ≠ {} ⟷
a ∈ *S* ∧ *b* ∈ *S* ∧ *connected_component_set S a* = *connected_component_set S b*
by (*auto simp*: *connected_component_nonoverlap*)

lemma *connected_component_sym_eq*: *connected_component S x y* ⟷ *connected_component S y x*
using *connected_component_sym* **by** *blast*

lemma *connected_component_eq_eq*:
connected_component_set S x = *connected_component_set S y* ⟷
x ∉ *S* ∧ *y* ∉ *S* ∨ *x* ∈ *S* ∧ *y* ∈ *S* ∧ *connected_component S x y*
by (*metis connected_component_eq connected_component_eq_empty connected_component_refl mem_Collect_eq*)

lemma *connected_iff_connected_component_eq*:
connected S ⟷ (∀*x* ∈ *S*. ∀*y* ∈ *S*. *connected_component_set S x* = *connected_component_set S y*)

by (*simp add*: *connected_component_eq_eq connected_iff_connected_component*)

lemma *connected_component_idemp*:
connected_component_set (*connected_component_set S x*) *x* = *connected_component_set S x*
by (*metis Int_absorb connected_component_disjoint connected_component_eq_empty connected_component_eq_self connected_connected_component*)

lemma *connected_component_unique*:
⟦*x* ∈ *c*; *c* ⊆ *S*; *connected c*;
⋀*c'*. ⟦*x* ∈ *c'*; *c'* ⊆ *S*; *connected c'*⟧ ⟹ *c'* ⊆ *c*⟧
⟹ *connected_component_set S x* = *c*
by (*meson connected_component_maximal connected_component_subset connected_connected_component subsetD subset_antisym*)

lemma *joinable_connected_component_eq*:
⟦*connected T*; *T* ⊆ *S*;
connected_component_set S x ∩ *T* ≠ {};
connected_component_set S y ∩ *T* ≠ {}⟧
⟹ *connected_component_set S x* = *connected_component_set S y*
by (*metis* (*full_types*) *subsetD connected_component_eq connected_component_maximal disjoint_iff_not_equal*)

lemma *Union_connected_component*: ⋃(*connected_component_set S* ‘ *S*) = *S*
proof
show ⋃(*connected_component_set S* ‘ *S*) ⊆ *S*
by (*simp add*: *SUP_least connected_component_subset*)
qed (*use connected_component_refl_eq* **in** *force*)

lemma *complement_connected_component_unions*:
S − *connected_component_set S x* =
⋃(*connected_component_set S* ‘ *S* − {*connected_component_set S x*})
(**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs*
by (*metis Diff_subset Diff_subset_conv Sup_insert Union_connected_component insert_Diff_single*)
show *?rhs* ⊆ *?lhs*
by *clarsimp* (*metis connected_component_eq_eq connected_component_in*)
qed

lemma *connected_component_intermediate_subset*:
⟦*connected_component_set U a* ⊆ *T*; *T* ⊆ *U*⟧
⟹ *connected_component_set T a* = *connected_component_set U a*
by (*metis connected_component_idemp connected_component_mono subset_antisym*)

lemma *connected_component_homeomorphismI*:
assumes *homeomorphism A B f g connected_component A x y*
shows *connected_component B* (*f x*) (*f y*)

proof −
from *assms* **obtain** *T* **where** *T*: *connected T T* ⊆ *A x* ∈ *T y* ∈ *T*
unfolding *connected_component_def* **by** *blast*
have *connected* (*f* ‘ *T*) *f* ‘ *T* ⊆ *B f x* ∈ *f* ‘ *T f y* ∈ *f* ‘ *T*
using *assms T continuous_on_subset*[*of A f T*]
by (*auto intro*!: *connected_continuous_image simp*: *homeomorphism_def*)
thus *?thesis*
unfolding *connected_component_def* **by** *blast*
qed

lemma *connected_component_homeomorphism_iff*:
assumes *homeomorphism A B f g*
shows *connected_component A x y* ⟷ *x* ∈ *A* ∧ *y* ∈ *A* ∧ *connected_component B* (*f x*) (*f y*)
by (*metis assms connected_component_homeomorphismI connected_component_in homeomorphism_apply1 homeomorphism_sym*)

lemma *connected_component_set_homeomorphism*:
assumes *homeomorphism A B f g x* ∈ *A*
shows *connected_component_set B* (*f x*) = *f* ‘ *connected_component_set A x*
(**is** *?lhs* = *?rhs*)
proof −
have *y* ∈ *?lhs* ⟷ *y* ∈ *?rhs* **for** *y*
by (*smt* (*verit*, *best*) *assms connected_component_homeomorphism_iff homeomorphism_def image_iff mem_Collect_eq*)
thus *?thesis*
by *blast*
qed

2.4.3 The set of connected components of a set

definition *components*:: *'a*::*topological_space set* ⇒ *'a set set*
where *components S* ≡ *connected_component_set S* ‘ *S*

lemma *components_iff*: *S* ∈ *components U* ⟷ (∃ *x*. *x* ∈ *U* ∧ *S* = *connected_component_set U x*)
by (*auto simp*: *components_def*)

lemma *componentsI*: *x* ∈ *U* ⟹ *connected_component_set U x* ∈ *components U*
by (*auto simp*: *components_def*)

lemma *componentsE*:
assumes *S* ∈ *components U*
obtains *x* **where** *x* ∈ *U S* = *connected_component_set U x*
using *assms* **by** (*auto simp*: *components_def*)

lemma *Union_components* [*simp*]: ⋃(*components U*) = *U*
by (*simp add*: *Union_connected_component components_def*)

lemma *pairwise_disjoint_components*: *pairwise* ($\lambda X\ Y.\ X \cap Y = \{\}$) (*components* U)
unfolding *pairwise_def*
by (*metis* (*full_types*) *components_iff connected_component_nonoverlap*)

lemma *in_components_nonempty*: $C \in$ *components* $S \Longrightarrow C \neq \{\}$
by (*metis components_iff connected_component_eq_empty*)

lemma *in_components_subset*: $C \in$ *components* $S \Longrightarrow C \subseteq S$
using *Union_components* **by** *blast*

lemma *in_components_connected*: $C \in$ *components* $S \Longrightarrow$ *connected* C
by (*metis components_iff connected_connected_component*)

lemma *in_components_maximal*:
$C \in$ *components* $S \longleftrightarrow$
$C \neq \{\} \wedge C \subseteq S \wedge$ *connected* $C \wedge (\forall d.\ d \neq \{\} \wedge C \subseteq d \wedge d \subseteq S \wedge$ *connected* $d \longrightarrow d = C)$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
assume L: *?lhs*
then have $C \subseteq S$ *connected* C
by (*simp_all add*: *in_components_subset in_components_connected*)
then show *?rhs*
by (*metis* (*full_types*) L *components_iff connected_component_maximal connected_component_refl empty_iff mem_Collect_eq subsetD subset_antisym*)
next
show *?rhs* $\Longrightarrow$ *?lhs*
by (*metis bot.extremum_uniqueI components_iff connected_component_eq_empty connected_component_maximal connected_component_subset connected_connected_component subset_emptyI*)
qed

lemma *joinable_components_eq*:
connected $T \wedge T \subseteq S \wedge$ *c1* $\in$ *components* $S \wedge$ *c2* $\in$ *components* $S \wedge$ *c1* $\cap\ T \neq \{\} \wedge$ *c2* $\cap\ T \neq \{\} \Longrightarrow$ *c1* $=$ *c2*
by (*metis* (*full_types*) *components_iff joinable_connected_component_eq*)

lemma *closed_components*: $[\![$*closed* S; $C \in$ *components* $S]\!] \Longrightarrow$ *closed* C
by (*metis closed_connected_component components_iff*)

lemma *components_nonoverlap*:
$[\![C \in$ *components* S; $C' \in$ *components* $S]\!] \Longrightarrow (C \cap C' = \{\}) \longleftrightarrow (C \neq C')$
by (*metis* (*full_types*) *components_iff connected_component_nonoverlap*)

lemma *components_eq*: $[\![C \in$ *components* S; $C' \in$ *components* $S]\!] \Longrightarrow (C = C' \longleftrightarrow C \cap C' \neq \{\})$
by (*metis components_nonoverlap*)

lemma *components_eq_empty* [*simp*]: *components S* = {} ⟷ *S* = {}
by (*simp add*: *components_def*)

lemma *components_empty* [*simp*]: *components* {} = {}
by *simp*

lemma *connected_eq_connected_components_eq*: *connected S* ⟷ (∀ *C* ∈ *components S*. ∀ *C'* ∈ *components S*. *C* = *C'*)
by (*metis* (*no_types*, *opaque_lifting*) *components_iff connected_component_eq_eq connected_iff_connected_component*)

lemma *components_eq_sing_iff*: *components S* = {*S*} ⟷ *connected S* ∧ *S* ≠ {} (**is** *?lhs* ⟷ *?rhs*)
proof
show *?rhs* ⟹ *?lhs*
by (*metis components_iff connected_component_eq_self equals0I insert_iff mk_disjoint_insert*)
qed (*use in_components_connected* **in** *fastforce*)

lemma *components_eq_sing_exists*: (∃ *a*. *components S* = {*a*}) ⟷ *connected S* ∧ *S* ≠ {}
by (*metis Union_components ccpo_Sup_singleton components_eq_sing_iff*)

lemma *connected_eq_components_subset_sing*: *connected S* ⟷ *components S* ⊆ {*S*}
by (*metis components_eq_empty components_eq_sing_iff connected_empty subset_singleton_iff*)

lemma *connected_eq_components_subset_sing_exists*: *connected S* ⟷ (∃ *a*. *components S* ⊆ {*a*})
by (*metis components_eq_sing_exists connected_eq_components_subset_sing subset_singleton_iff*)

lemma *in_components_self*: *S* ∈ *components S* ⟷ *connected S* ∧ *S* ≠ {}
by (*metis components_empty components_eq_sing_iff empty_iff in_components_connected insertI1*)

lemma *components_maximal*: ⟦*C* ∈ *components S*; *connected T*; *T* ⊆ *S*; *C* ∩ *T* ≠ {}⟧ ⟹ *T* ⊆ *C*
by (*smt* (*verit*, *best*) *Int_Un_eq*(*4*) *Un_upper1 bot_eq_sup_iff connected_Un in_components_maximal inf.orderE sup.mono sup.orderI*)

lemma *exists_component_superset*: ⟦*T* ⊆ *S*; *S* ≠ {}; *connected T*⟧ ⟹ ∃ *C*. *C* ∈ *components S* ∧ *T* ⊆ *C*
by (*meson componentsI connected_component_maximal equals0I subset_eq*)

lemma *components_intermediate_subset*: ⟦*S* ∈ *components U*; *S* ⊆ *T*; *T* ⊆ *U*⟧ ⟹ *S* ∈ *components T*

by (*smt* (*verit*, *best*) *dual_order.trans in_components_maximal*)

lemma *in_components_unions_complement*: $C \in$ *components* $S \Longrightarrow S - C = \bigcup$(*components* $S - \{C\}$)
by (*metis complement_connected_component_unions components_def components_iff*)

lemma *connected_intermediate_closure*:
assumes *cs*: *connected* S **and** *st*: $S \subseteq T$ **and** *ts*: $T \subseteq$ *closure* S
shows *connected* T
using *assms* **unfolding** *connected_def*
by (*smt* (*verit*) *Int_assoc inf.absorb_iff2 inf_bot_left open_Int_closure_eq_empty*)

lemma *closedin_connected_component*: *closedin* (*top_of_set* S) (*connected_component_set* S x)
proof (*cases connected_component_set* S $x = \{\}$)
case *True*
then show *?thesis*
by (*metis closedin_empty*)
next
case *False*
then obtain y **where** y: *connected_component* S x y **and** $x \in S$
using *connected_component_eq_empty* **by** *blast*
have $*$: *connected_component_set* S $x \subseteq S \cap$ *closure* (*connected_component_set* S x)
by (*auto simp*: *closure_def connected_component_in*)
have $**$: $x \in$ *closure* (*connected_component_set* S x)
by (*simp add*: ‹$x \in S$› *closure_def*)
have $S \cap$ *closure* (*connected_component_set* S x) $\subseteq$ *connected_component_set* S x **if** *connected_component* S x y
proof (*rule connected_component_maximal*)
show *connected* ($S \cap$ *closure* (*connected_component_set* S x))
using $*$ *connected_intermediate_closure* **by** *blast*
qed (*use* ‹$x \in S$› $**$ **in** *auto*)
with y $*$ **show** *?thesis*
by (*auto simp*: *closedin_closed*)
qed

lemma *closedin_component*:
$C \in$ *components* $S \Longrightarrow$ *closedin* (*top_of_set* S) C
using *closedin_connected_component componentsE* **by** *blast*

2.4.4 Proving a function is constant on a connected set by proving that a level set is open

lemma *continuous_levelset_openin_cases*:
fixes f :: _ $\Rightarrow$ *'b*::*t1_space*
shows *connected* $S \Longrightarrow$ *continuous_on* S $f \Longrightarrow$
openin (*top_of_set* S) $\{x \in S.\ f\ x = a\}$

$\Longrightarrow (\forall x \in S.\ f\ x \neq a) \vee (\forall x \in S.\ f\ x = a)$
unfolding *connected_clopen*
using *continuous_closedin_preimage_constant* **by** *auto*

lemma *continuous_levelset_openin*:
fixes $f :: \_ \Rightarrow 'b$::*t1_space*
shows *connected* $S \Longrightarrow$ *continuous_on* $S\ f \Longrightarrow$
$\quad$ *openin* (*top_of_set* S) $\{x \in S.\ f\ x = a\} \Longrightarrow$
$\quad$ $(\exists x \in S.\ f\ x = a) \Longrightarrow (\forall x \in S.\ f\ x = a)$
using *continuous_levelset_openin_cases*[*of* $S\ f$]
by *meson*

lemma *continuous_levelset_open*:
fixes $f :: \_ \Rightarrow 'b$::*t1_space*
assumes S: *connected* S *continuous_on* $S\ f$
and a: *open* $\{x \in S.\ f\ x = a\}$ $\exists x \in S.\ f\ x = a$
shows $\forall x \in S.\ f\ x = a$
using a *continuous_levelset_openin*[*OF* S, *of* a, *unfolded openin_open*]
by *fast*

2.4.5 Preservation of Connectedness

lemma *homeomorphic_connectedness*:
assumes S *homeomorphic* T
shows *connected* $S \longleftrightarrow$ *connected* T
using *assms* **unfolding** *homeomorphic_def homeomorphism_def* **by** (*metis connected_continuous_image*)

lemma *connected_monotone_quotient_preimage*:
assumes *connected* T
and *contf*: *continuous_on* $S\ f$ **and** *fim*: f ‘ $S = T$
and *opT*: $\bigwedge U.\ U \subseteq T$
$\Longrightarrow$ *openin* (*top_of_set* S) $(S \cap f -$‘ $U) \longleftrightarrow$
openin (*top_of_set* T) U
and *connT*: $\bigwedge y.\ y \in T \Longrightarrow$ *connected* $(S \cap f -$‘ $\{y\})$
shows *connected* S
proof (*rule connectedI*)
fix $U\ V$
assume *open* U **and** *open* V **and** $U \cap S \neq \{\}$ **and** $V \cap S \neq \{\}$
and $U \cap V \cap S = \{\}$ **and** $S \subseteq U \cup V$
moreover
have *disjoint*: f ‘ $(S \cap U) \cap f$ ‘ $(S \cap V) = \{\}$
proof −
have *False* **if** $y \in f$ ‘ $(S \cap U) \cap f$ ‘ $(S \cap V)$ **for** y
proof −
have $y \in T$
using *fim that* **by** *blast*
show *?thesis*
using *connectedD* [*OF connT* [*OF* ‹$y \in T$›] ‹*open* U› ‹*open* V›]

```
            ‹S ⊆ U ∪ V› ‹U ∩ V ∩ S = {}› that by fastforce
    qed
    then show ?thesis by blast
  qed
  ultimately have UU: (S ∩ f −‘ f ‘ (S ∩ U)) = S ∩ U and VV: (S ∩ f −‘ f ‘
(S ∩ V)) = S ∩ V
    by auto
  have opeU: openin (top_of_set T) (f ‘ (S ∩ U))
   by (metis UU ‹open U› fim image_Int_subset le_inf_iff opT openin_open_Int)
  have opeV: openin (top_of_set T) (f ‘ (S ∩ V))
   by (metis opT fim VV ‹open V› openin_open_Int image_Int_subset inf.bounded_iff)
  have T ⊆ f ‘ (S ∩ U) ∪ f ‘ (S ∩ V)
    using ‹S ⊆ U ∪ V› fim by auto
  then show False
    using ‹connected T› disjoint opeU opeV ‹U ∩ S ≠ {}› ‹V ∩ S ≠ {}›
    by (auto simp: connected_openin)
qed

lemma connected_open_monotone_preimage:
  assumes contf: continuous_on S f and fim: f ‘ S = T
    and ST: ⋀C. openin (top_of_set S) C ⟹ openin (top_of_set T) (f ‘ C)
    and connT: ⋀y. y ∈ T ⟹ connected (S ∩ f −‘ {y})
    and connected C C ⊆ T
  shows connected (S ∩ f −‘ C)
proof −
  have contf': continuous_on (S ∩ f −‘ C) f
    by (meson contf continuous_on_subset inf_le1)
  have eqC: f ‘ (S ∩ f −‘ C) = C
    using ‹C ⊆ T› fim by blast
  show ?thesis
  proof (rule connected_monotone_quotient_preimage [OF ‹connected C› contf'
eqC])
    show connected (S ∩ f −‘ C ∩ f −‘ {y}) if y ∈ C for y
      by (metis Int_assoc Int_empty_right Int_insert_right_if1 assms(6) connT
in_mono that vimage_Int)
    have ⋀U. openin (top_of_set (S ∩ f −‘ C)) U
            ⟹ openin (top_of_set C) (f ‘ U)
      using open_map_restrict [OF _ ST ‹C ⊆ T›] by metis
    then show ⋀D. D ⊆ C
        ⟹ openin (top_of_set (S ∩ f −‘ C)) (S ∩ f −‘ C ∩ f −‘ D) =
           openin (top_of_set C) D
      using open_map_imp_quotient_map [of (S ∩ f −‘ C) f] contf' by (simp
add: eqC)
  qed
qed


lemma connected_closed_monotone_preimage:
  assumes contf: continuous_on S f and fim: f ‘ S = T
```

```
    and ST: ⋀C. closedin (top_of_set S) C ⟹ closedin (top_of_set T) (f ‘ C)
    and connT: ⋀y. y ∈ T ⟹ connected (S ∩ f −‘ {y})
    and connected C C ⊆ T
  shows connected (S ∩ f −‘ C)
proof −
  have contf': continuous_on (S ∩ f −‘ C) f
    by (meson contf continuous_on_subset inf_le1)
  have eqC: f ‘ (S ∩ f −‘ C) = C
    using ‹C ⊆ T› fim by blast
  show ?thesis
  proof (rule connected_monotone_quotient_preimage [OF ‹connected C› contf' eqC])
    show connected (S ∩ f −‘ C ∩ f −‘ {y}) if y ∈ C for y
      by (metis Int_assoc Int_empty_right Int_insert_right_if1 ‹C ⊆ T› connT subsetD that vimage_Int)
    have ⋀U. closedin (top_of_set (S ∩ f −‘ C)) U
            ⟹ closedin (top_of_set C) (f ‘ U)
      using closed_map_restrict [OF _ ST ‹C ⊆ T›] by metis
    then show ⋀D. D ⊆ C
        ⟹ openin (top_of_set (S ∩ f −‘ C)) (S ∩ f −‘ C ∩ f −‘ D) =
           openin (top_of_set C) D
      using closed_map_imp_quotient_map [of (S ∩ f −‘ C) f] contf' by (simp add: eqC)
  qed
qed
```

2.4.6 Lemmas about components

See Newman IV, 3.3 and 3.4.

```
lemma connected_Un_clopen_in_complement:
  fixes S U :: 'a::metric_space set
  assumes connected S connected U S ⊆ U
    and opeT: openin (top_of_set (U − S)) T
    and cloT: closedin (top_of_set (U − S)) T
  shows connected (S ∪ T)
proof −
  have ∗: ⟦⋀x y. P x y ⟷ P y x; ⋀x y. P x y ⟹ S ⊆ x ∨ S ⊆ y;
           ⋀x y. ⟦P x y; S ⊆ x⟧ ⟹ False⟧ ⟹ ¬(∃ x y. (P x y)) for P
    by metis
  show ?thesis
    unfolding connected_closedin_eq
  proof (rule ∗)
    fix H1 H2
    assume H: closedin (top_of_set (S ∪ T)) H1 ∧
              closedin (top_of_set (S ∪ T)) H2 ∧
              H1 ∪ H2 = S ∪ T ∧ H1 ∩ H2 = {} ∧ H1 ≠ {} ∧ H2 ≠ {}
    then have clo: closedin (top_of_set S) (S ∩ H1)
                   closedin (top_of_set S) (S ∩ H2)
      by (metis Un_upper1 closedin_closed_subset inf_commute)+
```

moreover have $S \cap ((S \cup T) \cap H1) \cup S \cap ((S \cup T) \cap H2) = S$
using H **by** *blast*
moreover have $H1 \cap (S \cap ((S \cup T) \cap H2)) = \{\}$
using H **by** *blast*
ultimately have $S \cap H1 = \{\} \vee S \cap H2 = \{\}$
by (*smt* (*verit*) *Int_assoc* ‹*connected S*› *connected_closedin_eq inf_commute inf_sup_absorb*)
then show $S \subseteq H1 \vee S \subseteq H2$
using H ‹*connected S*› **unfolding** *connected_closedin* **by** *blast*
next
fix *H1 H2*
assume H: *closedin* (*top_of_set* $(S \cup T)$) *H1* $\wedge$
closedin (*top_of_set* $(S \cup T)$) *H2* $\wedge$
$H1 \cup H2 = S \cup T \wedge H1 \cap H2 = \{\} \wedge H1 \neq \{\} \wedge H2 \neq \{\}$
and $S \subseteq H1$
then have *H2T*: $H2 \subseteq T$
by *auto*
have $T \subseteq U$
using *Diff_iff opeT openin_imp_subset* **by** *auto*
with ‹$S \subseteq U$› **have** *Ueq*: $U = (U - S) \cup (S \cup T)$
by *auto*
have *openin* (*top_of_set* $((U - S) \cup (S \cup T))$) *H2*
proof (*rule openin_subtopology_Un*)
show *openin* (*top_of_set* $(S \cup T)$) *H2*
by (*metis Diff_cancel H Un_Diff Un_Diff_Int closedin_subset openin_closedin_eq topspace_euclidean_subtopology*)
then show *openin* (*top_of_set* $(U - S)$) *H2*
by (*meson H2T Un_upper2 opeT openin_subset_trans openin_trans*)
qed
moreover have *closedin* (*top_of_set* $((U - S) \cup (S \cup T))$) *H2*
proof (*rule closedin_subtopology_Un*)
show *closedin* (*top_of_set* $(U - S)$) *H2*
using *H H2T cloT closedin_subset_trans*
by (*blast intro*: *closedin_subtopology_Un closedin_trans*)
qed (*simp add*: *H*)
ultimately have *H2*: $H2 = \{\} \vee H2 = U$
using *Ueq* ‹*connected U*› **unfolding** *connected_clopen* **by** *metis*
then have $H2 \subseteq S$
by (*metis Diff_partition H Un_Diff_cancel Un_subset_iff* ‹$H2 \subseteq T$› *assms*(*3*) *inf.orderE opeT openin_imp_subset*)
moreover have $T \subseteq H2 - S$
by (*metis* (*no_types*) *H2 H opeT openin_closedin_eq topspace_euclidean_subtopology*)
ultimately show *False*
using *H* ‹$S \subseteq H1$› **by** *blast*
qed *blast*
qed

proposition *component_diff_connected*:

fixes S :: $'a$::*metric_space set*
assumes *connected S connected U S* $\subseteq$ U **and** C: $C \in$ *components* $(U - S)$
shows *connected*$(U - C)$
using ‹*connected S*› **unfolding** *connected_closedin_eq not_ex de_Morgan_conj*
proof *clarify*
fix *H3 H4*
assume *clo3*: *closedin* (*top_of_set* $(U - C)$) *H3*
and *clo4*: *closedin* (*top_of_set* $(U - C)$) *H4*
and *H34*: *H3* $\cup$ *H4* $= U - C$ *H3* $\cap$ *H4* $= \{\}$ **and** *H3* $\neq \{\}$ **and** *H4* $\neq \{\}$
and $*$ [*rule_format*]:
$\forall$ *H1 H2*. $\neg$ *closedin* (*top_of_set S*) *H1* $\vee$
$\neg$ *closedin* (*top_of_set S*) *H2* $\vee$
H1 $\cup$ *H2* $\neq$ *S* $\vee$ *H1* $\cap$ *H2* $\neq \{\}$ $\vee$ $\neg$ *H1* $\neq \{\}$ $\vee$ $\neg$ *H2* $\neq \{\}$
then have *H3* $\subseteq U-C$ **and** *ope3*: *openin* (*top_of_set* $(U - C)$) $(U - C -$ *H3*$)$
and *H4* $\subseteq U-C$ **and** *ope4*: *openin* (*top_of_set* $(U - C)$) $(U - C -$ *H4*$)$
by (*auto simp*: *closedin_def*)
have *C* $\neq \{\}$ *C* $\subseteq U-S$ *connected C*
using *C in_components_nonempty in_components_subset in_components_maximal*
by *blast*+
have *cCH3*: *connected* $(C \cup$ *H3*$)$
proof (*rule connected_Un_clopen_in_complement* [*OF* ‹*connected C*› ‹*connected U*› _ _ *clo3*])
show *openin* (*top_of_set* $(U - C)$) *H3*
by (*metis Diff_cancel Un_Diff Un_Diff_Int* ‹*H3* $\cap$ *H4* $= \{\}$› ‹*H3* $\cup$ *H4* $=$ $U - C$› *ope4*)
qed (*use clo3* ‹*C* $\subseteq U - S$› **in** *auto*)
have *cCH4*: *connected* $(C \cup$ *H4*$)$
proof (*rule connected_Un_clopen_in_complement* [*OF* ‹*connected C*› ‹*connected U*› _ _ *clo4*])
show *openin* (*top_of_set* $(U - C)$) *H4*
by (*metis Diff_cancel Diff_triv Int_Un_eq*(2) *Un_Diff H34 inf_commute ope3*)
qed (*use clo4* ‹*C* $\subseteq U - S$› **in** *auto*)
have *closedin* (*top_of_set S*) (*S* $\cap$ *H3*) *closedin* (*top_of_set S*) (*S* $\cap$ *H4*)
using *clo3 clo4* ‹*S* $\subseteq U$› ‹*C* $\subseteq U - S$› **by** (*auto simp*: *closedin_closed*)
moreover have *S* $\cap$ *H3* $\neq \{\}$
using *components_maximal* [*OF C cCH3*] ‹*C* $\neq \{\}$› ‹*C* $\subseteq U - S$› ‹*H3* $\neq \{\}$› ‹*H3* $\subseteq U - C$› **by** *auto*
moreover have *S* $\cap$ *H4* $\neq \{\}$
using *components_maximal* [*OF C cCH4*] ‹*C* $\neq \{\}$› ‹*C* $\subseteq U - S$› ‹*H4* $\neq \{\}$› ‹*H4* $\subseteq U - C$› **by** *auto*
ultimately show *False*
using $*$ [*of S* $\cap$ *H3 S* $\cap$ *H4*] ‹*H3* $\cap$ *H4* $= \{\}$› ‹*C* $\subseteq U - S$› ‹*H3* $\cup$ *H4* $= U - C$› ‹*S* $\subseteq U$›
by *auto*
qed

2.4.7 Constancy of a function from a connected set into a finite, disconnected or discrete set

Still missing: versions for a set that is smaller than R, or countable.

lemma *continuous_disconnected_range_constant*:
assumes *S*: *connected S*
and *conf*: *continuous_on S f*
and *fim*: $f \in S \to T$
and *cct*: $\bigwedge y.\ y \in T \Longrightarrow$ *connected_component_set T y* $= \{y\}$
shows *f constant_on S*
proof (*cases* $S = \{\}$)
case *True* **then show** *?thesis*
by (*simp add*: *constant_on_def*)
next
case *False*
then have f ‘ $S \subseteq \{f\ x\}$ **if** $x \in S$ **for** x
by (*metis PiE S cct connected_component_maximal connected_continuous_image* [*OF conf*] *fim image_eqI*
image_subset_iff that)
with *False* **show** *?thesis*
unfolding *constant_on_def* **by** *blast*
qed

This proof requires the existence of two separate values of the range type.

lemma *finite_range_constant_imp_connected*:
assumes $\bigwedge f$::*'a*::*topological_space* $\Rightarrow$ *'b*::*real_normed_algebra_1*.
$[\![$*continuous_on S f*; *finite*(f ‘ S)$]\!] \Longrightarrow$ *f constant_on S*
shows *connected S*
proof −
{ fix *T U*
assume *clt*: *closedin* (*top_of_set S*) *T*
and *clu*: *closedin* (*top_of_set S*) *U*
and *tue*: $T \cap U = \{\}$ **and** *tus*: $T \cup U = S$
have *continuous_on* ($T \cup U$) (λx. *if* $x \in T$ *then 0 else 1*)
using *clt clu tue* **by** (*intro continuous_on_cases_local*) (*auto simp*: *tus*)
then have *conif*: *continuous_on S* (λx. *if* $x \in T$ *then 0 else 1*)
using *tus* **by** *blast*
have *fi*: *finite* ((λx. *if* $x \in T$ *then 0 else 1*) ‘ S)
by (*rule finite_subset* [*of* _ $\{0,1\}$]) *auto*
have $T = \{\} \vee U = \{\}$
using *assms* [*OF conif fi*] *tus* [*symmetric*]
by (*auto simp*: *Ball_def constant_on_def*) (*metis IntI empty_iff one_neq_zero tue*)
}
then show *?thesis*
by (*simp add*: *connected_closedin_eq*)
qed

end

theory *Function_Topology*
imports
Elementary_Topology
Abstract_Limits
Connected
begin

2.5 Function Topology

We want to define the general product topology.

The product topology on a product of topological spaces is generated by the sets which are products of open sets along finitely many coordinates, and the whole space along the other coordinates. This is the coarsest topology for which the projection to each factor is continuous.

To form a product of objects in Isabelle/HOL, all these objects should be subsets of a common type 'a. The product is then $Pi_E\ I\ X$, the set of elements from $'i$ to $'a$ such that the i-th coordinate belongs to $X\ i$ for all $i \in I$.

Hence, to form a product of topological spaces, all these spaces should be subsets of a common type. This means that type classes can not be used to define such a product if one wants to take the product of different topological spaces (as the type 'a can only be given one structure of topological space using type classes). On the other hand, one can define different topologies (as introduced in *thy*) on one type, and these topologies do not need to share the same maximal open set. Hence, one can form a product of topologies in this sense, and this works well. The big caveat is that it does not interact well with the main body of topology in Isabelle/HOL defined in terms of type classes... For instance, continuity of maps is not defined in this setting.

As the product of different topological spaces is very important in several areas of mathematics (for instance adeles), I introduce below the product topology in terms of topologies, and reformulate afterwards the consequences in terms of type classes (which are of course very handy for applications).

Given this limitation, it looks to me that it would be very beneficial to revamp the theory of topological spaces in Isabelle/HOL in terms of topologies, and keep the statements involving type classes as consequences of more general statements in terms of topologies (but I am probably too naive here).

Here is an example of a reformulation using topologies. Let

$$\begin{aligned}&\mathit{continuous\_map}\ T1\ T2\ f =\\ &\quad((\forall\ U.\ \mathit{openin}\ T2\ U \longrightarrow \mathit{openin}\ T1\ (f-`U \cap \mathit{topspace}(T1)))\\ &\qquad\wedge\ (f`(\mathit{topspace}\ T1) \subseteq (\mathit{topspace}\ T2)))\end{aligned}$$

be the natural continuity definition of a map from the topology *T1* to the topology *T2*. Then the current *continuous_on* (with type classes) can be redefined as

continuous_on s f =
continuous_map (*top_of_set s*) (*topology euclidean*) *f*

In fact, I need *continuous_map* to express the continuity of the projection on subfactors for the product topology, in Lemma *continuous_on_restrict_product_topology*, and I show the above equivalence in Lemma *continuous_map_iff_continuous*.

I only develop the basics of the product topology in this theory. The most important missing piece is Tychonov theorem, stating that a product of compact spaces is always compact for the product topology, even when the product is not finite (or even countable).

I realized afterwards that this theory has a lot in common with `~~/src/HOL/Library/Finite_Map.thy`.

2.5.1 The product topology

We can now define the product topology, as generated by the sets which are products of open sets along finitely many coordinates, and the whole space along the other coordinates. Equivalently, it is generated by sets which are one open set along one single coordinate, and the whole space along other coordinates. In fact, this is only equivalent for nonempty products, but for the empty product the first formulation is better (the second one gives an empty product space, while an empty product should have exactly one point, equal to *undefined* along all coordinates.

So, we use the first formulation, which moreover seems to give rise to more straightforward proofs.

definition *product_topology*::('*i* ⇒ ('*a topology*)) ⇒ ('*i set*) ⇒ (('*i* ⇒ '*a*) *topology*)
where *product_topology T I* =
topology_generated_by {(Π$_E$ *i*∈*I*. *X i*) |*X*. (∀ *i*. *openin* (*T i*) (*X i*)) ∧ *finite* {*i*. *X i* ≠ *topspace* (*T i*)}}

abbreviation *powertop_real* :: '*a set* ⇒ ('*a* ⇒ *real*) *topology*
where *powertop_real* ≡ *product_topology* (λ*i*. *euclideanreal*)

The total set of the product topology is the product of the total sets along each coordinate.

proposition *product_topology*:
product_topology X I =
topology
(*arbitrary union_of*
((*finite intersection_of*
(λ*F*. ∃ *i U*. *F* = {*f*. *f i* ∈ *U*} ∧ *i* ∈ *I* ∧ *openin* (*X i*) *U*))

```
      relative_to (Π_E i∈I. topspace (X i))))
  (is _ = topology (_ union_of ((_ intersection_of ?Ψ) relative_to ?TOP)))
proof −
  let ?Ω = (λF. ∃ Y. F = Pi_E I Y ∧ (∀ i. openin (X i) (Y i)) ∧ finite {i. Y i ≠
topspace (X i)})
  have ∗: (finite′ intersection_of ?Ω) A = (finite intersection_of ?Ψ relative_to
?TOP) A for A
  proof −
    have 1: ∃ U. (∃𝒰. finite 𝒰 ∧ 𝒰 ⊆ Collect ?Ψ ∧ ⋂𝒰 = U) ∧ ?TOP ∩ U =
⋂𝒰
      if 𝒰: 𝒰 ⊆ Collect ?Ω and finite′ 𝒰 A = ⋂𝒰 𝒰 ≠ {} for 𝒰
    proof −
      have ∀ U ∈ 𝒰. ∃ Y. U = Pi_E I Y ∧ (∀ i. openin (X i) (Y i)) ∧ finite {i. Y
i ≠ topspace (X i)}
        using 𝒰 by auto
      then obtain Y where Y: ⋀U. U ∈ 𝒰 ⟹ U = Pi_E I (Y U) ∧ (∀ i. openin
(X i) (Y U i)) ∧ finite {i. (Y U) i ≠ topspace (X i)}
        by metis
      obtain U where U ∈ 𝒰
        using ‹𝒰 ≠ {}› by blast
      let ?F = λU. (λi. {f. f i ∈ Y U i}) ` {i ∈ I. Y U i ≠ topspace (X i)}
      show ?thesis
      proof (intro conjI exI)
        show finite (⋃ U∈𝒰. ?F U)
          using Y ‹finite′ 𝒰› by auto
        show ?TOP ∩ ⋂ (⋃ U∈𝒰. ?F U) = ⋂𝒰
        proof
          have ∗: f ∈ U
            if U ∈ 𝒰 and ∀ V∈𝒰. ∀ i. i ∈ I ∧ Y V i ≠ topspace (X i) ⟶ f i ∈ Y
V i
              and ∀ i∈I. f i ∈ topspace (X i) and f ∈ extensional I for f U
            by (smt (verit) PiE_iff Y that)
          show ?TOP ∩ ⋂ (⋃ U∈𝒰. ?F U) ⊆ ⋂𝒰
            by (auto simp: PiE_iff ∗)
          show ⋂𝒰 ⊆ ?TOP ∩ ⋂ (⋃ U∈𝒰. ?F U)
            using Y openin_subset ‹finite′ 𝒰› by fastforce
        qed
      qed (use Y openin_subset in ‹blast+›)
    qed
    have 2: ∃𝒰′. finite′ 𝒰′ ∧ 𝒰′ ⊆ Collect ?Ω ∧ ⋂𝒰′ = ?TOP ∩ ⋂𝒰
      if 𝒰: 𝒰 ⊆ Collect ?Ψ and finite 𝒰 for 𝒰
    proof (cases 𝒰={})
      case True
      then show ?thesis
        apply (rule_tac x={?TOP} in exI, simp)
        apply (rule_tac x=λi. topspace (X i) in exI)
        apply force
        done
    next
```

```
      case False
      then obtain U where U ∈ 𝒰
        by blast
      have ∀ U ∈ 𝒰. ∃ i Y. U = {f. f i ∈ Y} ∧ i ∈ I ∧ openin (X i) Y
        using 𝒰 by auto
      then obtain J Y where
        Y: ⋀U. U ∈ 𝒰 ⟹ U = {f. f (J U) ∈ Y U} ∧ J U ∈ I ∧ openin (X (J
U)) (Y U)
        by metis
      let ?G = λU. Π_E i∈I. if i = J U then Y U else topspace (X i)
      show ?thesis
      proof (intro conjI exI)
        show finite (?G ` 𝒰) ?G ` 𝒰 ≠ {}
          using ‹finite 𝒰› ‹U ∈ 𝒰› by blast+
        have ∗: ⋀U. U ∈ 𝒰 ⟹ openin (X (J U)) (Y U)
          using Y by force
        show ?G ` 𝒰 ⊆ {Pi_E I Y |Y. (∀ i. openin (X i) (Y i)) ∧ finite {i. Y i ≠
topspace (X i)}}
          apply clarsimp
          apply (rule_tac x= (λi. if i = J U then Y U else topspace (X i)) in exI)
          apply (auto simp: ∗)
          done
      next
        show (⋂ U∈𝒰. ?G U) = ?TOP ∩ ⋂𝒰
        proof
          have (Π_E i∈I. if i = J U then Y U else topspace (X i)) ⊆ (Π_E i∈I.
topspace (X i))
            by (simp add: PiE_mono Y ‹U ∈ 𝒰› openin_subset)
          then have (⋂ U∈𝒰. ?G U) ⊆ ?TOP
            using ‹U ∈ 𝒰› by fastforce
          moreover have (⋂ U∈𝒰. ?G U) ⊆ ⋂𝒰
            using PiE_mem Y by fastforce
          ultimately show (⋂ U∈𝒰. ?G U) ⊆ ?TOP ∩ ⋂𝒰
            by auto
        qed (use Y in fastforce)
      qed
    qed
    show ?thesis
      unfolding relative_to_def intersection_of_def
      by (safe; blast dest!: 1 2)
  qed
  show ?thesis
    unfolding product_topology_def generate_topology_on_eq
    apply (rule arg_cong [where f = topology])
    apply (rule arg_cong [where f = (union_of)arbitrary])
    apply (force simp: ∗)
    done
qed
```

lemma *topspace_product_topology* [*simp*]:
topspace (*product_topology T I*) = (Π_E *i*∈*I*. *topspace*(*T i*))
proof
show *topspace* (*product_topology T I*) ⊆ (Π_E *i*∈*I*. *topspace* (*T i*))
unfolding *product_topology_def topology_generated_by_topspace*
unfolding *topspace_def* **by** *auto*
have (Π_E *i*∈*I*. *topspace* (*T i*)) ∈ {(Π_E *i*∈*I*. *X i*) |*X*. (∀ *i*. *openin* (*T i*) (*X i*)) ∧ *finite* {*i*. *X i* ≠ *topspace* (*T i*)}}
using *openin_topspace not_finite_existsD* **by** *auto*
then show (Π_E *i*∈*I*. *topspace* (*T i*)) ⊆ *topspace* (*product_topology T I*)
unfolding *product_topology_def* **using** *PiE_def* **by** (*auto*)
qed

lemma *product_topology_trivial_iff*:
product_topology X I = *trivial_topology* ⟷ (∃ *i* ∈ *I*. *X i* = *trivial_topology*)
by (*auto simp*: *PiE_eq_empty_iff simp flip*: *null_topspace_iff_trivial*)

lemma *topspace_product_topology_alt*:
topspace (*product_topology X I*) = {*x* ∈ *extensional I*. ∀ *i* ∈ *I*. *x i* ∈ *topspace*(*X i*)}
by (*fastforce simp*: *PiE_iff*)

lemma *product_topology_basis*:
assumes ⋀*i*. *openin* (*T i*) (*X i*) *finite* {*i*. *X i* ≠ *topspace* (*T i*)}
shows *openin* (*product_topology T I*) (Π_E *i*∈*I*. *X i*)
unfolding *product_topology_def*
by (*rule topology_generated_by_Basis*) (*use assms* **in** *auto*)

proposition *product_topology_open_contains_basis*:
assumes *openin* (*product_topology T I*) *U x* ∈ *U*
shows ∃ *X*. *x* ∈ (Π_E *i*∈*I*. *X i*) ∧ (∀ *i*. *openin* (*T i*) (*X i*)) ∧ *finite* {*i*. *X i* ≠ *topspace* (*T i*)} ∧ (Π_E *i*∈*I*. *X i*) ⊆ *U*
proof −
define *IT* **where** *IT* ≡ λ*X*. {*i*. *X i* ≠ *topspace* (*T i*)}
have *generate_topology_on* {(Π_E *i*∈*I*. *X i*) |*X*. (∀ *i*. *openin* (*T i*) (*X i*)) ∧ *finite* (*IT X*)} *U*
using *assms* **unfolding** *product_topology_def IT_def* **by** (*intro openin_topology_generated_by*) *auto*
then have ⋀*x*. *x*∈*U* ⟹ ∃ *X*. *x* ∈ (Π_E *i*∈*I*. *X i*) ∧ (∀ *i*. *openin* (*T i*) (*X i*)) ∧ *finite* (*IT X*) ∧ (Π_E *i*∈*I*. *X i*) ⊆ *U*
proof *induction*
case (*Int U V x*)
then obtain *XU XV* **where** *H*:
x ∈ *Pi*$_E$ *I XU* ⋀*i*. *openin* (*T i*) (*XU i*) *finite* (*IT XU*) *Pi*$_E$ *I XU* ⊆ *U*
x ∈ *Pi*$_E$ *I XV* ⋀*i*. *openin* (*T i*) (*XV i*) *finite* (*IT XV*) *Pi*$_E$ *I XV* ⊆ *V*
by (*meson Int_iff*)
define *X* **where** *X* = (λ*i*. *XU i* ∩ *XV i*)
have *Pi*$_E$ *I X* ⊆ *Pi*$_E$ *I XU* ∩ *Pi*$_E$ *I XV*
by (*auto simp add*: *PiE_iff X_def*)

```
    then have Pi_E I X ⊆ U ∩ V using H by auto
    moreover have ∀ i. openin (T i) (X i)
      unfolding X_def using H by auto
    moreover have finite (IT X)
      apply (rule rev_finite_subset[of IT XU ∪ IT XV])
      using H by (auto simp: X_def IT_def)
    moreover have x ∈ Pi_E I X
      unfolding X_def using H by auto
    ultimately show ?case
      by auto
  next
    case (UN K x)
    then obtain k where k ∈ K x ∈ k by auto
    with ‹k ∈ K› UN show ?case
      by (meson Sup_upper2)
  qed auto
  then show ?thesis using ‹x ∈ U› IT_def by blast
qed

lemma product_topology_empty_discrete:
   product_topology T {} = discrete_topology {(λx. undefined)}
  by (simp add: subtopology_eq_discrete_topology_sing)

lemma openin_product_topology:
   openin (product_topology X I) =
    arbitrary union_of
          ((finite intersection_of (λF. (∃ i U. F = {f. f i ∈ U} ∧ i ∈ I ∧ openin
(X i) U)))
           relative_to topspace (product_topology X I))
  by (simp add: istopology_subbase product_topology)

lemma subtopology_product_topology:
   subtopology (product_topology X I) (Π_E i∈I. (S i)) = product_topology (λi.
subtopology (X i) (S i)) I
proof −
  let ?P = λF. ∃ i U. F = {f. f i ∈ U} ∧ i ∈ I ∧ openin (X i) U
  let ?X = Π_E i∈I. topspace (X i)
  have finite intersection_of ?P relative_to ?X ∩ Pi_E I S =
        finite intersection_of (?P relative_to ?X ∩ Pi_E I S) relative_to ?X ∩ Pi_E
I S
    by (rule finite_intersection_of_relative_to)
  also have ... = finite intersection_of
                   ((λF. ∃ i U. F = {f. f i ∈ U} ∧ i ∈ I ∧ (openin (X i) relative_to
S i) U)
                      relative_to ?X ∩ Pi_E I S)
                  relative_to ?X ∩ Pi_E I S
    apply (rule arg_cong2 [where f = (relative_to)])
     apply (rule arg_cong [where f = (intersection_of)finite])
     apply (rule ext)
```

```
    apply (auto simp: relative_to_def intersection_of_def)
   done
 finally
 have finite intersection_of ?P relative_to ?X ∩ Pi_E I S =
       finite intersection_of
        (λF. ∃i U. F = {f. f i ∈ U} ∧ i ∈ I ∧ (openin (X i) relative_to S i) U)
        relative_to ?X ∩ Pi_E I S
   by (metis finite_intersection_of_relative_to)
 then show ?thesis
 unfolding topology_eq
 apply clarify
 apply (simp add: openin_product_topology flip: openin_relative_to)
 apply (simp add: arbitrary_union_of_relative_to flip: PiE_Int)
 done
qed

lemma product_topology_base_alt:
  finite intersection_of (λF. (∃i U. F = {f. f i ∈ U} ∧ i ∈ I ∧ openin (X i) U))
   relative_to (Π_E i∈I. topspace (X i)) =
   (λF. (∃U. F =  Pi_E I U ∧ finite {i ∈ I. U i ≠ topspace(X i)} ∧ (∀i ∈ I.
openin (X i) (U i))))
  (is ?lhs = ?rhs)
proof −
 have (∀F. ?lhs F ⟶ ?rhs F)
   unfolding all_relative_to all_intersection_of topspace_product_topology
 proof clarify
   fix ℱ
   assume finite ℱ and ℱ ⊆ {{f. f i ∈ U} |i U. i ∈ I ∧ openin (X i) U}
   then show ∃U. (Π_E i∈I. topspace (X i)) ∩ ⋂ℱ = Pi_E I U ∧
        finite {i ∈ I. U i ≠ topspace (X i)} ∧ (∀i∈I. openin (X i) (U i))
   proof (induction)
     case (insert F ℱ)
     then obtain U where eq: (Π_E i∈I. topspace (X i)) ∩ ⋂ℱ = Pi_E I U
       and fin: finite {i ∈ I. U i ≠ topspace (X i)}
       and ope: ⋀i. i ∈ I ⟹ openin (X i) (U i)
       by auto
     obtain i V where F = {f. f i ∈ V} i ∈ I openin (X i) V
       using insert by auto
     let ?U = λj. U j ∩ (if j = i then V else topspace(X j))
     show ?case
     proof (intro exI conjI)
       show (Π_E i∈I. topspace (X i)) ∩ ⋂(insert F ℱ) = Pi_E I ?U
       using eq  PiE_mem ‹i ∈ I›  by (auto simp: ‹F = {f. f i ∈ V}›) fastforce
     next
       show finite {i ∈ I. ?U i ≠ topspace (X i)}
         by (rule rev_finite_subset [OF finite.insertI [OF fin]]) auto
     next
       show ∀i∈I. openin (X i) (?U i)
         by (simp add: ‹openin (X i) V› ope openin_Int)
```

```
    qed
  qed (auto intro: dest: not_finite_existsD)
 qed
 moreover have (∀ F. ?rhs F ⟶ ?lhs F)
 proof clarify
  fix U :: 'a ⇒ 'b set
  assume fin: finite {i ∈ I. U i ≠ topspace (X i)} and ope: ∀ i∈I. openin (X i)
(U i)
  let ?U = ⋂ i∈{i ∈ I. U i ≠ topspace (X i)}. {x. x i ∈ U i}
  show ?lhs (Pi_E I U)
   unfolding relative_to_def topspace_product_topology
  proof (intro exI conjI)
   show (finite intersection_of (λF. ∃ i U. F = {f. f i ∈ U} ∧ i ∈ I ∧ openin
(X i) U)) ?U
    using fin ope by (intro finite_intersection_of_Inter finite_intersection_of_inc)
auto
    show (Π_E i∈I. topspace (X i)) ∩ ?U = Pi_E I U
     using ope openin_subset by fastforce
  qed
 qed
 ultimately show ?thesis
  by meson
qed

corollary openin_product_topology_alt:
 openin (product_topology X I) S ⟷
  (∀ x ∈ S. ∃ U. finite {i ∈ I. U i ≠ topspace(X i)} ∧
            (∀ i ∈ I. openin (X i) (U i)) ∧ x ∈ Pi_E I U ∧ Pi_E I U ⊆ S)
 unfolding openin_product_topology arbitrary_union_of_alt product_topology_base_alt
topspace_product_topology
 by (smt (verit, best))

lemma closure_of_product_topology:
 (product_topology X I) closure_of (PiE I S) = PiE I (λi. (X i) closure_of (S
i))
proof −
 have ∗: (∀ T. f ∈ T ∧ openin (product_topology X I) T ⟶ (∃ y∈Pi_E I S. y ∈
T))
      ⟷ (∀ i ∈ I. ∀ T. f i ∈ T ∧ openin (X i) T ⟶ S i ∩ T ≠ {})
  (is ?lhs = ?rhs)
  if top: ⋀i. i ∈ I ⟹ f i ∈ topspace (X i) and ext: f ∈ extensional I for f
 proof
  assume L[rule_format]: ?lhs
  show ?rhs
  proof clarify
   fix i T
   assume i ∈ I f i ∈ T openin (X i) T S i ∩ T = {}
   then have openin (product_topology X I) ((Π_E i∈I. topspace (X i)) ∩ {x. x
i ∈ T})
```

by (*force simp*: *openin_product_topology intro*: *arbitrary_union_of_inc relative_to_inc finite_intersection_of_inc*)
then show *False*
using *L* [*of topspace* (*product_topology X I*) ∩ {*f*. *f i* ∈ *T*}] ‹*S i* ∩ *T* = {}› ‹*f i* ∈ *T*› ‹*i* ∈ *I*›
by (*auto simp*: *top ext PiE_iff*)
qed
next
assume *R* [*rule_format*]: *?rhs*
show *?lhs*
proof (*clarsimp simp*: *openin_product_topology union_of_def arbitrary_def*)
fix *𝒰 U*
assume
𝒰: *𝒰* ⊆ *Collect*
(*finite intersection_of* (λ*F*. ∃ *i U*. *F* = {*x*. *x i* ∈ *U*} ∧ *i* ∈ *I* ∧ *openin* (*X i*) *U*) *relative_to*
(Π$_E$ *i*∈*I*. *topspace* (*X i*))) **and**
f ∈ *U U* ∈ *𝒰*
then have (*finite intersection_of* (λ*F*. ∃ *i U*. *F* = {*x*. *x i* ∈ *U*} ∧ *i* ∈ *I* ∧ *openin* (*X i*) *U*)
relative_to (Π$_E$ *i*∈*I*. *topspace* (*X i*))) *U*
by *blast*
with ‹*f* ∈ *U*› ‹*U* ∈ *𝒰*›
obtain *𝒯* **where** *finite 𝒯*
and *𝒯*: ⋀*C*. *C* ∈ *𝒯* ⟹ ∃ *i* ∈ *I*. ∃ *V*. *openin* (*X i*) *V* ∧ *C* = {*x*. *x i* ∈ *V*}
and *topspace* (*product_topology X I*) ∩ ⋂ *𝒯* ⊆ *U f* ∈ *topspace* (*product_topology X I*) ∩ ⋂ *𝒯*
apply (*clarsimp simp add*: *relative_to_def intersection_of_def*)
apply (*rule that*, *auto dest*!: *subsetD*)
done
then have *f* ∈ *PiE I* (*topspace* ∘ *X*) *f* ∈ ⋂*𝒯* **and** *subU*: *PiE I* (*topspace* ∘ *X*) ∩ ⋂*𝒯* ⊆ *U*
by (*auto simp*: *PiE_iff*)
have ∗: *f i* ∈ *topspace* (*X i*) ∩ ⋂{*U*. *openin* (*X i*) *U* ∧ {*x*. *x i* ∈ *U*} ∈ *𝒯*}
∧ *openin* (*X i*) (*topspace* (*X i*) ∩ ⋂{*U*. *openin* (*X i*) *U* ∧ {*x*. *x i* ∈ *U*} ∈ *𝒯*})
if *i* ∈ *I* **for** *i*
proof −
have *finite* ((λ*U*. {*x*. *x i* ∈ *U*}) −‘ *𝒯*)
proof (*rule finite_vimageI* [*OF* ‹*finite 𝒯*›])
show *inj* (λ*U*. {*x*. *x i* ∈ *U*})
by (*auto simp*: *inj_on_def*)
qed
then have *fin*: *finite* {*U*. *openin* (*X i*) *U* ∧ {*x*. *x i* ∈ *U*} ∈ *𝒯*}
by (*rule rev_finite_subset*) *auto*
have *openin* (*X i*) (⋂ (*insert* (*topspace* (*X i*)) {*U*. *openin* (*X i*) *U* ∧ {*x*. *x i* ∈ *U*} ∈ *𝒯*}))
by (*rule openin_Inter*) (*auto simp*: *fin*)
then show *?thesis*

```
        using ‹f ∈ ⋂ 𝒯› by (fastforce simp: that top)
      qed
      define Φ where Φ ≡ λi. topspace (X i) ∩ ⋂{U. openin (X i) U ∧ {f. f i ∈
U} ∈ 𝒯}
      have ∀ i ∈ I. ∃ x. x ∈ S i ∩ Φ i
        using R [OF _ *] unfolding Φ_def by blast
      then obtain ϑ where ϑ [rule_format]: ∀ i ∈ I. ϑ i ∈ S i ∩ Φ i
        by metis
      show ∃ y∈Pi_E I S. ∃ x∈𝒰. y ∈ x
      proof
        show ∃ U ∈ 𝒰. (λi ∈ I. ϑ i) ∈ U
        proof
          have restrict ϑ I ∈ PiE I (topspace ∘ X) ∩ ⋂𝒯
            using 𝒯 by (fastforce simp: Φ_def PiE_def dest: ϑ)
          then show restrict ϑ I ∈ U
            using subU by blast
        qed (rule ‹U ∈ 𝒰›)
      next
        show (λi ∈ I. ϑ i) ∈ Pi_E I S
          using ϑ by simp
      qed
    qed
  qed
  show ?thesis
    apply (simp add: * closure_of_def PiE_iff set_eq_iff cong: conj_cong)
    by metis
qed

corollary closedin_product_topology:
   closedin (product_topology X I) (PiE I S) ⟷ PiE I S = {} ∨ (∀ i ∈ I. closedin
(X i) (S i))
  by (smt (verit, best) PiE_eq closedin_empty closure_of_eq closure_of_product_topology)

corollary closedin_product_topology_singleton:
   f ∈ extensional I ⟹ closedin (product_topology X I) {f} ⟷ (∀ i ∈ I. closedin
(X i) {f i})
  using PiE_singleton closedin_product_topology [of X I]
  by (metis (no_types, lifting) all_not_in_conv insertI1)

lemma product_topology_empty:
   product_topology X {} = topology (λS. S ∈ {{},{λk. undefined}})
  unfolding product_topology union_of_def intersection_of_def arbitrary_def rel-
ative_to_def
  by (auto intro: arg_cong [where f=topology])

lemma openin_product_topology_empty: openin (product_topology X {}) S ⟷
S ∈ {{},{λk. undefined}}
  unfolding union_of_def intersection_of_def arbitrary_def relative_to_def openin_product_topology
  by auto
```

The basic property of the product topology is the continuity of projections:

```
lemma continuous_map_product_coordinates [simp]:
  assumes i ∈ I
  shows continuous_map (product_topology T I) (T i) (λx. x i)
proof −
  {
    fix U assume openin (T i) U
    define X where X = (λj. if j = i then U else topspace (T j))
    then have ∗: (λx. x i) −‘ U ∩ (Π_E i∈I. topspace (T i)) = (Π_E j∈I. X j)
      unfolding X_def using assms openin_subset[OF ‹openin (T i) U›]
      by (auto simp add: PiE_iff, auto, metis subsetCE)
    have ∗∗: (∀ i. openin (T i) (X i)) ∧ finite {i. X i ≠ topspace (T i)}
      unfolding X_def using ‹openin (T i) U› by auto
   have openin (product_topology T I) ((λx. x i) −‘ U ∩ (Π_E i∈I. topspace (T
i)))
      unfolding product_topology_def
      apply (rule topology_generated_by_Basis)
      apply (subst ∗)
      using ∗∗ by auto
  }
  then show ?thesis unfolding continuous_map_alt
    by (auto simp add: assms PiE_iff)
qed

lemma continuous_map_coordinatewise_then_product [intro]:
  assumes ⋀i. i ∈ I ⟹ continuous_map T1 (T i) (λx. f x i)
        ⋀i x. i ∉ I ⟹ x ∈ topspace T1 ⟹ f x i = undefined
  shows continuous_map T1 (product_topology T I) f
unfolding product_topology_def
proof (rule continuous_on_generated_topo)
  fix U assume U ∈ {Pi_E I X |X. (∀ i. openin (T i) (X i)) ∧ finite {i. X i ≠
topspace (T i)}}
  then obtain X where H: U = Pi_E I X ⋀i. openin (T i) (X i) finite {i. X i ≠
topspace (T i)}
    by blast
  define J where J = {i ∈ I. X i ≠ topspace (T i)}
  have finite J J ⊆ I unfolding J_def using H(3) by auto
  have (λx. f x i)−‘(topspace(T i)) ∩ topspace T1 = topspace T1 if i ∈ I for i
    using that assms(1) by (simp add: continuous_map_preimage_topspace)
  then have ∗: (λx. f x i)−‘(X i) ∩ topspace T1 = topspace T1 if i ∈ I−J for i
    using that unfolding J_def by auto
   have f−‘U ∩ topspace T1 = (⋂i∈I. (λx. f x i)−‘(X i) ∩ topspace T1) ∩
(topspace T1)
    by (subst H(1), auto simp add: PiE_iff assms)
  also have ... = (⋂i∈J. (λx. f x i)−‘(X i) ∩ topspace T1) ∩ (topspace T1)
    using ∗ ‹J ⊆ I› by auto
  also have openin T1 (...)
    using H(2) ‹J ⊆ I› ‹finite J› assms(1) by blast
```

ultimately show *openin T1* (*f*−‘*U* ∩ *topspace T1*) **by** *simp*
next
have *f* ∈ *topspace T1* → *topspace* (*product_topology T I*)
using *assms continuous_map_funspace* **by** (*force simp*: *Pi_iff*)
then show *f* ‘*topspace T1* ⊆ ⋃{*Pi*$_E$ *I X* |*X*. (∀ *i*. *openin* (*T i*) (*X i*)) ∧ *finite* {*i*. *X i* ≠ *topspace* (*T i*)}}
by (*fastforce simp add*: *product_topology_def Pi_iff*)
qed

lemma *continuous_map_product_then_coordinatewise* [*intro*]:
assumes *continuous_map T1* (*product_topology T I*) *f*
shows ⋀*i*. *i* ∈ *I* ⟹ *continuous_map T1* (*T i*) (λ*x*. *f x i*)
⋀*i x*. *i* ∉ *I* ⟹ *x* ∈ *topspace T1* ⟹ *f x i* = *undefined*
proof −
fix *i* **assume** *i* ∈ *I*
have (λ*x*. *f x i*) = (λ*y*. *y i*) *o f* **by** *auto*
also have *continuous_map T1* (*T i*) (...)
by (*metis* ‹*i* ∈ *I*› *assms continuous_map_compose continuous_map_product_coordinates*)
ultimately show *continuous_map T1* (*T i*) (λ*x*. *f x i*)
by *simp*
next
fix *i x* **assume** *i* ∉ *I x* ∈ *topspace T1*
have *f x* ∈ *topspace* (*product_topology T I*)
using *assms* ‹*x* ∈ *topspace T1*› **unfolding** *continuous_map_def* **by** *auto*
then have *f x* ∈ (Π$_E$ *i*∈*I*. *topspace* (*T i*))
using *topspace_product_topology* **by** *metis*
then show *f x i* = *undefined*
using ‹*i* ∉ *I*› **by** (*auto simp add*: *PiE_iff extensional_def*)
qed

lemma *continuous_on_restrict*:
assumes *J* ⊆ *I*
shows *continuous_map* (*product_topology T I*) (*product_topology T J*) (λ*x*. *restrict x J*)
proof (*rule continuous_map_coordinatewise_then_product*)
fix *i* **assume** *i* ∈ *J*
then have (λ*x*. *restrict x J i*) = (λ*x*. *x i*) **unfolding** *restrict_def* **by** *auto*
then show *continuous_map* (*product_topology T I*) (*T i*) (λ*x*. *restrict x J i*)
using ‹*i* ∈ *J*› ‹*J* ⊆ *I*› **by** *auto*
next
fix *i* **assume** *i* ∉ *J*
then show *restrict x J i* = *undefined* **for** *x*::*'a* ⇒ *'b*
unfolding *restrict_def* **by** *auto*
qed

Powers of a single topological space as a topological space, using type classes

instantiation *fun* :: (*type*, *topological_space*) *topological_space*

begin

definition *open_fun_def*:
open U = openin (product_topology (λi. euclidean) UNIV) U

instance proof
have *topspace (product_topology (λ(i::'a). euclidean::('b topology)) UNIV) = UNIV*
unfolding *topspace_product_topology topspace_euclidean* **by** *auto*
then show *open (UNIV::('a ⇒ 'b) set)*
unfolding *open_fun_def* **by** *(metis openin_topspace)*
qed *(auto simp add: open_fun_def)*

end

lemma *open_PiE [intro?]*:
fixes *X::'i ⇒ ('b::topological_space) set*
assumes *⋀i. open (X i) finite {i. X i ≠ UNIV}*
shows *open (Pi$_E$ UNIV X)*
by *(simp add: assms open_fun_def product_topology_basis)*

lemma *euclidean_product_topology*:
product_topology (λi. euclidean::('b::topological_space) topology) UNIV = euclidean
by *(metis open_openin topology_eq open_fun_def)*

proposition *product_topology_basis'*:
fixes *x::'i ⇒ 'a* **and** *U::'i ⇒ ('b::topological_space) set*
assumes *finite I ⋀i. i ∈ I ⟹ open (U i)*
shows *open {f. ∀i∈I. f (x i) ∈ U i}*
proof −
define *V* **where** *V ≡ (λy. if y ∈ x'I then ⋂{U i|i. i∈I ∧ x i = y} else UNIV)*
define *X* **where** *X ≡ (λy. if y ∈ x'I then V y else UNIV)*
have *∗: open (X i)* **for** *i*
unfolding *X_def V_def* **using** *assms* **by** *auto*
then have *open (Pi$_E$ UNIV X)*
by *(simp add: X_def assms(1) open_PiE)*
moreover have *Pi$_E$ UNIV X = {f. ∀i∈I. f (x i) ∈ U i}*
by *(fastforce simp add: PiE_iff X_def V_def split: if_split_asm)*
ultimately show *?thesis* **by** *simp*
qed

The results proved in the general situation of products of possibly different spaces have their counterparts in this simpler setting.

lemma *continuous_on_product_coordinates [simp]*:
continuous_on UNIV (λx. x i::('b::topological_space))
using *continuous_map_product_coordinates [of _ UNIV λi. euclidean]*
by *(metis (no_types) continuous_map_iff_continuous euclidean_product_topology iso_tuple_UNIV_I subtopology_UNIV)*

lemma *continuous_on_coordinatewise_then_product* [*continuous_intros*]:
fixes *f* :: *'a*::*topological_space* ⇒ *'b* ⇒ *'c*::*topological_space*
assumes ⋀*i*. *continuous_on S* (λ*x*. *f x i*)
shows *continuous_on S f*
by (*metis UNIV_I assms continuous_map_iff_continuous euclidean_product_topology continuous_map_coordinatewise_then_product*)

lemma *continuous_on_product_then_coordinatewise*:
assumes *continuous_on S f*
shows *continuous_on S* (λ*x*. *f x i*)
by (*metis UNIV_I assms continuous_map_iff_continuous continuous_map_product_then_coordinate*
euclidean_product_topology)

lemma *continuous_on_coordinatewise_iff*:
fixes *f* :: (*'a* ⇒ *real*) ⇒ *'b* ⇒ *real*
shows *continuous_on* (*A* ∩ *S*) *f* ⟷ (∀ *i*. *continuous_on* (*A* ∩ *S*) (λ*x*. *f x i*))
by (*auto simp*: *continuous_on_product_then_coordinatewise continuous_on_coordinatewise_then_pro*

lemma *continuous_map_span_sum*:
fixes *B* :: *'a*::*real_normed_vector set*
assumes *biB*: ⋀*i*. *i* ∈ *I* ⟹ *b i* ∈ *B*
shows *continuous_map euclidean* (*top_of_set* (*span B*)) (λ*x*. ∑ *i*∈*I*. *x i* ∗*R b*
i)
proof (*rule continuous_map_euclidean_top_of_set*)
show (λ*x*. ∑ *i*∈*I*. *x i* ∗*R b i*) −‘ *span B* = *UNIV*
by *auto* (*meson biB lessThan_iff span_base span_scale span_sum*)
show *continuous_on UNIV* (λ*x*. ∑ *i*∈ *I*. *x i* ∗*R b i*)
by (*intro continuous_intros*) *auto*
qed

Topological countability for product spaces

The next two lemmas are useful to prove first or second countability of product spaces, but they have more to do with countability and could be put in the corresponding theory.

lemma *countable_nat_product_event_const*:
fixes *F*::*'a set* **and** *a*::*'a*
assumes *a* ∈ *F countable F*
shows *countable* {*x*::(*nat* ⇒ *'a*). (∀ *i*. *x i* ∈ *F*) ∧ *finite* {*i*. *x i* ≠ *a*}}
proof −
have ∗: {*x*::(*nat* ⇒ *'a*). (∀ *i*. *x i* ∈ *F*) ∧ *finite* {*i*. *x i* ≠ *a*}}
⊆ (⋃ *N*. {*x*. (∀ *i*. *x i* ∈ *F*) ∧ (∀ *i*≥*N*. *x i* = *a*)})
using *infinite_nat_iff_unbounded_le* **by** *fastforce*
have *countable* {*x*. (∀ *i*. *x i* ∈ *F*) ∧ (∀ *i*≥*N*. *x i* = *a*)} **for** *N*::*nat*
proof (*induction N*)
case *0*
have {*x*. (∀ *i*. *x i* ∈ *F*) ∧ (∀ *i*≥(*0*::*nat*). *x i* = *a*)} = {(λ*i*. *a*)}
using ‹*a* ∈ *F*› **by** *auto*

```
    then show ?case by auto
  next
    case (Suc N)
    define f::((nat ⇒ 'a) × 'a) ⇒ (nat ⇒ 'a)
      where f = (λ(x, b). x(N:=b))
     have {x. (∀ i. x i ∈ F) ∧ (∀ i≥Suc N. x i = a)} ⊆ f‘({x. (∀ i. x i ∈ F) ∧
(∀ i≥N. x i = a)} × F)
    proof (auto)
      fix x assume H: ∀ i::nat. x i ∈ F ∀ i≥Suc N. x i = a
      have f (x(N:=a), x N) = x
        unfolding f_def by auto
      moreover have (x(N:=a), x N) ∈ {x. (∀ i. x i ∈ F) ∧ (∀ i≥N. x i = a)} ×
F
        using H ‹a ∈ F› by auto
      ultimately show x ∈ f ‘ ({x. (∀ i. x i ∈ F) ∧ (∀ i≥N. x i = a)} × F)
        by (metis (no_types, lifting) image_eqI)
    qed
    moreover have countable ({x. (∀ i. x i ∈ F) ∧ (∀ i≥N. x i = a)} × F)
      using Suc.IH assms(2) by auto
    ultimately show ?case
      by (meson countable_image countable_subset)
  qed
  then show ?thesis using countable_subset[OF *] by auto
qed

lemma countable_product_event_const:
  fixes F::('a::countable) ⇒ 'b set and b::'b
  assumes ⋀i. countable (F i)
  shows countable {f::('a ⇒ 'b). (∀ i. f i ∈ F i) ∧ (finite {i. f i ≠ b})}
proof −
  define G where G = (⋃ i. F i) ∪ {b}
  have countable G unfolding G_def using assms by auto
  have b ∈ G unfolding G_def by auto
  define pi where pi = (λ(x::(nat ⇒ 'b)). (λ i::'a. x ((to_nat::('a ⇒ nat)) i)))
  have {f::('a ⇒ 'b). (∀ i. f i ∈ F i) ∧ (finite {i. f i ≠ b})}
        ⊆ pi‘{g::(nat ⇒ 'b). (∀ j. g j ∈ G) ∧ (finite {j. g j ≠ b})}
  proof (auto)
    fix f assume H: ∀ i. f i ∈ F i finite {i. f i ≠ b}
    define I where I = {i. f i ≠ b}
    define g where g = (λj. if j ∈ to_nat‘I then f (from_nat j) else b)
    have {j. g j ≠ b} ⊆ to_nat‘I unfolding g_def by auto
    then have finite {j. g j ≠ b}
      unfolding I_def using H(2) using finite_surj by blast
    moreover have g j ∈ G for j
      unfolding g_def G_def using H by auto
    ultimately have g ∈ {g::(nat ⇒ 'b). (∀ j. g j ∈ G) ∧ (finite {j. g j ≠ b})}
      by auto
    moreover have f = pi g
      unfolding pi_def g_def I_def using H by fastforce
```

```
    ultimately show f ∈ pi‘{g. (∀ j. g j ∈ G) ∧ finite {j. g j ≠ b}}
      by auto
  qed
  then show ?thesis
    using countable_nat_product_event_const[OF ‹b ∈ G› ‹countable G›]
    by (meson countable_image countable_subset)
qed

instance fun :: (countable, first_countable_topology) first_countable_topology
proof
  fix x::'a ⇒ 'b
  have ∃ A::('b ⇒ nat ⇒ 'b set). ∀ x. (∀ i. x ∈ A x i ∧ open (A x i)) ∧ (∀ S. open
S ∧ x ∈ S ⟶ (∃ i. A x i ⊆ S))
    apply (rule choice) using first_countable_basis by auto
  then obtain A::('b ⇒ nat ⇒ 'b set) where A: ⋀x i. x ∈ A x i
                                              ⋀x i. open (A x i)
                                            ⋀x S. open S ⟹ x ∈ S ⟹ (∃ i. A x i ⊆ S)
    by metis
```

$B\ i$ is a countable basis of neighborhoods of x_i.

```
  define B where B = (λi. (A (x i))‘UNIV ∪ {UNIV})
  have countB: countable (B i) for i unfolding B_def by auto
  have open_B: ⋀X i. X ∈ B i ⟹ open X
    by (auto simp: B_def A)
  define K where K = {Pi_E UNIV X | X. (∀ i. X i ∈ B i) ∧ finite {i. X i ≠
UNIV}}
  have Pi_E UNIV (λi. UNIV) ∈ K
    unfolding K_def B_def by auto
  then have K ≠ {} by auto
  have countable {X. (∀ i. X i ∈ B i) ∧ finite {i. X i ≠ UNIV}}
    by (simp add: countB countable_product_event_const)
  moreover have K = (λX. Pi_E UNIV X)‘{X. (∀ i. X i ∈ B i) ∧ finite {i. X i
≠ UNIV}}
    unfolding K_def by auto
  ultimately have countable K by auto
  have I: x ∈ k if k ∈ K for k
    using that unfolding K_def B_def apply auto using A(1) by auto
  have II: open k if k ∈ K for k
    using that unfolding K_def by (blast intro: open_B open_PiE)
  have Inc: ∃ k∈K. k ⊆ U if open U ∧ x ∈ U for U
  proof −
    have openin (product_topology (λi. euclidean) UNIV) U x ∈ U
      using ‹open U ∧ x ∈ U› unfolding open_fun_def by auto
    with product_topology_open_contains_basis[OF this]
    have ∃ X. x ∈ (Π_E i∈UNIV. X i) ∧ (∀ i. open (X i)) ∧ finite {i. X i ≠ UNIV}
∧ (Π_E i∈UNIV. X i) ⊆ U
      by simp
    then obtain X where H: x ∈ (Π_E i∈UNIV. X i)
                          ⋀i. open (X i)
```

finite $\{i.\ X\ i \neq UNIV\}$
$(\Pi_E\ i{\in}UNIV.\ X\ i) \subseteq U$
by *auto*
define I **where** $I = \{i.\ X\ i \neq UNIV\}$
define Y **where** $Y = (\lambda i.\ if\ i \in I\ then\ (SOME\ y.\ y \in B\ i \wedge y \subseteq X\ i)\ else\ UNIV)$
have $*$: $\exists\, y.\ y \in B\ i \wedge y \subseteq X\ i$ **for** i
unfolding *B_def* **using** $A(3)[OF\ H(2)]\ H(1)$ **by** (*metis PiE_E UNIV_I UnCI image_iff*)
have $**$: $Y\ i \in B\ i \wedge Y\ i \subseteq X\ i$ **for** i
proof (*cases* $i \in I$)
case *True*
then show *?thesis*
by (*metis* (*mono_tags*, *lifting*) $*$ *Nitpick.Eps_psimp Y_def*)
next
case *False*
then show *?thesis* **by** (*simp add*: *B_def I_def Y_def*)
qed
have $\{i.\ Y\ i \neq UNIV\} \subseteq I$
unfolding *Y_def* **by** *auto*
with $**$ **have** $(\forall\, i.\ Y\ i \in B\ i) \wedge$ *finite* $\{i.\ Y\ i \neq UNIV\}$
using $H(3)$ *I_def finite_subset* **by** *blast*
then have $Pi_E\ UNIV\ Y \in K$
unfolding *K_def* **by** *auto*
have $Y\ i \subseteq X\ i$ **for** i
using $**$ **by** *auto*
then have $Pi_E\ UNIV\ Y \subseteq U$
by (*metis* $H(4)$ *PiE_mono subset_trans*)
then show *?thesis* **using** ‹$Pi_E\ UNIV\ Y \in K$› **by** *auto*
qed
show $\exists\, L.\ (\forall\, (i{::}nat).\ x \in L\ i \wedge open\ (L\ i)) \wedge (\forall\, U.\ open\ U \wedge x \in U \longrightarrow (\exists\, i.\ L\ i \subseteq U))$
using ‹*countable K*› *I II Inc* **by** (*simp add*: *first_countableI*)
qed

proposition *product_topology_countable_basis*:
shows $\exists\, K{::}((\,'a{::}countable \Rightarrow\, 'b{::}second\_countable\_topology)\ set\ set).$
topological_basis K $\wedge$ *countable K* $\wedge$
$(\forall\, k{\in}K.\ \exists\, X.\ (k = Pi_E\ UNIV\ X) \wedge (\forall\, i.\ open\ (X\ i)) \wedge$ *finite* $\{i.\ X\ i \neq UNIV\})$
proof −
obtain $B{::}\,'b\ set\ set$ **where** B: *countable B* $\wedge$ *topological_basis B*
using *ex_countable_basis* **by** *auto*
then have $B \neq \{\}$ **by** (*meson UNIV_I empty_iff open_UNIV topological_basisE*)
define *B2* **where** $B2 = B \cup \{UNIV\}$
have *countable B2*
unfolding *B2_def* **using** B **by** *auto*
have *open U* **if** $U \in B2$ **for** U
using *that* **unfolding** *B2_def* **using** B *topological_basis_open* **by** *auto*

define K **where** $K = \{Pi_E\ UNIV\ X \mid X.\ (\forall i::'a.\ X\ i \in B2) \wedge finite\ \{i.\ X\ i \neq UNIV\}\}$
have i: $\forall k{\in}K.\ \exists X.\ (k = Pi_E\ UNIV\ X) \wedge (\forall i.\ open\ (X\ i)) \wedge finite\ \{i.\ X\ i \neq UNIV\}$
unfolding K_def **using** ‹$\bigwedge U.\ U \in B2 \Longrightarrow open\ U$› **by** *auto*

have *countable* $\{X.\ (\forall (i::'a).\ X\ i \in B2) \wedge finite\ \{i.\ X\ i \neq UNIV\}\}$
using ‹*countable B2*› **by** (*intro countable_product_event_const*) *auto*
moreover have $K = (\lambda X.\ Pi_E\ UNIV\ X)\text{`}\{X.\ (\forall i.\ X\ i \in B2) \wedge finite\ \{i.\ X\ i \neq UNIV\}\}$
unfolding K_def **by** *auto*
ultimately have *ii*: *countable K* **by** *auto*

have *iii*: *topological_basis K*
proof (*rule topological_basisI*)
fix U **and** $x::'a{\Rightarrow}'b$ **assume** *open U x* ∈ *U*
then have *openin* (*product_topology* (λ*i. euclidean*) *UNIV*) *U*
unfolding *open_fun_def* **by** *auto*
with *product_topology_open_contains_basis*[*OF this* ‹*x* ∈ *U*›]
obtain X **where** H: $x \in (\Pi_E\ i{\in}UNIV.\ X\ i)$
$\bigwedge i.\ open\ (X\ i)$
$finite\ \{i.\ X\ i \neq UNIV\}$
$(\Pi_E\ i{\in}UNIV.\ X\ i) \subseteq U$
by *auto*
then have $x\ i \in X\ i$ **for** i **by** *auto*
define I **where** $I = \{i.\ X\ i \neq UNIV\}$
define Y **where** $Y = (\lambda i.\ if\ i \in I\ then\ (SOME\ y.\ y \in B2 \wedge y \subseteq X\ i \wedge x\ i \in y)\ else\ UNIV)$
have $*$: $\exists y.\ y \in B2 \wedge y \subseteq X\ i \wedge x\ i \in y$ **for** i
unfolding *B2_def* **using** *B* ‹*open* (*X i*)› ‹*x i* ∈ *X i*› **by** (*meson UnCI topological_basisE*)
have $**$: $Y\ i \in B2 \wedge Y\ i \subseteq X\ i \wedge x\ i \in Y\ i$ **for** i
using *someI_ex*[*OF* ∗] **by** (*simp add*: *B2_def I_def Y_def*)
have $\{i.\ Y\ i \neq UNIV\} \subseteq I$
unfolding *Y_def* **by** *auto*
then have $(\forall i.\ Y\ i \in B2) \wedge finite\ \{i.\ Y\ i \neq UNIV\}$
using ∗∗ *H*(*3*) *I_def finite_subset* **by** *blast*
then have $Pi_E\ UNIV\ Y \in K$
unfolding *K_def* **by** *auto*
then show $\exists V{\in}K.\ x \in V \wedge V \subseteq U$
by (*meson* ∗∗ *H*(*4*) *PiE_I PiE_mono UNIV_I order.trans*)
next
fix U **assume** $U \in K$
show *open U*
using ‹*U* ∈ *K*› **unfolding** *open_fun_def K_def* **by** *clarify* (*metis* ‹*U* ∈ *K*› *i open_PiE open_fun_def*)
qed

show *?thesis* **using** *i ii iii* **by** *auto*
qed

instance *fun* :: (*countable*, *second_countable_topology*) *second_countable_topology*
proof
show $\exists B{::}('a \Rightarrow 'b)\ set\ set.\ countable\ B \wedge open = generate\_topology\ B$
using *product_topology_countable_basis topological_basis_imp_subbasis*
by *auto*
qed

2.5.2 The Alexander subbase theorem

theorem *Alexander_subbase*:
assumes X: *topology* (*arbitrary union_of* (*finite intersection_of* $(\lambda x.\ x \in \mathcal{B})$ *relative_to* $\bigcup\mathcal{B}$)) $= X$
and *fin*: $\bigwedge C.\ [\![C \subseteq \mathcal{B};\ \bigcup C = topspace\ X]\!] \Longrightarrow \exists C'.\ finite\ C' \wedge C' \subseteq C \wedge \bigcup C' = topspace\ X$
shows *compact_space X*
proof −
have *UB*: $\bigcup\mathcal{B} = topspace\ X$
by (*simp flip*: X)
have *False* **if** $\mathcal{U}$: $\forall U{\in}\mathcal{U}.\ openin\ X\ U$ **and** *sub*: $topspace\ X \subseteq \bigcup\mathcal{U}$
and *neg*: $\bigwedge\mathcal{F}.\ [\![\mathcal{F} \subseteq \mathcal{U};\ finite\ \mathcal{F}]\!] \Longrightarrow \neg\ topspace\ X \subseteq \bigcup\mathcal{F}$ **for** $\mathcal{U}$
proof −
define $\mathcal{A}$ **where** $\mathcal{A} \equiv \{\mathcal{C}.\ (\forall U \in \mathcal{C}.\ openin\ X\ U) \wedge topspace\ X \subseteq \bigcup\mathcal{C} \wedge (\forall\mathcal{F}.\ finite\ \mathcal{F} \longrightarrow \mathcal{F} \subseteq \mathcal{C} \longrightarrow {\sim}(topspace\ X \subseteq \bigcup\mathcal{F}))\}$
have *1*: $\mathcal{A} \neq \{\}$
unfolding *A_def* **using** *sub* $\mathcal{U}$ *neg* **by** *force*
have *2*: $\bigcup\mathcal{C} \in \mathcal{A}$ **if** $\mathcal{C}{\neq}\{\}$ **and** $\mathcal{C}$: *subset.chain* $\mathcal{A}$ $\mathcal{C}$ **for** $\mathcal{C}$
unfolding *A_def*
proof (*intro CollectI conjI ballI allI impI notI*)
show *openin X U* **if** U: $U \in \bigcup\mathcal{C}$ **for** U
using U $\mathcal{C}$ **unfolding** *A_def subset_chain_def* **by** *force*
have $\mathcal{C} \subseteq \mathcal{A}$
using *subset_chain_def* $\mathcal{C}$ **by** *blast*
with *that A_def* **show** *UUC*: $topspace\ X \subseteq \bigcup(\bigcup\mathcal{C})$
by *blast*
show *False* **if** *finite* $\mathcal{F}$ **and** $\mathcal{F} \subseteq \bigcup\mathcal{C}$ **and** $topspace\ X \subseteq \bigcup\mathcal{F}$ **for** $\mathcal{F}$
proof −
obtain $\mathcal{B}$ **where** $\mathcal{B} \in \mathcal{C}$ $\mathcal{F} \subseteq \mathcal{B}$
by (*metis Sup_empty* $\mathcal{C}$ ‹$\mathcal{F} \subseteq \bigcup\mathcal{C}$› ‹*finite* $\mathcal{F}$› *UUC empty_subsetI finite.emptyI finite_subset_Union_chain neg*)
then show *False*
using *A_def* ‹$\mathcal{C} \subseteq \mathcal{A}$› ‹*finite* $\mathcal{F}$› ‹$topspace\ X \subseteq \bigcup\mathcal{F}$› **by** *blast*
qed
qed
obtain $\mathcal{K}$ **where** $\mathcal{K} \in \mathcal{A}$ **and** $\bigwedge X.\ [\![X{\in}\mathcal{A};\ \mathcal{K} \subseteq X]\!] \Longrightarrow X = \mathcal{K}$
using *subset_Zorn_nonempty* [*OF 1 2*] **by** *metis*
then have $*$: $\bigwedge\mathcal{W}.\ [\![\bigwedge W.\ W{\in}\mathcal{W} \Longrightarrow openin\ X\ W;\ topspace\ X \subseteq \bigcup\mathcal{W};\ \mathcal{K} \subseteq$

$\mathcal{W}$;
$\bigwedge\mathcal{F}.\ [\![\mathit{finite}\ \mathcal{F};\ \mathcal{F} \subseteq \mathcal{W};\ \mathit{topspace}\ X \subseteq \bigcup\mathcal{F}]\!] \Longrightarrow \mathit{False}]\!]$
$\Longrightarrow \mathcal{W} = \mathcal{K}$
and *ope*: $\forall\, U{\in}\mathcal{K}.\ \mathit{openin}\ X\ U$ **and** *top*: $\mathit{topspace}\ X \subseteq \bigcup\mathcal{K}$
and *non*: $\bigwedge\mathcal{F}.\ [\![\mathit{finite}\ \mathcal{F};\ \mathcal{F} \subseteq \mathcal{K};\ \mathit{topspace}\ X \subseteq \bigcup\mathcal{F}]\!] \Longrightarrow \mathit{False}$
unfolding $\mathcal{A}\_\mathit{def}$ **by** *simp_all metis+*
then obtain x **where** $x \in \mathit{topspace}\ X\ x \notin \bigcup(\mathcal{B} \cap \mathcal{K})$
proof −
have $\bigcup(\mathcal{B} \cap \mathcal{K}) \neq \bigcup\mathcal{B}$
by (*metis* ‹$\bigcup\mathcal{B} = \mathit{topspace}\ X$› *fin inf.bounded_iff non order_refl*)
then have $\exists\, a.\ a \notin \bigcup(\mathcal{B} \cap \mathcal{K}) \wedge a \in \bigcup\mathcal{B}$
by *blast*
then show *?thesis*
using *that* **by** (*metis* $U\mathcal{B}$)
qed
obtain C **where** C: $\mathit{openin}\ X\ C\ C \in \mathcal{K}\ x \in C$
using ‹$x \in \mathit{topspace}\ X$› *ope top* **by** *auto*
then have $C \subseteq \mathit{topspace}\ X$
by (*metis openin_subset*)
then have (*arbitrary union_of* (*finite intersection_of* ($\lambda x.\ x \in \mathcal{B}$) *relative_to* $\bigcup\mathcal{B}$)) C
using *openin_subbase* C **unfolding** X [*symmetric*] **by** *blast*
moreover have $C \neq \mathit{topspace}\ X$
using ‹$\mathcal{K} \in \mathcal{A}$› ‹$C \in \mathcal{K}$› **unfolding** $\mathcal{A}\_\mathit{def}$ **by** *blast*
ultimately obtain $\mathcal{V}\ W$ **where** W: (*finite intersection_of* ($\lambda x.\ x \in \mathcal{B}$) *relative_to topspace* X) W
and $x \in W\ W \in \mathcal{V}\ \bigcup\mathcal{V} \neq \mathit{topspace}\ X\ C = \bigcup\mathcal{V}$
using C **by** (*auto simp*: *union_of_def* $U\mathcal{B}$)
then have $\bigcup\mathcal{V} \subseteq \mathit{topspace}\ X$
by (*metis* ‹$C \subseteq \mathit{topspace}\ X$›)
then have $\mathit{topspace}\ X \notin \mathcal{V}$
using ‹$\bigcup\mathcal{V} \neq \mathit{topspace}\ X$› **by** *blast*
then obtain $\mathcal{B}'$ **where** $\mathcal{B}'$: $\mathit{finite}\ \mathcal{B}'\ \mathcal{B}' \subseteq \mathcal{B}\ x \in \bigcap\mathcal{B}'\ W = \mathit{topspace}\ X \cap \bigcap\mathcal{B}'$
using W ‹$x \in W$› **unfolding** *relative_to_def intersection_of_def* **by** *auto*
then have $\bigcap\mathcal{B}' \subseteq \bigcup\mathcal{B}$
using ‹$W \in \mathcal{V}$› ‹$\bigcup\mathcal{V} \neq \mathit{topspace}\ X$› ‹$\bigcup\mathcal{V} \subseteq \mathit{topspace}\ X$› **by** *blast*
then have $\bigcap\mathcal{B}' \subseteq C$
using $U\mathcal{B}$ ‹$C = \bigcup\mathcal{V}$› ‹$W = \mathit{topspace}\ X \cap \bigcap\mathcal{B}'$› ‹$W \in \mathcal{V}$› **by** *auto*
have $\forall\, b \in \mathcal{B}'.\ \exists\, C'.\ \mathit{finite}\ C' \wedge C' \subseteq \mathcal{K} \wedge \mathit{topspace}\ X \subseteq \bigcup(\mathit{insert}\ b\ C')$
proof
fix b
assume $b \in \mathcal{B}'$
have $\mathit{insert}\ b\ \mathcal{K} = \mathcal{K}$ **if** *neg*: $\neg\ (\exists\, C'.\ \mathit{finite}\ C' \wedge C' \subseteq \mathcal{K} \wedge \mathit{topspace}\ X \subseteq \bigcup(\mathit{insert}\ b\ C'))$
proof (*rule* $*$)
show $\mathit{openin}\ X\ W$ **if** $W \in \mathit{insert}\ b\ \mathcal{K}$ **for** W
using *that*
proof
have $b \in \mathcal{B}$

```
          using ‹b ∈ ℬ'› ‹ℬ' ⊆ ℬ› by blast
        then have ∃𝒰. finite 𝒰 ∧ 𝒰 ⊆ ℬ ∧ ⋂𝒰 = b
          by (rule_tac x={b} in exI) auto
        moreover have ⋃ℬ ∩ b = b
          using ℬ'(2) ‹b ∈ ℬ'› by auto
        ultimately show openin X W if W = b
          using  that ‹b ∈ ℬ'›
          apply (simp add: openin_subbase flip: X)
            apply (auto simp: arbitrary_def intersection_of_def relative_to_def
intro!: union_of_inc)
          done
        show openin X W if W ∈ 𝒦
          by (simp add: ‹W ∈ 𝒦› ope)
      qed
    next
      show topspace X ⊆ ⋃ (insert b 𝒦)
        using top by auto
    next
      show False if finite ℱ and ℱ ⊆ insert b 𝒦 topspace X ⊆ ⋃ℱ for ℱ
      proof −
        have insert b (ℱ ∩ 𝒦) = ℱ
          using non that by blast
        then show False
          by (metis Int_lower2 finite_insert neg that(1) that(3))
      qed
    qed auto
    then show ∃ C'. finite C' ∧ C' ⊆ 𝒦 ∧ topspace X ⊆ ⋃(insert b C')
      using ‹b ∈ ℬ'› ‹x ∉ ⋃(ℬ ∩ 𝒦)› ℬ'
      by (metis IntI InterE Union_iff subsetD insertI1)
  qed
  then obtain F where F: ∀ b ∈ ℬ'. finite (F b) ∧ F b ⊆ 𝒦 ∧ topspace X ⊆
⋃(insert b (F b))
    by metis
  let ?𝒟 = insert C (⋃(F ‘ ℬ'))
  show False
  proof (rule non)
    have topspace X ⊆ (⋂ b ∈ ℬ'. ⋃(insert b (F b)))
      using F by (simp add: INT_greatest)
    also have ... ⊆ ⋃ ?𝒟
      using ‹⋂ℬ' ⊆ C› by force
    finally show topspace X ⊆ ⋃ ?𝒟 .
    show ?𝒟 ⊆ 𝒦
      using ‹C ∈ 𝒦› F by auto
    show finite ?𝒟
      using ‹finite ℬ'› F by auto
  qed
 qed
 then show ?thesis
  by (force simp: compact_space_def compactin_def)
```

qed

corollary *Alexander_subbase_alt*:
assumes $U \subseteq \bigcup \mathcal{B}$
and *fin*: $\bigwedge C.\ [\![C \subseteq \mathcal{B};\ U \subseteq \bigcup C]\!] \Longrightarrow \exists\, C'.$ *finite* $C' \wedge C' \subseteq C \wedge U \subseteq \bigcup C'$
and X: *topology*
(*arbitrary union_of*
(*finite intersection_of* $(\lambda x.\ x \in \mathcal{B})$ *relative_to* U)) $= X$
shows *compact_space* X
proof −
have *topspace* $X = U$
using X *topspace_subbase* **by** *fastforce*
have *eq*: $\bigcup$ (*Collect* $((\lambda x.\ x \in \mathcal{B})$ *relative_to* $U)) = U$
unfolding *relative_to_def*
using ‹$U \subseteq \bigcup \mathcal{B}$› **by** *blast*
have $*$: $\exists \mathcal{F}.$ *finite* $\mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{C} \wedge \bigcup \mathcal{F} =$ *topspace* X
if $\mathcal{C} \subseteq$ *Collect* $((\lambda x.\ x \in \mathcal{B})$ *relative_to topspace* $X)$ **and** UC: $\bigcup \mathcal{C} =$ *topspace* X **for** $\mathcal{C}$
proof −
have $\mathcal{C} \subseteq (\lambda U.$ *topspace* $X \cap U)$ ‘ $\mathcal{B}$
using *that* **by** (*auto simp*: *relative_to_def*)
then obtain $\mathcal{B}'$ **where** $\mathcal{B}' \subseteq \mathcal{B}$ **and** $\mathcal{B}'$: $\mathcal{C} = (\cap)$ (*topspace* X) ‘ $\mathcal{B}'$
by (*auto simp*: *subset_image_iff*)
moreover have $U \subseteq \bigcup \mathcal{B}'$
using $\mathcal{B}'$ ‹*topspace* $X = U$› UC **by** *auto*
ultimately obtain $\mathcal{C}'$ **where** *finite* $\mathcal{C}'$ $\mathcal{C}' \subseteq \mathcal{B}'$ $U \subseteq \bigcup \mathcal{C}'$
using *fin* [*of* $\mathcal{B}'$] ‹*topspace* $X = U$› ‹$U \subseteq \bigcup \mathcal{B}$› **by** *blast*
then show *?thesis*
unfolding $\mathcal{B}'$ *ex_finite_subset_image* ‹*topspace* $X = U$› **by** *auto*
qed
show *?thesis*
apply (*rule Alexander_subbase* [**where** $\mathcal{B} =$ *Collect* $((\lambda x.\ x \in \mathcal{B})$ *relative_to* (*topspace* X))])
apply (*simp flip*: X)
apply (*metis finite_intersection_of_relative_to eq*)
apply (*blast intro*: $*$)
done
qed

proposition *continuous_map_componentwise*:
continuous_map X (*product_topology* Y I) $f \longleftrightarrow$
f ‘ (*topspace* X) $\subseteq$ *extensional* $I \wedge (\forall k \in I.$ *continuous_map* X (Y k) $(\lambda x.\ f\ x\ k))$
(**is** *?lhs* $\longleftrightarrow$ _ $\wedge$ *?rhs*)
proof (*cases* $\forall x \in$ *topspace* $X.\ f\ x \in$ *extensional* I)
case *True*
then have f ‘ (*topspace* X) $\subseteq$ *extensional* I
by *force*

```
  moreover have ?rhs if L: ?lhs
  proof −
    have openin X {x ∈ topspace X. f x k ∈ U} if k ∈ I and openin (Y k) U for
k U
    proof −
      have openin (product_topology Y I) ({Y. Y k ∈ U} ∩ (Π_E i∈I. topspace (Y
i)))
      apply (simp add: openin_product_topology flip: arbitrary_union_of_relative_to)
        apply (simp add: relative_to_def)
      using that apply (blast intro: arbitrary_union_of_inc finite_intersection_of_inc)
        done
      with that have openin X {x ∈ topspace X. f x ∈ ({Y. Y k ∈ U} ∩ (Π_E i∈I.
topspace (Y i)))}
        using L unfolding continuous_map_def by blast
      moreover have {x ∈ topspace X. f x ∈ ({Y. Y k ∈ U} ∩ (Π_E i∈I. topspace
(Y i)))} = {x ∈ topspace X. f x k ∈ U}
        using L by (auto simp: continuous_map_def)
      ultimately show ?thesis
        by metis
    qed
    with that
    show ?thesis
      by (auto simp: continuous_map_def)
  qed
  moreover have ?lhs if ?rhs
  proof −
    have 1: ⋀x. x ∈ topspace X ⟹ f x ∈ (Π_E i∈I. topspace (Y i))
      using that True by (auto simp: continuous_map_def PiE_iff)
    have 2: {x ∈ S. ∃ T∈𝒯. f x ∈ T} = (⋃ T∈𝒯. {x ∈ S. f x ∈ T}) for S 𝒯
      by blast
    have 3: {x ∈ S. ∀ U∈𝒰. f x ∈ U} = (⋂ (insert S ((λU. {x ∈ S. f x ∈ U}) ‘
𝒰))) for S 𝒰
      by blast
    show ?thesis
      unfolding continuous_map_def openin_product_topology arbitrary_def
    proof (clarsimp simp: all_union_of 1 2)
      fix 𝒯
      assume 𝒯: 𝒯 ⊆ Collect (finite intersection_of (λF. ∃ i U. F = {f. f i ∈ U}
∧ i ∈ I ∧ openin (Y i) U)
                    relative_to (Π_E i∈I. topspace (Y i)))
      show openin X (⋃ T∈𝒯. {x ∈ topspace X. f x ∈ T})
      proof (rule openin_Union; clarify)
        fix S T
        assume T ∈ 𝒯
        obtain 𝒰 where T = (Π_E i∈I. topspace (Y i)) ∩ ⋂𝒰 and finite 𝒰
          𝒰 ⊆ {{f. f i ∈ U} |i U. i ∈ I ∧ openin (Y i) U}
            using subsetD [OF 𝒯 ‹T ∈ 𝒯›] by (auto simp: intersection_of_def
relative_to_def)
        with that show openin X {x ∈ topspace X. f x ∈ T}
```

```
      apply (simp add: continuous_map_def 1 cong: conj_cong)
      unfolding 3
      apply (rule openin_Inter; auto)
      done
    qed
  qed
 qed
 ultimately show ?thesis
  by metis
qed (auto simp: continuous_map_def PiE_def)


lemma continuous_map_componentwise_UNIV:
  continuous_map X (product_topology Y UNIV) f ⟷ (∀k. continuous_map X
(Y k) (λx. f x k))
 by (simp add: continuous_map_componentwise)

lemma continuous_map_product_projection [continuous_intros]:
  k ∈ I ⟹ continuous_map (product_topology X I) (X k) (λx. x k)
 using continuous_map_componentwise [of product_topology X I X I id] by simp

declare continuous_map_from_subtopology [OF continuous_map_product_projection,
continuous_intros]

proposition open_map_product_projection:
 assumes i ∈ I
 shows open_map (product_topology Y I) (Y i) (λf. f i)
 unfolding openin_product_topology all_union_of arbitrary_def open_map_def
image_Union
proof clarify
 fix 𝒱
 assume 𝒱: 𝒱 ⊆ Collect
            (finite intersection_of
             (λF. ∃i U. F = {f. f i ∈ U} ∧ i ∈ I ∧ openin (Y i) U) relative_to
                    topspace (product_topology Y I))
 show openin (Y i) (⋃x∈𝒱. (λf. f i) ` x)
 proof (rule openin_Union, clarify)
  fix S V
  assume V ∈ 𝒱
  obtain ℱ where finite ℱ
   and V: V = (Π_E i∈I. topspace (Y i)) ∩ ⋂ℱ
   and ℱ: ℱ ⊆ {{f. f i ∈ U} |i U. i ∈ I ∧ openin (Y i) U}
   using subsetD [OF 𝒱 ‹V ∈ 𝒱›]
   by (auto simp: intersection_of_def relative_to_def)
  show openin (Y i) ((λf. f i) ` V)
  proof (subst openin_subopen; clarify)
   fix x f
   assume f ∈ V
   let ?T = {a ∈ topspace(Y i).
```

```
                (λj∈I. f j)(i:=a) ∈ (Π_E i∈I. topspace (Y i)) ∩ ⋂F}
    show ∃ T. openin (Y i) T ∧ f i ∈ T ∧ T ⊆ (λf. f i) ` V
    proof (intro exI conjI)
      show openin (Y i) ?T
      proof (rule openin_continuous_map_preimage)
        have continuous_map (Y i) (Y k) (λx. if k = i then x else f k) if k ∈ I
for k
        proof (cases k=i)
          case True
          then show ?thesis
            by (metis (mono_tags) continuous_map_id eq_id_iff)
        next
          case False
          then show ?thesis
            by simp (metis IntD1 PiE_iff V ‹f ∈ V› that)
        qed
        then show continuous_map (Y i) (product_topology Y I)
              (λx. (λj∈I. f j)(i:=x))
          by (auto simp: continuous_map_componentwise assms extensional_def
restrict_def)
      next
        have openin (product_topology Y I) (Π_E i∈I. topspace (Y i))
          by (metis openin_topspace topspace_product_topology)
      moreover have openin (product_topology Y I) (⋂B∈F. (Π_E i∈I. topspace
(Y i)) ∩ B)
                    if F ≠ {}
        proof −
          show ?thesis
          proof (rule openin_Inter)
                show ⋀X. X ∈ (∩) (Π_E i∈I. topspace (Y i)) ` F ⟹ openin
(product_topology Y I) X
              unfolding openin_product_topology relative_to_def
              apply (clarify intro!: arbitrary_union_of_inc)
              using subsetD [OF F]
            by (metis (mono_tags, lifting) finite_intersection_of_inc mem_Collect_eq
topspace_product_topology)
          qed (use ‹finite F› ‹F ≠ {}› in auto)
        qed
        ultimately show openin (product_topology Y I) ((Π_E i∈I. topspace (Y
i)) ∩ ⋂F)
          by (auto simp only: Int_Inter_eq split: if_split)
      qed
    next
      have eqf: (λj∈I. f j)(i:=f i) = f
        using PiE_arb V ‹f ∈ V› by force
      show f i ∈ ?T
        using V assms ‹f ∈ V› by (auto simp: PiE_iff eqf)
    next
      show ?T ⊆ (λf. f i) ` V
```

```
        unfolding V by (auto simp: intro!: rev_image_eqI)
      qed
    qed
  qed
qed

lemma retraction_map_product_projection:
  assumes i ∈ I
  shows (retraction_map (product_topology X I) (X i) (λx. x i) ⟷
        ((product_topology X I) = trivial_topology) ⟶ (X i) = trivial_topology)
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    using retraction_imp_surjective_map
    by (metis image_empty subtopology_eq_discrete_topology_empty)
next
  assume R: ?rhs
  show ?lhs
  proof (cases (product_topology X I) = trivial_topology)
    case True
    then show ?thesis
      using R by (auto simp: retraction_map_def retraction_maps_def)
  next
    case False
    have ∗: ∃ g. continuous_map (X i) (product_topology X I) g ∧ (∀ x∈topspace
(X i). g x i = x)
      if z: z ∈ (Π_E i∈I. topspace (X i)) for z
    proof −
      have cm: continuous_map (X i) (X j) (λx. if j = i then x else z j) if j ∈ I
for j
        using ‹j ∈ I› z by (case_tac j = i) auto
      show ?thesis
        using ‹i ∈ I› that
        by (rule_tac x=λx j. if j = i then x else z j in exI) (auto simp: continu-
ous_map_componentwise PiE_iff extensional_def cm)
    qed
    with ‹i ∈ I› False assms show ?thesis
    by (auto simp: retraction_map_def retraction_maps_def simp flip: null_topspace_iff_trivial)
  qed
qed
```

2.5.3 Open Pi-sets in the product topology

```
proposition openin_PiE_gen:
  openin (product_topology X I) (PiE I S) ⟷
        PiE I S = {} ∨
        finite {i ∈ I. S i ≠ topspace (X i)} ∧ (∀ i ∈ I. openin (X i) (S i))
  (is ?lhs ⟷ _ ∨ ?rhs)
```

proof (*cases PiE I S = {}*)
case *False*
moreover have *?lhs = ?rhs*
proof
assume *L*: *?lhs*
moreover
obtain *z* **where** *z*: *z ∈ PiE I S*
using *False* **by** *blast*
ultimately obtain *U* **where** *fin*: *finite {i ∈ I. U i ≠ topspace (X i)}*
and *Pi$_E$ I U ≠ {}*
and *sub*: *Pi$_E$ I U ⊆ Pi$_E$ I S*
by (*fastforce simp add*: *openin_product_topology_alt*)
then have ∗: *⋀i. i ∈ I ⟹ U i ⊆ S i*
by (*simp add*: *subset_PiE*)
show *?rhs*
proof (*intro conjI ballI*)
show *finite {i ∈ I. S i ≠ topspace (X i)}*
apply (*rule finite_subset* [*OF _ fin*], *clarify*)
using ∗
by (*metis False L openin_subset topspace_product_topology subset_PiE subset_antisym*)
next
fix *i* :: *'a*
assume *i ∈ I*
then show *openin (X i) (S i)*
using *open_map_product_projection* [*of i I X*] *L*
apply (*simp add*: *open_map_def*)
apply (*drule_tac x=PiE I S* **in** *spec*)
apply (*simp add*: *False image_projection_PiE split*: *if_split_asm*)
done
qed
next
assume *?rhs*
then show *?lhs*
unfolding *openin_product_topology*
by (*intro arbitrary_union_of_inc*) (*auto simp*: *product_topology_base_alt*)
qed
ultimately show *?thesis*
by *simp*
qed *simp*

corollary *openin_PiE*:
finite I ⟹ openin (product_topology X I) (PiE I S) ⟷ PiE I S = {} ∨ (∀ i ∈ I. openin (X i) (S i))
by (*simp add*: *openin_PiE_gen*)

proposition *compact_space_product_topology*:
compact_space(product_topology X I) ⟷

```
    (product_topology X I) = trivial_topology ∨ (∀ i ∈ I. compact_space(X i))
    (is ?lhs = ?rhs)
proof (cases (product_topology X I) = trivial_topology)
  case False
  then obtain z where z: z ∈ (Π_E i∈I. topspace(X i))
    by (auto simp flip: null_topspace_iff_trivial)
  show ?thesis
  proof
    assume L: ?lhs
    show ?rhs
    proof (clarsimp simp add: False compact_space_def)
      fix i
      assume i ∈ I
      with L have continuous_map (product_topology X I) (X i) (λf. f i)
        by (simp add: continuous_map_product_projection)
      moreover
      have ⋀x. x ∈ topspace (X i) ⟹ x ∈ (λf. f i) ` (Π_E i∈I. topspace (X i))
        using ‹i ∈ I› z by (rule_tac x=z(i:=x) in image_eqI) auto
      then have (λf. f i) ` (Π_E i∈I. topspace (X i)) = topspace (X i)
        using ‹i ∈ I› z by auto
      ultimately show compactin (X i) (topspace (X i))
      by (metis L compact_space_def image_compactin topspace_product_topology)
    qed
  next
    assume R: ?rhs
    show ?lhs
    proof (cases I = {})
      case True
      with R show ?thesis
        by (simp add: compact_space_def)
    next
      case False
      then obtain i where i ∈ I
        by blast
      show ?thesis
        using R
      proof
        assume com [rule_format]: ∀ i∈I. compact_space (X i)
        let ?C = {{f. f i ∈ U} |i U. i ∈ I ∧ openin (X i) U}
        show compact_space (product_topology X I)
        proof (rule Alexander_subbase_alt)
          show topspace (product_topology X I) ⊆ ⋃ ?C
            unfolding topspace_product_topology using ‹i ∈ I› by blast
        next
          fix C
          assume Csub: C ⊆ ?C and UC: topspace (product_topology X I) ⊆ ⋃ C
          define D where D ≡ λi. {U. openin (X i) U ∧ {f. f i ∈ U} ∈ C}
          show ∃ C'. finite C' ∧ C' ⊆ C ∧ topspace (product_topology X I) ⊆ ⋃ C'
          proof (cases ∃ i. i ∈ I ∧ topspace (X i) ⊆ ⋃(D i))
```

```
          case True
          then obtain i where i ∈ I
                and i: topspace (X i) ⊆ ⋃(𝒟 i)
            unfolding 𝒟_def by blast
          then have ∗: ⋀𝒰. ⟦Ball 𝒰 (openin (X i)); topspace (X i) ⊆ ⋃𝒰⟧ ⟹
                       ∃ℱ. finite ℱ ∧ ℱ ⊆ 𝒰 ∧ topspace (X i) ⊆ ⋃ℱ
              using com [OF ‹i ∈ I›] by (auto simp: compact_space_def compactin_def)
          have topspace (X i) ⊆ ⋃(𝒟 i)
            using i by auto
          with ∗ obtain ℱ where finite ℱ ∧ ℱ ⊆ (𝒟 i) ∧ topspace (X i) ⊆ ⋃ℱ
            unfolding 𝒟_def by fastforce
          with ‹i ∈ I› show ?thesis
            unfolding 𝒟_def
            by (rule_tac x=(λU. {x. x i ∈ U}) ` ℱ in exI) auto
        next
          case False
          then have ∀ i ∈ I. ∃ y. y ∈ topspace (X i) ∧ y ∉ ⋃(𝒟 i)
            by force
          then obtain g where g: ⋀i. i ∈ I ⟹ g i ∈ topspace (X i) ∧ g i ∉ ⋃(𝒟 i)
            by metis
        then have (λi. if i ∈ I then g i else undefined) ∈ topspace (product_topology X I)
            by (simp add: PiE_I)
          moreover have (λi. if i ∈ I then g i else undefined) ∉ ⋃C
            using Csub g unfolding 𝒟_def by force
          ultimately show ?thesis
            using UC by blast
        qed
      qed (simp add: product_topology)
    qed simp
  qed
 qed
qed auto

corollary compactin_PiE:
  compactin (product_topology X I) (PiE I S) ⟷
      PiE I S = {} ∨ (∀ i ∈ I. compactin (X i) (S i))
 by (fastforce simp add: compactin_subspace subtopology_product_topology compact_space_product_topology
     subset_PiE product_topology_trivial_iff subtopology_trivial_iff)

lemma in_product_topology_closure_of:
  z ∈ (product_topology X I) closure_of S
      ⟹ i ∈ I ⟹ z i ∈ ((X i) closure_of ((λx. x i) ` S))
 using continuous_map_product_projection
 by (force simp: continuous_map_eq_image_closure_subset image_subset_iff)
```

lemma *homeomorphic_space_singleton_product*:
product_topology X {k} homeomorphic_space (X k)
unfolding *homeomorphic_space*
apply (*rule_tac x=λx. x k* **in** *exI*)
apply (*rule bijective_open_imp_homeomorphic_map*)
apply (*simp_all add*: *continuous_map_product_projection open_map_product_projection*)
unfolding *PiE_over_singleton_iff*
apply (*auto simp*: *image_iff inj_on_def*)
done

2.5.4 Relationship with connected spaces, paths, etc.

proposition *connected_space_product_topology*:
connected_space(*product_topology X I*) ⟷
(∃ *i* ∈ *I*. *X i* = *trivial_topology*) ∨ (∀ *i* ∈ *I*. *connected_space*(*X i*))
(**is** *?lhs* ⟷ *?eq* ∨ *?rhs*)
proof (*cases ?eq*)
case *False*
moreover have *?lhs* = *?rhs*
proof
assume *?lhs*
moreover
have *connectedin*(*X i*) (*topspace*(*X i*))
if *i* ∈ *I* **and** *ci*: *connectedin*(*product_topology X I*) (*topspace*(*product_topology X I*)) **for** *i*
proof −
have *cm*: *continuous_map* (*product_topology X I*) (*X i*) (λ*f*. *f i*)
by (*simp add*: ‹*i* ∈ *I*› *continuous_map_product_projection*)
show *?thesis*
using *connectedin_continuous_map_image* [*OF cm ci*] ‹*i* ∈ *I*›
by (*simp add*: *False image_projection_PiE PiE_eq_empty_iff*)
qed
ultimately show *?rhs*
by (*meson connectedin_topspace*)
next
assume *cs* [*rule_format*]: *?rhs*
have *False*
if *disj*: *U* ∩ *V* = {} **and** *subUV*: (Π$_E$ *i*∈*I*. *topspace* (*X i*)) ⊆ *U* ∪ *V*
and *U*: *openin* (*product_topology X I*) *U*
and *V*: *openin* (*product_topology X I*) *V*
and *U* ≠ {} *V* ≠ {}
for *U V*
proof −
obtain *f* **where** *f* ∈ *U*
using ‹*U* ≠ {}› **by** *blast*
then have *f*: *f* ∈ (Π$_E$ *i*∈*I*. *topspace* (*X i*))
using *U openin_subset* **by** *fastforce*
have *U* ⊆ *topspace*(*product_topology X I*) *V* ⊆ *topspace*(*product_topology X I*)

```
      using U V openin_subset by blast+
    moreover have (Π_E i∈I. topspace (X i)) ⊆ U
    proof −
      obtain C where (finite intersection_of (λF. ∃i U. F = {x. x i ∈ U} ∧ i
∈ I ∧ openin (X i) U) relative_to
          (Π_E i∈I. topspace (X i))) C C ⊆ U f ∈ C
        using U ‹f ∈ U› unfolding openin_product_topology union_of_def by
auto
      then obtain 𝒯 where finite 𝒯
       and t: ⋀C. C ∈ 𝒯 ⟹ ∃i u. (i ∈ I ∧ openin (X i) u) ∧ C = {x. x i ∈ u}
        and subU: topspace (product_topology X I) ∩ ⋂𝒯 ⊆ U
        and ftop: f ∈ topspace (product_topology X I)
        and fint: f ∈ ⋂ 𝒯
        by (fastforce simp: relative_to_def intersection_of_def subset_iff)
      let ?L = ⋃ C∈𝒯. {i. (λx. x i) ` C ⊂ topspace (X i)}
      obtain L where finite L
       and L: ⋀i U. ⟦i ∈ I; openin (X i) U; U ⊂ topspace(X i); {x. x i ∈ U} ∈
𝒯⟧ ⟹ i ∈ L
      proof
        show finite ?L
        proof (rule finite_Union)
          fix M
          assume M ∈ (λC. {i. (λx. x i) ` C ⊂ topspace (X i)}) ` 𝒯
         then obtain C where C ∈ 𝒯 and C: M = {i. (λx. x i) ` C ⊂ topspace
(X i)}
            by blast
          then obtain j V where j ∈ I and ope: openin (X j) V and Ceq: C =
{x. x j ∈ V}
            using t by meson
          then have f j ∈ V
            using ‹C ∈ 𝒯› fint by force
          then have (λx. x k) ` {x. x j ∈ V} = UNIV if k ≠ j for k
            using that
            apply (clarsimp simp add: set_eq_iff)
            apply (rule_tac x=f(k:=x) in image_eqI, auto)
            done
          then have {i. (λx. x i) ` C ⊂ topspace (X i)} ⊆ {j}
            using Ceq by auto
          then show finite M
            using C finite_subset by fastforce
        qed (use ‹finite 𝒯› in blast)
      next
        fix i U
        assume  i ∈ I and ope: openin (X i) U and psub: U ⊂ topspace (X i)
and int: {x. x i ∈ U} ∈ 𝒯
        then show i ∈ ?L
          by (rule_tac a={x. x i ∈ U} in UN_I) (force+)
      qed
      show ?thesis
```

```
      proof
        fix h
        assume h: h ∈ (Π_E i∈I. topspace (X i))
        define g where g ≡ λi. if i ∈ L then f i else h i
        have gin: g ∈ (Π_E i∈I. topspace (X i))
          unfolding g_def using f h by auto
        moreover have g ∈ X if X ∈ 𝒯 for X
          using fint openin_subset t [OF that] L g_def h that by fastforce
        ultimately have g ∈ U
          using subU by auto
        have h ∈ U if finite M h ∈ PiE I (topspace ∘ X) {i ∈ I. h i ≠ g i} ⊆ M
for M h
          using that
        proof (induction arbitrary: h)
          case empty
          then show ?case
            using PiE_ext ‹g ∈ U› gin by force
        next
          case (insert i M)
          define f where f ≡ h(i:=g i)
          have fin: f ∈ PiE I (topspace ∘ X)
            unfolding f_def using gin insert.prems(1) by auto
          have subM: {j ∈ I. f j ≠ g j} ⊆ M
            unfolding f_def using insert.prems(2) by auto
          have f ∈ U
            using insert.IH [OF fin subM] .
          show ?case
          proof (cases h ∈ V)
            case True
            show ?thesis
            proof (cases i ∈ I)
              case True
              let ?U = {x ∈ topspace(X i). h(i:=x) ∈ U}
              let ?V = {x ∈ topspace(X i). h(i:=x) ∈ V}
              have False
              proof (rule connected_spaceD [OF cs [OF ‹i ∈ I›]])
               have ⋀k. k ∈ I ⟹ continuous_map (X i) (X k) (λx. if k = i then
x else h k)
                    using continuous_map_eq_topcontinuous_at insert.prems(1)
topcontinuous_at_def by fastforce
               then have cm: continuous_map (X i) (product_topology X I) (λx.
h(i:=x))
                 using ‹i ∈ I› insert.prems(1)
                by (auto simp: continuous_map_componentwise extensional_def)
               show openin (X i) ?U
                 by (rule openin_continuous_map_preimage [OF cm U])
               show openin (X i) ?V
                 by (rule openin_continuous_map_preimage [OF cm V])
               show topspace (X i) ⊆ ?U ∪ ?V
```

```
            proof clarsimp
              fix x
              assume x ∈ topspace (X i) and h(i:=x) ∉ V
              with True subUV ‹h ∈ Pi_E I (topspace ∘ X)›
              show h(i:=x) ∈ U
                by (force dest: subsetD [where c=h(i:=x)])
            qed
            show ?U ∩ ?V = {}
              using disj by blast
            show ?U ≠ {}
              using True ‹f ∈ U› f_def gin by auto
            show ?V ≠ {}
              using True ‹h ∈ V› V openin_subset by fastforce
          qed
          then show ?thesis ..
        next
          case False
          show ?thesis
            using insert.prems(1) by (metis False gin PiE_E ‹f ∈ U› f_def
fun_upd_triv)
        qed
      next
        case False
        then show ?thesis
          using subUV insert.prems(1) by auto
      qed
    qed
    then show h ∈ U
      unfolding g_def using PiE_iff ‹finite L› h by fastforce
  qed
 qed
 ultimately show ?thesis
  using disj inf_absorb2 ‹V ≠ {}› by fastforce
qed
then show ?lhs
 unfolding connected_space_def
 by auto
qed
ultimately show ?thesis
 by simp
qed (metis connected_space_trivial_topology product_topology_trivial_iff)


lemma connectedin_PiE:
  connectedin (product_topology X I) (PiE I S) ⟷
    PiE I S = {} ∨ (∀ i ∈ I. connectedin (X i) (S i))
 by (auto simp: connectedin_def subtopology_product_topology connected_space_product_topology
subset_PiE
    PiE_eq_empty_iff subtopology_trivial_iff)
```

lemma *path_connected_space_product_topology*:
  path_connected_space(*product_topology X I*) ⟷
   topspace(*product_topology X I*) = {} ∨ (∀ *i* ∈ *I*. *path_connected_space*(*X i*))
 (**is** *?lhs* ⟷ *?eq* ∨ *?rhs*)
proof (*cases ?eq*)
 case *False*
 moreover have *?lhs* = *?rhs*
 proof
  assume *L*: *?lhs*
  show *?rhs*
  proof (*clarsimp simp flip*: *path_connectedin_topspace*)
   fix *i* :: *'a*
   assume *i* ∈ *I*
   have *cm*: *continuous_map* (*product_topology X I*) (*X i*) (λ*f*. *f i*)
    by (*simp add*: ‹*i* ∈ *I*› *continuous_map_product_projection*)
   show *path_connectedin* (*X i*) (*topspace* (*X i*))
       using *path_connectedin_continuous_map_image* [*OF cm L* [*unfolded path_connectedin_topspace* [*symmetric*]]]
    by (*metis* ‹*i* ∈ *I*› *False retraction_imp_surjective_map retraction_map_product_projection topspace_discrete_topology*)
  qed
 next
  assume *R* [*rule_format*]: *?rhs*
  show *?lhs*
   unfolding *path_connected_space_def topspace_product_topology*
  proof *clarify*
   fix *x y*
   assume *x*: *x* ∈ (Π$_E$ *i*∈*I*. *topspace* (*X i*)) **and** *y*: *y* ∈ (Π$_E$ *i*∈*I*. *topspace* (*X i*))
   have ∀ *i*. ∃ *g*. *i* ∈ *I* ⟶ *pathin* (*X i*) *g* ∧ *g* 0 = *x i* ∧ *g* 1 = *y i*
    using *PiE_mem R path_connected_space_def x y* **by** *force*
   then obtain *g* **where** *g*: ⋀*i*. *i* ∈ *I* ⟹ *pathin* (*X i*) (*g i*) ∧ *g i* 0 = *x i* ∧ *g i* 1 = *y i*
    by *metis*
   with *x y* **show** ∃ *g*. *pathin* (*product_topology X I*) *g* ∧ *g* 0 = *x* ∧ *g* 1 = *y*
    apply (*rule_tac x*=λ*a*. λ*i* ∈ *I*. *g i a* **in** *exI*)
    apply (*force simp*: *pathin_def continuous_map_componentwise*)
    done
  qed
 qed
 ultimately show *?thesis*
  by *simp*
next
qed (*force simp*: *path_connected_space_topspace_empty iff*: *null_topspace_iff_trivial*)

lemma *path_connectedin_PiE*:
  path_connectedin (*product_topology X I*) (*PiE I S*) ⟷
   PiE I S = {} ∨ (∀ *i* ∈ *I*. *path_connectedin* (*X i*) (*S i*))

by (*fastforce simp add*: *path_connectedin_def subtopology_product_topology path_connected_space_product_topol*
subset_PiE PiE_eq_empty_iff topspace_subtopology_subset)

2.5.5 Projections from a function topology to a component

lemma *quotient_map_product_projection*:
assumes $i \in I$
shows *quotient_map*(*product_topology X I*) (*X i*) ($\lambda x.\ x\ i$) $\longleftrightarrow$
((*product_topology X I*) = *trivial_topology* $\longrightarrow$ (*X i*) = *trivial_topology*)
by (*metis* (*no_types*) *assms image_is_empty null_topspace_iff_trivial quotient_imp_surjective_map*

retraction_imp_quotient_map retraction_map_product_projection)

lemma *product_topology_homeomorphic_component*:
assumes $i \in I$ $\bigwedge j.\ [\![j \in I; j \neq i]\!] \Longrightarrow \exists a.$ *topspace*(*X j*) = {*a*}
shows *product_topology X I homeomorphic_space* (*X i*)
proof −
have *quotient_map* (*product_topology X I*) (*X i*) ($\lambda x.\ x\ i$)
using *assms* **by** (*metis* (*full_types*) *discrete_topology_unique empty_not_insert*

product_topology_trivial_iff quotient_map_product_projection)
moreover
have *inj_on* ($\lambda x.\ x\ i$) ($\Pi_E\ i{\in}I.$ *topspace* (*X i*))
using *assms* **by** (*auto simp*: *inj_on_def PiE_iff*) (*metis extensionalityI singletonD*)
ultimately show *?thesis*
unfolding *homeomorphic_space_def*
by (*rule_tac x*=$\lambda x.\ x\ i$ **in** *exI*) (*simp add*: *homeomorphic_map_def flip*: *homeomorphic_map_maps*)
qed

lemma *topological_property_of_product_component*:
assumes *major*: *P* (*product_topology X I*)
and *minor*: $\bigwedge z\ i.\ [\![z \in (\Pi_E\ i{\in}I.\ topspace(X\ i)); P(product\_topology\ X\ I); i \in I]\!]$
$\Longrightarrow P(subtopology\ (product\_topology\ X\ I)\ (PiE\ I\ (\lambda j.\ if\ j = i\ then\ topspace(X\ i)\ else\ \{z\ j\})))$
(**is** $\bigwedge z\ i.\ [\![\_; \_; \_]\!] \Longrightarrow P\ (?SX\ z\ i)$)
and *PQ*: $\bigwedge X\ X'.\ X$ *homeomorphic_space* $X' \Longrightarrow (P\ X \longleftrightarrow Q\ X')$
shows ($\exists i{\in}I.$ (*X i*) = *trivial_topology*) $\vee$ ($\forall i \in I.\ Q(X\ i)$)
proof −
have *Q*(*X i*) **if** $\forall i{\in}I.$ (*X i*) $\neq$ *trivial_topology* $i \in I$ **for** *i*
proof −
from *that* **obtain** *f* **where** *f*: $f \in (\Pi_E\ i{\in}I.$ *topspace* (*X i*))
by (*meson null_topspace_iff_trivial PiE_eq_empty_iff ex_in_conv*)
have *?SX f i homeomorphic_space X i*
using *f product_topology_homeomorphic_component* [*OF* ‹$i \in I$›, *of* $\lambda j.$ *subtopology* (*X j*) (*if j* = *i then topspace* (*X i*) *else* {*f j*})]
by (*force simp add*: *subtopology_product_topology*)

```
    then show ?thesis
      using minor [OF f major ‹i ∈ I›] PQ by auto
  qed
  then show ?thesis by metis
qed
```

2.5.6 Limits

The original HOL Light proof was a mess, yuk

```
lemma limitin_componentwise:
   limitin (product_topology X I) f l F ⟷
        l ∈ extensional I ∧
        eventually (λa. f a ∈ topspace(product_topology X I)) F ∧
        (∀ i ∈ I. limitin (X i) (λc. f c i) (l i) F)
    (is ?L ⟷ _ ∧ ?R1 ∧ ?R2)
proof (cases l ∈ extensional I)
  case l: True
  show ?thesis
  proof (cases ∀ i∈I. l i ∈ topspace (X i))
    case True
    have ?R1 if ?L
      by (metis limitin_subtopology subtopology_topspace that)
    moreover
    have ?R2 if ?L
      unfolding limitin_def
    proof (intro conjI strip)
      fix i U
      assume i ∈ I and U: openin (X i) U ∧ l i ∈ U
    then have openin (product_topology X I) ({y. y i ∈ U} ∩ topspace (product_topology
X I))
      unfolding openin_product_topology arbitrary_union_of_relative_to [symmetric]
        apply (simp add: relative_to_def topspace_product_topology_alt)
      by (smt (verit, del_insts) Collect_cong arbitrary_union_of_inc finite_intersection_of_inc
inf_commute)
      moreover have l ∈ {y. y i ∈ U} ∩ topspace (product_topology X I)
        using U True l by (auto simp: extensional_def)
    ultimately have eventually (λx. f x ∈ {y. y i ∈ U} ∩ topspace (product_topology
X I)) F
        by (metis limitin_def that)
      then show ∀ F x in F. f x i ∈ U
        by (simp add: eventually_conj_iff)
    qed (use True in auto)
    moreover
    have ?L if R1: ?R1 and R2: ?R2
      unfolding limitin_def openin_product_topology all_union_of imp_conjL
arbitrary_def
    proof (intro conjI strip)
      show l: l ∈ topspace (product_topology X I)
        by (simp add: PiE_iff True l)
```

fix $\mathcal{V}$
assume $\mathcal{V} \subseteq$ *Collect* (*finite intersection_of* ($\lambda F.\ \exists i\ U.\ F = \{f.\ f\ i \in U\} \wedge i \in I \wedge$ *openin* ($X\ i$) U)
relative_to topspace (*product_topology* $X\ I$))
and $l \in \bigcup\ \mathcal{V}$
then obtain $\mathcal{W}$ **where** *finite* $\mathcal{W}$ **and** $\mathcal{W}X$: $\forall X{\in}\mathcal{W}.\ l \in X$
and $\mathcal{W}$: $\bigwedge C.\ C \in \mathcal{W} \Longrightarrow C \in \{\{x.\ x\ i \in U\}\ |i\ U.\ i \in I \wedge$ *openin* (X i) $U\}$
and $\mathcal{W}\mathcal{V}$: *topspace* (*product_topology* $X\ I$) $\cap \bigcap\ \mathcal{W} \in \mathcal{V}$
by (*fastforce simp*: *intersection_of_def relative_to_def subset_eq*)
have $\forall_F\ x$ *in* $F.\ f\ x \in$ *topspace* (*product_topology* $X\ I$) $\cap \bigcap \mathcal{W}$
proof −
have $\bigwedge W.\ W \in \{\{x.\ x\ i \in U\} \mid i\ U.\ i \in I \wedge$ *openin* ($X\ i$) $U\} \Longrightarrow W \in \mathcal{W} \Longrightarrow \forall_F\ x$ *in* $F.\ f\ x \in W$
using $\mathcal{W}X$ *R2* **by** (*auto simp*: *limitin_def*)
with $\mathcal{W}$ **have** $\forall_F\ x$ *in* $F.\ \forall\ W{\in}\mathcal{W}.\ f\ x \in W$
by (*simp add*: ‹*finite* $\mathcal{W}$› *eventually_ball_finite*)
with *R1* **show** *?thesis*
by (*simp add*: *eventually_conj_iff*)
qed
then show $\forall_F\ x$ *in* $F.\ f\ x \in \bigcup \mathcal{V}$
by (*smt* (*verit*, *ccfv_threshold*) $\mathcal{W}\mathcal{V}$ *UnionI eventually_mono*)
qed
ultimately show *?thesis*
using l **by** *blast*
next
case *False*
then show *?thesis*
by (*metis PiE_iff limitin_def topspace_product_topology*)
qed
next
case *False*
then show *?thesis*
by (*simp add*: *limitin_def PiE_iff*)
qed

end

2.6 The binary product topology

theory *Product_Topology*
imports *Function_Topology*
begin

2.7 Product Topology

2.7.1 Definition

definition *prod_topology* :: *'a topology* ⇒ *'b topology* ⇒ (*'a* × *'b*) *topology* **where**

prod_topology X Y ≡ topology (arbitrary union_of (λU. U ∈ {S × T |S T. openin X S ∧ openin Y T}))

lemma *open_product_open*:
assumes *open A*
shows ∃𝒰. 𝒰 ⊆ {S × T |S T. open S ∧ open T} ∧ ⋃ 𝒰 = A
proof −
obtain *f g* **where** ∗: ⋀u. u ∈ A ⟹ *open* (*f u*) ∧ *open* (*g u*) ∧ u ∈ (*f u*) × (*g u*) ∧ (*f u*) × (*g u*) ⊆ A
using *open_prod_def* [*of A*] *assms* **by** *metis*
let *?𝒰* = (λu. *f u* × *g u*) ‘ A
show *?thesis*
by (*rule_tac x*=*?𝒰* **in** *exI*) (*auto simp*: *dest*: ∗)
qed

lemma *open_product_open_eq*: (*arbitrary union_of* (λU. ∃ S T. U = S × T ∧ *open S* ∧ *open T*)) = *open*
by (*force simp*: *union_of_def arbitrary_def intro*: *open_product_open open_Times*)

lemma *openin_prod_topology*:
openin (*prod_topology X Y*) = *arbitrary union_of* (λU. U ∈ {S × T |S T. *openin X S* ∧ *openin Y T*})
unfolding *prod_topology_def*
proof (*rule topology_inverse′*)
show *istopology* (*arbitrary union_of* (λU. U ∈ {S × T |S T. *openin X S* ∧ *openin Y T*}))
apply (*rule istopology_base*, *simp*)
by (*metis openin_Int Times_Int_Times*)
qed

lemma *topspace_prod_topology* [*simp*]:
topspace (*prod_topology X Y*) = *topspace X* × *topspace Y*
proof −
have *topspace*(*prod_topology X Y*) = ⋃ (*Collect* (*openin* (*prod_topology X Y*))) (**is** _ = *?Z*)
unfolding *topspace_def* **..**
also have ... = *topspace X* × *topspace Y*
proof
show *?Z* ⊆ *topspace X* × *topspace Y*
apply (*auto simp*: *openin_prod_topology union_of_def arbitrary_def*)
using *openin_subset* **by** *force*+
next
have ∗: ∃ *A B*. *topspace X* × *topspace Y* = *A* × *B* ∧ *openin X A* ∧ *openin Y B*
by *blast*
show *topspace X* × *topspace Y* ⊆ *?Z*
apply (*rule Union_upper*)
using ∗ **by** (*simp add*: *openin_prod_topology arbitrary_union_of_inc*)
qed

finally show *?thesis* **.**
qed

lemma *prod_topology_trivial_iff* [*simp*]:
prod_topology X Y = trivial_topology ⟷ *X = trivial_topology* ∨ *Y = trivial_topology*
by (*metis* (*full_types*) *Sigma_empty1 null_topspace_iff_trivial subset_empty times_subset_iff topspace_prod_topology*)

lemma *subtopology_Times*:
shows *subtopology* (*prod_topology X Y*) (*S* × *T*) = *prod_topology* (*subtopology X S*) (*subtopology Y T*)
proof −
have ((λ*U*. ∃*S T*. *U* = *S* × *T* ∧ *openin X S* ∧ *openin Y T*) *relative_to S* × *T*) =
(λ*U*. ∃*S*′ *T*′. *U* = *S*′ × *T*′ ∧ (*openin X relative_to S*) *S*′ ∧ (*openin Y relative_to T*) *T*′)
by (*auto simp*: *relative_to_def Times_Int_Times fun_eq_iff*) *metis*
then show *?thesis*
by (*simp add*: *topology_eq openin_prod_topology arbitrary_union_of_relative_to flip*: *openin_relative_to*)
qed

lemma *prod_topology_subtopology*:
prod_topology (*subtopology X S*) *Y* = *subtopology* (*prod_topology X Y*) (*S* × *topspace Y*)
prod_topology X (*subtopology Y T*) = *subtopology* (*prod_topology X Y*) (*topspace X* × *T*)
by (*auto simp*: *subtopology_Times*)

lemma *prod_topology_discrete_topology*:
discrete_topology (*S* × *T*) = *prod_topology* (*discrete_topology S*) (*discrete_topology T*)
by (*auto simp*: *discrete_topology_unique openin_prod_topology intro*: *arbitrary_union_of_inc*)

lemma *prod_topology_euclidean* [*simp*]: *prod_topology euclidean euclidean* = *euclidean*
by (*simp add*: *prod_topology_def open_product_open_eq*)

lemma *prod_topology_subtopology_eu* [*simp*]:
prod_topology (*subtopology euclidean S*) (*subtopology euclidean T*) = *subtopology euclidean* (*S* × *T*)
by (*simp add*: *prod_topology_subtopology subtopology_subtopology Times_Int_Times*)

lemma *openin_prod_topology_alt*:
openin (*prod_topology X Y*) *S* ⟷
(∀*x y*. (*x*,*y*) ∈ *S* ⟶ (∃*U V*. *openin X U* ∧ *openin Y V* ∧ *x* ∈ *U* ∧ *y* ∈ *V* ∧ *U* × *V* ⊆ *S*))
apply (*auto simp*: *openin_prod_topology arbitrary_union_of_alt*, *fastforce*)

by (*metis mem_Sigma_iff*)

lemma *open_map_fst*: *open_map* (*prod_topology X Y*) *X fst*
unfolding *open_map_def openin_prod_topology_alt*
by (*force simp*: *openin_subopen* [*of X fst* ‘ *_*] *intro*: *subset_fst_imageI*)

lemma *open_map_snd*: *open_map* (*prod_topology X Y*) *Y snd*
unfolding *open_map_def openin_prod_topology_alt*
by (*force simp*: *openin_subopen* [*of Y snd* ‘ *_*] *intro*: *subset_snd_imageI*)

lemma *openin_prod_Times_iff*:
openin (*prod_topology X Y*) (*S* × *T*) ⟷ *S* = {} ∨ *T* = {} ∨ *openin X S* ∧ *openin Y T*
proof (*cases S* = {} ∨ *T* = {})
case *False*
then show *?thesis*
apply (*simp add*: *openin_prod_topology_alt openin_subopen* [*of X S*] *openin_subopen* [*of Y T*] *times_subset_iff*, *safe*)
apply (*meson*|*force*)+
done
qed *force*

lemma *closure_of_Times*:
(*prod_topology X Y*) *closure_of* (*S* × *T*) = (*X closure_of S*) × (*Y closure_of T*) (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs*
by (*clarsimp simp*: *closure_of_def openin_prod_topology_alt*) *blast*
show *?rhs* ⊆ *?lhs*
by (*clarsimp simp*: *closure_of_def openin_prod_topology_alt*) (*meson SigmaI subsetD*)
qed

lemma *closedin_prod_Times_iff*:
closedin (*prod_topology X Y*) (*S* × *T*) ⟷ *S* = {} ∨ *T* = {} ∨ *closedin X S* ∧ *closedin Y T*
by (*auto simp*: *closure_of_Times times_eq_iff simp flip*: *closure_of_eq*)

lemma *interior_of_Times*: (*prod_topology X Y*) *interior_of* (*S* × *T*) = (*X interior_of S*) × (*Y interior_of T*)
proof (*rule interior_of_unique*)
show (*X interior_of S*) × *Y interior_of T* ⊆ *S* × *T*
by (*simp add*: *Sigma_mono interior_of_subset*)
show *openin* (*prod_topology X Y*) ((*X interior_of S*) × *Y interior_of T*)
by (*simp add*: *openin_prod_Times_iff*)
next
show *T′* ⊆ (*X interior_of S*) × *Y interior_of T* **if** *T′* ⊆ *S* × *T openin* (*prod_topology X Y*) *T′* **for** *T′*

```
  proof (clarsimp; intro conjI)
    fix a :: 'a and b :: 'b
    assume (a, b) ∈ T'
    with that obtain U V where UV: openin X U openin Y V a ∈ U b ∈ V U ×
V ⊆ T'
      by (metis openin_prod_topology_alt)
    then show a ∈ X interior_of S
      using interior_of_maximal_eq that(1) by fastforce
    show b ∈ Y interior_of T
      using UV interior_of_maximal_eq that(1)
      by (metis SigmaI mem_Sigma_iff subset_eq)
  qed
qed
```

Missing the opposite direction. Does it hold? A converse is proved for proper maps, a stronger condition

```
lemma closed_map_prod:
  assumes closed_map (prod_topology X Y) (prod_topology X' Y') (λ(x,y). (f x,
g y))
  shows (prod_topology X Y) = trivial_topology ∨ closed_map X X' f ∧ closed_map
Y Y' g
proof (cases (prod_topology X Y) = trivial_topology)
  case False
  then have ne: topspace X ≠ {} topspace Y ≠ {}
    by (auto simp flip: null_topspace_iff_trivial)
  have closed_map X X' f
    unfolding closed_map_def
  proof (intro strip)
    fix C
    assume closedin X C
    show closedin X' (f ‘ C)
    proof (cases C={})
      case False
      with assms have closedin (prod_topology X' Y') ((λ(x,y). (f x, g y)) ‘ (C ×
topspace Y))
        by (simp add: ‹closedin X C› closed_map_def closedin_prod_Times_iff)
      with False ne show ?thesis
      by (simp add: image_paired_Times closedin_Times closedin_prod_Times_iff)
    qed auto
  qed
  moreover
  have closed_map Y Y' g
    unfolding closed_map_def
  proof (intro strip)
    fix C
    assume closedin Y C
    show closedin Y' (g ‘ C)
    proof (cases C={})
      case False
```

```
      with assms have closedin (prod_topology X' Y') ((λ(x,y). (f x, g y)) ‘
(topspace X × C))
       by (simp add: ‹closedin Y C› closed_map_def closedin_prod_Times_iff)
     with False ne show ?thesis
     by (simp add: image_paired_Times closedin_Times closedin_prod_Times_iff)
   qed auto
  qed
  ultimately show ?thesis
   by (auto simp: False)
qed auto
```

2.7.2 Continuity

```
lemma continuous_map_pairwise:
  continuous_map Z (prod_topology X Y) f ⟷ continuous_map Z X (fst ∘ f)
∧ continuous_map Z Y (snd ∘ f)
  (is ?lhs = ?rhs)
proof −
  let ?g = fst ∘ f and ?h = snd ∘ f
  have f: f x = (?g x, ?h x) for x
   by auto
  show ?thesis
  proof (cases ?g ∈ topspace Z → topspace X ∧ ?h ∈ topspace Z → topspace Y)
   case True
   show ?thesis
   proof safe
    assume continuous_map Z (prod_topology X Y) f
    then have openin Z {x ∈ topspace Z. fst (f x) ∈ U} if openin X U for U
     unfolding continuous_map_def using True that
     apply clarify
     apply (drule_tac x=U × topspace Y in spec)
       by (auto simp: openin_prod_Times_iff mem_Times_iff Pi_iff cong:
conj_cong)
    with True show continuous_map Z X (fst ∘ f)
     by (auto simp: continuous_map_def)
   next
    assume continuous_map Z (prod_topology X Y) f
    then have openin Z {x ∈ topspace Z. snd (f x) ∈ V} if openin Y V for V
     unfolding continuous_map_def using True that
     apply clarify
     apply (drule_tac x=topspace X × V in spec)
    by (simp add: openin_prod_Times_iff mem_Times_iff Pi_iff cong: conj_cong)
    with True show continuous_map Z Y (snd ∘ f)
     by (auto simp: continuous_map_def)
   next
    assume Z: continuous_map Z X (fst ∘ f) continuous_map Z Y (snd ∘ f)
    have ∗: openin Z {x ∈ topspace Z. f x ∈ W}
     if ⋀w. w ∈ W ⟹ ∃ U V. openin X U ∧ openin Y V ∧ w ∈ U × V ∧ U
× V ⊆ W for W
```

proof (*subst openin_subopen*, *clarify*)
fix $x :: 'a$
assume $x \in$ *topspace* Z **and** $f\ x \in W$
with *that* [*OF* ‹$f\ x \in W$›]
obtain $U\ V$ **where** UV: *openin* $X\ U$ *openin* $Y\ V\ f\ x \in U \times V\ U \times V \subseteq W$
by *auto*
with $Z\ \ UV$ **show** $\exists T.$ *openin* $Z\ T \wedge x \in T \wedge T \subseteq \{x \in$ *topspace* $Z.\ f\ x \in W\}$
apply (*rule_tac* x=$\{x \in$ *topspace* $Z.\ ?g\ x \in U\} \cap \{x \in$ *topspace* $Z.\ ?h\ x \in V\}$ **in** *exI*)
apply (*auto simp*: ‹$x \in$ *topspace* Z› *continuous_map_def*)
done
qed
show *continuous_map* Z (*prod_topology* $X\ Y$) f
using *True* **by** (*force simp*: *continuous_map_def openin_prod_topology_alt mem_Times_iff* $*$)
qed
qed (*force simp*: *continuous_map_def*)
qed

lemma *continuous_map_paired*:
continuous_map Z (*prod_topology* $X\ Y$) ($\lambda x.$ ($f\ x$,$g\ x$)) $\longleftrightarrow$ *continuous_map* $Z\ X\ f \wedge$ *continuous_map* $Z\ Y\ g$
by (*simp add*: *continuous_map_pairwise o_def*)

lemma *continuous_map_pairedI* [*continuous_intros*]:
⟦*continuous_map* $Z\ X\ f$; *continuous_map* $Z\ Y\ g$⟧ $\Longrightarrow$ *continuous_map* Z (*prod_topology* $X\ Y$) ($\lambda x.$ ($f\ x$,$g\ x$))
by (*simp add*: *continuous_map_pairwise o_def*)

lemma *continuous_map_fst* [*continuous_intros*]: *continuous_map* (*prod_topology* $X\ Y$) X *fst*
using *continuous_map_pairwise* [*of prod_topology* $X\ Y\ X\ Y$ *id*]
by (*simp add*: *continuous_map_pairwise*)

lemma *continuous_map_snd* [*continuous_intros*]: *continuous_map* (*prod_topology* $X\ Y$) Y *snd*
using *continuous_map_pairwise* [*of prod_topology* $X\ Y\ X\ Y$ *id*]
by (*simp add*: *continuous_map_pairwise*)

lemma *continuous_map_fst_of* [*continuous_intros*]:
continuous_map Z (*prod_topology* $X\ Y$) $f \Longrightarrow$ *continuous_map* $Z\ X$ (*fst* $\circ\ f$)
by (*simp add*: *continuous_map_pairwise*)

lemma *continuous_map_snd_of* [*continuous_intros*]:
continuous_map Z (*prod_topology* $X\ Y$) $f \Longrightarrow$ *continuous_map* $Z\ Y$ (*snd* $\circ\ f$)
by (*simp add*: *continuous_map_pairwise*)

lemma *continuous_map_prod_fst*:

$i \in I \Longrightarrow$ *continuous_map* (*prod_topology* (*product_topology* ($\lambda i.\ Y$) I) X) Y ($\lambda x.$ *fst* x i)
using *continuous_map_componentwise_UNIV continuous_map_fst* **by** *fastforce*

lemma *continuous_map_prod_snd*:
$i \in I \Longrightarrow$ *continuous_map* (*prod_topology* X (*product_topology* ($\lambda i.\ Y$) I)) Y ($\lambda x.$ *snd* x i)
using *continuous_map_componentwise_UNIV continuous_map_snd* **by** *fastforce*

lemma *continuous_map_if_iff* [*simp*]: *continuous_map* X Y ($\lambda x.$ *if* P *then* f x *else* g x) $\longleftrightarrow$ *continuous_map* X Y (*if* P *then* f *else* g)
by *simp*

lemma *continuous_map_if* [*continuous_intros*]: $[\![P \Longrightarrow$ *continuous_map* X Y f; $\sim P \Longrightarrow$ *continuous_map* X Y $g]\!]$
$\Longrightarrow$ *continuous_map* X Y ($\lambda x.$ *if* P *then* f x *else* g x)
by *simp*

lemma *prod_topology_trivial1* [*simp*]: *prod_topology trivial_topology* Y = *trivial_topology*
using *continuous_map_fst continuous_map_on_empty2* **by** *blast*

lemma *prod_topology_trivial2* [*simp*]: *prod_topology* X *trivial_topology* = *trivial_topology*
using *continuous_map_snd continuous_map_on_empty2* **by** *blast*

lemma *continuous_map_subtopology_fst* [*continuous_intros*]: *continuous_map* (*subtopology* (*prod_topology* X Y) Z) X *fst*
using *continuous_map_from_subtopology continuous_map_fst* **by** *force*

lemma *continuous_map_subtopology_snd* [*continuous_intros*]: *continuous_map* (*subtopology* (*prod_topology* X Y) Z) Y *snd*
using *continuous_map_from_subtopology continuous_map_snd* **by** *force*

lemma *quotient_map_fst* [*simp*]:
quotient_map(*prod_topology* X Y) X *fst* $\longleftrightarrow$ (Y = *trivial_topology* $\longrightarrow$ X = *trivial_topology*)
apply (*simp add*: *continuous_open_quotient_map open_map_fst continuous_map_fst*)
by (*metis null_topspace_iff_trivial*)

lemma *quotient_map_snd* [*simp*]:
quotient_map(*prod_topology* X Y) Y *snd* $\longleftrightarrow$ (X = *trivial_topology* $\longrightarrow$ Y = *trivial_topology*)
apply (*simp add*: *continuous_open_quotient_map open_map_snd continuous_map_snd*)
by (*metis null_topspace_iff_trivial*)

lemma *retraction_map_fst*:
retraction_map (*prod_topology* X Y) X *fst* $\longleftrightarrow$ (Y = *trivial_topology* $\longrightarrow$ X

= *trivial_topology*)
proof (*cases* *Y* = *trivial_topology*)
case *True*
then show *?thesis*
using *continuous_map_image_subset_topspace*
by (*auto simp*: *retraction_map_def retraction_maps_def continuous_map_pairwise*)
next
case *False*
have ∃ *g*. *continuous_map* *X* (*prod_topology* *X* *Y*) *g* ∧ (∀ *x*∈*topspace* *X*. *fst* (*g* *x*) = *x*)
if *y*: *y* ∈ *topspace* *Y* **for** *y*
by (*rule_tac* *x*=λ*x*. (*x*,*y*) **in** *exI*) (*auto simp*: *y continuous_map_paired*)
with *False* **have** *retraction_map* (*prod_topology* *X* *Y*) *X fst*
by (*fastforce simp*: *retraction_map_def retraction_maps_def continuous_map_fst*)
with *False* **show** *?thesis*
by *simp*
qed

lemma *retraction_map_snd*:
retraction_map (*prod_topology* *X* *Y*) *Y snd* ⟷ (*X* = *trivial_topology* ⟶ *Y* = *trivial_topology*)
proof (*cases* *X* = *trivial_topology*)
case *True*
then show *?thesis*
using *continuous_map_image_subset_topspace*
by (*fastforce simp*: *retraction_map_def retraction_maps_def continuous_map_fst*)
next
case *False*
have ∃ *g*. *continuous_map* *Y* (*prod_topology* *X* *Y*) *g* ∧ (∀ *y*∈*topspace* *Y*. *snd* (*g* *y*) = *y*)
if *x*: *x* ∈ *topspace* *X* **for** *x*
by (*rule_tac* *x*=λ*y*. (*x*,*y*) **in** *exI*) (*auto simp*: *x continuous_map_paired*)
with *False* **have** *retraction_map* (*prod_topology* *X* *Y*) *Y snd*
by (*fastforce simp*: *retraction_map_def retraction_maps_def continuous_map_snd*)
with *False* **show** *?thesis*
by *simp*
qed

lemma *continuous_map_of_fst*:
continuous_map (*prod_topology* *X* *Y*) *Z* (*f* ∘ *fst*) ⟷ *Y* = *trivial_topology* ∨ *continuous_map* *X* *Z* *f*
proof (*cases* *Y* = *trivial_topology*)
case *True*
then show *?thesis*
by (*simp add*: *continuous_map_on_empty*)
next
case *False*
then show *?thesis*

by (*simp add*: *continuous_compose_quotient_map_eq*)
qed

lemma *continuous_map_of_snd*:
continuous_map (*prod_topology X Y*) *Z* (*f* ∘ *snd*) ⟷ *X* = *trivial_topology* ∨ *continuous_map Y Z f*
proof (*cases X* = *trivial_topology*)
case *True*
then show *?thesis*
by (*simp add*: *continuous_map_on_empty*)
next
case *False*
then show *?thesis*
by (*simp add*: *continuous_compose_quotient_map_eq*)
qed

lemma *continuous_map_prod_top*:
continuous_map (*prod_topology X Y*) (*prod_topology X′ Y′*) (λ(*x*,*y*). (*f x*, *g y*)) ⟷
(*prod_topology X Y*) = *trivial_topology* ∨ *continuous_map X X′ f* ∧ *continuous_map Y Y′ g*
proof (*cases* (*prod_topology X Y*) = *trivial_topology*)
case *False*
then show *?thesis*
by (*auto simp*: *continuous_map_paired case_prod_unfold*
continuous_map_of_fst [*unfolded o_def*] *continuous_map_of_snd* [*unfolded o_def*])
qed *auto*

lemma *in_prod_topology_closure_of*:
assumes *z* ∈ (*prod_topology X Y*) *closure_of S*
shows *fst z* ∈ *X closure_of* (*fst* ‘ *S*) *snd z* ∈ *Y closure_of* (*snd* ‘ *S*)
using *assms continuous_map_eq_image_closure_subset continuous_map_fst*
apply *fastforce*
using *assms continuous_map_eq_image_closure_subset continuous_map_snd*
apply *fastforce*
done

proposition *compact_space_prod_topology*:
compact_space(*prod_topology X Y*) ⟷ (*prod_topology X Y*) = *trivial_topology* ∨ *compact_space X* ∧ *compact_space Y*
proof (*cases* (*prod_topology X Y*) = *trivial_topology*)
case *True*
then show *?thesis*
by *fastforce*
next
case *False*
then have *non_mt*: *topspace X* ≠ {} *topspace Y* ≠ {}

by *auto*
have *compact_space X compact_space Y* **if** *compact_space*(*prod_topology X Y*)
proof −
have *compactin X* (*fst* ‘ (*topspace X* × *topspace Y*))
by (*metis compact_space_def continuous_map_fst image_compactin that topspace_prod_topology*)
moreover
have *compactin Y* (*snd* ‘ (*topspace X* × *topspace Y*))
by (*metis compact_space_def continuous_map_snd image_compactin that topspace_prod_topology*)
ultimately show *compact_space X compact_space Y*
using *non_mt* **by** (*auto simp*: *compact_space_def*)
qed
moreover
define $\mathcal{X}$ **where** $\mathcal{X} \equiv (\lambda V.\ \mathit{topspace}\ X \times V)$ ‘ *Collect* (*openin Y*)
define $\mathcal{Y}$ **where** $\mathcal{Y} \equiv (\lambda U.\ U \times \mathit{topspace}\ Y)$ ‘ *Collect* (*openin X*)
have *compact_space*(*prod_topology X Y*) **if** *compact_space X compact_space Y*
proof (*rule Alexander_subbase_alt*)
show *toptop*: *topspace X* × *topspace Y* ⊆ $\bigcup(\mathcal{X} \cup \mathcal{Y})$
unfolding $\mathcal{X}$*_def* $\mathcal{Y}$*_def* **by** *auto*
fix $\mathcal{C}$ :: (′*a* × ′*b*) *set set*
assume $\mathcal{C}$: $\mathcal{C} \subseteq \mathcal{X} \cup \mathcal{Y}$ *topspace X* × *topspace Y* ⊆ $\bigcup\mathcal{C}$
then obtain $\mathcal{X}'$ $\mathcal{Y}'$ **where** *XY*: $\mathcal{X}' \subseteq \mathcal{X}$ $\mathcal{Y}' \subseteq \mathcal{Y}$ **and** $\mathcal{C}$*eq*: $\mathcal{C} = \mathcal{X}' \cup \mathcal{Y}'$
using *subset_UnE* **by** *metis*
then have *sub*: *topspace X* × *topspace Y* ⊆ $\bigcup(\mathcal{X}' \cup \mathcal{Y}')$
using $\mathcal{C}$ **by** *simp*
obtain $\mathcal{U}$ $\mathcal{V}$ **where** $\mathcal{U}$: $\bigwedge U.\ U \in \mathcal{U} \Longrightarrow \mathit{openin}\ X\ U\ \mathcal{Y}' = (\lambda U.\ U \times \mathit{topspace}\ Y)$ ‘ $\mathcal{U}$
and $\mathcal{V}$: $\bigwedge V.\ V \in \mathcal{V} \Longrightarrow \mathit{openin}\ Y\ V\ \mathcal{X}' = (\lambda V.\ \mathit{topspace}\ X \times V)$ ‘ $\mathcal{V}$
using *XY* **by** (*clarsimp simp add*: $\mathcal{X}$*_def* $\mathcal{Y}$*_def subset_image_iff*) (*force simp*: *subset_iff*)
have ∃$\mathcal{D}$. *finite* $\mathcal{D}$ ∧ $\mathcal{D} \subseteq \mathcal{X}' \cup \mathcal{Y}'$ ∧ *topspace X* × *topspace Y* ⊆ $\bigcup$ $\mathcal{D}$
proof −
have *topspace X* ⊆ $\bigcup\mathcal{U}$ ∨ *topspace Y* ⊆ $\bigcup\mathcal{V}$
using $\mathcal{U}$ $\mathcal{V}$ $\mathcal{C}$ $\mathcal{C}$*eq* **by** *auto*
then have ∗: ∃$\mathcal{D}$. *finite* $\mathcal{D}$ ∧
(∀ *x* ∈ $\mathcal{D}$. *x* ∈ ($\lambda V.$ *topspace X* × *V*) ‘ $\mathcal{V}$ ∨ *x* ∈ ($\lambda U.$ *U* × *topspace Y*) ‘ $\mathcal{U}$) ∧
(*topspace X* × *topspace Y* ⊆ $\bigcup\mathcal{D}$)
proof
assume *topspace X* ⊆ $\bigcup\mathcal{U}$
with ‹*compact_space X*› $\mathcal{U}$ **obtain** $\mathcal{F}$ **where** *finite* $\mathcal{F}$ $\mathcal{F} \subseteq \mathcal{U}$ *topspace X* ⊆ $\bigcup\mathcal{F}$
by (*meson compact_space_alt*)
with *that* **show** *?thesis*
by (*rule_tac x*=($\lambda D.$ *D* × *topspace Y*) ‘ $\mathcal{F}$ **in** *exI*) *auto*
next
assume *topspace Y* ⊆ $\bigcup\mathcal{V}$
with ‹*compact_space Y*› $\mathcal{V}$ **obtain** $\mathcal{F}$ **where** *finite* $\mathcal{F}$ $\mathcal{F} \subseteq \mathcal{V}$ *topspace Y*

```
⊆ ⋃ℱ
        by (meson compact_space_alt)
      with that show ?thesis
        by (rule_tac x=(λC. topspace X × C) ` ℱ in exI) auto
    qed
    then show ?thesis
      using that 𝒰 𝒱 by blast
  qed
  then show ∃𝒟. finite 𝒟 ∧ 𝒟 ⊆ 𝒞 ∧ topspace X × topspace Y ⊆ ⋃ 𝒟
    using 𝒞 𝒞eq by blast
next
  have (finite intersection_of (λx. x ∈ 𝒳 ∨ x ∈ 𝒴) relative_to topspace X ×
topspace Y)
        = (λU. ∃S T. U = S × T ∧ openin X S ∧ openin Y T)
    (is ?lhs = ?rhs)
  proof −
    have ?rhs U if ?lhs U for U
    proof −
     have topspace X × topspace Y ∩ ⋂ T ∈ {A × B |A B. A ∈ Collect (openin
X) ∧ B ∈ Collect (openin Y)}
       if finite T T ⊆ 𝒳 ∪ 𝒴 for T
       using that
      proof induction
       case (insert B ℬ)
       then show ?case
         unfolding 𝒳_def 𝒴_def
         apply (simp add: Int_ac subset_eq image_def)
        apply (metis (no_types) openin_Int openin_topspace Times_Int_Times)
         done
      qed auto
      then show ?thesis
       using that
       by (auto simp: subset_eq  elim!: relative_toE intersection_ofE)
    qed
    moreover
    have ?lhs Z if Z: ?rhs Z for Z
    proof −
      obtain U V where Z = U × V openin X U openin Y V
       using Z by blast
      then have UV: U × V = (topspace X × topspace Y) ∩ (U × V)
       by (simp add: Sigma_mono inf_absorb2 openin_subset)
      moreover
      have ?lhs ((topspace X × topspace Y) ∩ (U × V))
      proof (rule relative_to_inc)
       show (finite intersection_of (λx. x ∈ 𝒳 ∨ x ∈ 𝒴)) (U × V)
         apply (simp add: intersection_of_def 𝒳_def 𝒴_def)
         apply (rule_tac x={(U × topspace Y),(topspace X × V)} in exI)
          using ‹openin X U› ‹openin Y V› openin_subset UV apply (fastforce
simp:)
```

```
          done
        qed
        ultimately show ?thesis
          using ‹Z = U × V› by auto
      qed
      ultimately show ?thesis
        by meson
    qed
    then show topology (arbitrary union_of (finite intersection_of (λx. x ∈ 𝒳 ∪
𝒴)
          relative_to (topspace X × topspace Y))) =
        prod_topology X Y
      by (simp add: prod_topology_def)
  qed
  ultimately show ?thesis
    using False by blast
qed

lemma compactin_Times:
  compactin (prod_topology X Y) (S × T) ⟷ S = {} ∨ T = {} ∨ compactin X
S ∧ compactin Y T
 by (auto simp: compactin_subspace subtopology_Times compact_space_prod_topology
subtopology_trivial_iff)
```

2.7.3 Homeomorphic maps

```
lemma homeomorphic_maps_prod:
  homeomorphic_maps (prod_topology X Y) (prod_topology X' Y') (λ(x,y). (f x,
g y)) (λ(x,y). (f' x, g' y)) ⟷
        (prod_topology X Y) = trivial_topology ∧ (prod_topology X' Y') = triv-
ial_topology
     ∨ homeomorphic_maps X X' f f' ∧ homeomorphic_maps Y Y' g g'  (is ?lhs
= ?rhs)
proof
  show ?lhs ⟹ ?rhs
   by (fastforce simp: homeomorphic_maps_def continuous_map_prod_top ball_conj_distrib)
next
  show ?rhs ⟹ ?lhs
  by (auto simp: homeomorphic_maps_def continuous_map_prod_top)
qed


lemma homeomorphic_maps_swap:
  homeomorphic_maps (prod_topology X Y) (prod_topology Y X) (λ(x,y). (y,x))
(λ(y,x). (x,y))
 by (auto simp: homeomorphic_maps_def case_prod_unfold continuous_map_fst
continuous_map_pairedI continuous_map_snd)

lemma homeomorphic_map_swap:
```

homeomorphic_map (*prod_topology X Y*) (*prod_topology Y X*) (λ(*x*,*y*). (*y*,*x*))
using *homeomorphic_map_maps homeomorphic_maps_swap* **by** *metis*

lemma *homeomorphic_space_prod_topology_swap*:
(*prod_topology X Y*) *homeomorphic_space* (*prod_topology Y X*)
using *homeomorphic_map_swap homeomorphic_space* **by** *blast*

lemma *embedding_map_graph*:
embedding_map X (*prod_topology X Y*) (λ*x*. (*x*, *f x*)) ⟷ *continuous_map X Y f*
(**is** *?lhs* = *?rhs*)
proof
assume *L*: *?lhs*
have *snd* ∘ (λ*x*. (*x*, *f x*)) = *f*
by *force*
moreover have *continuous_map X Y* (*snd* ∘ (λ*x*. (*x*, *f x*)))
using *L* **unfolding** *embedding_map_def*
by (*meson continuous_map_in_subtopology continuous_map_snd_of homeomorphic_imp_continuous_map*)
ultimately show *?rhs*
by *simp*
next
assume *R*: *?rhs*
then show *?lhs*
unfolding *homeomorphic_map_maps embedding_map_def homeomorphic_maps_def*
by (*rule_tac x*=*fst* **in** *exI*)
(*auto simp*: *continuous_map_in_subtopology continuous_map_paired continuous_map_from_subtopology*
continuous_map_fst)
qed

lemma *homeomorphic_space_prod_topology*:
⟦*X homeomorphic_space X′*; *Y homeomorphic_space Y′*⟧
⟹ *prod_topology X Y homeomorphic_space prod_topology X′ Y′*
using *homeomorphic_maps_prod* **unfolding** *homeomorphic_space_def* **by** *blast*

lemma *prod_topology_homeomorphic_space_left*:
Y = *discrete_topology* {*b*} ⟹ *prod_topology X Y homeomorphic_space X*
unfolding *homeomorphic_space_def*
apply (*rule_tac x*=*fst* **in** *exI*)
apply (*simp add*: *homeomorphic_map_def inj_on_def discrete_topology_unique flip*: *homeomorphic_map_maps*)
done

lemma *prod_topology_homeomorphic_space_right*:
X = *discrete_topology* {*a*} ⟹ *prod_topology X Y homeomorphic_space Y*
unfolding *homeomorphic_space_def*
by (*meson homeomorphic_space_def homeomorphic_space_prod_topology_swap homeomorphic_space_trans prod_topology_homeomorphic_space_left*)

lemma *homeomorphic_space_prod_topology_sing1*:
 $b \in$ *topspace* $Y \Longrightarrow X$ *homeomorphic_space* (*prod_topology* X (*subtopology* Y $\{b\}$))
 by (*metis empty_subsetI homeomorphic_space_sym insert_subset prod_topology_homeomorphic_space_left subtopology_eq_discrete_topology_sing topspace_subtopology_subset*)

lemma *homeomorphic_space_prod_topology_sing2*:
 $a \in$ *topspace* $X \Longrightarrow Y$ *homeomorphic_space* (*prod_topology* (*subtopology* X $\{a\}$) Y)
 by (*metis empty_subsetI homeomorphic_space_sym insert_subset prod_topology_homeomorphic_space_right subtopology_eq_discrete_topology_sing topspace_subtopology_subset*)

lemma *topological_property_of_prod_component*:
 assumes *major*: P(*prod_topology* X Y)
 and X: $\bigwedge x$. $[\![x \in$ *topspace* X; P(*prod_topology* X Y)$]\!] \Longrightarrow P$(*subtopology* (*prod_topology* X Y) ($\{x\} \times$ *topspace* Y))
 and Y: $\bigwedge y$. $[\![y \in$ *topspace* Y; P(*prod_topology* X Y)$]\!] \Longrightarrow P$(*subtopology* (*prod_topology* X Y) (*topspace* $X \times \{y\}$))
 and PQ: $\bigwedge X\ X'$. X *homeomorphic_space* $X' \Longrightarrow (P\ X \longleftrightarrow Q\ X')$
 and PR: $\bigwedge X\ X'$. X *homeomorphic_space* $X' \Longrightarrow (P\ X \longleftrightarrow R\ X')$
 shows (*prod_topology* X Y) = *trivial_topology* $\vee$ $Q\ X \wedge R\ Y$
proof −
 have $Q\ X \wedge R\ Y$ **if** *topspace*(*prod_topology* X Y) $\neq \{\}$
 proof −
 from *that* **obtain** a b **where** a: $a \in$ *topspace* X **and** b: $b \in$ *topspace* Y
 by *force*
 show *?thesis*
 using X [*OF a major*] **and** Y [*OF b major*] *homeomorphic_space_prod_topology_sing1* [*OF b, of X*] *homeomorphic_space_prod_topology_sing2* [*OF a, of Y*]
 by (*simp add*: *subtopology_Times*) (*meson PQ PR homeomorphic_space_prod_topology_sing2 homeomorphic_space_sym*)
 qed
 then show *?thesis* **by** *force*
qed

lemma *limitin_pairwise*:
 limitin (*prod_topology* X Y) f l $F \longleftrightarrow$ *limitin* X (*fst* $\circ$ f) (*fst* l) $F \wedge$ *limitin* Y (*snd* $\circ$ f) (*snd* l) F
 (**is** *?lhs* = *?rhs*)
proof
 assume *?lhs*
 then obtain a b **where** *ev*: $\bigwedge U$. $[\![(a,b) \in U$; *openin* (*prod_topology* X Y) $U]\!] \Longrightarrow \forall_F\ x$ *in* F. $f\ x \in U$
 and a: $a \in$ *topspace* X **and** b: $b \in$ *topspace* Y **and** l: $l = (a,b)$
 by (*auto simp*: *limitin_def*)
 moreover have $\forall_F\ x$ *in* F. *fst* $(f\ x) \in U$ **if** *openin* X U $a \in U$ **for** U
 proof −

```
    have ∀F c in F. f c ∈ U × topspace Y
     using b that ev [of U × topspace Y] by (auto simp: openin_prod_topology_alt)
    then show ?thesis
     by (rule eventually_mono) (metis (mono_tags, lifting) SigmaE2 prod.collapse)
  qed
  moreover have ∀F x in F. snd (f x) ∈ U if openin Y U b ∈ U for U
  proof −
    have ∀F c in F. f c ∈ topspace X × U
     using a that ev [of topspace X × U] by (auto simp: openin_prod_topology_alt)
    then show ?thesis
     by (rule eventually_mono) (metis (mono_tags, lifting) SigmaE2 prod.collapse)
  qed
  ultimately show ?rhs
    by (simp add: limitin_def)
next
  have limitin (prod_topology X Y) f (a,b) F
    if limitin X (fst ∘ f) a F limitin Y (snd ∘ f) b F for a b
    using that
  proof (clarsimp simp: limitin_def)
    fix Z :: ('a × 'b) set
    assume a: a ∈ topspace X ∀U. openin X U ∧ a ∈ U ⟶ (∀F x in F. fst (f
x) ∈ U)
      and b: b ∈ topspace Y ∀U. openin Y U ∧ b ∈ U ⟶ (∀F x in F. snd (f x)
∈ U)
      and Z: openin (prod_topology X Y) Z (a, b) ∈ Z
    then obtain U V where openin X U openin Y V a ∈ U b ∈ V U × V ⊆ Z
      using Z by (force simp: openin_prod_topology_alt)
    then have ∀F x in F. fst (f x) ∈ U ∀F x in F. snd (f x) ∈ V
      by (simp_all add: a b)
    then show ∀F x in F. f x ∈ Z
      by (rule eventually_elim2) (use ‹U × V ⊆ Z› subsetD in auto)
  qed
  then show ?rhs ⟹ ?lhs
    by (metis prod.collapse)
qed

proposition connected_space_prod_topology:
   connected_space(prod_topology X Y) ⟷
   (prod_topology X Y) = trivial_topology ∨ connected_space X ∧ connected_space
Y (is ?lhs=?rhs)
proof (cases (prod_topology X Y) = trivial_topology)
  case True
  then show ?thesis
    by auto
next
  case False
  then have nonempty: topspace X ≠ {} topspace Y ≠ {}
    by force+
  show ?thesis
```

```
  proof
    assume ?lhs
    then show ?rhs
     by (metis connected_space_quotient_map_image nonempty quotient_map_fst
quotient_map_snd
         subtopology_eq_discrete_topology_empty)
  next
    assume ?rhs
    then have conX: connected_space X and conY: connected_space Y
      using False by blast+
    have False
      if openin (prod_topology X Y) U and openin (prod_topology X Y) V
        and UV: topspace X × topspace Y ⊆ U ∪ V U ∩ V = {}
        and U ≠ {} and V ≠ {}
      for U V
    proof −
      have Usub: U ⊆ topspace X × topspace Y and Vsub: V ⊆ topspace X ×
topspace Y
        using that by (metis openin_subset topspace_prod_topology)+
      obtain a b where ab: (a,b) ∈ U and a: a ∈ topspace X and b: b ∈ topspace
Y
        using ‹U ≠ {}› Usub by auto
      have ¬ topspace X × topspace Y ⊆ U
        using Usub Vsub ‹U ∩ V = {}› ‹V ≠ {}› by auto
      then obtain x y where x: x ∈ topspace X and y: y ∈ topspace Y and (x,y)
∉ U
        by blast
      have oX: openin X {x ∈ topspace X. (x,y) ∈ U} openin X {x ∈ topspace X.
(x,y) ∈ V}
       and oY: openin Y {y ∈ topspace Y. (a,y) ∈ U} openin Y {y ∈ topspace Y.
(a,y) ∈ V}
      by (force intro: openin_continuous_map_preimage [where Y = prod_topology
X Y]
           simp: that continuous_map_pairwise o_def x y a)+
      have 1: topspace Y ⊆ {y ∈ topspace Y. (a,y) ∈ U} ∪ {y ∈ topspace Y. (a,y)
∈ V}
        using a that(3) by auto
      have 2: {y ∈ topspace Y. (a,y) ∈ U} ∩ {y ∈ topspace Y. (a,y) ∈ V} = {}
        using that(4) by auto
      have 3: {y ∈ topspace Y. (a,y) ∈ U} ≠ {}
        using ab b by auto
      have 4: {y ∈ topspace Y. (a,y) ∈ V} ≠ {}
      proof −
        show ?thesis
          using connected_spaceD [OF conX oX] UV ‹(x,y) ∉ U› a x y
                disjoint_iff_not_equal by blast
      qed
      show ?thesis
        using connected_spaceD [OF conY oY 1 2 3 4] by auto
```

qed
then show *?lhs*
unfolding *connected_space_def topspace_prod_topology* **by** *blast*
qed
qed

lemma *connectedin_Times*:
connectedin (*prod_topology X Y*) (*S* × *T*) ⟷
S = {} ∨ *T* = {} ∨ *connectedin X S* ∧ *connectedin Y T*
by (*auto simp*: *connectedin_def subtopology_Times connected_space_prod_topology subtopology_trivial_iff*)

end

2.8 T1 and Hausdorff spaces

theory *T1_Spaces*
imports *Product_Topology*
begin

2.9 T1 spaces with equivalences to many naturally "nice" properties.

definition *t1_space* **where**
t1_space X ≡ ∀ *x* ∈ *topspace X*. ∀ *y* ∈ *topspace X*. *x*≠*y* ⟶ (∃ *U*. *openin X U* ∧ *x* ∈ *U* ∧ *y* ∉ *U*)

lemma *t1_space_expansive*:
⟦*topspace Y* = *topspace X*; ⋀*U*. *openin X U* ⟹ *openin Y U*⟧ ⟹ *t1_space X* ⟹ *t1_space Y*
by (*metis t1_space_def*)

lemma *t1_space_alt*:
t1_space X ⟷ (∀ *x* ∈ *topspace X*. ∀ *y* ∈ *topspace X*. *x*≠*y* ⟶ (∃ *U*. *closedin X U* ∧ *x* ∈ *U* ∧ *y* ∉ *U*))
by (*metis DiffE DiffI closedin_def openin_closedin_eq t1_space_def*)

lemma *t1_space_empty* [*iff*]: *t1_space trivial_topology*
by (*simp add*: *t1_space_def*)

lemma *t1_space_derived_set_of_singleton*:
t1_space X ⟷ (∀ *x* ∈ *topspace X*. *X derived_set_of* {*x*} = {})
apply (*simp add*: *t1_space_def derived_set_of_def*, *safe*)
apply (*metis openin_topspace*)
by *force*

lemma *t1_space_derived_set_of_finite*:
t1_space X ⟷ (∀ *S*. *finite S* ⟶ *X derived_set_of S* = {})

proof (*intro iffI allI impI*)
fix $S :: 'a\ set$
assume *finite S*
then have *fin*: *finite* ((λ*x*. {*x*}) ‘ (*topspace X* ∩ *S*))
by *blast*
assume *t1_space X*
then have *X derived_set_of* (⋃ *x* ∈ *topspace X* ∩ *S*. {*x*}) = {}
unfolding *derived_set_of_Union* [*OF fin*]
by (*auto simp*: *t1_space_derived_set_of_singleton*)
then have *X derived_set_of* (*topspace X* ∩ *S*) = {}
by *simp*
then show *X derived_set_of S* = {}
by *simp*
qed (*auto simp*: *t1_space_derived_set_of_singleton*)

lemma *t1_space_closedin_singleton*:
t1_space X ⟷ (∀ *x* ∈ *topspace X*. *closedin X* {*x*})
apply (*rule iffI*)
apply (*simp add*: *closedin_contains_derived_set t1_space_derived_set_of_singleton*)
using *t1_space_alt* **by** *auto*

lemma *continuous_closed_imp_proper_map*:
⟦*compact_space X*; *t1_space Y*; *continuous_map X Y f*; *closed_map X Y f*⟧
⟹ *proper_map X Y f*
unfolding *proper_map_def*
by (*smt* (*verit*) *closedin_compact_space closedin_continuous_map_preimage*
Collect_cong singleton_iff t1_space_closedin_singleton)

lemma *t1_space_euclidean*: *t1_space* (*euclidean* :: *'a*::*metric_space topology*)
by (*simp add*: *t1_space_closedin_singleton*)

lemma *closedin_t1_singleton*:
⟦*t1_space X*; *a* ∈ *topspace X*⟧ ⟹ *closedin X* {*a*}
by (*simp add*: *t1_space_closedin_singleton*)

lemma *t1_space_closedin_finite*:
t1_space X ⟷ (∀ *S*. *finite S* ∧ *S* ⊆ *topspace X* ⟶ *closedin X S*)
apply (*rule iffI*)
apply (*simp add*: *closedin_contains_derived_set t1_space_derived_set_of_finite*)
by (*simp add*: *t1_space_closedin_singleton*)

lemma *closure_of_singleton*:
t1_space X ⟹ *X closure_of* {*a*} = (*if a* ∈ *topspace X then* {*a*} *else* {})
by (*simp add*: *closure_of_eq t1_space_closedin_singleton closure_of_eq_empty_gen*)

lemma *separated_in_singleton*:
assumes *t1_space X*
shows *separatedin X* {*a*} *S* ⟷ *a* ∈ *topspace X* ∧ *S* ⊆ *topspace X* ∧ (*a* ∉ *X*
closure_of S)

separatedin X S {a} ⟷ *a ∈ topspace X ∧ S ⊆ topspace X ∧ (a ∉ X closure_of S)*
unfolding *separatedin_def*
using *assms closure_of closure_of_singleton* **by** *fastforce+*

lemma *t1_space_openin_delete*:
t1_space X ⟷ *(∀ U x. openin X U ∧ x ∈ U ⟶ openin X (U − {x}))*
apply (*rule iffI*)
apply (*meson closedin_t1_singleton in_mono openin_diff openin_subset*)
by (*simp add*: *closedin_def t1_space_closedin_singleton*)

lemma *t1_space_openin_delete_alt*:
t1_space X ⟷ *(∀ U x. openin X U ⟶ openin X (U − {x}))*
by (*metis Diff_empty Diff_insert0 t1_space_openin_delete*)

lemma *t1_space_singleton_Inter_open*:
t1_space X ⟷ *(∀ x ∈ topspace X. ⋂{U. openin X U ∧ x ∈ U} = {x})* (**is** *?P=?Q*)
and *t1_space_Inter_open_supersets*:
t1_space X ⟷ *(∀ S. S ⊆ topspace X ⟶ ⋂{U. openin X U ∧ S ⊆ U} = S)* (**is** *?P=?R*)
proof −
have *?R* ⟹ *?Q*
apply *clarify*
apply (*drule_tac x={x}* **in** *spec*, *simp*)
done
moreover have *?Q* ⟹ *?P*
apply (*clarsimp simp add*: *t1_space_def*)
apply (*drule_tac x=x* **in** *bspec*)
apply (*simp_all add*: *set_eq_iff*)
by (*metis* (*no_types*, *lifting*))
moreover have *?P* ⟹ *?R*
proof (*clarsimp simp add*: *t1_space_closedin_singleton*, *rule subset_antisym*)
fix *S*
assume *S*: *∀ x∈topspace X. closedin X {x} S ⊆ topspace X*
then show *⋂ {U. openin X U ∧ S ⊆ U} ⊆ S*
apply *clarsimp*
by (*metis Diff_insert_absorb Set.set_insert closedin_def openin_topspace subset_insert*)
qed *force*
ultimately show *?P=?Q ?P=?R*
by *auto*
qed

lemma *t1_space_derived_set_of_infinite_openin*:
t1_space X ⟷
(∀ S. X derived_set_of S =
{x ∈ topspace X. ∀ U. x ∈ U ∧ openin X U ⟶ infinite(S ∩ U)})

(**is** _ = *?rhs*)
proof
assume *t1_space X*
show *?rhs*
proof *safe*
fix *S x U*
assume *x* ∈ *X derived_set_of S x* ∈ *U openin X U finite* (*S* ∩ *U*)
with ‹*t1_space X*› **show** *False*
apply (*simp add*: *t1_space_derived_set_of_finite*)
by (*metis IntI empty_iff empty_subsetI inf_commute openin_Int_derived_set_of_subset subset_antisym*)
next
fix *S x*
have *eq*: (∃ *y*. (*y* ≠ *x*) ∧ *y* ∈ *S* ∧ *y* ∈ *T*) ⟷ ~((*S* ∩ *T*) ⊆ {*x*}) **for** *x S T*
by *blast*
assume *x* ∈ *topspace X* ∀ *U*. *x* ∈ *U* ∧ *openin X U* ⟶ *infinite* (*S* ∩ *U*)
then show *x* ∈ *X derived_set_of S*
apply (*clarsimp simp add*: *derived_set_of_def eq*)
by (*meson finite.emptyI finite.insertI finite_subset*)
qed (*auto simp*: *in_derived_set_of*)
qed (*auto simp*: *t1_space_derived_set_of_singleton*)

lemma *finite_t1_space_imp_discrete_topology*:
⟦*topspace X* = *U*; *finite U*; *t1_space X*⟧ ⟹ *X* = *discrete_topology U*
by (*metis discrete_topology_unique_derived_set t1_space_derived_set_of_finite*)

lemma *t1_space_subtopology*: *t1_space X* ⟹ *t1_space*(*subtopology X U*)
by (*simp add*: *derived_set_of_subtopology t1_space_derived_set_of_finite*)

lemma *closedin_derived_set_of_gen*:
t1_space X ⟹ *closedin X* (*X derived_set_of S*)
apply (*clarsimp simp add*: *in_derived_set_of closedin_contains_derived_set derived_set_of_subset_topspace*)
by (*metis DiffD2 insert_Diff insert_iff t1_space_openin_delete*)

lemma *derived_set_of_derived_set_subset_gen*:
t1_space X ⟹ *X derived_set_of* (*X derived_set_of S*) ⊆ *X derived_set_of S*
by (*meson closedin_contains_derived_set closedin_derived_set_of_gen*)

lemma *subtopology_eq_discrete_topology_gen_finite*:
⟦*t1_space X*; *finite S*⟧ ⟹ *subtopology X S* = *discrete_topology*(*topspace X* ∩ *S*)
by (*simp add*: *subtopology_eq_discrete_topology_gen t1_space_derived_set_of_finite*)

lemma *subtopology_eq_discrete_topology_finite*:
⟦*t1_space X*; *S* ⊆ *topspace X*; *finite S*⟧
⟹ *subtopology X S* = *discrete_topology S*
by (*simp add*: *subtopology_eq_discrete_topology_eq t1_space_derived_set_of_finite*)

lemma *t1_space_closed_map_image*:
  ⟦*closed_map X Y f*; *f ' (topspace X) = topspace Y*; *t1_space X*⟧ ⟹ *t1_space Y*
  by (*metis closed_map_def finite_subset_image t1_space_closedin_finite*)

lemma *homeomorphic_t1_space*: *X homeomorphic_space Y* ⟹ (*t1_space X* ⟷ *t1_space Y*)
  apply (*clarsimp simp add*: *homeomorphic_space_def*)
  by (*meson homeomorphic_eq_everything_map homeomorphic_maps_map t1_space_closed_map_ima*

proposition *t1_space_product_topology*:
  t1_space (*product_topology X I*)
⟷ (*product_topology X I*) = *trivial_topology* ∨ (∀ *i* ∈ *I*. *t1_space* (*X i*))
proof (*cases* (*product_topology X I*) = *trivial_topology*)
  case *True*
  then show *?thesis*
    using *True t1_space_empty* **by** *force*
next
  case *False*
  then obtain *f* **where** *f*: *f* ∈ (Π$_E$ *i*∈*I*. *topspace*(*X i*))
    using *discrete_topology_unique* **by** (*fastforce iff*: *null_topspace_iff_trivial*)
  have *t1_space* (*product_topology X I*) ⟷ (∀ *i*∈*I*. *t1_space* (*X i*))
  proof (*intro iffI ballI*)
    show *t1_space* (*X i*) **if** *t1_space* (*product_topology X I*) **and** *i* ∈ *I* **for** *i*
    proof −
      have *clo*: ⋀*h*. *h* ∈ (Π$_E$ *i*∈*I*. *topspace* (*X i*)) ⟹ *closedin* (*product_topology X I*) {*h*}
        using *that* **by** (*simp add*: *t1_space_closedin_singleton*)
      show *?thesis*
        unfolding *t1_space_closedin_singleton*
      proof *clarify*
        show *closedin* (*X i*) {*xi*} **if** *xi* ∈ *topspace* (*X i*) **for** *xi*
          using *clo* [*of* λ*j* ∈ *I*. *if i=j then xi else f j*] *f that* ‹*i* ∈ *I*›
          by (*fastforce simp add*: *closedin_product_topology_singleton*)
      qed
    qed
  next
  next
    show *t1_space* (*product_topology X I*) **if** ∀ *i*∈*I*. *t1_space* (*X i*)
      using *that*
    by (*simp add*: *t1_space_closedin_singleton Ball_def PiE_iff closedin_product_topology_singleton*)
  qed
  then show *?thesis*
    using *False* **by** *blast*
qed

lemma *t1_space_prod_topology*:
  t1_space(*prod_topology X Y*) ⟷ (*prod_topology X Y*) = *trivial_topology* ∨ *t1_space X* ∧ *t1_space Y*

proof (*cases* (*prod_topology X Y*) = *trivial_topology*)
case *True* **then show** *?thesis*
by *auto*
next
case *False*
have *eq*: {(*x*,*y*)} = {*x*} × {*y*} **for** *x*::'*a* **and** *y*::'*b*
by *simp*
have *t1_space* (*prod_topology X Y*) ⟷ (*t1_space X* ∧ *t1_space Y*)
using *False*
apply(*simp add*: *t1_space_closedin_singleton closedin_prod_Times_iff eq*
del: *insert_Times_insert flip*: *null_topspace_iff_trivial ex_in_conv*)
by *blast*
with *False* **show** *?thesis*
by *simp*
qed

2.9.1 Hausdorff Spaces

definition *Hausdorff_space*
where
Hausdorff_space X ≡
∀ *x y*. *x* ∈ *topspace X* ∧ *y* ∈ *topspace X* ∧ (*x* ≠ *y*)
⟶ (∃ *U V*. *openin X U* ∧ *openin X V* ∧ *x* ∈ *U* ∧ *y* ∈ *V* ∧ *disjnt U V*)

lemma *Hausdorff_space_expansive*:
⟦*Hausdorff_space X*; *topspace X* = *topspace Y*; ⋀*U*. *openin X U* ⟹ *openin Y U*⟧ ⟹ *Hausdorff_space Y*
by (*metis Hausdorff_space_def*)

lemma *Hausdorff_space_topspace_empty* [*iff*]: *Hausdorff_space trivial_topology*
by (*simp add*: *Hausdorff_space_def*)

lemma *Hausdorff_imp_t1_space*:
Hausdorff_space X ⟹ *t1_space X*
by (*metis Hausdorff_space_def disjnt_iff t1_space_def*)

lemma *closedin_derived_set_of*:
Hausdorff_space X ⟹ *closedin X* (*X derived_set_of S*)
by (*simp add*: *Hausdorff_imp_t1_space closedin_derived_set_of_gen*)

lemma *t1_or_Hausdorff_space*:
t1_space X ∨ *Hausdorff_space X* ⟷ *t1_space X*
using *Hausdorff_imp_t1_space* **by** *blast*

lemma *Hausdorff_space_sing_Inter_opens*:
⟦*Hausdorff_space X*; *a* ∈ *topspace X*⟧ ⟹ ⋂{*u*. *openin X u* ∧ *a* ∈ *u*} = {*a*}
using *Hausdorff_imp_t1_space t1_space_singleton_Inter_open* **by** *force*

lemma *Hausdorff_space_subtopology*:
assumes *Hausdorff_space X* **shows** *Hausdorff_space*(*subtopology X S*)
proof −
have ∗: *disjnt U V* ⟹ *disjnt* (*S* ∩ *U*) (*S* ∩ *V*) **for** *U V*
by (*simp add*: *disjnt_iff*)
from *assms* **show** *?thesis*
apply (*simp add*: *Hausdorff_space_def openin_subtopology_alt*)
apply (*fast intro*: ∗ *elim*!: *all_forward*)
done
qed

lemma *Hausdorff_space_compact_separation*:
assumes *X*: *Hausdorff_space X* **and** *S*: *compactin X S* **and** *T*: *compactin X T* **and** *disjnt S T*
obtains *U V* **where** *openin X U openin X V S* ⊆ *U T* ⊆ *V disjnt U V*
proof (*cases S* = {})
case *True*
then show *thesis*
by (*metis* ‹*compactin X T*› *compactin_subset_topspace disjnt_empty1 empty_subsetI openin_empty openin_topspace that*)
next
case *False*
have ∀ *x* ∈ *S*. ∃ *U V*. *openin X U* ∧ *openin X V* ∧ *x* ∈ *U* ∧ *T* ⊆ *V* ∧ *disjnt U V*
proof
fix *a*
assume *a* ∈ *S*
then have *a* ∉ *T*
by (*meson assms*(*4*) *disjnt_iff*)
have *a*: *a* ∈ *topspace X*
using *S* ‹*a* ∈ *S*› *compactin_subset_topspace* **by** *blast*
show ∃ *U V*. *openin X U* ∧ *openin X V* ∧ *a* ∈ *U* ∧ *T* ⊆ *V* ∧ *disjnt U V*
proof (*cases T* = {})
case *True*
then show *?thesis*
using *a disjnt_empty2 openin_empty* **by** *blast*
next
case *False*
have ∀ *x* ∈ *topspace X* − {*a*}. ∃ *U V*. *openin X U* ∧ *openin X V* ∧ *x* ∈ *U* ∧ *a* ∈ *V* ∧ *disjnt U V*
using *X a* **by** (*simp add*: *Hausdorff_space_def*)
then obtain *U V* **where** *UV*: ∀ *x* ∈ *topspace X* − {*a*}. *openin X* (*U x*) ∧ *openin X* (*V x*) ∧ *x* ∈ *U x* ∧ *a* ∈ *V x* ∧ *disjnt* (*U x*) (*V x*)
by *metis*
with ‹*a* ∉ *T*› *compactin_subset_topspace* [*OF T*]
have *Topen*: ∀ *W* ∈ *U* ‘ *T*. *openin X W* **and** *Tsub*: *T* ⊆ ⋃ (*U* ‘ *T*)
by *auto*
then obtain *F* **where** *F*: *finite F F* ⊆ *U* ‘ *T* **and** *T* ⊆ ⋃*F*
using *T* **unfolding** *compactin_def* **by** *meson*

```
    then obtain F where F: finite F F ⊆ T ℱ = U ‘ F and SUF: T ⊆ ⋃(U ‘
F) and a ∉ F
      using finite_subset_image [OF ℱ] ‹a ∉ T› by (metis subsetD)
    have U: ⋀x. ⟦x ∈ topspace X; x ≠ a⟧ ⟹ openin X (U x)
      and V: ⋀x. ⟦x ∈ topspace X; x ≠ a⟧ ⟹ openin X (V x)
      and disj: ⋀x. ⟦x ∈ topspace X; x ≠ a⟧ ⟹ disjnt (U x) (V x)
      using UV by blast+
    show ?thesis
    proof (intro exI conjI)
      have F ≠ {}
        using False SUF by blast
      with ‹a ∉ F› show openin X (⋂(V ‘ F))
        using F compactin_subset_topspace [OF T] by (force intro: V)
      show openin X (⋃(U ‘ F))
        using F Topen Tsub by (force intro: U)
      show disjnt (⋂(V ‘ F)) (⋃(U ‘ F))
        using disj
        apply (auto simp: disjnt_def)
        using ‹F ⊆ T› ‹a ∉ F› compactin_subset_topspace [OF T] by blast
      show a ∈ (⋂(V ‘ F))
        using ‹F ⊆ T› T UV ‹a ∉ T› compactin_subset_topspace by blast
    qed (auto simp: SUF)
  qed
 qed
 then obtain U V where UV: ∀x ∈ S. openin X (U x) ∧ openin X (V x) ∧ x
∈ U x ∧ T ⊆ V x ∧ disjnt (U x) (V x)
  by metis
 then have S ⊆ ⋃ (U ‘ S)
  by auto
 moreover have ∀W ∈ U ‘ S. openin X W
  using UV by blast
 ultimately obtain I where I: S ⊆ ⋃ (U ‘ I) I ⊆ S finite I
  by (metis S compactin_def finite_subset_image)
 show thesis
 proof
  show openin X (⋃(U ‘ I))
   using ‹I ⊆ S› UV by blast
  show openin X (⋂ (V ‘ I))
   using False UV ‹I ⊆ S› ‹S ⊆ ⋃ (U ‘ I)› ‹finite I› by blast
  show disjnt (⋃(U ‘ I)) (⋂ (V ‘ I))
   by simp (meson UV ‹I ⊆ S› disjnt_subset2 in_mono le_INF_iff order_refl)
 qed (use UV I in auto)
qed


lemma Hausdorff_space_compact_sets:
 Hausdorff_space X ⟷
   (∀S T. compactin X S ∧ compactin X T ∧ disjnt S T
         ⟶ (∃U V. openin X U ∧ openin X V ∧ S ⊆ U ∧ T ⊆ V ∧ disjnt U
```

V))
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (*meson Hausdorff_space_compact_separation*)
next
assume *R* [*rule_format*]: *?rhs*
show *?lhs*
proof (*clarsimp simp add*: *Hausdorff_space_def*)
fix *x y*
assume *x* ∈ *topspace X y* ∈ *topspace X x* ≠ *y*
then show ∃ *U*. *openin X U* ∧ (∃ *V*. *openin X V* ∧ *x* ∈ *U* ∧ *y* ∈ *V* ∧ *disjnt U V*)
using *R* [*of* {*x*} {*y*}] **by** *auto*
qed
qed

lemma *compactin_imp_closedin*:
assumes *X*: *Hausdorff_space X* **and** *S*: *compactin X S* **shows** *closedin X S*
proof −
have *S* ⊆ *topspace X*
by (*simp add*: *assms compactin_subset_topspace*)
moreover
have ∃ *T*. *openin X T* ∧ *x* ∈ *T* ∧ *T* ⊆ *topspace X* − *S* **if** *x* ∈ *topspace X x* ∉ *S* **for** *x*
using *Hausdorff_space_compact_separation* [*OF X _ S*, *of* {*x*}] *that*
apply (*simp add*: *disjnt_def*)
by (*metis Diff_mono Diff_triv openin_subset*)
ultimately show *?thesis*
using *closedin_def openin_subopen* **by** *force*
qed

lemma *closedin_Hausdorff_singleton*:
⟦*Hausdorff_space X*; *x* ∈ *topspace X*⟧ ⟹ *closedin X* {*x*}
by (*simp add*: *Hausdorff_imp_t1_space closedin_t1_singleton*)

lemma *closedin_Hausdorff_sing_eq*:
Hausdorff_space X ⟹ *closedin X* {*x*} ⟷ *x* ∈ *topspace X*
by (*meson closedin_Hausdorff_singleton closedin_subset insert_subset*)

lemma *Hausdorff_space_discrete_topology* [*simp*]:
Hausdorff_space (*discrete_topology U*)
unfolding *Hausdorff_space_def*
by (*metis Hausdorff_space_compact_sets Hausdorff_space_def compactin_discrete_topology equalityE openin_discrete_topology*)

lemma *compactin_Int*:
⟦*Hausdorff_space X*; *compactin X S*; *compactin X T*⟧ ⟹ *compactin X* (*S* ∩ *T*)

by (*simp add*: *closed_Int_compactin compactin_imp_closedin*)

lemma *finite_topspace_imp_discrete_topology*:
⟦*topspace X* = *U*; *finite U*; *Hausdorff_space X*⟧ ⟹ *X* = *discrete_topology U*
using *Hausdorff_imp_t1_space finite_t1_space_imp_discrete_topology* **by** *blast*

lemma *derived_set_of_finite*:
⟦*Hausdorff_space X*; *finite S*⟧ ⟹ *X derived_set_of S* = {}
using *Hausdorff_imp_t1_space t1_space_derived_set_of_finite* **by** *auto*

lemma *infinite_perfect_set*:
⟦*Hausdorff_space X*; *S* ⊆ *X derived_set_of S*; *S* ≠ {}⟧ ⟹ *infinite S*
using *derived_set_of_finite* **by** *blast*

lemma *derived_set_of_singleton*:
Hausdorff_space X ⟹ *X derived_set_of* {*x*} = {}
by (*simp add*: *derived_set_of_finite*)

lemma *closedin_Hausdorff_finite*:
⟦*Hausdorff_space X*; *S* ⊆ *topspace X*; *finite S*⟧ ⟹ *closedin X S*
by (*simp add*: *compactin_imp_closedin finite_imp_compactin_eq*)

lemma *open_in_Hausdorff_delete*:
⟦*Hausdorff_space X*; *openin X S*⟧ ⟹ *openin X* (*S* − {*x*})
using *Hausdorff_imp_t1_space t1_space_openin_delete_alt* **by** *auto*

lemma *closedin_Hausdorff_finite_eq*:
⟦*Hausdorff_space X*; *finite S*⟧ ⟹ *closedin X S* ⟷ *S* ⊆ *topspace X*
by (*meson closedin_Hausdorff_finite closedin_def*)

lemma *derived_set_of_infinite_openin*:
Hausdorff_space X
⟹ *X derived_set_of S* =
{*x* ∈ *topspace X*. ∀ *U*. *x* ∈ *U* ∧ *openin X U* ⟶ *infinite*(*S* ∩ *U*)}
using *Hausdorff_imp_t1_space t1_space_derived_set_of_infinite_openin* **by** *fastforce*

lemma *Hausdorff_space_discrete_compactin*:
Hausdorff_space X
⟹ *S* ∩ *X derived_set_of S* = {} ∧ *compactin X S* ⟷ *S* ⊆ *topspace X* ∧ *finite S*
using *derived_set_of_finite discrete_compactin_eq_finite* **by** *fastforce*

lemma *Hausdorff_space_finite_topspace*:
Hausdorff_space X ⟹ *X derived_set_of* (*topspace X*) = {} ∧ *compact_space X* ⟷ *finite*(*topspace X*)
using *derived_set_of_finite discrete_compact_space_eq_finite* **by** *auto*

lemma *derived_set_of_derived_set_subset*:

Hausdorff_space X $\Longrightarrow$ *X derived_set_of* (*X derived_set_of S*) $\subseteq$ *X derived_set_of S*
by (*simp add*: *Hausdorff_imp_t1_space derived_set_of_derived_set_subset_gen*)

lemma *Hausdorff_space_injective_preimage*:
assumes *Hausdorff_space Y* **and** *cmf*: *continuous_map X Y f* **and** *inj_on f* (*topspace X*)
shows *Hausdorff_space X*
unfolding *Hausdorff_space_def*
proof *clarify*
fix *x y*
assume *x*: $x \in$ *topspace X* **and** *y*: $y \in$ *topspace X* **and** $x \neq y$
then obtain *U V* **where** *openin Y U openin Y V f x* $\in$ *U f y* $\in$ *V disjnt U V*
using *assms*
by (*smt* (*verit*, *ccfv_threshold*) *Hausdorff_space_def continuous_map image_subset_iff inj_on_def*)
show $\exists$ *U V*. *openin X U* $\wedge$ *openin X V* $\wedge$ $x \in$ *U* $\wedge$ $y \in$ *V* $\wedge$ *disjnt U V*
proof (*intro exI conjI*)
show *openin X* {$x \in$ *topspace X*. *f x* $\in$ *U*}
using ‹*openin Y U*› *cmf continuous_map* **by** *fastforce*
show *openin X* {$x \in$ *topspace X*. *f x* $\in$ *V*}
using ‹*openin Y V*› *cmf openin_continuous_map_preimage* **by** *blast*
show *disjnt* {$x \in$ *topspace X*. *f x* $\in$ *U*} {$x \in$ *topspace X*. *f x* $\in$ *V*}
using ‹*disjnt U V*› **by** (*auto simp add*: *disjnt_def*)
qed (*use x* ‹*f x* $\in$ *U*› *y* ‹*f y* $\in$ *V*› **in** *auto*)
qed

lemma *homeomorphic_Hausdorff_space*:
X homeomorphic_space Y $\Longrightarrow$ *Hausdorff_space X* $\longleftrightarrow$ *Hausdorff_space Y*
unfolding *homeomorphic_space_def homeomorphic_maps_map*
by (*auto simp*: *homeomorphic_eq_everything_map Hausdorff_space_injective_preimage*)

lemma *Hausdorff_space_retraction_map_image*:
⟦*retraction_map X Y r*; *Hausdorff_space X*⟧ $\Longrightarrow$ *Hausdorff_space Y*
unfolding *retraction_map_def*
using *Hausdorff_space_subtopology homeomorphic_Hausdorff_space retraction_maps_section_image2*
by *blast*

lemma *compact_Hausdorff_space_optimal*:
assumes *eq*: *topspace Y* = *topspace X* **and** *XY*: $\bigwedge$*U*. *openin X U* $\Longrightarrow$ *openin Y U*
and *Hausdorff_space X compact_space Y*
shows *Y* = *X*
proof −
have $\bigwedge$*U*. *closedin X U* $\Longrightarrow$ *closedin Y U*
using *XY* **using** *topology_finer_closedin* [*OF eq*]
by *metis*
have *openin Y S* = *openin X S* **for** *S*

by (*metis XY assms*(3) *assms*(4) *closedin_compact_space compactin_contractive compactin_imp_closedin eq openin_closedin_eq*)
then show *?thesis*
by (*simp add*: *topology_eq*)
qed

lemma *continuous_map_imp_closed_graph*:
assumes *f*: *continuous_map X Y f* **and** *Y*: *Hausdorff_space Y*
shows *closedin* (*prod_topology X Y*) ((λ*x*. (*x*,*f x*)) ' *topspace X*)
unfolding *closedin_def*
proof
show (λ*x*. (*x*, *f x*)) ' *topspace X* ⊆ *topspace* (*prod_topology X Y*)
using *continuous_map_def f* **by** *fastforce*
show *openin* (*prod_topology X Y*) (*topspace* (*prod_topology X Y*) − (λ*x*. (*x*, *f x*)) ' *topspace X*)
unfolding *openin_prod_topology_alt*
proof (*intro allI impI*)
show ∃ *U V*. *openin X U* ∧ *openin Y V* ∧ *x* ∈ *U* ∧ *y* ∈ *V* ∧ *U* × *V* ⊆ *topspace* (*prod_topology X Y*) − (λ*x*. (*x*, *f x*)) ' *topspace X*
if (*x*,*y*) ∈ *topspace* (*prod_topology X Y*) − (λ*x*. (*x*, *f x*)) ' *topspace X*
for *x y*
proof −
have *x* ∈ *topspace X* **and** *y*: *y* ∈ *topspace Y y* ≠ *f x*
using *that* **by** *auto*
then have *f x* ∈ *topspace Y*
using *continuous_map_image_subset_topspace f* **by** *blast*
then obtain *U V* **where** *UV*: *openin Y U openin Y V f x* ∈ *U y* ∈ *V disjnt U V*
using *Y y Hausdorff_space_def* **by** *metis*
show *?thesis*
proof (*intro exI conjI*)
show *openin X* {*x* ∈ *topspace X*. *f x* ∈ *U*}
using ‹*openin Y U*› *f openin_continuous_map_preimage* **by** *blast*
show {*x* ∈ *topspace X*. *f x* ∈ *U*} × *V* ⊆ *topspace* (*prod_topology X Y*) − (λ*x*. (*x*, *f x*)) ' *topspace X*
using *UV* **by** (*auto simp*: *disjnt_iff dest*: *openin_subset*)
qed (*use UV* ‹*x* ∈ *topspace X*› **in** *auto*)
qed
qed
qed

lemma *continuous_imp_closed_map*:
⟦*continuous_map X Y f*; *compact_space X*; *Hausdorff_space Y*⟧ ⟹ *closed_map X Y f*
by (*meson closed_map_def closedin_compact_space compactin_imp_closedin image_compactin*)

lemma *continuous_imp_quotient_map*:
⟦*continuous_map X Y f*; *compact_space X*; *Hausdorff_space Y*; *f* ' (*topspace*

```
X) = topspace Y⟧
      ⟹ quotient_map X Y f
 by (simp add: continuous_imp_closed_map continuous_closed_imp_quotient_map)

lemma continuous_imp_homeomorphic_map:
  ⟦continuous_map X Y f; compact_space X; Hausdorff_space Y;
    f ‘ (topspace X) = topspace Y; inj_on f (topspace X)⟧
      ⟹ homeomorphic_map X Y f
 by (simp add: continuous_imp_closed_map bijective_closed_imp_homeomorphic_map)

lemma continuous_imp_embedding_map:
  ⟦continuous_map X Y f; compact_space X; Hausdorff_space Y; inj_on f (topspace
X)⟧
      ⟹ embedding_map X Y f
 by (simp add: continuous_imp_closed_map injective_closed_imp_embedding_map)

lemma continuous_inverse_map:
  assumes compact_space X Hausdorff_space Y
    and cmf: continuous_map X Y f and gf: ⋀x. x ∈ topspace X ⟹ g(f x) = x
    and Sf:  S ⊆ f ‘ (topspace X)
  shows continuous_map (subtopology Y S) X g
proof (rule continuous_map_from_subtopology_mono [OF _ ‹S ⊆ f ‘ (topspace
X)›])
  show continuous_map (subtopology Y (f ‘ (topspace X))) X g
    unfolding continuous_map_closedin
  proof (intro conjI ballI allI impI)
    show g ∈ topspace (subtopology Y (f ‘ topspace X)) → topspace X
      using gf by auto
  next
    fix C
    assume C: closedin X C
    show closedin (subtopology Y (f ‘ topspace X))
          {x ∈ topspace (subtopology Y (f ‘ topspace X)). g x ∈ C}
    proof (rule compactin_imp_closedin)
      show Hausdorff_space (subtopology Y (f ‘ topspace X))
        using Hausdorff_space_subtopology [OF ‹Hausdorff_space Y›] by blast
      have compactin Y (f ‘ C)
       using C cmf image_compactin closedin_compact_space [OF ‹compact_space
X›] by blast
      moreover have {x ∈ topspace Y. x ∈ f ‘ topspace X ∧ g x ∈ C} = f ‘ C
       using closedin_subset [OF C] cmf by (auto simp: gf continuous_map_def)
      ultimately have compactin Y {x ∈ topspace Y. x ∈ f ‘ topspace X ∧ g x ∈
C}
        by simp
      then show compactin (subtopology Y (f ‘ topspace X))
                  {x ∈ topspace (subtopology Y (f ‘ topspace X)). g x ∈ C}
        by (auto simp add: compactin_subtopology)
    qed
  qed
```

qed

lemma *closed_map_paired_continuous_map_right*:
⟦*continuous_map X Y f*; *Hausdorff_space Y*⟧ ⟹ *closed_map X* (*prod_topology X Y*) (λ*x*. (*x*,*f x*))
by (*simp add*: *continuous_map_imp_closed_graph embedding_map_graph embedding_imp_closed_map*)

lemma *closed_map_paired_continuous_map_left*:
assumes *f*: *continuous_map X Y f* **and** *Y*: *Hausdorff_space Y*
shows *closed_map X* (*prod_topology Y X*) (λ*x*. (*f x*,*x*))
proof −
have *eq*: (λ*x*. (*f x*,*x*)) = (λ(*a*,*b*). (*b*,*a*)) ∘ (λ*x*. (*x*,*f x*))
by *auto*
show *?thesis*
unfolding *eq*
proof (*rule closed_map_compose*)
show *closed_map X* (*prod_topology X Y*) (λ*x*. (*x*, *f x*))
using *Y closed_map_paired_continuous_map_right f* **by** *blast*
show *closed_map* (*prod_topology X Y*) (*prod_topology Y X*) (λ(*a*, *b*). (*b*, *a*))
by (*metis homeomorphic_map_swap homeomorphic_imp_closed_map*)
qed
qed

lemma *proper_map_paired_continuous_map_right*:
⟦*continuous_map X Y f*; *Hausdorff_space Y*⟧
⟹ *proper_map X* (*prod_topology X Y*) (λ*x*. (*x*,*f x*))
using *closed_injective_imp_proper_map closed_map_paired_continuous_map_right*
by (*metis* (*mono_tags*, *lifting*) *Pair_inject inj_onI*)

lemma *proper_map_paired_continuous_map_left*:
⟦*continuous_map X Y f*; *Hausdorff_space Y*⟧
⟹ *proper_map X* (*prod_topology Y X*) (λ*x*. (*f x*,*x*))
using *closed_injective_imp_proper_map closed_map_paired_continuous_map_left*
by (*metis* (*mono_tags*, *lifting*) *Pair_inject inj_onI*)

lemma *Hausdorff_space_prod_topology*:
Hausdorff_space(*prod_topology X Y*) ⟷ (*prod_topology X Y*) = *trivial_topology*
∨ *Hausdorff_space X* ∧ *Hausdorff_space Y*
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (*rule topological_property_of_prod_component*) (*auto simp*: *Hausdorff_space_subtopology homeomorphic_Hausdorff_space*)
next
assume *R*: *?rhs*
show *?lhs*
proof (*cases* (*topspace X* × *topspace Y*) = {})

case *False*
with *R* **have** *ne*: *topspace X* ≠ {} *topspace Y* ≠ {} **and** *X*: *Hausdorff_space X* **and** *Y*: *Hausdorff_space Y*
by *auto*
show *?thesis*
unfolding *Hausdorff_space_def*
proof *clarify*
fix *x y x′ y′*
assume *xy*: (*x*, *y*) ∈ *topspace* (*prod_topology X Y*)
and *xy′*: (*x′*,*y′*) ∈ *topspace* (*prod_topology X Y*)
and ∗: ∄ *U V*. *openin* (*prod_topology X Y*) *U* ∧ *openin* (*prod_topology X Y*) *V*
∧ (*x*, *y*) ∈ *U* ∧ (*x′*, *y′*) ∈ *V* ∧ *disjnt U V*
have *False* **if** *x* ≠ *x′* ∨ *y* ≠ *y′*
using *that*
proof
assume *x* ≠ *x′*
then obtain *U V* **where** *openin X U openin X V x* ∈ *U x′* ∈ *V disjnt U V*
by (*metis Hausdorff_space_def X mem_Sigma_iff topspace_prod_topology xy xy′*)
let *?U* = *U* × *topspace Y*
let *?V* = *V* × *topspace Y*
have *openin* (*prod_topology X Y*) *?U openin* (*prod_topology X Y*) *?V*
by (*simp_all add*: *openin_prod_Times_iff* ‹*openin X U*› ‹*openin X V*›)
moreover have *disjnt ?U ?V*
by (*simp add*: ‹*disjnt U V*›)
ultimately show *False*
using ∗ ‹*x* ∈ *U*› ‹*x′* ∈ *V*› *xy xy′* **by** (*metis SigmaD2 SigmaI topspace_prod_topology*)
next
assume *y* ≠ *y′*
then obtain *U V* **where** *openin Y U openin Y V y* ∈ *U y′* ∈ *V disjnt U V*
by (*metis Hausdorff_space_def Y mem_Sigma_iff topspace_prod_topology xy xy′*)
let *?U* = *topspace X* × *U*
let *?V* = *topspace X* × *V*
have *openin* (*prod_topology X Y*) *?U openin* (*prod_topology X Y*) *?V*
by (*simp_all add*: *openin_prod_Times_iff* ‹*openin Y U*› ‹*openin Y V*›)
moreover have *disjnt ?U ?V*
by (*simp add*: ‹*disjnt U V*›)
ultimately show *False*
using ∗ ‹*y* ∈ *U*› ‹*y′* ∈ *V*› *xy xy′* **by** (*metis SigmaD1 SigmaI topspace_prod_topology*)
qed
then show *x* = *x′* ∧ *y* = *y′*
by *blast*
qed
qed *force*
qed

lemma *Hausdorff_space_product_topology*:
Hausdorff_space (*product_topology X I*) ⟷ (Π$_E$ *i*∈*I*. *topspace*(*X i*)) = {} ∨ (∀ *i* ∈ *I*. *Hausdorff_space* (*X i*))
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (*simp add*: *Hausdorff_space_subtopology PiE_eq_empty_iff homeomorphic_Hausdorff_space*
topological_property_of_product_component)
next
assume *R*: *?rhs*
show *?lhs*
proof (*cases* (Π$_E$ *i*∈*I*. *topspace*(*X i*)) = {})
case *True*
then show *?thesis*
by (*simp add*: *Hausdorff_space_def*)
next
case *False*
have ∃ *U V*. *openin* (*product_topology X I*) *U* ∧ *openin* (*product_topology X I*) *V* ∧ *f* ∈ *U* ∧ *g* ∈ *V* ∧ *disjnt U V*
if *f*: *f* ∈ (Π$_E$ *i*∈*I*. *topspace* (*X i*)) **and** *g*: *g* ∈ (Π$_E$ *i*∈*I*. *topspace* (*X i*)) **and** *f* ≠ *g*
for *f g* :: *'a* ⇒ *'b*
proof −
obtain *m* **where** *f m* ≠ *g m*
using ‹*f* ≠ *g*› **by** *blast*
then have *m* ∈ *I*
using *f g* **by** *fastforce*
then have *Hausdorff_space* (*X m*)
using *False that R* **by** *blast*
then obtain *U V* **where** *U*: *openin* (*X m*) *U* **and** *V*: *openin* (*X m*) *V* **and** *f m* ∈ *U g m* ∈ *V disjnt U V*
by (*metis Hausdorff_space_def PiE_mem* ‹*f m* ≠ *g m*› ‹*m* ∈ *I*› *f g*)
show *?thesis*
proof (*intro exI conjI*)
let *?U* = (Π$_E$ *i*∈*I*. *topspace*(*X i*)) ∩ {*x*. *x m* ∈ *U*}
let *?V* = (Π$_E$ *i*∈*I*. *topspace*(*X i*)) ∩ {*x*. *x m* ∈ *V*}
show *openin* (*product_topology X I*) *?U openin* (*product_topology X I*) *?V*
using ‹*m* ∈ *I*› *U V*
by (*force simp add*: *openin_product_topology intro*: *arbitrary_union_of_inc relative_to_inc finite_intersection_of_inc*)+
show *f* ∈ *?U*
using ‹*f m* ∈ *U*› *f* **by** *blast*
show *g* ∈ *?V*
using ‹*g m* ∈ *V*› *g* **by** *blast*
show *disjnt ?U ?V*
using ‹*disjnt U V*› **by** (*auto simp*: *PiE_def Pi_def disjnt_def*)
qed

```
    qed
    then show ?thesis
      by (simp add: Hausdorff_space_def)
  qed
qed

lemma Hausdorff_space_closed_neighbourhood:
   Hausdorff_space X ⟷
    (∀ x ∈ topspace X. ∃ U C. openin X U ∧ closedin X C ∧
                    Hausdorff_space(subtopology X C) ∧ x ∈ U ∧ U ⊆ C) (is _ =
?rhs)
proof
  assume R: ?rhs
  show Hausdorff_space X
    unfolding Hausdorff_space_def
  proof clarify
    fix x y
    assume x: x ∈ topspace X and y: y ∈ topspace X and x ≠ y
    obtain T C where ∗: openin X T closedin X C x ∈ T T ⊆ C
                and C: Hausdorff_space (subtopology X C)
      by (meson R ‹x ∈ topspace X›)
    show ∃ U V. openin X U ∧ openin X V ∧ x ∈ U ∧ y ∈ V ∧ disjnt U V
    proof (cases y ∈ C)
      case True
      with ∗ C obtain U V where U: openin (subtopology X C) U
                           and V: openin (subtopology X C) V
                           and x ∈ U y ∈ V disjnt U V
        unfolding Hausdorff_space_def
      by (smt (verit, best) ‹x ≠ y› closedin_subset subsetD topspace_subtopology_subset)
       then obtain U' V' where UV': U = U' ∩ C openin X U' V = V' ∩ C
openin X V'
        by (meson openin_subtopology)
      have disjnt (T ∩ U') V'
        using ‹disjnt U V› UV' ‹T ⊆ C› by (force simp: disjnt_iff)
      with ‹T ⊆ C› have disjnt (T ∩ U') (V' ∪ (topspace X − C))
        unfolding disjnt_def by blast
      moreover
      have openin X (T ∩ U')
        by (simp add: ‹openin X T› ‹openin X U'› openin_Int)
      moreover have openin X (V' ∪ (topspace X − C))
        using ‹closedin X C› ‹openin X V'› by auto
      ultimately show ?thesis
        using UV' ‹x ∈ T› ‹x ∈ U› ‹y ∈ V› by blast
    next
      case False
      with ∗ y show ?thesis
        by (force simp: closedin_def disjnt_def)
    qed
  qed
```

qed *fastforce*

end

2.10 Lindelöf spaces

theory *Lindelof_Spaces*
imports *T1_Spaces*
begin

definition *Lindelof_space* **where**
Lindelof_space X ≡
∀$\mathcal{U}$. (∀ *U* ∈ $\mathcal{U}$. *openin X U*) ∧ ⋃$\mathcal{U}$ = *topspace X*
⟶ (∃$\mathcal{V}$. *countable* $\mathcal{V}$ ∧ $\mathcal{V}$ ⊆ $\mathcal{U}$ ∧ ⋃$\mathcal{V}$ = *topspace X*)

lemma *Lindelof_spaceD*:
⟦*Lindelof_space X*; ⋀*U*. *U* ∈ $\mathcal{U}$ ⟹ *openin X U*; ⋃$\mathcal{U}$ = *topspace X*⟧
⟹ ∃$\mathcal{V}$. *countable* $\mathcal{V}$ ∧ $\mathcal{V}$ ⊆ $\mathcal{U}$ ∧ ⋃$\mathcal{V}$ = *topspace X*
by (*auto simp*: *Lindelof_space_def*)

lemma *Lindelof_space_alt*:
Lindelof_space X ⟷
(∀$\mathcal{U}$. (∀ *U* ∈ $\mathcal{U}$. *openin X U*) ∧ *topspace X* ⊆ ⋃$\mathcal{U}$
⟶ (∃$\mathcal{V}$. *countable* $\mathcal{V}$ ∧ $\mathcal{V}$ ⊆ $\mathcal{U}$ ∧ *topspace X* ⊆ ⋃$\mathcal{V}$))
unfolding *Lindelof_space_def*
using *openin_subset* **by** *fastforce*

lemma *compact_imp_Lindelof_space*:
compact_space X ⟹ *Lindelof_space X*
unfolding *Lindelof_space_def compact_space*
by (*meson uncountable_infinite*)

lemma *Lindelof_space_topspace_empty*:
topspace X = {} ⟹ *Lindelof_space X*
using *compact_imp_Lindelof_space compact_space_trivial_topology* **by** *force*

lemma *Lindelof_space_Union*:
assumes $\mathcal{U}$: *countable* $\mathcal{U}$ **and** *lin*: ⋀*U*. *U* ∈ $\mathcal{U}$ ⟹ *Lindelof_space* (*subtopology X U*)
shows *Lindelof_space* (*subtopology X* (⋃$\mathcal{U}$))
proof −
have ∃$\mathcal{V}$. *countable* $\mathcal{V}$ ∧ $\mathcal{V}$ ⊆ $\mathcal{F}$ ∧ ⋃$\mathcal{U}$ ∩ ⋃$\mathcal{V}$ = *topspace X* ∩ ⋃$\mathcal{U}$
if $\mathcal{F}$: $\mathcal{F}$ ⊆ *Collect* (*openin X*) **and** *UF*: ⋃$\mathcal{U}$ ∩ ⋃$\mathcal{F}$ = *topspace X* ∩ ⋃$\mathcal{U}$
for $\mathcal{F}$
proof −
have ⋀*U*. ⟦*U* ∈ $\mathcal{U}$; *U* ∩ ⋃$\mathcal{F}$ = *topspace X* ∩ *U*⟧
⟹ ∃$\mathcal{V}$. *countable* $\mathcal{V}$ ∧ $\mathcal{V}$ ⊆ $\mathcal{F}$ ∧ *U* ∩ ⋃$\mathcal{V}$ = *topspace X* ∩ *U*
using *lin* $\mathcal{F}$
unfolding *Lindelof_space_def openin_subtopology_alt Ball_def subset_iff*

```
[symmetric]
      by (simp add: all_subset_image imp_conjL ex_countable_subset_image)
    then obtain g where g: ⋀U. ⟦U ∈ 𝒰; U ∩ ⋃ℱ = topspace X ∩ U⟧
                           ⟹ countable (g U) ∧ (g U) ⊆ ℱ ∧ U ∩ ⋃(g U) =
topspace X ∩ U
      by metis
    show ?thesis
    proof (intro exI conjI)
      show countable (⋃(g ` 𝒰))
        using Int_commute UF g  by (fastforce intro: countable_UN [OF 𝒰])
      show ⋃(g ` 𝒰) ⊆ ℱ
        using g UF by blast
      show ⋃𝒰 ∩ ⋃(⋃(g ` 𝒰)) = topspace X ∩ ⋃𝒰
      proof
        show ⋃𝒰 ∩ ⋃(⋃(g ` 𝒰)) ⊆ topspace X ∩ ⋃𝒰
          using g UF by blast
        show topspace X ∩ ⋃𝒰 ⊆ ⋃𝒰 ∩ ⋃(⋃(g ` 𝒰))
        proof clarsimp
          show ∃y∈𝒰. ∃W∈g y. x ∈ W
            if x ∈ topspace X x ∈ V V ∈ 𝒰 for x V
          proof −
            have V ∩ ⋃ℱ = topspace X ∩ V
              using UF ‹V ∈ 𝒰› by blast
            with that g [OF ‹V ∈ 𝒰›]  show ?thesis by blast
          qed
        qed
      qed
    qed
  qed
  then show ?thesis
       unfolding Lindelof_space_def openin_subtopology_alt Ball_def subset_iff
[symmetric]
      by (simp add: all_subset_image imp_conjL ex_countable_subset_image)
qed

lemma countable_imp_Lindelof_space:
  assumes countable(topspace X)
  shows Lindelof_space X
proof −
  have Lindelof_space (subtopology X (⋃x ∈ topspace X. {x}))
  proof (rule Lindelof_space_Union)
    show countable ((λx. {x}) ` topspace X)
      using assms by blast
    show Lindelof_space (subtopology X U)
      if U ∈ (λx. {x}) ` topspace X for U
    proof −
      have compactin X U
        using that by force
      then show ?thesis
```

```
      by (meson compact_imp_Lindelof_space compact_space_subtopology)
    qed
  qed
  then show ?thesis
    by simp
qed
lemma Lindelof_space_subtopology:
  Lindelof_space(subtopology X S) ⟷
      (∀𝒰. (∀ U ∈ 𝒰. openin X U) ∧ topspace X ∩ S ⊆ ⋃𝒰
          ⟶ (∃ V. countable V ∧ V ⊆ 𝒰 ∧ topspace X ∩ S ⊆ ⋃ V))
proof −
  have ∗: (S ∩ ⋃𝒰 = topspace X ∩ S) = (topspace X ∩ S ⊆ ⋃𝒰)
    if ⋀x. x ∈ 𝒰 ⟹ openin X x for 𝒰
    by (blast dest: openin_subset [OF that])
  moreover have (𝒱 ⊆ 𝒰 ∧ S ∩ ⋃𝒱 = topspace X ∩ S) = (𝒱 ⊆ 𝒰 ∧ topspace X
∩ S ⊆ ⋃𝒱)
    if ∀x. x ∈ 𝒰 ⟶ openin X x topspace X ∩ S ⊆ ⋃𝒰 countable 𝒱 for 𝒰 𝒱
    using that ∗ by blast
  ultimately show ?thesis
    unfolding Lindelof_space_def openin_subtopology_alt Ball_def
     apply (simp add: all_subset_image imp_conjL ex_countable_subset_image
flip: subset_iff)
    apply (intro all_cong1 imp_cong ex_cong, auto)
    done
qed

lemma Lindelof_space_subtopology_subset:
  S ⊆ topspace X
      ⟹ (Lindelof_space(subtopology X S) ⟷
          (∀𝒰. (∀ U ∈ 𝒰. openin X U) ∧ S ⊆ ⋃𝒰
              ⟶ (∃ V. countable V ∧ V ⊆ 𝒰 ∧ S ⊆ ⋃ V)))
 by (metis Lindelof_space_subtopology topspace_subtopology topspace_subtopology_subset)

lemma Lindelof_space_closedin_subtopology:
  assumes X: Lindelof_space X and clo: closedin X S
  shows Lindelof_space (subtopology X S)
proof −
  have S ⊆ topspace X
    by (simp add: clo closedin_subset)
  then show ?thesis
  proof (clarsimp simp add: Lindelof_space_subtopology_subset)
    show ∃ V. countable V ∧ V ⊆ ℱ ∧ S ⊆ ⋃ V
      if ∀ U∈ℱ. openin X U and S ⊆ ⋃ℱ for ℱ
    proof −
      have ∃𝒱. countable 𝒱 ∧ 𝒱 ⊆ insert (topspace X − S) ℱ ∧ ⋃𝒱 = topspace X
      proof (rule Lindelof_spaceD [OF X, of insert (topspace X − S) ℱ])
        show openin X U
          if U ∈ insert (topspace X − S) ℱ for U
          using that ‹∀ U∈ℱ. openin X U› clo by blast
```

```
      show ⋃(insert (topspace X − S) ℱ) = topspace X
        apply auto
        apply (meson in_mono openin_closedin_eq that(1))
        using UnionE ‹S ⊆ ⋃ℱ› by auto
      qed
      then obtain 𝒱 where countable 𝒱 𝒱 ⊆ insert (topspace X − S) ℱ ⋃𝒱 =
topspace X
        by metis
      with ‹S ⊆ topspace X›
      show ?thesis
        by (rule_tac x=(𝒱 − {topspace X − S}) in exI) auto
    qed
  qed
qed

lemma Lindelof_space_continuous_map_image:
   assumes X: Lindelof_space X and f: continuous_map X Y f and fim: f ‘
(topspace X) = topspace Y
  shows Lindelof_space Y
proof −
  have ∃𝒱. countable 𝒱 ∧ 𝒱 ⊆ 𝒰 ∧ ⋃𝒱 = topspace Y
    if 𝒰: ⋀U. U ∈ 𝒰 ⟹ openin Y U and UU: ⋃𝒰 = topspace Y for 𝒰
  proof −
    define 𝒱 where 𝒱 ≡ (λU. {x ∈ topspace X. f x ∈ U}) ‘ 𝒰
    have ⋀V. V ∈ 𝒱 ⟹ openin X V
      unfolding 𝒱_def using 𝒰 continuous_map f by fastforce
    moreover have ⋃𝒱 = topspace X
      unfolding 𝒱_def using UU fim by fastforce
    ultimately have ∃𝒲. countable 𝒲 ∧ 𝒲 ⊆ 𝒱 ∧ ⋃𝒲 = topspace X
      using X by (simp add: Lindelof_space_def)
    then obtain 𝒞 where countable 𝒞 𝒞 ⊆ 𝒰 and 𝒞: (⋃U∈𝒞. {x ∈ topspace X. f
x ∈ U}) = topspace X
      by (metis (no_types, lifting) 𝒱_def countable_subset_image)
    moreover have ⋃𝒞 = topspace Y
    proof
      show ⋃𝒞 ⊆ topspace Y
        using UU 𝒞 ‹𝒞 ⊆ 𝒰› by fastforce
      have y ∈ ⋃𝒞 if y ∈ topspace Y for y
      proof −
        obtain x where x ∈ topspace X y = f x
          using that fim by (metis ‹y ∈ topspace Y› imageE)
        with 𝒞 show ?thesis by auto
      qed
      then show topspace Y ⊆ ⋃𝒞 by blast
    qed
    ultimately show ?thesis
      by blast
  qed
  then show ?thesis
```

unfolding *Lindelof_space_def*
by *auto*
qed

lemma *Lindelof_space_quotient_map_image*:
⟦*quotient_map X Y q*; *Lindelof_space X*⟧ ⟹ *Lindelof_space Y*
by (*meson Lindelof_space_continuous_map_image quotient_imp_continuous_map quotient_imp_surjective_map*)

lemma *Lindelof_space_retraction_map_image*:
⟦*retraction_map X Y r*; *Lindelof_space X*⟧ ⟹ *Lindelof_space Y*
using *Abstract_Topology.retraction_imp_quotient_map Lindelof_space_quotient_map_image*
by *blast*

lemma *locally_finite_cover_of_Lindelof_space*:
assumes *X*: *Lindelof_space X* **and** *UU*: *topspace X* ⊆ ⋃ *𝒰* **and** *fin*: *locally_finite_in X 𝒰*
shows *countable 𝒰*
proof −
have *UU_eq*: ⋃ *𝒰* = *topspace X*
by (*meson UU fin locally_finite_in_def subset_antisym*)
obtain *T* **where** *T*: ⋀*x*. *x* ∈ *topspace X* ⟹ *openin X* (*T x*) ∧ *x* ∈ *T x* ∧ *finite* {*U* ∈ *𝒰*. *U* ∩ *T x* ≠ {}}
using *fin* **unfolding** *locally_finite_in_def* **by** *meson*
then obtain *I* **where** *countable I I* ⊆ *topspace X* **and** *I*: *topspace X* ⊆ ⋃ (*T* ‘ *I*)
using *X* **unfolding** *Lindelof_space_alt*
by (*drule_tac x*=*image T* (*topspace X*) **in** *spec*) (*auto simp*: *ex_countable_subset_image*)
show *?thesis*
proof (*rule countable_subset*)
have ⋀*i*. *i* ∈ *I* ⟹ *countable* {*U* ∈ *𝒰*. *U* ∩ *T i* ≠ {}}
using *T*
by (*meson* ‹*I* ⊆ *topspace X*› *in_mono uncountable_infinite*)
then show *countable* (*insert* {} (⋃ *i*∈*I*. {*U* ∈ *𝒰*. *U* ∩ *T i* ≠ {}}))
by (*simp add*: ‹*countable I*›)
qed (*use UU_eq I* **in** *auto*)
qed

lemma *Lindelof_space_proper_map_preimage*:
assumes *f*: *proper_map X Y f* **and** *Y*: *Lindelof_space Y*
shows *Lindelof_space X*
proof (*clarsimp simp*: *Lindelof_space_alt*)
show ∃ *𝒱*. *countable 𝒱* ∧ *𝒱* ⊆ *𝒰* ∧ *topspace X* ⊆ ⋃ *𝒱*
if *𝒰*: ∀ *U*∈*𝒰*. *openin X U* **and** *sub_UU*: *topspace X* ⊆ ⋃ *𝒰* **for** *𝒰*
proof −
have ∃ *𝒱*. *finite 𝒱* ∧ *𝒱* ⊆ *𝒰* ∧ {*x* ∈ *topspace X*. *f x* = *y*} ⊆ ⋃ *𝒱* **if** *y* ∈ *topspace Y* **for** *y*
proof (*rule compactinD*)

show *compactin X* $\{x \in$ *topspace X*. *f x* $= y\}$
using *f proper_map_def that* **by** *fastforce*
qed (*use sub_UU* $\mathcal{U}$ **in** *auto*)
then obtain $\mathcal{V}$ **where** $\mathcal{V}$: $\bigwedge y$. $y \in$ *topspace Y* $\Longrightarrow$ *finite* $(\mathcal{V}\ y) \wedge \mathcal{V}\ y \subseteq \mathcal{U} \wedge$ $\{x \in$ *topspace X*. *f x* $= y\} \subseteq \bigcup(\mathcal{V}\ y)$
by *meson*
define $\mathcal{W}$ **where** $\mathcal{W} \equiv (\lambda y.$ *topspace Y* $-$ *image f* (*topspace X* $- \bigcup(\mathcal{V}\ y)))$ ‘ *topspace Y*
have $\forall\, U \in \mathcal{W}$. *openin Y U*
using *f* $\mathcal{U}$ $\mathcal{V}$ **unfolding** $\mathcal{W}$*_def proper_map_def closed_map_def*
by (*simp add*: *closedin_diff openin_Union openin_diff subset_iff*)
moreover have *topspace Y* $\subseteq \bigcup\mathcal{W}$
using $\mathcal{V}$ **unfolding** $\mathcal{W}$*_def* **by** *clarsimp fastforce*
ultimately have $\exists\mathcal{V}$. *countable* $\mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{W} \wedge$ *topspace Y* $\subseteq \bigcup\mathcal{V}$
using *Y* **by** (*simp add*: *Lindelof_space_alt*)
then obtain *I* **where** *countable I I* $\subseteq$ *topspace Y*
and *I*: *topspace Y* $\subseteq (\bigcup i{\in}I.$ *topspace Y* $-$ *f* ‘ (*topspace X* $- \bigcup(\mathcal{V}\ i)))$
unfolding $\mathcal{W}$*_def ex_countable_subset_image* **by** *metis*
show *?thesis*
proof (*intro exI conjI*)
have $\bigwedge i.\ i \in I \Longrightarrow$ *countable* $(\mathcal{V}\ i)$
by (*meson* $\mathcal{V}$ ‹*I* $\subseteq$ *topspace Y*› *in_mono uncountable_infinite*)
with ‹*countable I*› **show** *countable* $(\bigcup(\mathcal{V}$ ‘ *I*))
by *auto*
show $\bigcup(\mathcal{V}$ ‘ *I*) $\subseteq \mathcal{U}$
using $\mathcal{V}$ ‹*I* $\subseteq$ *topspace Y*› **by** *fastforce*
show *topspace X* $\subseteq \bigcup(\bigcup(\mathcal{V}$ ‘ *I*))
proof
show $x \in \bigcup\ (\bigcup\ (\mathcal{V}$ ‘ *I*)) **if** $x \in$ *topspace X* **for** *x*
proof $-$
have *f x* $\in$ *topspace Y*
using *f proper_map_imp_subset_topspace that* **by** *fastforce*
then show *?thesis*
using *that I* **by** *auto*
qed
qed
qed
qed
qed

lemma *Lindelof_space_perfect_map_image*:
$[\![$*Lindelof_space X*; *perfect_map X Y f*$]\!] \Longrightarrow$ *Lindelof_space Y*
using *Lindelof_space_quotient_map_image perfect_imp_quotient_map* **by** *blast*

lemma *Lindelof_space_perfect_map_image_eq*:
perfect_map X Y f $\Longrightarrow$ *Lindelof_space X* $\longleftrightarrow$ *Lindelof_space Y*
using *Lindelof_space_perfect_map_image Lindelof_space_proper_map_preimage perfect_map_def* **by** *blast*

end

Chapter 3

Functional Analysis

3.1 A decision procedure for metric spaces

theory *Metric_Arith*
  imports *HOL.Real_Vector_Spaces*
begin

A port of the decision procedure "Formalization of metric spaces in HOL Light" [4] for the type class *metric_space*, with the *Argo* solver as backend.

named_theorems *metric_prenex*
named_theorems *metric_nnf*
named_theorems *metric_unfold*
named_theorems *metric_pre_arith*

lemmas *pre_arith_simps* =
  max.bounded_iff max_less_iff_conj
  le_max_iff_disj less_max_iff_disj
  simp_thms HOL.eq_commute
declare *pre_arith_simps* [*metric_pre_arith*]

lemmas *unfold_simps* =
  Un_iff subset_iff disjoint_iff_not_equal
  Ball_def Bex_def
declare *unfold_simps* [*metric_unfold*]

declare *HOL.nnf_simps*(*4*) [*metric_prenex*]

lemma *imp_prenex* [*metric_prenex*]:
  $\bigwedge P\ Q.\ (\exists x.\ P\ x) \longrightarrow Q \equiv \forall x.\ (P\ x \longrightarrow Q)$
  $\bigwedge P\ Q.\ P \longrightarrow (\exists x.\ Q\ x) \equiv \exists x.\ (P \longrightarrow Q\ x)$
  $\bigwedge P\ Q.\ (\forall x.\ P\ x) \longrightarrow Q \equiv \exists x.\ (P\ x \longrightarrow Q)$
  $\bigwedge P\ Q.\ P \longrightarrow (\forall x.\ Q\ x) \equiv \forall x.\ (P \longrightarrow Q\ x)$
  by *simp_all*

lemma *ex_prenex* [*metric_prenex*]:

$\bigwedge P\ Q.\ (\exists\, x.\ P\ x) \wedge Q \equiv \exists\, x.\ (P\ x \wedge Q)$
$\bigwedge P\ Q.\ P \wedge (\exists\, x.\ Q\ x) \equiv \exists\, x.\ (P \wedge Q\ x)$
$\bigwedge P\ Q.\ (\exists\, x.\ P\ x) \vee Q \equiv \exists\, x.\ (P\ x \vee Q)$
$\bigwedge P\ Q.\ P \vee (\exists\, x.\ Q\ x) \equiv \exists\, x.\ (P \vee Q\ x)$
$\bigwedge P.\ \neg(\exists\, x.\ P\ x) \equiv \forall\, x.\ \neg P\ x$
by *simp_all*

lemma *all_prenex* [*metric_prenex*]:
$\bigwedge P\ Q.\ (\forall\, x.\ P\ x) \wedge Q \equiv \forall\, x.\ (P\ x \wedge Q)$
$\bigwedge P\ Q.\ P \wedge (\forall\, x.\ Q\ x) \equiv \forall\, x.\ (P \wedge Q\ x)$
$\bigwedge P\ Q.\ (\forall\, x.\ P\ x) \vee Q \equiv \forall\, x.\ (P\ x \vee Q)$
$\bigwedge P\ Q.\ P \vee (\forall\, x.\ Q\ x) \equiv \forall\, x.\ (P \vee Q\ x)$
$\bigwedge P.\ \neg(\forall\, x.\ P\ x) \equiv \exists\, x.\ \neg P\ x$
by *simp_all*

lemma *nnf_thms* [*metric_nnf*]:
$(\neg\ (P \wedge Q)) = (\neg\ P \vee \neg\ Q)$
$(\neg\ (P \vee Q)) = (\neg\ P \wedge \neg\ Q)$
$(P \longrightarrow Q) = (\neg\ P \vee Q)$
$(P = Q) \longleftrightarrow (P \vee \neg\ Q) \wedge (\neg\ P \vee Q)$
$(\neg\ (P = Q)) \longleftrightarrow (\neg\ P \vee \neg\ Q) \wedge (P \vee Q)$
$(\neg\ \neg\ P) = P$
by *blast+*

lemmas *nnf_simps* = *nnf_thms linorder_not_less linorder_not_le*
declare *nnf_simps*[*metric_nnf*]

lemma *ball_insert*: $(\forall\, x{\in}\mathit{insert}\ a\ B.\ P\ x) = (P\ a \wedge (\forall\, x{\in}B.\ P\ x))$
by *blast*

lemma *Sup_insert_insert*:
fixes *a*::*real*
shows *Sup* (*insert a* (*insert b s*)) = *Sup* (*insert* (*max a b*) *s*)
by (*simp add*: *Sup_real_def*)

lemma *real_abs_dist*: $|\mathit{dist}\ x\ y| = \mathit{dist}\ x\ y$
by *simp*

lemma *maxdist_thm* [*THEN HOL.eq_reflection*]:
assumes *finite s x* ∈ *s y* ∈ *s*
defines $\bigwedge a.\ f\ a \equiv |\mathit{dist}\ x\ a - \mathit{dist}\ a\ y|$
shows *dist x y* = *Sup* (*f* ‘ *s*)
proof −
have *dist x y* ≤ *Sup* (*f* ‘ *s*)
proof −
have *finite* (*f* ‘ *s*)
by (*simp add*: ‹*finite s*›)
moreover have $|\mathit{dist}\ x\ y - \mathit{dist}\ y\ y| \in f$ ‘ *s*
by (*metis* ‹*y* ∈ *s*› *f_def imageI*)

```
    ultimately show ?thesis
      using le_cSup_finite by simp
  qed
  also have Sup (f ‘ s) ≤ dist x y
    using ‹x ∈ s› cSUP_least[of s f] abs_dist_diff_le
    unfolding f_def
    by blast
  finally show ?thesis .
qed

theorem metric_eq_thm [THEN HOL.eq_reflection]:
  x ∈ s ⟹ y ∈ s ⟹ x = y ⟷ (∀ a∈s. dist x a = dist y a)
  by auto

ML_file ‹metric_arith.ML›

method_setup metric = ‹
  Scan.succeed (SIMPLE_METHOD' o Metric_Arith.metric_arith_tac)
› prove simple linear statements in metric spaces (∀∃_p fragment)

end
```

3.2 Elementary Metric Spaces

```
theory Elementary_Metric_Spaces
  imports
    Abstract_Topology_2
    Metric_Arith
begin
```

3.2.1 Open and closed balls

```
definition ball :: 'a::metric_space ⇒ real ⇒ 'a set
  where ball x e = {y. dist x y < e}

definition cball :: 'a::metric_space ⇒ real ⇒ 'a set
  where cball x e = {y. dist x y ≤ e}

definition sphere :: 'a::metric_space ⇒ real ⇒ 'a set
  where sphere x e = {y. dist x y = e}

lemma mem_ball [simp, metric_unfold]: y ∈ ball x e ⟷ dist x y < e
  by (simp add: ball_def)

lemma mem_cball [simp, metric_unfold]: y ∈ cball x e ⟷ dist x y ≤ e
  by (simp add: cball_def)

lemma mem_sphere [simp]: y ∈ sphere x e ⟷ dist x y = e
  by (simp add: sphere_def)
```

lemma *ball_trivial* [*simp*]: *ball x 0* = {}
by *auto*

lemma *cball_trivial* [*simp*]: *cball x 0* = {*x*}
by *auto*

lemma *sphere_trivial* [*simp*]: *sphere x 0* = {*x*}
by *auto*

lemma *disjoint_ballI*: *dist x y* ≥ *r*+*s* ⟹ *ball x r* ∩ *ball y s* = {}
using *dist_triangle_less_add not_le* **by** *fastforce*

lemma *disjoint_cballI*: *dist x y* > *r* + *s* ⟹ *cball x r* ∩ *cball y s* = {}
by (*metis add_mono disjoint_iff_not_equal dist_triangle2 dual_order.trans leD mem_cball*)

lemma *sphere_empty* [*simp*]: *r* < *0* ⟹ *sphere a r* = {}
for *a* :: *'a*::*metric_space*
by *auto*

lemma *centre_in_ball* [*simp*]: *x* ∈ *ball x e* ⟷ *0* < *e*
by *simp*

lemma *centre_in_cball* [*simp*]: *x* ∈ *cball x e* ⟷ *0* ≤ *e*
by *simp*

lemma *ball_subset_cball* [*simp*, *intro*]: *ball x e* ⊆ *cball x e*
by (*simp add*: *subset_eq*)

lemma *mem_ball_imp_mem_cball*: *x* ∈ *ball y e* ⟹ *x* ∈ *cball y e*
by *auto*

lemma *sphere_cball* [*simp*,*intro*]: *sphere z r* ⊆ *cball z r*
by *force*

lemma *cball_diff_sphere*: *cball a r* − *sphere a r* = *ball a r*
by *auto*

lemma *subset_ball*[*intro*]: *d* ≤ *e* ⟹ *ball x d* ⊆ *ball x e*
by *auto*

lemma *subset_cball*[*intro*]: *d* ≤ *e* ⟹ *cball x d* ⊆ *cball x e*
by *auto*

lemma *mem_ball_leI*: *x* ∈ *ball y e* ⟹ *e* ≤ *f* ⟹ *x* ∈ *ball y f*
by *auto*

lemma *mem_cball_leI*: *x* ∈ *cball y e* ⟹ *e* ≤ *f* ⟹ *x* ∈ *cball y f*

by *auto*

lemma *cball_trans*: $y \in cball\ z\ b \Longrightarrow x \in cball\ y\ a \Longrightarrow x \in cball\ z\ (b + a)$
by *metric*

lemma *ball_max_Un*: $ball\ a\ (max\ r\ s) = ball\ a\ r \cup ball\ a\ s$
by *auto*

lemma *ball_min_Int*: $ball\ a\ (min\ r\ s) = ball\ a\ r \cap ball\ a\ s$
by *auto*

lemma *cball_max_Un*: $cball\ a\ (max\ r\ s) = cball\ a\ r \cup cball\ a\ s$
by *auto*

lemma *cball_min_Int*: $cball\ a\ (min\ r\ s) = cball\ a\ r \cap cball\ a\ s$
by *auto*

lemma *cball_diff_eq_sphere*: $cball\ a\ r - ball\ a\ r = sphere\ a\ r$
by *auto*

lemma *open_ball* [*intro*, *simp*]: *open* (*ball x e*)
proof −
have *open* (*dist x* −‘ {..<*e*})
by (*intro open_vimage open_lessThan continuous_intros*)
also have *dist x* −‘ {..<*e*} = *ball x e*
by *auto*
finally show *?thesis* **.**
qed

lemma *open_contains_ball*: $open\ S \longleftrightarrow (\forall x{\in}S.\ \exists e{>}0.\ ball\ x\ e \subseteq S)$
by (*simp add*: *open_dist subset_eq Ball_def dist_commute*)

lemma *openI* [*intro?*]: $(\bigwedge x.\ x{\in}S \Longrightarrow \exists e{>}0.\ ball\ x\ e \subseteq S) \Longrightarrow open\ S$
by (*auto simp*: *open_contains_ball*)

lemma *openE*[*elim?*]:
assumes *open S x*∈*S*
obtains *e* **where** *e*>*0 ball x e* ⊆ *S*
using *assms* **unfolding** *open_contains_ball* **by** *auto*

lemma *open_contains_ball_eq*: $open\ S \Longrightarrow x{\in}S \longleftrightarrow (\exists e{>}0.\ ball\ x\ e \subseteq S)$
by (*metis open_contains_ball subset_eq centre_in_ball*)

lemma *ball_eq_empty*[*simp*]: $ball\ x\ e = \{\} \longleftrightarrow e \leq 0$
unfolding *mem_ball set_eq_iff*
by (*simp add*: *not_less*) *metric*

lemma *ball_empty*: $e \leq 0 \Longrightarrow ball\ x\ e = \{\}$
by *simp*

lemma *closed_cball* [*iff*]: *closed* (*cball x e*)
proof −
  have *closed* (*dist x* −‘ {..*e*})
    by (*intro closed_vimage closed_atMost continuous_intros*)
  also have *dist x* −‘ {..*e*} = *cball x e*
    by *auto*
  finally show *?thesis* **.**
qed

lemma *open_contains_cball*: *open S* ⟷ (∀ *x*∈*S*. ∃ *e*>*0*. *cball x e* ⊆ *S*)
proof −
  {
    fix *x* **and** *e*::*real*
    assume *x*∈*S e*>*0 ball x e* ⊆ *S*
    then have ∃ *d*>*0*. *cball x d* ⊆ *S*
      unfolding *subset_eq* **by** (*rule_tac x*=*e*/*2* **in** *exI*, *auto*)
  }
  moreover
  {
    fix *x* **and** *e*::*real*
    assume *x*∈*S e*>*0 cball x e* ⊆ *S*
    then have ∃ *d*>*0*. *ball x d* ⊆ *S*
      using *mem_ball_imp_mem_cball* **by** *blast*
  }
  ultimately show *?thesis*
    unfolding *open_contains_ball* **by** *auto*
qed

lemma *open_contains_cball_eq*: *open S* ⟹ (∀ *x*. *x* ∈ *S* ⟷ (∃ *e*>*0*. *cball x e* ⊆ *S*))
  by (*metis open_contains_cball subset_eq order_less_imp_le centre_in_cball*)

lemma *eventually_nhds_ball*: *d* > *0* ⟹ *eventually* (λ*x*. *x* ∈ *ball z d*) (*nhds z*)
  by (*rule eventually_nhds_in_open*) *simp_all*

lemma *eventually_at_ball*: *d* > *0* ⟹ *eventually* (λ*t*. *t* ∈ *ball z d* ∧ *t* ∈ *A*) (*at z within A*)
  unfolding *eventually_at* **by** (*intro exI*[*of* _ *d*]) (*simp_all add*: *dist_commute*)

lemma *eventually_at_ball'*: *d* > *0* ⟹ *eventually* (λ*t*. *t* ∈ *ball z d* ∧ *t* ≠ *z* ∧ *t* ∈ *A*) (*at z within A*)
  unfolding *eventually_at* **by** (*intro exI*[*of* _ *d*]) (*simp_all add*: *dist_commute*)

lemma *at_within_ball*: *e* > *0* ⟹ *dist x y* < *e* ⟹ *at y within ball x e* = *at y*
  by (*subst at_within_open*) *auto*

lemma *atLeastAtMost_eq_cball*:
  fixes *a b*::*real*

shows $\{a\ ..\ b\} = cball\ ((a + b)/2)\ ((b - a)/2)$
by (*auto simp*: *dist_real_def field_simps*)

lemma *cball_eq_atLeastAtMost*:
fixes *a b*::*real*
shows $cball\ a\ b = \{a - b\ ..\ a + b\}$
by (*auto simp*: *dist_real_def*)

lemma *greaterThanLessThan_eq_ball*:
fixes *a b*::*real*
shows $\{a <..< b\} = ball\ ((a + b)/2)\ ((b - a)/2)$
by (*auto simp*: *dist_real_def field_simps*)

lemma *ball_eq_greaterThanLessThan*:
fixes *a b*::*real*
shows $ball\ a\ b = \{a - b <..< a + b\}$
by (*auto simp*: *dist_real_def*)

lemma *interior_ball* [*simp*]: $interior\ (ball\ x\ e) = ball\ x\ e$
by (*simp add*: *interior_open*)

lemma *cball_eq_empty* [*simp*]: $cball\ x\ e = \{\} \longleftrightarrow e < 0$
by (*smt* (*verit*, *best*) *Diff_empty ball_eq_empty cball_diff_sphere centre_in_ball centre_in_cball sphere_empty*)

lemma *cball_empty* [*simp*]: $e < 0 \Longrightarrow cball\ x\ e = \{\}$
by *simp*

lemma *cball_sing*:
fixes $x :: 'a$::*metric_space*
shows $e = 0 \Longrightarrow cball\ x\ e = \{x\}$
by *simp*

lemma *ball_divide_subset*: $d \geq 1 \Longrightarrow ball\ x\ (e/d) \subseteq ball\ x\ e$
by (*metis ball_eq_empty div_by_1 frac_le linear subset_ball zero_less_one*)

lemma *ball_divide_subset_numeral*: $ball\ x\ (e\ /\ numeral\ w) \subseteq ball\ x\ e$
using *ball_divide_subset one_le_numeral* **by** *blast*

lemma *cball_divide_subset*: $d \geq 1 \Longrightarrow cball\ x\ (e/d) \subseteq cball\ x\ e$
by (*smt* (*verit*, *best*) *cball_empty div_by_1 frac_le subset_cball zero_le_divide_iff*)

lemma *cball_divide_subset_numeral*: $cball\ x\ (e\ /\ numeral\ w) \subseteq cball\ x\ e$
using *cball_divide_subset one_le_numeral* **by** *blast*

lemma *cball_scale*:
assumes $a \neq 0$
shows $(\lambda x.\ a *_R x)\ `\ cball\ c\ r = cball\ (a *_R c :: 'a :: real\_normed\_vector)\ (|a| * r)$

```
proof −
  have 1: (λx. a *R x) ‘ cball c r ⊆ cball (a *R c) (|a| * r) if a ≠ 0 for a r and
c :: 'a
  proof safe
    fix x
    assume x: x ∈ cball c r
    have dist (a *R c) (a *R x) = norm (a *R c − a *R x)
      by (auto simp: dist_norm)
    also have a *R c − a *R x = a *R (c − x)
      by (simp add: algebra_simps)
    finally show a *R x ∈ cball (a *R c) (|a| * r)
      using that x by (auto simp: dist_norm)
  qed

  have cball (a *R c) (|a| * r) = (λx. a *R x) ‘ (λx. inverse a *R x) ‘ cball (a *R
c) (|a| * r)
    unfolding image_image using assms by simp
  also have ... ⊆ (λx. a *R x) ‘ cball (inverse a *R (a *R c)) (|inverse a| * (|a|
* r))
    using assms by (intro image_mono 1) auto
  also have ... = (λx. a *R x) ‘ cball c r
    using assms by (simp add: algebra_simps)
  finally have cball (a *R c) (|a| * r) ⊆ (λx. a *R x) ‘ cball c r .
  moreover from assms have (λx. a *R x) ‘ cball c r ⊆ cball (a *R c) (|a| * r)
    by (intro 1) auto
  ultimately show ?thesis by blast
qed

lemma ball_scale:
  assumes a ≠ 0
  shows   (λx. a *R x) ‘ ball c r = ball (a *R c :: 'a :: real_normed_vector) (|a|
* r)
proof −
  have 1: (λx. a *R x) ‘ ball c r ⊆ ball (a *R c) (|a| * r) if a ≠ 0 for a r and c
:: 'a
  proof safe
    fix x
    assume x: x ∈ ball c r
    have dist (a *R c) (a *R x) = norm (a *R c − a *R x)
      by (auto simp: dist_norm)
    also have a *R c − a *R x = a *R (c − x)
      by (simp add: algebra_simps)
    finally show a *R x ∈ ball (a *R c) (|a| * r)
      using that x by (auto simp: dist_norm)
  qed

  have ball (a *R c) (|a| * r) = (λx. a *R x) ‘ (λx. inverse a *R x) ‘ ball (a *R c)
(|a| * r)
    unfolding image_image using assms by simp
```

```
  also have ... ⊆ (λx. a *R x) ‘ ball (inverse a *R (a *R c)) (|inverse a| * (|a| *
r))
    using assms by (intro image_mono 1) auto
  also have ... = (λx. a *R x) ‘ ball c r
    using assms by (simp add: algebra_simps)
  finally have ball (a *R c) (|a| * r) ⊆ (λx. a *R x) ‘ ball c r .
  moreover from assms have (λx. a *R x) ‘ ball c r ⊆ ball (a *R c) (|a| * r)
    by (intro 1) auto
  ultimately show ?thesis by blast
qed

lemma frequently_atE:
  fixes x :: 'a :: metric_space
  assumes frequently P (at x within s)
  shows   ∃f. filterlim f (at x within s) sequentially ∧ (∀n. P (f n))
proof −
  have ∃y. y ∈ s ∩ (ball x (1 / real (Suc n)) − {x}) ∧ P y for n
  proof −
    have ∃z∈s. z ≠ x ∧ dist z x < (1 / real (Suc n)) ∧ P z
    by (metis assms divide_pos_pos frequently_at of_nat_0_less_iff zero_less_Suc
zero_less_one)
    then show ?thesis
      by (auto simp: dist_commute conj_commute)
  qed
  then obtain f where f: ⋀n. f n ∈ s ∩ (ball x (1 / real (Suc n)) − {x}) ∧ P (f
n)
    by metis
  have filterlim f (nhds x) sequentially
    unfolding tendsto_iff
  proof clarify
    fix e :: real
    assume e: e > 0
    then obtain n where n: Suc n > 1 / e
      by (meson le_nat_floor lessI not_le)
    have dist (f k) x < e if k ≥ n for k
    proof −
      have dist (f k) x < 1 / real (Suc k)
        using f[of k] by (auto simp: dist_commute)
      also have ... ≤ 1 / real (Suc n)
        using that by (intro divide_left_mono) auto
      also have ... < e
        using n e by (simp add: field_simps)
      finally show ?thesis .
    qed
    thus ∀F k in sequentially. dist (f k) x < e
      unfolding eventually_at_top_linorder by blast
  qed
  moreover have f n ≠ x for n
    using f[of n] by auto
```

ultimately have *filterlim f* (*at x within s*) *sequentially*
using *f* **by** (*auto simp*: *filterlim_at*)
with *f* **show** *?thesis*
by *blast*
qed

3.2.2 Limit Points

lemma *islimpt_approachable*:
fixes *x* :: *'a*::*metric_space*
shows *x islimpt S* ⟷ (∀ *e*>*0*. ∃ *x'*∈*S*. *x'* ≠ *x* ∧ *dist x' x* < *e*)
unfolding *islimpt_iff_eventually eventually_at* **by** *fast*

lemma *islimpt_approachable_le*: *x islimpt S* ⟷ (∀ *e*>*0*. ∃ *x'*∈ *S*. *x'* ≠ *x* ∧ *dist x' x* ≤ *e*)
for *x* :: *'a*::*metric_space*
unfolding *islimpt_approachable*
using *approachable_lt_le2* [**where** *f*=λ*y*. *dist y x* **and** *P*=λ*y*. *y* ∉ *S* ∨ *y* = *x* **and** *Q*=λ*x*. *True*]
by *auto*

lemma *limpt_of_limpts*: *x islimpt* {*y*. *y islimpt S*} ⟹ *x islimpt S*
for *x* :: *'a*::*metric_space*
by (*metis islimpt_def islimpt_eq_acc_point mem_Collect_eq*)

lemma *closed_limpts*: *closed* {*x*::*'a*::*metric_space*. *x islimpt S*}
using *closed_limpt limpt_of_limpts* **by** *blast*

lemma *limpt_of_closure*: *x islimpt closure S* ⟷ *x islimpt S*
for *x* :: *'a*::*metric_space*
by (*auto simp*: *closure_def islimpt_Un dest*: *limpt_of_limpts*)

lemma *islimpt_eq_infinite_ball*: *x islimpt S* ⟷ (∀ *e*>*0*. *infinite*(*S* ∩ *ball x e*))
unfolding *islimpt_eq_acc_point*
by (*metis open_ball Int_commute Int_mono finite_subset open_contains_ball_eq subset_eq*)

lemma *islimpt_eq_infinite_cball*: *x islimpt S* ⟷ (∀ *e*>*0*. *infinite*(*S* ∩ *cball x e*))
unfolding *islimpt_eq_infinite_ball*
by (*metis open_ball ball_subset_cball centre_in_ball finite_Int inf.absorb_iff2 inf_assoc open_contains_cball_eq*)

3.2.3 Perfect Metric Spaces

lemma *perfect_choose_dist*: *0* < *r* ⟹ ∃ *a*. *a* ≠ *x* ∧ *dist a x* < *r*
for *x* :: *'a*::{*perfect_space*,*metric_space*}
using *islimpt_UNIV* [*of x*] **by** (*simp add*: *islimpt_approachable*)

lemma *cball_eq_sing*:
fixes *x* :: *'a*::{*metric_space*,*perfect_space*}

shows *cball x e* = {*x*} ⟷ *e* = *0*
by (*smt* (*verit*, *best*) *open_ball ball_eq_empty ball_subset_cball cball_empty cball_trivial*
not_open_singleton subset_singleton_iff)

3.2.4 Finite and discrete

lemma *finite_ball_include*:
fixes *a* :: *'a*::*metric_space*
assumes *finite S*
shows ∃ *e*>*0*. *S* ⊆ *ball a e*
using *assms*
proof *induction*
case (*insert x S*)
then obtain *e0* **where** *e0*>*0* **and** *e0*:*S* ⊆ *ball a e0* **by** *auto*
define *e* **where** *e* = *max e0* (*2* ∗ *dist a x*)
have *e*>*0* **unfolding** *e_def* **using** ‹*e0*>*0*› **by** *auto*
moreover have *insert x S* ⊆ *ball a e*
using *e0* ‹*e*>*0*› **unfolding** *e_def* **by** *auto*
ultimately show *?case* **by** *auto*
qed (*auto intro*: *zero_less_one*)

lemma *finite_set_avoid*:
fixes *a* :: *'a*::*metric_space*
assumes *finite S*
shows ∃ *d*>*0*. ∀ *x*∈*S*. *x* ≠ *a* ⟶ *d* ≤ *dist a x*
using *assms*
proof *induction*
case (*insert x S*)
then obtain *d* **where** *d* > *0* **and** *d*: ∀ *x*∈*S*. *x* ≠ *a* ⟶ *d* ≤ *dist a x*
by *blast*
show *?case*
by (*smt* (*verit*, *del_insts*) *dist_pos_lt insert.IH insert_iff*)
qed (*auto intro*: *zero_less_one*)

lemma *discrete_imp_closed*:
fixes *S* :: *'a*::*metric_space set*
assumes *e*: *0* < *e*
and *d*: ∀ *x* ∈ *S*. ∀ *y* ∈ *S*. *dist y x* < *e* ⟶ *y* = *x*
shows *closed S*
proof −
have *False* **if** *C*: ⋀*e*. *e*>*0* ⟹ ∃ *x'*∈*S*. *x'* ≠ *x* ∧ *dist x' x* < *e* **for** *x*
proof −
from *e* **have** *e2*: *e*/*2* > *0* **by** *arith*
from *C*[*OF e2*] **obtain** *y* **where** *y*: *y* ∈ *S y* ≠ *x dist y x* < *e*/*2*
by *blast*
from *e2 y*(*2*) **have** *mp*: *min* (*e*/*2*) (*dist x y*) > *0*
by *simp*
from *d y C*[*OF mp*] **show** *?thesis*

by *metric*
qed
then show *?thesis*
by (*metis islimpt_approachable closed_limpt* [**where** *'a='a*])
qed

lemma *discrete_imp_not_islimpt*:
assumes *e*: *0 < e*
and *d*: $\bigwedge x\ y.\ x \in S \Longrightarrow y \in S \Longrightarrow dist\ y\ x < e \Longrightarrow y = x$
shows $\neg$ *x islimpt S*
proof
assume *x islimpt S*
hence *x islimpt S − {x}*
by (*meson islimpt_punctured*)
moreover from *assms* **have** *closed* (*S − {x}*)
by (*intro discrete_imp_closed*) *auto*
ultimately show *False*
unfolding *closed_limpt* **by** *blast*
qed

3.2.5 Interior

lemma *mem_interior*: $x \in interior\ S \longleftrightarrow (\exists\, e{>}0.\ ball\ x\ e \subseteq S)$
using *open_contains_ball_eq* [**where** *S=interior S*]
by (*simp add*: *open_subset_interior*)

lemma *mem_interior_cball*: $x \in interior\ S \longleftrightarrow (\exists\, e{>}0.\ cball\ x\ e \subseteq S)$
by (*meson ball_subset_cball interior_subset mem_interior open_contains_cball open_interior*
subset_trans)

lemma *ball_iff_cball*: $(\exists\, r{>}0.\ ball\ x\ r \subseteq U) \longleftrightarrow (\exists\, r{>}0.\ cball\ x\ r \subseteq U)$
by (*meson mem_interior mem_interior_cball*)

3.2.6 Frontier

lemma *frontier_straddle*:
fixes *a* :: *'a*::*metric_space*
shows $a \in frontier\ S \longleftrightarrow (\forall\, e{>}0.\ (\exists\, x{\in}S.\ dist\ a\ x < e) \wedge (\exists\, x.\ x \notin S \wedge dist\ a\ x < e))$
unfolding *frontier_def closure_interior*
by (*auto simp*: *mem_interior subset_eq ball_def*)

3.2.7 Limits

proposition *Lim*: $(f \longrightarrow l)\ net \longleftrightarrow trivial\_limit\ net \vee (\forall\, e{>}0.\ eventually\ (\lambda x.\ dist\ (f\ x)\ l < e)\ net)$
by (*auto simp*: *tendsto_iff trivial_limit_eq*)

Show that they yield usual definitions in the various cases.

proposition *Lim_within_le*: $(f \longrightarrow l)(at\ a\ within\ S) \longleftrightarrow$
$(\forall e{>}0.\ \exists d{>}0.\ \forall x{\in}S.\ 0 < dist\ x\ a \wedge dist\ x\ a \leq d \longrightarrow dist\ (f\ x)\ l < e)$
by (*auto simp*: *tendsto_iff eventually_at_le*)

proposition *Lim_within*: $(f \longrightarrow l)\ (at\ a\ within\ S) \longleftrightarrow$
$(\forall e{>}0.\ \exists d{>}0.\ \forall x \in S.\ 0 < dist\ x\ a \wedge dist\ x\ a < d \longrightarrow dist\ (f\ x)\ l < e)$
by (*auto simp*: *tendsto_iff eventually_at*)

corollary *Lim_withinI* [*intro?*]:
assumes $\bigwedge e.\ e > 0 \Longrightarrow \exists d{>}0.\ \forall x \in S.\ 0 < dist\ x\ a \wedge dist\ x\ a < d \longrightarrow dist\ (f\ x)\ l \leq e$
shows $(f \longrightarrow l)\ (at\ a\ within\ S)$
unfolding *Lim_within* **by** (*smt* (*verit*, *ccfv_SIG*) *assms zero_less_dist_iff*)

proposition *Lim_at*: $(f \longrightarrow l)\ (at\ a) \longleftrightarrow$
$(\forall e{>}0.\ \exists d{>}0.\ \forall x.\ 0 < dist\ x\ a \wedge dist\ x\ a < d \longrightarrow dist\ (f\ x)\ l < e)$
by (*auto simp*: *tendsto_iff eventually_at*)

lemma *Lim_transform_within_set*:
fixes $a :: 'a{::}metric\_space$ **and** $l :: 'b{::}metric\_space$
shows $[\![(f \longrightarrow l)\ (at\ a\ within\ S);\ eventually\ (\lambda x.\ x \in S \longleftrightarrow x \in T)\ (at\ a)]\!]$
$\Longrightarrow (f \longrightarrow l)\ (at\ a\ within\ T)$
by (*simp add*: *eventually_at Lim_within*) (*smt* (*verit*, *best*))

Another limit point characterization.

lemma *limpt_sequential_inj*:
fixes $x :: 'a{::}metric\_space$
shows $x\ islimpt\ S \longleftrightarrow$
$(\exists f.\ (\forall n{::}nat.\ f\ n \in S - \{x\}) \wedge inj\ f \wedge (f \longrightarrow x)\ sequentially)$
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then have $\forall e{>}0.\ \exists x'{\in}S.\ x' \neq x \wedge dist\ x'\ x < e$
by (*force simp*: *islimpt_approachable*)
then obtain *y* **where** *y*: $\bigwedge e.\ e{>}0 \Longrightarrow y\ e \in S \wedge y\ e \neq x \wedge dist\ (y\ e)\ x < e$
by *metis*
define *f* **where** $f \equiv rec\_nat\ (y\ 1)\ (\lambda n\ fn.\ y\ (min\ (inverse(2\ \widehat{}\ (Suc\ n)))\ (dist\ fn\ x)))$
have [*simp*]: $f\ 0 = y\ 1$
and *fSuc*: $f(Suc\ n) = y\ (min\ (inverse(2\ \widehat{}\ (Suc\ n)))\ (dist\ (f\ n)\ x))$ **for** *n*
by (*simp_all add*: *f_def*)
have *f*: $f\ n \in S \wedge (f\ n \neq x) \wedge dist\ (f\ n)\ x < inverse(2\ \widehat{}\ n)$ **for** *n*
proof (*induction n*)
case *0* **show** *?case*
by (*simp add*: *y*)
next
case (*Suc n*) **then show** *?case*
by (*smt* (*verit*, *best*) *fSuc dist_pos_lt inverse_positive_iff_positive y zero_less_power*)
qed

```
  show ?rhs
  proof (intro exI conjI allI)
    show ⋀n. f n ∈ S − {x}
      using f by blast
    have dist (f n) x < dist (f m) x if m < n for m n
    using that
    proof (induction n)
      case 0 then show ?case by simp
    next
      case (Suc n)
      then consider m < n | m = n using less_Suc_eq by blast
      then show ?case
      proof cases
        assume m < n
       have dist (f(Suc n)) x = dist (y (min (inverse(2 ^(Suc n))) (dist (f n) x)))
x
          by (simp add: fSuc)
        also have ... < dist (f n) x
          by (metis dist_pos_lt f min.strict_order_iff min_less_iff_conj y)
        also have ... < dist (f m) x
          using Suc.IH ‹m < n› by blast
        finally show ?thesis .
      next
        assume m = n then show ?case
          by (smt (verit, best) dist_pos_lt f fSuc y)
      qed
    qed
    then show inj f
      by (metis less_irrefl linorder_injI)
    have ⋀e n. ⟦0 < e; nat ⌈1 / e⌉ ≤ n⟧ ⟹ dist (f n) x < e
      apply (rule less_trans [OF f [THEN conjunct2, THEN conjunct2]])
      by (simp add: divide_simps order_le_less_trans)
    then show f ⟶ x
      using lim_sequentially by blast
  qed
next
  assume ?rhs
  then show ?lhs
    by (fastforce simp add: islimpt_approachable lim_sequentially)
qed

lemma Lim_dist_ubound:
  assumes ¬(trivial_limit net)
    and (f ⟶ l) net
    and eventually (λx. dist a (f x) ≤ e) net
  shows dist a l ≤ e
  using assms by (fast intro: tendsto_le tendsto_intros)
```

3.2.8 Continuity

Derive the epsilon-delta forms, which we often use as "definitions"

proposition *continuous_within_eps_delta*:
continuous (*at x within s*) *f* ⟷ (∀ *e*>*0*. ∃ *d*>*0*. ∀ *x′*∈ *s*. *dist x′ x* < *d* −−> *dist* (*f x′*) (*f x*) < *e*)
unfolding *continuous_within* **and** *Lim_within* **by** *fastforce*

corollary *continuous_at_eps_delta*:
continuous (*at x*) *f* ⟷ (∀ *e* > *0*. ∃ *d* > *0*. ∀ *x′*. *dist x′ x* < *d* ⟶ *dist* (*f x′*) (*f x*) < *e*)
using *continuous_within_eps_delta* [*of x UNIV f*] **by** *simp*

lemma *continuous_at_right_real_increasing*:
fixes *f* :: *real* ⇒ *real*
assumes *nondecF*: ⋀*x y*. *x* ≤ *y* ⟹ *f x* ≤ *f y*
shows *continuous* (*at_right a*) *f* ⟷ (∀ *e*>*0*. ∃ *d*>*0*. *f* (*a* + *d*) − *f a* < *e*)
apply (*simp add*: *greaterThan_def dist_real_def continuous_within Lim_within_le*)
apply (*intro all_cong ex_cong*)
by (*smt* (*verit*, *best*) *nondecF*)

lemma *continuous_at_left_real_increasing*:
assumes *nondecF*: ⋀ *x y*. *x* ≤ *y* ⟹ *f x* ≤ ((*f y*) :: *real*)
shows (*continuous* (*at_left* (*a* :: *real*)) *f*) ⟷ (∀ *e* > *0*. ∃ *delta* > *0*. *f a* − *f* (*a* − *delta*) < *e*)
apply (*simp add*: *lessThan_def dist_real_def continuous_within Lim_within_le*)
apply (*intro all_cong ex_cong*)
by (*smt* (*verit*) *nondecF*)

Versions in terms of open balls.

lemma *continuous_within_ball*:
continuous (*at x within S*) *f* ⟷
(∀ *e* > *0*. ∃ *d* > *0*. *f* ‘ (*ball x d* ∩ *S*) ⊆ *ball* (*f x*) *e*)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
{
fix *e* :: *real*
assume *e* > *0*
then obtain *d* **where** *d*>*0* **and** *d*: ∀ *y*∈*S*. *0* < *dist y x* ∧ *dist y x* < *d* ⟶ *dist* (*f y*) (*f x*) < *e*
using ‹*?lhs*›[*unfolded continuous_within Lim_within*] **by** *auto*
{ fix *y*
assume *y* ∈ *f* ‘ (*ball x d* ∩ *S*) **then have** *y* ∈ *ball* (*f x*) *e*
using *d* ‹*e* > *0*› **by** (*auto simp*: *dist_commute*)
}
then have ∃ *d*>*0*. *f* ‘ (*ball x d* ∩ *S*) ⊆ *ball* (*f x*) *e*
using ‹*d* > *0*› **by** *blast*
}

then show *?rhs* **by** *auto*
next
assume *?rhs*
then show *?lhs*
apply (*simp add*: *continuous_within Lim_within ball_def subset_eq*)
by (*metis* (*mono_tags*, *lifting*) *Int_iff dist_commute mem_Collect_eq*)
qed

lemma *continuous_at_ball*:
continuous (*at x*) *f* ⟷ (∀ *e*>0. ∃ *d*>0. *f* ‘ (*ball x d*) ⊆ *ball* (*f x*) *e*)
apply (*simp add*: *continuous_at Lim_at subset_eq Ball_def Bex_def image_iff*)
by (*smt* (*verit*, *ccfv_threshold*) *dist_commute dist_self*)

Define setwise continuity in terms of limits within the set.

lemma *continuous_on_iff*:
continuous_on s f ⟷
(∀ *x*∈*s*. ∀ *e*>0. ∃ *d*>0. ∀ *x*′∈*s*. *dist x*′ *x* < *d* ⟶ *dist* (*f x*′) (*f x*) < *e*)
unfolding *continuous_on_def Lim_within*
by (*metis dist_pos_lt dist_self*)

lemma *continuous_within_E*:
assumes *continuous* (*at x within S*) *f e*>0
obtains *d* **where** *d*>0 ⋀*x*′. ⟦*x*′∈ *S*; *dist x*′ *x* ≤ *d*⟧ ⟹ *dist* (*f x*′) (*f x*) < *e*
using *assms* **unfolding** *continuous_within_eps_delta*
by (*metis dense order_le_less_trans*)

lemma *continuous_onI* [*intro?*]:
assumes ⋀*x e*. ⟦*e* > 0; *x* ∈ *S*⟧ ⟹ ∃ *d*>0. ∀ *x*′∈*S*. *dist x*′ *x* < *d* ⟶ *dist* (*f x*′) (*f x*) ≤ *e*
shows *continuous_on S f*
apply (*simp add*: *continuous_on_iff*, *clarify*)
apply (*rule ex_forward* [*OF assms* [*OF half_gt_zero*]], *auto*)
done

Some simple consequential lemmas.

lemma *continuous_onE*:
assumes *continuous_on s f x*∈*s e*>0
obtains *d* **where** *d*>0 ⋀*x*′. ⟦*x*′ ∈ *s*; *dist x*′ *x* ≤ *d*⟧ ⟹ *dist* (*f x*′) (*f x*) < *e*
using *assms*
unfolding *continuous_on_iff* **by** (*metis dense order_le_less_trans*)

The usual transformation theorems.

lemma *continuous_transform_within*:
fixes *f g* :: *'a*::*metric_space* ⇒ *'b*::*topological_space*
assumes *continuous* (*at x within s*) *f*
and *0* < *d*
and *x* ∈ *s*
and ⋀*x*′. ⟦*x*′ ∈ *s*; *dist x*′ *x* < *d*⟧ ⟹ *f x*′ = *g x*′
shows *continuous* (*at x within s*) *g*

using *assms*
unfolding *continuous_within* **by** (*force intro*: *Lim_transform_within*)

3.2.9 Closure and Limit Characterization

lemma *closure_approachable*:
fixes S :: *'a*::*metric_space set*
shows $x \in$ *closure* $S \longleftrightarrow (\forall\, e{>}0.\ \exists\, y{\in}S.\ dist\ y\ x < e)$
using *dist_self* **by** (*force simp*: *closure_def islimpt_approachable*)

lemma *closure_approachable_le*:
fixes S :: *'a*::*metric_space set*
shows $x \in$ *closure* $S \longleftrightarrow (\forall\, e{>}0.\ \exists\, y{\in}S.\ dist\ y\ x \leq e)$
unfolding *closure_approachable*
using *dense* **by** *force*

lemma *closure_approachableD*:
assumes $x \in$ *closure* S $e{>}0$
shows $\exists\, y{\in}S.\ dist\ x\ y < e$
using *assms* **unfolding** *closure_approachable* **by** (*auto simp*: *dist_commute*)

lemma *closed_approachable*:
fixes S :: *'a*::*metric_space set*
shows *closed* $S \Longrightarrow (\forall\, e{>}0.\ \exists\, y{\in}S.\ dist\ y\ x < e) \longleftrightarrow x \in S$
by (*metis closure_closed closure_approachable*)

lemma *closure_contains_Inf*:
fixes S :: *real set*
assumes $S \neq \{\}$ *bdd_below* S
shows *Inf* $S \in$ *closure* S
proof −
have $*$: $\forall\, x{\in}S.\ Inf\ S \leq x$
using *cInf_lower*[*of* _ S] *assms* **by** *metis*
{ fix e :: *real*
assume $e > 0$
then have *Inf* $S <$ *Inf* $S + e$ **by** *simp*
with *assms* **obtain** x **where** $x \in S$ $x <$ *Inf* $S + e$
using *cInf_lessD* **by** *blast*
with $*$ **have** $\exists\, x{\in}S.\ dist\ x\ (Inf\ S) < e$
using *dist_real_def* **by** *force*
}
then show *?thesis* **unfolding** *closure_approachable* **by** *auto*
qed

lemma *closure_contains_Sup*:
fixes S :: *real set*
assumes $S \neq \{\}$ *bdd_above* S
shows *Sup* $S \in$ *closure* S
proof −

```
  have ∗: ∀ x∈S. x ≤ Sup S
    using cSup_upper[of _ S] assms by metis
  {
    fix e :: real
    assume e > 0
    then have Sup S − e < Sup S by simp
    with assms obtain x where x ∈ S Sup S − e < x
      using less_cSupE by blast
    with ∗ have ∃ x∈S. dist x (Sup S) < e
      using dist_real_def by force
  }
  then show ?thesis unfolding closure_approachable by auto
qed

lemma not_trivial_limit_within_ball:
  ¬ trivial_limit (at x within S) ⟷ (∀ e>0. S ∩ ball x e − {x} ≠ {})
  (is ?lhs ⟷ ?rhs)
proof
  show ?rhs if ?lhs
  proof −
    { fix e :: real
      assume e > 0
      then obtain y where y ∈ S − {x} and dist y x < e
        using ‹?lhs› not_trivial_limit_within[of x S] closure_approachable[of x S
− {x}]
        by auto
      then have y ∈ S ∩ ball x e − {x}
        unfolding ball_def by (simp add: dist_commute)
      then have S ∩ ball x e − {x} ≠ {} by blast
    }
    then show ?thesis by auto
  qed
  show ?lhs if ?rhs
  proof −
    { fix e :: real
      assume e > 0
      then obtain y where y ∈ S ∩ ball x e − {x}
        using ‹?rhs› by blast
      then have y ∈ S − {x} and dist y x < e
        unfolding ball_def by (simp_all add: dist_commute)
      then have ∃ y ∈ S − {x}. dist y x < e
        by auto
    }
    then show ?thesis
      using not_trivial_limit_within[of x S] closure_approachable[of x S − {x}]
      by auto
  qed
qed
```

3.2.10 Boundedness

definition (**in** *metric_space*) *bounded* :: *'a set* ⇒ *bool*
where *bounded S* ⟷ (∃ *x e*. ∀ *y*∈*S*. *dist x y* ≤ *e*)

lemma *bounded_subset_cball*: *bounded S* ⟷ (∃ *e x*. *S* ⊆ *cball x e* ∧ *0* ≤ *e*)
unfolding *bounded_def subset_eq* **by** *auto* (*meson order_trans zero_le_dist*)

lemma *bounded_any_center*: *bounded S* ⟷ (∃ *e*. ∀ *y*∈*S*. *dist a y* ≤ *e*)
unfolding *bounded_def*
by *auto* (*metis add.commute add_le_cancel_right dist_commute dist_triangle_le*)

lemma *bounded_iff*: *bounded S* ⟷ (∃ *a*. ∀ *x*∈*S*. *norm x* ≤ *a*)
unfolding *bounded_any_center* [**where** *a=0*]
by (*simp add*: *dist_norm*)

lemma *bdd_above_norm*: *bdd_above* (*norm* ‘ *X*) ⟷ *bounded X*
by (*simp add*: *bounded_iff bdd_above_def*)

lemma *bounded_norm_comp*: *bounded* ((λ*x*. *norm* (*f x*)) ‘ *S*) = *bounded* (*f* ‘ *S*)
by (*simp add*: *bounded_iff*)

lemma *boundedI*:
assumes ⋀*x*. *x* ∈ *S* ⟹ *norm x* ≤ *B*
shows *bounded S*
using *assms bounded_iff* **by** *blast*

lemma *bounded_empty* [*simp*]: *bounded* {}
by (*simp add*: *bounded_def*)

lemma *bounded_subset*: *bounded T* ⟹ *S* ⊆ *T* ⟹ *bounded S*
by (*metis bounded_def subset_eq*)

lemma *bounded_interior*[*intro*]: *bounded S* ⟹ *bounded*(*interior S*)
by (*metis bounded_subset interior_subset*)

lemma *bounded_closure*[*intro*]:
assumes *bounded S*
shows *bounded* (*closure S*)
proof −
from *assms* **obtain** *x* **and** *a* **where** *a*: ∀ *y*∈*S*. *dist x y* ≤ *a*
unfolding *bounded_def* **by** *auto*
{ fix *y*
assume *y* ∈ *closure S*
then obtain *f* **where** *f*: ∀ *n*. *f n* ∈ *S* (*f* ⟶ *y*) *sequentially*
unfolding *closure_sequential* **by** *auto*
have ∀ *n*. *f n* ∈ *S* ⟶ *dist x* (*f n*) ≤ *a* **using** *a* **by** *simp*
then have *eventually* (λ*n*. *dist x* (*f n*) ≤ *a*) *sequentially*
by (*simp add*: *f*(*1*))
then have *dist x y* ≤ *a*

```
    using Lim_dist_ubound f(2) trivial_limit_sequentially by blast
  }
  then show ?thesis
    unfolding bounded_def by auto
qed

lemma bounded_closure_image: bounded (f ‘ closure S) ⟹ bounded (f ‘ S)
  by (simp add: bounded_subset closure_subset image_mono)

lemma bounded_cball[simp,intro]: bounded (cball x e)
  unfolding bounded_def  using mem_cball by blast

lemma bounded_ball[simp,intro]: bounded (ball x e)
  by (metis ball_subset_cball bounded_cball bounded_subset)

lemma bounded_Un[simp]: bounded (S ∪ T) ⟷ bounded S ∧ bounded T
 by (auto simp: bounded_def) (metis Un_iff bounded_any_center le_max_iff_disj)

lemma bounded_Union[intro]: finite F ⟹ ∀ S∈F. bounded S ⟹ bounded (⋃ F)
  by (induct rule: finite_induct[of F]) auto

lemma bounded_UN [intro]: finite A ⟹ ∀ x∈A. bounded (B x) ⟹ bounded
(⋃ x∈A. B x)
  by auto

lemma bounded_insert [simp]: bounded (insert x S) ⟷ bounded S
proof −
  have ∀ y∈{x}. dist x y ≤ 0
    by simp
  then have bounded {x}
    unfolding bounded_def by fast
  then show ?thesis
    by (metis insert_is_Un bounded_Un)
qed

lemma bounded_subset_ballI: S ⊆ ball x r ⟹ bounded S
  by (meson bounded_ball bounded_subset)

lemma bounded_subset_ballD:
  assumes bounded S shows ∃ r. 0 < r ∧ S ⊆ ball x r
proof −
  obtain e::real and y where S ⊆ cball y e 0 ≤ e
    using assms by (auto simp: bounded_subset_cball)
  then show ?thesis
    by (intro exI[where x=dist x y + e + 1]) metric
qed

lemma finite_imp_bounded [intro]: finite S ⟹ bounded S
  by (induct set: finite) simp_all
```

lemma *bounded_Int*[*intro*]: *bounded S* ∨ *bounded T* ⟹ *bounded* (*S* ∩ *T*)
by (*metis Int_lower1 Int_lower2 bounded_subset*)

lemma *bounded_diff*[*intro*]: *bounded S* ⟹ *bounded* (*S* − *T*)
by (*metis Diff_subset bounded_subset*)

lemma *bounded_dist_comp*:
assumes *bounded* (*f* ‘ *S*) *bounded* (*g* ‘ *S*)
shows *bounded* ((λ*x*. *dist* (*f x*) (*g x*)) ‘ *S*)
proof −
from *assms* **obtain** *M1 M2* **where** ∗: *dist* (*f x*) *undefined* ≤ *M1 dist undefined* (*g x*) ≤ *M2* **if** *x* ∈ *S* **for** *x*
by (*auto simp*: *bounded_any_center*[*of* _ *undefined*] *dist_commute*)
have *dist* (*f x*) (*g x*) ≤ *M1* + *M2* **if** *x* ∈ *S* **for** *x*
using ∗[*OF that*]
by *metric*
then show *?thesis*
by (*auto intro*!: *boundedI*)
qed

lemma *bounded_Times*:
assumes *bounded s bounded t*
shows *bounded* (*s* × *t*)
proof −
obtain *x y a b* **where** ∀ *z*∈*s*. *dist x z* ≤ *a* ∀ *z*∈*t*. *dist y z* ≤ *b*
using *assms* [*unfolded bounded_def*] **by** *auto*
then have ∀ *z*∈*s* × *t*. *dist* (*x*, *y*) *z* ≤ *sqrt* (*a*^{2} + *b*^{2})
by (*auto simp*: *dist_Pair_Pair real_sqrt_le_mono add_mono power_mono*)
then show *?thesis* **unfolding** *bounded_any_center* [**where** *a*=(*x*, *y*)] **by** *auto*
qed

3.2.11 Compactness

lemma *compact_imp_bounded*:
assumes *compact U*
shows *bounded U*
proof −
have *compact U* ∀ *x*∈*U*. *open* (*ball x 1*) *U* ⊆ (⋃ *x*∈*U*. *ball x 1*)
using *assms* **by** *auto*
then obtain *D* **where** *D*: *D* ⊆ *U finite D U* ⊆ (⋃ *x*∈*D*. *ball x 1*)
by (*metis compactE_image*)
from ‹*finite D*› **have** *bounded* (⋃ *x*∈*D*. *ball x 1*)
by (*simp add*: *bounded_UN*)
then show *bounded U* **using** ‹*U* ⊆ (⋃ *x*∈*D*. *ball x 1*)›
by (*rule bounded_subset*)
qed

lemma *continuous_on_compact_bound*:

assumes *compact A continuous_on A f*
obtains *B* **where** $B \geq 0$ $\bigwedge x.\ x \in A \Longrightarrow norm\ (f\ x) \leq B$
proof −
have *compact* (*f* ‘ *A*) **by** (*metis assms compact_continuous_image*)
then obtain *B* **where** $\forall\, x{\in}A.\ norm\ (f\ x) \leq B$
by (*auto dest*!: *compact_imp_bounded simp*: *bounded_iff*)
hence *max B 0* ≥ 0 **and** $\forall\, x{\in}A.\ norm\ (f\ x) \leq max\ B\ 0$ **by** *auto*
thus *?thesis* **using** *that* **by** *blast*
qed

lemma *closure_Int_ball_not_empty*:
assumes $S \subseteq closure\ T\ x \in S\ r > 0$
shows $T \cap ball\ x\ r \neq \{\}$
using *assms centre_in_ball closure_iff_nhds_not_empty* **by** *blast*

lemma *compact_sup_maxdistance*:
fixes *S* :: *'a*::*metric_space set*
assumes *compact S*
and $S \neq \{\}$
shows $\exists\, x{\in}S.\ \exists\, y{\in}S.\ \forall\, u{\in}S.\ \forall\, v{\in}S.\ dist\ u\ v \leq dist\ x\ y$
proof −
have *compact* (*S* × *S*)
using ‹*compact S*› **by** (*intro compact_Times*)
moreover have $S \times S \neq \{\}$
using ‹$S \neq \{\}$› **by** *auto*
moreover have *continuous_on* (*S* × *S*) (λ*x*. *dist* (*fst x*) (*snd x*))
by (*intro continuous_at_imp_continuous_on ballI continuous_intros*)
ultimately show *?thesis*
using *continuous_attains_sup*[*of S* × *S* λ*x*. *dist* (*fst x*) (*snd x*)] **by** *auto*
qed

If A is a compact subset of an open set B in a metric space, then there exists an $\varepsilon > 0$ such that the Minkowski sum of A with an open ball of radius ε is also a subset of B.

lemma *compact_subset_open_imp_ball_epsilon_subset*:
assumes *compact A open B A* ⊆ *B*
obtains *e* **where** $e > 0$ $(\bigcup x{\in}A.\ ball\ x\ e) \subseteq B$
proof −
have $\forall\, x{\in}A.\ \exists\, e.\ e > 0 \wedge ball\ x\ e \subseteq B$
using *assms* **unfolding** *open_contains_ball* **by** *blast*
then obtain *e* **where** *e*: $\bigwedge x.\ x \in A \Longrightarrow e\ x > 0$ $\bigwedge x.\ x \in A \Longrightarrow ball\ x\ (e\ x) \subseteq B$
by *metis*
define *C* **where** *C* = *e* ‘ *A*
obtain *X* **where** *X*: *X* ⊆ *A finite X A* ⊆ $(\bigcup c{\in}X.\ ball\ c\ (e\ c\ /\ 2))$
using *assms*(*1*)
proof (*rule compactE_image*)
show *open* (*ball x* (*e x* / *2*)) **if** *x* ∈ *A* **for** *x*
by *simp*

```
    show A ⊆ (⋃c∈A. ball c (e c / 2))
      using e by auto
  qed auto

  define e' where e' = Min (insert 1 ((λx. e x / 2) ` X))
  have e' > 0
    unfolding e'_def using e X by (subst Min_gr_iff) auto
  have e': e' ≤ e x / 2 if x ∈ X for x
    using that X unfolding e'_def by (intro Min.coboundedI) auto

  show ?thesis
  proof
    show e' > 0
      by fact
  next
    show (⋃x∈A. ball x e') ⊆ B
    proof clarify
      fix x y assume xy: x ∈ A y ∈ ball x e'
      from xy(1) X obtain z where z: z ∈ X x ∈ ball z (e z / 2)
        by auto
      have dist y z ≤ dist x y + dist z x
        by (metis dist_commute dist_triangle)
      also have dist z x < e z / 2
        using xy z by auto
      also have dist x y < e'
        using xy by auto
      also have ... ≤ e z / 2
        using z by (intro e') auto
      finally have y ∈ ball z (e z)
        by (simp add: dist_commute)
      also have ... ⊆ B
        using z X by (intro e) auto
      finally show y ∈ B .
    qed
  qed
qed

lemma compact_subset_open_imp_cball_epsilon_subset:
  assumes compact A open B A ⊆ B
  obtains e where e > 0 (⋃x∈A. cball x e) ⊆ B
proof −
  obtain e where e > 0 and e: (⋃x∈A. ball x e) ⊆ B
    using compact_subset_open_imp_ball_epsilon_subset [OF assms] by blast
  then have (⋃x∈A. cball x (e / 2)) ⊆ (⋃x∈A. ball x e)
    by auto
  with ‹0 < e› that show ?thesis
    by (metis e half_gt_zero_iff order_trans)
qed
```

Totally bounded

proposition *seq_compact_imp_totally_bounded*:
assumes *seq_compact S*
shows $\forall e{>}0.\ \exists k.\ \mathit{finite}\ k \wedge k \subseteq S \wedge S \subseteq (\bigcup x{\in}k.\ \mathit{ball}\ x\ e)$
proof −
{ fix *e*::*real* **assume** $e > 0$ **assume** ∗: $\bigwedge k.\ \mathit{finite}\ k \Longrightarrow k \subseteq S \Longrightarrow \neg\ S \subseteq (\bigcup x{\in}k.\ \mathit{ball}\ x\ e)$
let *?Q* = $\lambda x\ n\ r.\ r \in S \wedge (\forall m < (n{::}\mathit{nat}).\ \neg\ (\mathit{dist}\ (x\ m)\ r < e))$
have $\exists x.\ \forall n{::}\mathit{nat}.\ ?Q\ x\ n\ (x\ n)$
proof (*rule dependent_wellorder_choice*)
fix *n x* **assume** $\bigwedge y.\ y < n \Longrightarrow ?Q\ x\ y\ (x\ y)$
then have $\neg\ S \subseteq (\bigcup x{\in}x\ `\ \{0..{<}n\}.\ \mathit{ball}\ x\ e)$
using ∗[*of x* ` {*0* ..< *n*}] **by** (*auto simp*: *subset_eq*)
then obtain *z* **where** $z{:}z{\in}S\ z \notin (\bigcup x{\in}x\ `\ \{0..{<}n\}.\ \mathit{ball}\ x\ e)$
unfolding *subset_eq* **by** *auto*
show $\exists r.\ ?Q\ x\ n\ r$
using *z* **by** *auto*
qed *simp*
then obtain *x* **where** $\forall n{::}\mathit{nat}.\ x\ n \in S$ **and** $x{:}\bigwedge n\ m.\ m < n \Longrightarrow \neg\ (\mathit{dist}\ (x\ m)\ (x\ n) < e)$
by *blast*
then obtain *l r* **where** $l \in S$ **and** *r*:*strict_mono r* **and** $((x \circ r) \longrightarrow l)$ *sequentially*
using *assms* **by** (*metis seq_compact_def*)
then have *Cauchy* $(x \circ r)$
using *LIMSEQ_imp_Cauchy* **by** *auto*
then obtain *N*::*nat* **where** $\bigwedge m\ n.\ N \leq m \Longrightarrow N \leq n \Longrightarrow \mathit{dist}\ ((x \circ r)\ m)\ ((x \circ r)\ n) < e$
unfolding *Cauchy_def* **using** ‹$e > 0$› **by** *blast*
then have *False*
using *x*[*of r N r* (*N*+*1*)] *r* **by** (*auto simp*: *strict_mono_def*) **}**
then show *?thesis*
by *metis*
qed

Heine-Borel theorem

proposition *seq_compact_imp_Heine_Borel*:
fixes *S* :: *'a* :: *metric_space set*
assumes *seq_compact S*
shows *compact S*
proof −
from *seq_compact_imp_totally_bounded*[*OF* ‹*seq_compact S*›]
obtain *f* **where** *f*: $\forall e{>}0.\ \mathit{finite}\ (f\ e) \wedge f\ e \subseteq S \wedge S \subseteq (\bigcup x{\in}f\ e.\ \mathit{ball}\ x\ e)$
unfolding *choice_iff'* **..**
define *K* **where** $K = (\lambda(x,\ r).\ \mathit{ball}\ x\ r)\ `\ ((\bigcup e \in \mathbb{Q} \cap \{0 <..\}.\ f\ e) \times \mathbb{Q})$
have *countably_compact S*
using ‹*seq_compact S*› **by** (*rule seq_compact_imp_countably_compact*)
then show *compact S*

```
  proof (rule countably_compact_imp_compact)
    show countable K
      unfolding K_def using f
      by (auto intro: countable_finite countable_subset countable_rat
            intro!: countable_image countable_SIGMA countable_UN)
    show ∀b∈K. open b by (auto simp: K_def)
  next
    fix T x
    assume T: open T x ∈ T and x: x ∈ S
    from openE[OF T] obtain e where 0 < e ball x e ⊆ T
      by auto
    then have 0 < e/2 ball x (e/2) ⊆ T
      by auto
    from Rats_dense_in_real[OF ‹0 < e/2›] obtain r where r ∈ ℚ 0 < r r <
e/2
      by auto
    from f[rule_format, of r] ‹0 < r› ‹x ∈ S› obtain k where k ∈ f r x ∈ ball k
r
      by auto
    from ‹r ∈ ℚ› ‹0 < r› ‹k ∈ f r› have ball k r ∈ K
      by (auto simp: K_def)
    then show ∃b∈K. x ∈ b ∧ b ∩ S ⊆ T
    proof (rule bexI[rotated], safe)
      fix y
      assume y ∈ ball k r
      with ‹r < e/2› ‹x ∈ ball k r› have dist x y < e
        by (intro dist_triangle_half_r [of k _ e]) (auto simp: dist_commute)
      with ‹ball x e ⊆ T› show y ∈ T
        by auto
    next
      show x ∈ ball k r by fact
    qed
  qed
qed

proposition compact_eq_seq_compact_metric:
  compact (S :: 'a::metric_space set) ⟷ seq_compact S
  using compact_imp_seq_compact seq_compact_imp_Heine_Borel by blast

proposition compact_def: — this is the definition of compactness in HOL Light
  compact (S :: 'a::metric_space set) ⟷
   (∀f. (∀n. f n ∈ S) ⟶ (∃l∈S. ∃r::nat⇒nat. strict_mono r ∧ (f ∘ r) ⟶⟶
l))
  unfolding compact_eq_seq_compact_metric seq_compact_def by auto
```

Complete the chain of compactness variants

```
proposition compact_eq_Bolzano_Weierstrass:
  fixes S :: 'a::metric_space set
```

shows *compact S* $\longleftrightarrow$ ($\forall$ *T*. *infinite T* $\wedge$ *T* $\subseteq$ *S* $\longrightarrow$ ($\exists$ *x* $\in$ *S*. *x islimpt T*))
by (*meson Bolzano_Weierstrass_imp_seq_compact Heine_Borel_imp_Bolzano_Weierstrass seq_compact_imp_Heine_Borel*)

proposition *Bolzano_Weierstrass_imp_bounded*:
($\bigwedge$*T*. $[\![$*infinite T*; *T* $\subseteq$ *S*$]\!]$ $\Longrightarrow$ ($\exists$ *x* $\in$ *S*. *x islimpt T*)) $\Longrightarrow$ *bounded S*
using *compact_imp_bounded* **unfolding** *compact_eq_Bolzano_Weierstrass* **by** *metis*

3.2.12 Banach fixed point theorem

theorem *banach_fix*:— TODO: rename to *Banach_fix*
assumes *s*: *complete s s* $\neq$ {}
and *c*: *0* $\leq$ *c c* $<$ *1*
and *f*: *f* ‘ *s* $\subseteq$ *s*
and *lipschitz*: $\forall$ *x*$\in$*s*. $\forall$ *y*$\in$*s*. *dist* (*f x*) (*f y*) $\leq$ *c* $*$ *dist x y*
shows $\exists!$*x*$\in$*s*. *f x* = *x*
proof −
from *c* **have** *1* − *c* $>$ *0* **by** *simp*

from *s*(*2*) **obtain** *z0* **where** *z0*: *z0* $\in$ *s* **by** *blast*
define *z* **where** *z n* = (*f* ^^ *n*) *z0* **for** *n*
with *f z0* **have** *z_in_s*: *z n* $\in$ *s* **for** *n* :: *nat*
by (*induct n*) *auto*
define *d* **where** *d* = *dist* (*z 0*) (*z 1*)

have *fzn*: *f* (*z n*) = *z* (*Suc n*) **for** *n*
by (*simp add*: *z_def*)
have *cf_z*: *dist* (*z n*) (*z* (*Suc n*)) $\leq$ (*c* ^ *n*) $*$ *d* **for** *n* :: *nat*
proof (*induct n*)
case *0*
then show *?case*
by (*simp add*: *d_def*)
next
case (*Suc m*)
with ‹*0* $\leq$ *c*› **have** *c* $*$ *dist* (*z m*) (*z* (*Suc m*)) $\leq$ *c* ^ *Suc m* $*$ *d*
using *mult_left_mono*[*of dist* (*z m*) (*z* (*Suc m*)) *c* ^ *m* $*$ *d c*] **by** *simp*
then show *?case*
using *lipschitz*[*THEN bspec*[**where** *x*=*z m*], *OF z_in_s*, *THEN bspec*[**where** *x*=*z* (*Suc m*)], *OF z_in_s*]
by (*simp add*: *fzn mult_le_cancel_left*)
qed

have *cf_z2*: (*1* − *c*) $*$ *dist* (*z m*) (*z* (*m* + *n*)) $\leq$ (*c* ^ *m*) $*$ *d* $*$ (*1* − *c* ^ *n*) **for** *n m* :: *nat*
proof (*induct n*)
case *0*
show *?case* **by** *simp*
next

```
    case (Suc k)
    from c have (1 - c) * dist (z m) (z (m + Suc k)) ≤
        (1 - c) * (dist (z m) (z (m + k)) + dist (z (m + k)) (z (Suc (m + k))))
      by (simp add: dist_triangle)
    also from c cf_z[of m + k] have ... ≤ (1 - c) * (dist (z m) (z (m + k)) +
c ^ (m + k) * d)
      by simp
    also from Suc have ... ≤ c ^ m * d * (1 - c ^ k) + (1 - c) * c ^ (m + k)
* d
      by (simp add: field_simps)
    also have ... = (c ^ m) * (d * (1 - c ^ k) + (1 - c) * c ^ k * d)
      by (simp add: power_add field_simps)
    also from c have ... ≤ (c ^ m) * d * (1 - c ^ Suc k)
      by (simp add: field_simps)
    finally show ?case by simp
  qed

  have ∃N. ∀m n. N ≤ m ∧ N ≤ n ⟶ dist (z m) (z n) < e if e > 0 for e
  proof (cases d = 0)
    case True
    from ‹1 - c > 0› have (1 - c) * x ≤ 0 ⟷ x ≤ 0 for x
      by (simp add: mult_le_0_iff)
    with c cf_z2[of 0] True have z n = z0 for n
      by (simp add: z_def)
    with ‹e > 0› show ?thesis by simp
  next
    case False
    with zero_le_dist[of z 0 z 1] have d > 0
      by (metis d_def less_le)
    with ‹1 - c > 0› ‹e > 0› have 0 < e * (1 - c) / d
      by simp
    with c obtain N where N: c ^ N < e * (1 - c) / d
      using real_arch_pow_inv[of e * (1 - c) / d c] by auto
    have *: dist (z m) (z n) < e if m > n and as: m ≥ N n ≥ N for m n :: nat
    proof -
      from c ‹n ≥ N› have *: c ^ n ≤ c ^ N
        using power_decreasing[OF ‹n≥N›, of c] by simp
      from c ‹m > n› have 1 - c ^ (m - n) > 0
        using power_strict_mono[of c 1 m - n] by simp
      with ‹d > 0› ‹0 < 1 - c› have **: d * (1 - c ^ (m - n)) / (1 - c) > 0
        by simp
      from cf_z2[of n m - n] ‹m > n›
      have dist (z m) (z n) ≤ c ^ n * d * (1 - c ^ (m - n)) / (1 - c)
      by (simp add: pos_le_divide_eq[OF ‹1 - c > 0›] mult.commute dist_commute)
      also have ... ≤ c ^ N * d * (1 - c ^ (m - n)) / (1 - c)
        using mult_right_mono[OF * order_less_imp_le[OF **]]
        by (simp add: mult.assoc)
      also have ... < (e * (1 - c) / d) * d * (1 - c ^ (m - n)) / (1 - c)
        using mult_strict_right_mono[OF N **] by (auto simp: mult.assoc)
```

```
    also from c ‹d > 0› ‹1 − c > 0› have ... = e * (1 − c ^(m − n))
      by simp
    also from c ‹1 − c ^(m − n) > 0› ‹e > 0› have ... ≤ e
      using mult_right_le_one_le[of e 1 − c ^(m − n)] by auto
    finally show ?thesis by simp
   qed
   have dist (z n) (z m) < e if N ≤ m N ≤ n for m n :: nat
   proof (cases n = m)
    case True
    with ‹e > 0› show ?thesis by simp
   next
    case False
    with *[of n m] *[of m n] and that show ?thesis
     by (auto simp: dist_commute nat_neq_iff)
   qed
   then show ?thesis by auto
  qed
  then have Cauchy z
   by (metis metric_CauchyI)
  then obtain x where x∈s and x:(z ⟶ x) sequentially
   using s(1)[unfolded compact_def complete_def, THEN spec[where x=z]] and
z_in_s by auto

  define e where e = dist (f x) x
  have e = 0
  proof (rule ccontr)
   assume e ≠ 0
   then have e > 0
    unfolding e_def using zero_le_dist[of f x x]
    by (metis dist_eq_0_iff dist_nz e_def)
   then obtain N where N:∀ n≥N. dist (z n) x < e/2
    using x[unfolded lim_sequentially, THEN spec[where x=e/2]] by auto
   then have N':dist (z N) x < e/2 by auto
   have *: c * dist (z N) x ≤ dist (z N) x
    unfolding mult_le_cancel_right2
    using zero_le_dist[of z N x] and c
    by (metis dist_eq_0_iff dist_nz order_less_asym less_le)
   have dist (f (z N)) (f x) ≤ c * dist (z N) x
    using lipschitz[THEN bspec[where x=z N], THEN bspec[where x=x]]
    using z_in_s[of N] ‹x∈s›
    using c
    by auto
   also have ... < e/2
    using N' and c using * by auto
   finally show False
    unfolding fzn
    using N[THEN spec[where x=Suc N]] and dist_triangle_half_r[of z (Suc
N) f x e x]
    unfolding e_def
```

```
    by auto
  qed
  then have f x = x by (auto simp: e_def)
  moreover have y = x if f y = y y ∈ s for y
  proof −
    from ‹x ∈ s› ‹f x = x› that have dist x y ≤ c ∗ dist x y
      using lipschitz[THEN bspec[where x=x], THEN bspec[where x=y]] by simp
    with c and zero_le_dist[of x y] have dist x y = 0
      by (simp add: mult_le_cancel_right1)
    then show ?thesis by simp
  qed
  ultimately show ?thesis
    using ‹x∈s› by blast
qed
```

3.2.13 Edelstein fixed point theorem

```
theorem Edelstein_fix:
  fixes S :: 'a::metric_space set
  assumes S: compact S S ≠ {}
    and gs: (g ‘ S) ⊆ S
    and dist: ∀ x∈S. ∀ y∈S. x ≠ y ⟶ dist (g x) (g y) < dist x y
  shows ∃!x∈S. g x = x
proof −
  let ?D = (λx. (x, x)) ‘ S
  have D: compact ?D ?D ≠ {}
    by (rule compact_continuous_image)
     (auto intro!: S continuous_Pair continuous_ident simp: continuous_on_eq_continuous_within)

  have ⋀x y e. x ∈ S ⟹ y ∈ S ⟹ 0 < e ⟹ dist y x < e ⟹ dist (g y) (g x)
< e
    using dist by fastforce
  then have continuous_on S g
    by (auto simp: continuous_on_iff)
  then have cont: continuous_on ?D (λx. dist ((g ∘ fst) x) (snd x))
    unfolding continuous_on_eq_continuous_within
    by (intro continuous_dist ballI continuous_within_compose)
       (auto intro!: continuous_fst continuous_snd continuous_ident simp: im-
age_image)

  obtain a where a ∈ S and le: ⋀x. x ∈ S ⟹ dist (g a) a ≤ dist (g x) x
    using continuous_attains_inf[OF D cont] by auto

  have g a = a
  proof (rule ccontr)
    assume g a ≠ a
    with ‹a ∈ S› gs have dist (g (g a)) (g a) < dist (g a) a
      by (intro dist[rule_format]) auto
    moreover have dist (g a) a ≤ dist (g (g a)) (g a)
```

```
    using ‹a ∈ S› gs by (intro le) auto
   ultimately show False by auto
 qed
 moreover have ⋀x. x ∈ S ⟹ g x = x ⟹ x = a
   using dist[THEN bspec[where x=a]] ‹g a = a› and ‹a∈S› by auto
 ultimately show ∃!x∈S. g x = x
   using ‹a ∈ S› by blast
qed
```

3.2.14 The diameter of a set

```
definition diameter :: 'a::metric_space set ⇒ real where
 diameter S = (if S = {} then 0 else SUP (x,y)∈S×S. dist x y)

lemma diameter_empty [simp]: diameter{} = 0
 by (auto simp: diameter_def)

lemma diameter_singleton [simp]: diameter{x} = 0
 by (auto simp: diameter_def)

lemma diameter_le:
 assumes S ≠ {} ∨ 0 ≤ d
   and no: ⋀x y. ⟦x ∈ S; y ∈ S⟧ ⟹ norm(x - y) ≤ d
 shows diameter S ≤ d
 using assms
 by (auto simp: dist_norm diameter_def intro: cSUP_least)

lemma diameter_bounded_bound:
 fixes S :: 'a :: metric_space set
 assumes S: bounded S x ∈ S y ∈ S
 shows dist x y ≤ diameter S
proof -
 from S obtain z d where z: ⋀x. x ∈ S ⟹ dist z x ≤ d
   unfolding bounded_def by auto
 have bdd_above (case_prod dist ` (S×S))
 proof (intro bdd_aboveI, safe)
   fix a b
   assume a ∈ S b ∈ S
   with z[of a] z[of b] dist_triangle[of a b z]
   show dist a b ≤ 2 * d
     by (simp add: dist_commute)
 qed
 moreover have (x,y) ∈ S×S using S by auto
 ultimately have dist x y ≤ (SUP (x,y)∈S×S. dist x y)
   by (rule cSUP_upper2) simp
 with ‹x ∈ S› show ?thesis
   by (auto simp: diameter_def)
qed
```

lemma *diameter_lower_bounded*:
fixes $S :: 'a :: metric\_space\ set$
assumes S: *bounded* S
and d: $0 < d$ $d <$ *diameter* S
shows $\exists\, x{\in}S.\ \exists\, y{\in}S.\ d < dist\ x\ y$
proof (*rule ccontr*)
assume *contr*: $\neg$ *?thesis*
moreover have $S \neq \{\}$
using d **by** (*auto simp*: *diameter_def*)
ultimately have *diameter* $S \leq d$
by (*auto simp*: *not_less diameter_def intro*!: *cSUP_least*)
with ‹$d <$ *diameter* S› **show** *False* **by** *auto*
qed

lemma *diameter_bounded*:
assumes *bounded* S
shows $\forall\, x{\in}S.\ \forall\, y{\in}S.\ dist\ x\ y \leq$ *diameter* S
and $\forall\, d{>}0.\ d <$ *diameter* $S \longrightarrow (\exists\, x{\in}S.\ \exists\, y{\in}S.\ dist\ x\ y > d)$
using *diameter_bounded_bound*[*of* S] *diameter_lower_bounded*[*of* S] *assms*
by *auto*

lemma *bounded_two_points*: *bounded* $S \longleftrightarrow (\exists\, e.\ \forall\, x{\in}S.\ \forall\, y{\in}S.\ dist\ x\ y \leq e)$
by (*meson bounded_def diameter_bounded*(*1*))

lemma *diameter_compact_attained*:
assumes *compact* S
and $S \neq \{\}$
shows $\exists\, x{\in}S.\ \exists\, y{\in}S.\ dist\ x\ y =$ *diameter* S
proof −
have b: *bounded* S **using** *assms*(*1*)
by (*rule compact_imp_bounded*)
then obtain $x\ y$ **where** *xys*: $x{\in}S\ y{\in}S$
and *xy*: $\forall\, u{\in}S.\ \forall\, v{\in}S.\ dist\ u\ v \leq dist\ x\ y$
using *compact_sup_maxdistance*[*OF assms*] **by** *auto*
then have *diameter* $S \leq dist\ x\ y$
unfolding *diameter_def* **by** (*force intro*!: *cSUP_least*)
then show *?thesis*
by (*metis b diameter_bounded_bound order_antisym xys*)
qed

lemma *diameter_ge_0*:
assumes *bounded* S **shows** $0 \leq$ *diameter* S
by (*metis all_not_in_conv assms diameter_bounded_bound diameter_empty dist_self order_refl*)

lemma *diameter_subset*:
assumes $S \subseteq T$ *bounded* T
shows *diameter* $S \leq$ *diameter* T
proof (*cases* $S = \{\} \vee T = \{\}$)

```
  case True
  with assms show ?thesis
    by (force simp: diameter_ge_0)
next
  case False
  then have bdd_above ((λx. case x of (x, xa) ⇒ dist x xa) ` (T × T))
    using ‹bounded T› diameter_bounded_bound by (force simp: bdd_above_def)
  with False ‹S ⊆ T› show ?thesis
    apply (simp add: diameter_def)
    apply (rule cSUP_subset_mono, auto)
    done
qed

lemma diameter_closure:
  assumes bounded S
  shows diameter(closure S) = diameter S
proof (rule order_antisym)
  have False if d_less_d: diameter S < diameter (closure S)
  proof −
    define d where d = diameter(closure S) − diameter(S)
    have d > 0
      using that by (simp add: d_def)
    then have dle: diameter(closure(S)) − d / 2 < diameter(closure(S))
      by simp
   have dd: diameter (closure S) − d / 2 = (diameter(closure(S)) + diameter(S))
/ 2
      by (simp add: d_def field_split_simps)
     have bocl: bounded (closure S)
      using assms by blast
    moreover have 0 ≤ diameter S
      using assms diameter_ge_0 by blast
    ultimately obtain x y where x ∈ closure S y ∈ closure S and xy: diame-
ter(closure(S)) − d / 2 < dist x y
      by (smt (verit) dle d_less_d d_def dd diameter_lower_bounded)
    then obtain x′ y′ where x′y′: x′ ∈ S dist x′ x < d/4 y′ ∈ S dist y′ y < d/4
    by (metis ‹0 < d› zero_less_divide_iff zero_less_numeral closure_approachable)
    then have dist x′ y′ ≤ diameter S
      using assms diameter_bounded_bound by blast
    with x′y′ have dist x y ≤ d / 4 + diameter S + d / 4
        by (meson add_mono dist_triangle dist_triangle3 less_eq_real_def or-
der_trans)
    then show ?thesis
      using xy d_def by linarith
  qed
  then show diameter (closure S) ≤ diameter S
    by fastforce
  next
    show diameter S ≤ diameter (closure S)
      by (simp add: assms bounded_closure closure_subset diameter_subset)
```

qed

proposition *Lebesgue_number_lemma*:
assumes *compact* S $\mathcal{C} \neq \{\}$ $S \subseteq \bigcup \mathcal{C}$ **and** *ope*: $\bigwedge B.\ B \in \mathcal{C} \Longrightarrow$ *open* B
obtains δ **where** $0 < \delta$ $\bigwedge T.\ [\![T \subseteq S;$ *diameter* $T < \delta]\!] \Longrightarrow \exists B \in \mathcal{C}.\ T \subseteq B$
proof (*cases* $S = \{\}$)
case *True*
then show *?thesis*
by (*metis* ‹$\mathcal{C} \neq \{\}$› *zero_less_one empty_subsetI equals0I subset_trans that*)
next
case *False*
{ fix x **assume** $x \in S$
then obtain C **where** C: $x \in C$ $C \in \mathcal{C}$
using ‹$S \subseteq \bigcup \mathcal{C}$› **by** *blast*
then obtain r **where** r: $r>0$ *ball* x $(2{*}r) \subseteq C$
by (*metis mult.commute mult_2_right not_le ope openE field_sum_of_halves zero_le_numeral zero_less_mult_iff*)
then have $\exists r\ C.\ r > 0 \wedge$ *ball* x $(2{*}r) \subseteq C \wedge C \in \mathcal{C}$
using C **by** *blast*
}
then obtain r **where** r: $\bigwedge x.\ x \in S \Longrightarrow r\ x > 0 \wedge (\exists C \in \mathcal{C}.$ *ball* x $(2{*}r\ x) \subseteq C)$
by *metis*
then have $S \subseteq (\bigcup x \in S.$ *ball* x $(r\ x))$
by *auto*
then obtain $\mathcal{T}$ **where** *finite* $\mathcal{T}$ $S \subseteq \bigcup \mathcal{T}$ **and** $\mathcal{T}$: $\mathcal{T} \subseteq (\lambda x.$ *ball* x $(r\ x))$ ‘ S
by (*rule compactE* [*OF* ‹*compact S*›]) *auto*
then obtain *S0* **where** *S0* $\subseteq S$ *finite S0* **and** *S0*: $\mathcal{T} = (\lambda x.$ *ball* x $(r\ x))$ ‘ *S0*
by (*meson finite_subset_image*)
then have *S0* $\neq \{\}$
using *False* ‹$S \subseteq \bigcup \mathcal{T}$› **by** *auto*
define δ **where** $\delta =$ *Inf* $(r$ ‘ *S0*$)$
have $\delta > 0$
using ‹*finite S0*› ‹*S0* $\subseteq S$› ‹*S0* $\neq \{\}$› r **by** (*auto simp*: δ*_def finite_less_Inf_iff*)
show *?thesis*
proof
show $0 < \delta$
by (*simp add*: ‹$0 < \delta$›)
show $\exists B \in \mathcal{C}.\ T \subseteq B$ **if** $T \subseteq S$ **and** *dia*: *diameter* $T < \delta$ **for** T
proof (*cases* $T = \{\}$)
case *True*
then show *?thesis*
using ‹$\mathcal{C} \neq \{\}$› **by** *blast*
next
case *False*
then obtain y **where** $y \in T$ **by** *blast*
then have $y \in S$
using ‹$T \subseteq S$› **by** *auto*
then obtain x **where** $x \in S0$ **and** x: $y \in$ *ball* x $(r\ x)$

using ‹$S \subseteq \bigcup \mathcal{T}$› *S0 that* **by** *blast*
have *ball y δ ⊆ ball y (r x)*
by (*metis δ_def* ‹*S0 ≠ {}*› ‹*finite S0*› ‹*x ∈ S0*› *empty_is_image finite_imageI finite_less_Inf_iff imageI less_irrefl not_le subset_ball*)
also have *... ⊆ ball x (2∗r x)*
using *x* **by** *metric*
finally obtain *C* **where** *C ∈ 𝒞 ball y δ ⊆ C*
by (*meson r* ‹*S0 ⊆ S*› ‹*x ∈ S0*› *dual_order.trans subsetCE*)
have *bounded T*
using ‹*compact S*› *bounded_subset compact_imp_bounded* ‹*T ⊆ S*› **by** *blast*
then have *T ⊆ ball y δ*
using ‹*y ∈ T*› *dia diameter_bounded_bound* **by** *fastforce*
then show *?thesis*
by (*meson* ‹*C ∈ 𝒞*› ‹*ball y δ ⊆ C*› *subset_eq*)
qed
qed
qed

3.2.15 Metric spaces with the Heine-Borel property

A metric space (or topological vector space) is said to have the Heine-Borel property if every closed and bounded subset is compact.

class *heine_borel = metric_space +*
assumes *bounded_imp_convergent_subsequence*:
bounded (range f) ⟹ ∃ l r. strict_mono (r::nat⇒nat) ∧ ((f ∘ r) ⟶ l) sequentially

proposition *bounded_closed_imp_seq_compact*:
fixes *S::'a::heine_borel set*
assumes *bounded S*
and *closed S*
shows *seq_compact S*
proof (*unfold seq_compact_def*, *clarify*)
fix *f :: nat ⇒ 'a*
assume *f*: *∀ n. f n ∈ S*
with ‹*bounded S*› **have** *bounded (range f)*
by (*auto intro*: *bounded_subset*)
obtain *l r* **where** *r*: *strict_mono (r :: nat ⇒ nat)* **and** *l*: *((f ∘ r) ⟶ l) sequentially*
using *bounded_imp_convergent_subsequence* [*OF* ‹*bounded (range f)*›] **by** *auto*
from *f* **have** *fr*: *∀ n. (f ∘ r) n ∈ S*
by *simp*
show *∃ l∈S. ∃ r. strict_mono r ∧ (f ∘ r) ⟶ l*
using *assms(2) closed_sequentially fr l r* **by** *blast*
qed

lemma *compact_eq_bounded_closed*:
fixes *S :: 'a::heine_borel set*
shows *compact S ⟷ bounded S ∧ closed S*

using *bounded_closed_imp_seq_compact compact_eq_seq_compact_metric compact_imp_bounded compact_imp_closed*
by *auto*

lemma *bounded_infinite_imp_islimpt*:
fixes $S :: 'a::heine\_borel\ set$
assumes $T \subseteq S$ *bounded* S *infinite* T
obtains x **where** x *islimpt* S
by (*meson assms closed_limpt compact_eq_Bolzano_Weierstrass compact_eq_bounded_closed islimpt_subset*)

lemma *compact_Inter*:
fixes $\mathcal{F} :: 'a :: heine\_borel\ set\ set$
assumes *com*: $\bigwedge S.\ S \in \mathcal{F} \Longrightarrow compact\ S$ **and** $\mathcal{F} \neq \{\}$
shows *compact*$(\bigcap\ \mathcal{F})$
using *assms*
by (*meson Inf_lower all_not_in_conv bounded_subset closed_Inter compact_eq_bounded_closed*)

lemma *compact_closure* [*simp*]:
fixes $S :: 'a::heine\_borel\ set$
shows *compact*(*closure* S) $\longleftrightarrow$ *bounded* S
by (*meson bounded_closure bounded_subset closed_closure closure_subset compact_eq_bounded_closed*)

instance *real* :: *heine_borel*
proof
fix $f :: nat \Rightarrow real$
assume f: *bounded* (*range* f)
obtain $r :: nat \Rightarrow nat$ **where** r: *strict_mono* r *monoseq* $(f \circ r)$
unfolding *comp_def* **by** (*metis seq_monosub*)
then have *Bseq* $(f \circ r)$
unfolding *Bseq_eq_bounded* **by** (*metis f BseqI' bounded_iff comp_apply rangeI*)
with r **show** $\exists\, l\ r.\ strict\_mono\ r \wedge (f \circ r) \longrightarrow l$
using *Bseq_monoseq_convergent*[*of* $f \circ r$] **by** (*auto simp*: *convergent_def*)
qed

lemma *compact_lemma_general*:
fixes $f :: nat \Rightarrow 'a$
fixes *proj*$::'a \Rightarrow 'b \Rightarrow 'c::heine\_borel$ (**infixl** *proj* 60)
fixes *unproj*:: $('b \Rightarrow 'c) \Rightarrow 'a$
assumes *finite_basis*: *finite basis*
assumes *bounded_proj*: $\bigwedge k.\ k \in basis \Longrightarrow bounded\ ((\lambda x.\ x\ proj\ k)\ `\ range\ f)$
assumes *proj_unproj*: $\bigwedge e\ k.\ k \in basis \Longrightarrow (unproj\ e)\ proj\ k = e\ k$
assumes *unproj_proj*: $\bigwedge x.\ unproj\ (\lambda k.\ x\ proj\ k) = x$
shows $\forall\, d \subseteq basis.\ \exists\, l::'a.\ \exists\ r::nat \Rightarrow nat.$
$strict\_mono\ r \wedge (\forall\, e>0.\ eventually\ (\lambda n.\ \forall\, i \in d.\ dist\ (f\ (r\ n)\ proj\ i)\ (l\ proj\ i) < e)\ sequentially)$
proof *safe*

```
  fix d :: 'b set
  assume d: d ⊆ basis
  with finite_basis have finite d
    by (blast intro: finite_subset)
  from this d show ∃ l::'a. ∃ r::nat⇒nat. strict_mono r ∧
    (∀ e>0. eventually (λn. ∀ i∈d. dist (f (r n) proj i) (l proj i) < e) sequentially)
  proof (induct d)
    case empty
    then show ?case
      unfolding strict_mono_def by auto
  next
    case (insert k d)
    have k[intro]: k ∈ basis
      using insert by auto
    have s': bounded ((λx. x proj k) ` range f)
      using k
      by (rule bounded_proj)
    obtain l1::'a and r1 where r1: strict_mono r1
      and lr1: ∀ e > 0. ∀ F n in sequentially. ∀ i∈d. dist (f (r1 n) proj i) (l1 proj
i) < e
      using insert by auto
    have f': ∀ n. f (r1 n) proj k ∈ (λx. x proj k) ` range f
      by simp
    have bounded (range (λi. f (r1 i) proj k))
      by (metis (lifting) bounded_subset f' image_subsetI s')
    then obtain l2 r2 where r2: strict_mono r2 and lr2: (λi. f (r1 (r2 i)) proj
k) ⟶ l2
      using bounded_imp_convergent_subsequence[of λi. f (r1 i) proj k]
      by (auto simp: o_def)
    define r where r = r1 ∘ r2
    have r: strict_mono r
      using r1 and r2 unfolding r_def o_def strict_mono_def by auto
    moreover
    define l where l = unproj (λi. if i = k then l2 else l1 proj i)
    { fix e::real
      assume e > 0
      from lr1 ‹e > 0› have N1: ∀ F n in sequentially. ∀ i∈d. dist (f (r1 n) proj
i) (l1 proj i) < e
        by blast
      from lr2 ‹e > 0› have N2: ∀ F n in sequentially. dist (f (r1 (r2 n)) proj k)
l2 < e
        by (rule tendstoD)
      from r2 N1 have N1': ∀ F n in sequentially. ∀ i∈d. dist (f (r1 (r2 n)) proj
i) (l1 proj i) < e
        by (rule eventually_subseq)
      have ∀ F n in sequentially. ∀ i∈insert k d. dist (f (r n) proj i) (l proj i) < e
        using N1' N2
          by eventually_elim (use insert.prems in ‹auto simp: l_def r_def o_def
proj_unproj›)
```

```
    }
    ultimately show ?case by auto
  qed
qed

lemma bounded_fst: bounded s ⟹ bounded (fst ‘ s)
  unfolding bounded_def
  by (metis (erased, opaque_lifting) dist_fst_le image_iff order_trans)

lemma bounded_snd: bounded s ⟹ bounded (snd ‘ s)
  unfolding bounded_def
  by (metis (no_types, opaque_lifting) dist_snd_le image_iff order.trans)

instance prod :: (heine_borel, heine_borel) heine_borel
proof
  fix f :: nat ⇒ 'a × 'b
  assume f: bounded (range f)
  then have bounded (fst ‘ range f)
    by (rule bounded_fst)
  then have s1: bounded (range (fst ∘ f))
    by (simp add: image_comp)
  obtain l1 r1 where r1: strict_mono r1 and l1: (λn. fst (f (r1 n))) ⟶ l1
   using bounded_imp_convergent_subsequence [OF s1] unfolding o_def by fast
  from f have s2: bounded (range (snd ∘ f ∘ r1))
    by (auto simp: image_comp intro: bounded_snd bounded_subset)
  obtain l2 r2 where r2: strict_mono r2 and l2: (λn. snd (f (r1 (r2 n)))) ⟶
l2
    using bounded_imp_convergent_subsequence [OF s2]
    unfolding o_def by fast
  have l1': ((λn. fst (f (r1 (r2 n)))) ⟶ l1) sequentially
    using LIMSEQ_subseq_LIMSEQ [OF l1 r2] unfolding o_def .
  have l: ((f ∘ (r1 ∘ r2)) ⟶ (l1, l2)) sequentially
    using tendsto_Pair [OF l1' l2] unfolding o_def by simp
  have r: strict_mono (r1 ∘ r2)
    using r1 r2 unfolding strict_mono_def by simp
  show ∃l r. strict_mono r ∧ ((f ∘ r) ⟶ l) sequentially
    using l r by fast
qed
```

3.2.16 Completeness

```
proposition (in metric_space) completeI:
  assumes ⋀f. ∀n. f n ∈ s ⟹ Cauchy f ⟹ ∃l∈s. f ⟶ l
  shows complete s
  using assms unfolding complete_def by fast

proposition (in metric_space) completeE:
  assumes complete s and ∀n. f n ∈ s and Cauchy f
  obtains l where l ∈ s and f ⟶ l
```

using *assms* **unfolding** *complete_def* **by** *fast*

lemma *compact_imp_complete*:
fixes $s :: 'a::metric\_space\ set$
assumes *compact s*
shows *complete s*
proof −
{
fix *f*
assume *as*: $(\forall n::nat.\ f\ n \in s)$ *Cauchy f*
from *as*(*1*) **obtain** *l r* **where** *lr*: $l \in s$ *strict_mono r* $(f \circ r) \longrightarrow l$
using *assms* **unfolding** *compact_def* **by** *blast*

note $lr' = seq\_suble\ [OF\ lr(2)]$
{
fix *e* :: *real*
assume $e > 0$
from *as*(*2*) **obtain** *N* **where** $N{:}\forall m\ n.\ N \leq m \wedge N \leq n \longrightarrow dist\ (f\ m)\ (f\ n) < e/2$
unfolding *Cauchy_def* **using** ‹$e > 0$› **by** (*meson half_gt_zero*)
then obtain *M* **where** $M{:}\forall n{\geq}M.\ dist\ ((f \circ r)\ n)\ l < e/2$
by (*metis dist_self lim_sequentially lr*(*3*))
{
fix *n* :: *nat*
assume *n*: $n \geq max\ N\ M$
have $dist\ ((f \circ r)\ n)\ l < e/2$
using *n M* **by** *auto*
moreover have $r\ n \geq N$
using $lr'[of\ n]$ *n* **by** *auto*
then have $dist\ (f\ n)\ ((f \circ r)\ n) < e/2$
using *N* **and** *n* **by** *auto*
ultimately have $dist\ (f\ n)\ l < e$ **using** *n M*
by *metric*
}
then have $\exists N.\ \forall n{\geq}N.\ dist\ (f\ n)\ l < e$ **by** *blast*
}
then have $\exists l{\in}s.\ (f \longrightarrow l)\ sequentially$ **using** ‹$l \in s$›
unfolding *lim_sequentially* **by** *auto*
}
then show *?thesis* **unfolding** *complete_def* **by** *auto*
qed

proposition *compact_eq_totally_bounded*:
$compact\ s \longleftrightarrow complete\ s \wedge (\forall e{>}0.\ \exists k.\ finite\ k \wedge s \subseteq (\bigcup x{\in}k.\ ball\ x\ e))$
(**is** _ $\longleftrightarrow$ *?rhs*)
proof
assume *assms*: *?rhs*
then obtain *k* **where** *k*: $\bigwedge e.\ 0 < e \Longrightarrow finite\ (k\ e)\ \bigwedge e.\ 0 < e \Longrightarrow s \subseteq (\bigcup x{\in}k$

```
e. ball x e)
    by (auto simp: choice_iff')

  show compact s
  proof cases
    assume s = {}
    then show compact s by (simp add: compact_def)
  next
    assume s ≠ {}
    show ?thesis
      unfolding compact_def
    proof safe
      fix f :: nat ⇒ 'a
      assume f: ∀ n. f n ∈ s

      define e where e n = 1 / (2 * Suc n) for n
      then have [simp]: ⋀n. 0 < e n by auto
      define B where B n U = (SOME b. infinite {n. f n ∈ b} ∧ (∃ x. b ⊆ ball x
(e n) ∩ U)) for n U
      {
        fix n U
        assume infinite {n. f n ∈ U}
        then have ∃ b∈k (e n). infinite {i∈{n. f n ∈ U}. f i ∈ ball b (e n)}
          using k f by (intro pigeonhole_infinite_rel) (auto simp: subset_eq)
        then obtain a where
          a ∈ k (e n)
          infinite {i ∈ {n. f n ∈ U}. f i ∈ ball a (e n)} ..
        then have ∃ b. infinite {i. f i ∈ b} ∧ (∃ x. b ⊆ ball x (e n) ∩ U)
          by (intro exI[of _ ball a (e n) ∩ U] exI[of _ a]) (auto simp: ac_simps)
        from someI_ex[OF this]
        have infinite {i. f i ∈ B n U} ∃ x. B n U ⊆ ball x (e n) ∩ U
          unfolding B_def by auto
      }
      note B = this

      define F where F = rec_nat (B 0 UNIV) B
      {
        fix n
        have infinite {i. f i ∈ F n}
          by (induct n) (auto simp: F_def B)
      }
      then have F: ⋀n. ∃ x. F (Suc n) ⊆ ball x (e n) ∩ F n
        using B by (simp add: F_def)
      then have F_dec: ⋀m n. m ≤ n ⟹ F n ⊆ F m
        using decseq_SucI[of F] by (auto simp: decseq_def)

      obtain sel where sel: ⋀k i. i < sel k i ⋀k i. f (sel k i) ∈ F k
      proof (atomize_elim, unfold all_conj_distrib[symmetric], intro choice allI)
        fix k i
```

```
      have infinite ({n. f n ∈ F k} − {.. i})
        using ‹infinite {n. f n ∈ F k}› by auto
      from infinite_imp_nonempty[OF this]
      show ∃x>i. f x ∈ F k
        by (simp add: set_eq_iff not_le conj_commute)
    qed

    define t where t = rec_nat (sel 0 0) (λn i. sel (Suc n) i)
    have strict_mono t
      unfolding strict_mono_Suc_iff by (simp add: t_def sel)
    moreover have ∀i. (f ∘ t) i ∈ s
      using f by auto
    moreover
    have t: (f ∘ t) n ∈ F n for n
      by (cases n) (simp_all add: t_def sel)

    have Cauchy (f ∘ t)
    proof (safe intro!: metric_CauchyI exI elim!: nat_approx_posE)
      fix r :: real and N n m
      assume 1 / Suc N < r Suc N ≤ n Suc N ≤ m
      then have (f ∘ t) n ∈ F (Suc N) (f ∘ t) m ∈ F (Suc N) 2 * e N < r
        using F_dec t by (auto simp: e_def field_simps)
      with F[of N] obtain x where dist x ((f ∘ t) n) < e N dist x ((f ∘ t) m)
< e N
        by (auto simp: subset_eq)
      with ‹2 * e N < r› show dist ((f ∘ t) m) ((f ∘ t) n) < r
        by metric
    qed

    ultimately show ∃l∈s. ∃r. strict_mono r ∧ (f ∘ r) ⟶ l
      using assms unfolding complete_def by blast
  qed
 qed
qed (metis compact_imp_complete compact_imp_seq_compact seq_compact_imp_totally_bounded)

lemma cauchy_imp_bounded:
 assumes Cauchy s
 shows bounded (range s)
proof −
 from assms obtain N :: nat where ∀m n. N ≤ m ∧ N ≤ n ⟶ dist (s m) (s
n) < 1
   by (meson Cauchy_def zero_less_one)
 then have N:∀n. N ≤ n ⟶ dist (s N) (s n) < 1 by auto
 moreover
 have bounded (s ‘ {0..N})
   using finite_imp_bounded[of s ‘ {1..N}] by auto
 then obtain a where a:∀x∈s ‘ {0..N}. dist (s N) x ≤ a
   unfolding bounded_any_center [where a=s N] by auto
 ultimately show ?thesis
```

```
    unfolding bounded_any_center [where a=s N]
    apply (rule_tac x=max a 1 in exI, auto)
    apply (erule_tac x=y in allE)
    apply (erule_tac x=y in ballE, auto)
    done
qed

instance heine_borel < complete_space
proof
  fix f :: nat ⇒ 'a assume Cauchy f
  then show convergent f
    unfolding convergent_def
   using Cauchy_converges_subseq cauchy_imp_bounded bounded_imp_convergent_subsequence
by blast
qed

lemma complete_UNIV: complete (UNIV :: ('a::complete_space) set)
proof (rule completeI)
  fix f :: nat ⇒ 'a assume Cauchy f
  then show ∃l∈UNIV. f ⟶ l
    using Cauchy_convergent convergent_def by blast
qed

lemma complete_imp_closed:
  fixes S :: 'a::metric_space set
  shows complete S ⟹ closed S
  by (metis LIMSEQ_imp_Cauchy LIMSEQ_unique closed_sequential_limits com-
pleteE)

lemma complete_Int_closed:
  fixes S :: 'a::metric_space set
  assumes complete S and closed t
  shows complete (S ∩ t)
  by (metis Int_iff assms closed_sequentially completeE completeI)

lemma complete_closed_subset:
  fixes S :: 'a::metric_space set
  assumes closed S and S ⊆ t and complete t
  shows complete S
  using assms complete_Int_closed [of t S] by (simp add: Int_absorb1)

lemma complete_eq_closed:
  fixes S :: ('a::complete_space) set
  shows complete S ⟷ closed S
proof
  assume closed S then show complete S
    using subset_UNIV complete_UNIV by (rule complete_closed_subset)
next
  assume complete S then show closed S
```

by (*rule complete_imp_closed*)
qed

lemma *convergent_eq_Cauchy*:
fixes $S :: nat \Rightarrow 'a::complete\_space$
shows $(\exists\, l.\ (S \longrightarrow l)\ sequentially) \longleftrightarrow Cauchy\ S$
unfolding *Cauchy_convergent_iff convergent_def* **..**

lemma *convergent_imp_bounded*:
fixes $S :: nat \Rightarrow 'a::metric\_space$
shows $(S \longrightarrow l)\ sequentially \Longrightarrow bounded\ (range\ S)$
by (*intro cauchy_imp_bounded LIMSEQ_imp_Cauchy*)

lemma *frontier_subset_compact*:
fixes $S :: 'a::heine\_borel\ set$
shows $compact\ S \Longrightarrow frontier\ S \subseteq S$
using *frontier_subset_closed compact_eq_bounded_closed*
by *blast*

lemma *banach_fix_type*:
fixes $f::'a::complete\_space \Rightarrow 'a$
assumes $c{:}0 \le c\ c < 1$
and $lipschitz{:}\forall x.\ \forall y.\ dist\ (f\ x)\ (f\ y) \le c * dist\ x\ y$
shows $\exists !x.\ (f\ x = x)$
using *assms banach_fix*[*OF complete_UNIV UNIV_not_empty assms*(*1*,*2*) *subset_UNIV*, *of f*]
by *auto*

3.2.17 Cauchy continuity

definition *Cauchy_continuous_on* **where**
$Cauchy\_continuous\_on \equiv \lambda S\ f.\ \forall \sigma.\ Cauchy\ \sigma \longrightarrow range\ \sigma \subseteq S \longrightarrow Cauchy\ (f \circ \sigma)$

lemma *continuous_closed_imp_Cauchy_continuous*:
fixes $S :: ('a::complete\_space)\ set$
shows $[\![continuous\_on\ S\ f;\ closed\ S]\!] \Longrightarrow Cauchy\_continuous\_on\ S\ f$
unfolding *Cauchy_continuous_on_def*
by (*metis LIMSEQ_imp_Cauchy completeE complete_eq_closed continuous_on_sequentially range_subsetD*)

lemma *uniformly_continuous_imp_Cauchy_continuous*:
fixes $f :: 'a::metric\_space \Rightarrow 'b::metric\_space$
shows $uniformly\_continuous\_on\ S\ f \Longrightarrow Cauchy\_continuous\_on\ S\ f$
by (*simp add*: *uniformly_continuous_on_def Cauchy_continuous_on_def Cauchy_def image_subset_iff*) *metis*

lemma *Cauchy_continuous_on_imp_continuous*:
fixes $f :: 'a::metric\_space \Rightarrow 'b::metric\_space$

assumes *Cauchy_continuous_on S f*
shows *continuous_on S f*
proof −
have *False* **if** x: $\forall\, n.\ \exists\, x'{\in}S.\ dist\ x'\ x < inverse(Suc\ n) \wedge \neg\ dist\ (f\ x')\ (f\ x) < \varepsilon$ $\varepsilon{>}0$ $x \in S$ **for** x **and** ε::*real*
proof −
obtain ϱ **where** ϱ: $\forall\, n.\ \varrho\ n \in S$ **and** dx: $\forall\, n.\ dist\ (\varrho\ n)\ x < inverse(Suc\ n)$ **and** dfx: $\forall\, n.\ \neg\ dist\ (f\ (\varrho\ n))\ (f\ x) < \varepsilon$
using x **by** *metis*
define σ **where** $\sigma \equiv \lambda n.\ if\ even\ n\ then\ \varrho\ n\ else\ x$
with ϱ ‹$x \in S$› **have** *range* $\sigma \subseteq S$
by *auto*
have $\sigma \longrightarrow x$
unfolding *tendsto_iff*
proof (*intro strip*)
fix e :: *real*
assume $e{>}0$
then obtain N **where** $inverse\ (Suc\ N) < e$
using *reals_Archimedean* **by** *blast*
then have $\forall\, n.\ N \leq n \longrightarrow dist\ (\varrho\ n)\ x < e$
by (*smt* (*verit*, *ccfv_SIG*) *dx inverse_Suc inverse_less_iff_less inverse_positive_iff_positive of_nat_Suc of_nat_mono*)
with ‹$e{>}0$› **show** $\forall_F\ n$ *in sequentially*. $dist\ (\sigma\ n)\ x < e$
by (*auto simp add*: *eventually_sequentially* σ_*def*)
qed
then have *Cauchy* σ
by (*intro LIMSEQ_imp_Cauchy*)
then have *Cf*: *Cauchy* $(f \circ \sigma)$
by (*meson Cauchy_continuous_on_def* ‹*range* $\sigma \subseteq S$› *assms*)
have $(f \circ \sigma) \longrightarrow f\ x$
unfolding *tendsto_iff*
proof (*intro strip*)
fix e :: *real*
assume $e{>}0$
then obtain N **where** N: $\forall\, m{\geq}N.\ \forall\, n{\geq}N.\ dist\ ((f \circ \sigma)\ m)\ ((f \circ \sigma)\ n) < e$
using *Cf* **unfolding** *Cauchy_def* **by** *presburger*
moreover have $(f \circ \sigma)\ (Suc(N{+}N)) = f\ x$
by (*simp add*: σ_*def*)
ultimately have $\forall\, n{\geq}N.\ dist\ ((f \circ \sigma)\ n)\ (f\ x) < e$
by (*metis add_Suc le_add2*)
then show $\forall_F\ n$ *in sequentially*. $dist\ ((f \circ \sigma)\ n)\ (f\ x) < e$
using *eventually_sequentially* **by** *blast*
qed
moreover have $\bigwedge n.\ \neg\ dist\ (f\ (\sigma\ (2{*}n)))\ (f\ x) < \varepsilon$
using *dfx* **by** (*simp add*: σ_*def*)
ultimately show *False*
using ‹$\varepsilon{>}0$› **by** (*fastforce simp*: *mult_2 nat_le_iff_add tendsto_iff eventually_sequentially*)
qed

then show *?thesis*
unfolding *continuous_on_iff* **by** (*meson inverse_Suc*)
qed

3.2.18 Finite intersection property

Also developed in HOL's toplogical spaces theory, but the Heine-Borel type class isn't available there.

lemma *closed_imp_fip*:
fixes S :: *'a*::*heine_borel set*
assumes *closed S*
and T: $T \in \mathcal{F}$ *bounded T*
and *clof*: $\bigwedge T.\ T \in \mathcal{F} \Longrightarrow$ *closed T*
and $\bigwedge \mathcal{F}'.\ [\![\mathit{finite}\ \mathcal{F}';\ \mathcal{F}' \subseteq \mathcal{F}]\!] \Longrightarrow S \cap \bigcap \mathcal{F}' \neq \{\}$
shows $S \cap \bigcap \mathcal{F} \neq \{\}$
proof −
have *compact* $(S \cap T)$
using ‹*closed S*› *clof compact_eq_bounded_closed T* **by** *blast*
then have $(S \cap T) \cap \bigcap \mathcal{F} \neq \{\}$
by (*smt* (*verit*, *best*) *Inf_insert Int_assoc assms compact_imp_fip finite_insert insert_subset*)
then show *?thesis* **by** *blast*
qed

lemma *closed_imp_fip_compact*:
fixes S :: *'a*::*heine_borel set*
shows
$[\![\mathit{closed}\ S;\ \bigwedge T.\ T \in \mathcal{F} \Longrightarrow \mathit{compact}\ T;$
$\bigwedge \mathcal{F}'.\ [\![\mathit{finite}\ \mathcal{F}';\ \mathcal{F}' \subseteq \mathcal{F}]\!] \Longrightarrow S \cap \bigcap \mathcal{F}' \neq \{\}]\!]$
$\Longrightarrow S \cap \bigcap \mathcal{F} \neq \{\}$
by (*metis closed_imp_fip compact_eq_bounded_closed equals0I finite.emptyI order.refl*)

lemma *closed_fip_Heine_Borel*:
fixes $\mathcal{F}$ :: *'a*::*heine_borel set set*
assumes $T \in \mathcal{F}$ *bounded T*
and $\bigwedge T.\ T \in \mathcal{F} \Longrightarrow$ *closed T*
and $\bigwedge \mathcal{F}'.\ [\![\mathit{finite}\ \mathcal{F}';\ \mathcal{F}' \subseteq \mathcal{F}]\!] \Longrightarrow \bigcap \mathcal{F}' \neq \{\}$
shows $\bigcap \mathcal{F} \neq \{\}$
using *closed_imp_fip* [*OF closed_UNIV*] *assms* **by** *auto*

lemma *compact_fip_Heine_Borel*:
fixes $\mathcal{F}$ :: *'a*::*heine_borel set set*
assumes *clof*: $\bigwedge T.\ T \in \mathcal{F} \Longrightarrow$ *compact T*
and *none*: $\bigwedge \mathcal{F}'.\ [\![\mathit{finite}\ \mathcal{F}';\ \mathcal{F}' \subseteq \mathcal{F}]\!] \Longrightarrow \bigcap \mathcal{F}' \neq \{\}$
shows $\bigcap \mathcal{F} \neq \{\}$
by (*metis InterI clof closed_fip_Heine_Borel compact_eq_bounded_closed equals0D none*)

lemma *compact_sequence_with_limit*:
fixes $f :: nat \Rightarrow 'a::heine\_borel$
shows $f \longrightarrow l \Longrightarrow compact\ (insert\ l\ (range\ f))$
by (*simp add*: *closed_limpt compact_eq_bounded_closed convergent_imp_bounded islimpt_insert sequence_unique_limpt*)

3.2.19 Properties of Balls and Spheres

lemma *compact_cball*[*simp*]:
fixes $x :: 'a::heine\_borel$
shows *compact* (*cball x e*)
using *compact_eq_bounded_closed bounded_cball closed_cball* **by** *blast*

lemma *compact_frontier_bounded*[*intro*]:
fixes $S :: 'a::heine\_borel\ set$
shows *bounded S* $\Longrightarrow$ *compact* (*frontier S*)
unfolding *frontier_def*
using *compact_eq_bounded_closed* **by** *blast*

lemma *compact_frontier*[*intro*]:
fixes $S :: 'a::heine\_borel\ set$
shows *compact S* $\Longrightarrow$ *compact* (*frontier S*)
using *compact_eq_bounded_closed compact_frontier_bounded* **by** *blast*

lemma *no_limpt_imp_countable*:
assumes $\bigwedge z.\ \neg z\ islimpt\ (X :: 'a :: \{real\_normed\_vector, heine\_borel\}\ set)$
shows *countable X*
proof −
have *countable* $(\bigcup n.\ cball\ 0\ (real\ n) \cap X)$
proof (*intro countable_UN*[*OF _ countable_finite*])
fix $n :: nat$
show *finite* $(cball\ 0\ (real\ n) \cap X)$
using *assms* **by** (*intro finite_not_islimpt_in_compact*) *auto*
qed *auto*
also have $(\bigcup n.\ cball\ 0\ (real\ n)) = (UNIV :: 'a\ set)$
proof *safe*
fix $z :: 'a$
have $norm\ z \geq 0$
by *simp*
hence $real\ (nat\ \lceil norm\ z \rceil) \geq norm\ z$
by *linarith*
thus $z \in (\bigcup n.\ cball\ 0\ (real\ n))$
by *auto*
qed *auto*
hence $(\bigcup n.\ cball\ 0\ (real\ n) \cap X) = X$
by *blast*
finally show *countable X* **.**
qed

3.2.20 Distance from a Set

lemma *distance_attains_sup*:
assumes *compact s s* ≠ {}
shows ∃ *x*∈*s*. ∀ *y*∈*s*. *dist a y* ≤ *dist a x*
proof (*rule continuous_attains_sup* [*OF assms*])
{
fix *x*
assume *x*∈*s*
have (*dist a* ⟶ *dist a x*) (*at x within s*)
by (*intro tendsto_dist tendsto_const tendsto_ident_at*)
}
then show *continuous_on s* (*dist a*)
unfolding *continuous_on* **..**
qed

For *minimal* distance, we only need closure, not compactness.

lemma *distance_attains_inf*:
fixes *a* :: *'a*::*heine_borel*
assumes *closed s* **and** *s* ≠ {}
obtains *x* **where** *x*∈*s* ⋀*y*. *y* ∈ *s* ⟹ *dist a x* ≤ *dist a y*
proof −
from *assms* **obtain** *b* **where** *b* ∈ *s* **by** *auto*
let *?B* = *s* ∩ *cball a* (*dist b a*)
have *?B* ≠ {} **using** ‹*b* ∈ *s*›
by (*auto simp*: *dist_commute*)
moreover have *continuous_on ?B* (*dist a*)
by (*auto intro*!: *continuous_at_imp_continuous_on continuous_dist continuous_ident continuous_const*)
moreover have *compact ?B*
by (*intro closed_Int_compact* ‹*closed s*› *compact_cball*)
ultimately obtain *x* **where** *x* ∈ *?B* ∀ *y*∈*?B*. *dist a x* ≤ *dist a y*
by (*metis continuous_attains_inf*)
with *that* **show** *?thesis* **by** *fastforce*
qed

3.2.21 Infimum Distance

definition *infdist x A* = (*if A* = {} *then 0 else INF a*∈*A*. *dist x a*)

lemma *bdd_below_image_dist*[*intro*, *simp*]: *bdd_below* (*dist x* ‘ *A*)
by (*auto intro*!: *zero_le_dist*)

lemma *infdist_notempty*: *A* ≠ {} ⟹ *infdist x A* = (*INF a*∈*A*. *dist x a*)
by (*simp add*: *infdist_def*)

lemma *infdist_nonneg*: *0* ≤ *infdist x A*
by (*auto simp*: *infdist_def intro*: *cINF_greatest*)

lemma *infdist_le*: *a* ∈ *A* ⟹ *infdist x A* ≤ *dist x a*

by (*auto intro*: *cINF_lower simp add*: *infdist_def*)

lemma *infdist_le2*: $a \in A \Longrightarrow dist\ x\ a \leq d \Longrightarrow infdist\ x\ A \leq d$
by (*auto intro*!: *cINF_lower2 simp add*: *infdist_def*)

lemma *infdist_zero*[*simp*]: $a \in A \Longrightarrow infdist\ a\ A = 0$
by (*auto intro*!: *antisym infdist_nonneg infdist_le2*)

lemma *infdist_Un_min*:
assumes $A \neq \{\}$ $B \neq \{\}$
shows $infdist\ x\ (A \cup B) = min\ (infdist\ x\ A)\ (infdist\ x\ B)$
using *assms* **by** (*simp add*: *infdist_def cINF_union inf_real_def*)

lemma *infdist_triangle*: $infdist\ x\ A \leq infdist\ y\ A + dist\ x\ y$
proof (*cases* $A = \{\}$)
case *True*
then show *?thesis* **by** (*simp add*: *infdist_def*)
next
case *False*
then obtain a **where** $a \in A$ **by** *auto*
have $infdist\ x\ A \leq Inf\ \{dist\ x\ y + dist\ y\ a\ |a.\ a \in A\}$
proof (*rule cInf_greatest*)
from ‹$A \neq \{\}$› **show** $\{dist\ x\ y + dist\ y\ a\ |a.\ a \in A\} \neq \{\}$
by *simp*
fix d
assume $d \in \{dist\ x\ y + dist\ y\ a\ |a.\ a \in A\}$
then obtain a **where** d: $d = dist\ x\ y + dist\ y\ a\ a \in A$
by *auto*
show $infdist\ x\ A \leq d$
unfolding *infdist_notempty*[*OF* ‹$A \neq \{\}$›]
proof (*rule cINF_lower2*)
show $a \in A$ **by** *fact*
show $dist\ x\ a \leq d$
unfolding d **by** (*rule dist_triangle*)
qed *simp*
qed
also have $\ldots = dist\ x\ y + infdist\ y\ A$
proof (*rule cInf_eq*, *safe*)
fix a
assume $a \in A$
then show $dist\ x\ y + infdist\ y\ A \leq dist\ x\ y + dist\ y\ a$
by (*auto intro*: *infdist_le*)
next
fix i
assume *inf*: $\bigwedge d.\ d \in \{dist\ x\ y + dist\ y\ a\ |a.\ a \in A\} \Longrightarrow i \leq d$
then have $i - dist\ x\ y \leq infdist\ y\ A$
unfolding *infdist_notempty*[*OF* ‹$A \neq \{\}$›] **using** ‹$a \in A$›
by (*intro cINF_greatest*) (*auto simp*: *field_simps*)
then show $i \leq dist\ x\ y + infdist\ y\ A$

```
    by simp
  qed
  finally show ?thesis by simp
qed

lemma infdist_triangle_abs: |infdist x A − infdist y A| ≤ dist x y
  by (metis (full_types) abs_diff_le_iff diff_le_eq dist_commute infdist_triangle)

lemma in_closure_iff_infdist_zero:
  assumes A ≠ {}
  shows x ∈ closure A ⟷ infdist x A = 0
proof
  assume x ∈ closure A
  show infdist x A = 0
  proof (rule ccontr)
    assume infdist x A ≠ 0
    with infdist_nonneg[of x A] have infdist x A > 0
      by auto
    then have ball x (infdist x A) ∩ closure A = {}
      by (smt (verit, best) ‹x ∈ closure A› closure_approachableD infdist_le)
    then have x ∉ closure A
      by (metis ‹0 < infdist x A› centre_in_ball disjoint_iff_not_equal)
    then show False using ‹x ∈ closure A› by simp
  qed
next
  assume x: infdist x A = 0
  then obtain a where a ∈ A
    by atomize_elim (metis all_not_in_conv assms)
  have False if e > 0 ¬ (∃ y∈A. dist y x < e) for e
  proof −
    have infdist x A ≥ e using ‹a ∈ A›
      unfolding infdist_def using that
      by (force simp: dist_commute intro: cINF_greatest)
    with x ‹e > 0› show False by auto
  qed
  then show x ∈ closure A
    using closure_approachable by blast
qed

lemma in_closed_iff_infdist_zero:
  assumes closed A A ≠ {}
  shows x ∈ A ⟷ infdist x A = 0
proof −
  have x ∈ closure A ⟷ infdist x A = 0
    by (simp add: ‹A ≠ {}› in_closure_iff_infdist_zero)
  with assms show ?thesis by simp
qed

lemma infdist_pos_not_in_closed:
```

assumes *closed S* $S \neq \{\}$ $x \notin S$
shows *infdist x S* > 0
by (*smt* (*verit*, *best*) *assms in_closed_iff_infdist_zero infdist_nonneg*)

lemma
infdist_attains_inf:
fixes X::'*a*::*heine_borel set*
assumes *closed X*
assumes $X \neq \{\}$
obtains x **where** $x \in X$ *infdist y X* $=$ *dist y x*
proof −
have *bdd_below* (*dist y* ‘ *X*)
by *auto*
from *distance_attains_inf*[*OF assms*, *of y*]
obtain x **where** $x \in X$ $\bigwedge z.\ z \in X \Longrightarrow$ *dist y x* $\leq$ *dist y z* **by** *auto*
then have *infdist y X* $=$ *dist y x*
by (*metis antisym assms*(*2*) *cINF_greatest infdist_def infdist_le*)
with ‹$x \in X$› **show** *?thesis* **..**
qed

Every metric space is a T4 space:

instance *metric_space* $\subseteq$ *t4_space*
proof
fix S T::'*a set* **assume** *H*: *closed S closed T S* $\cap$ $T = \{\}$
consider $S = \{\} \mid T = \{\} \mid S \neq \{\} \wedge T \neq \{\}$ **by** *auto*
then show $\exists U\ V.$ *open U* $\wedge$ *open V* $\wedge$ $S \subseteq U \wedge T \subseteq V \wedge U \cap V = \{\}$
proof (*cases*)
case *1* **then show** *?thesis* **by** *blast*
next
case *2* **then show** *?thesis* **by** *blast*
next
case *3*
define *U* **where** $U = (\bigcup x \in S.$ *ball x* ((*infdist x T*)/*2*))
have *A*: *open U* **unfolding** *U_def* **by** *auto*
have *infdist x T* > 0 **if** $x \in S$ **for** x
using *H that 3* **by** (*auto intro*!: *infdist_pos_not_in_closed*)
then have *B*: $S \subseteq U$ **unfolding** *U_def* **by** *auto*
define *V* **where** $V = (\bigcup x \in T.$ *ball x* ((*infdist x S*)/*2*))
have *C*: *open V* **unfolding** *V_def* **by** *auto*
have *infdist x S* > 0 **if** $x \in T$ **for** x
using *H that 3* **by** (*auto intro*!: *infdist_pos_not_in_closed*)
then have *D*: $T \subseteq V$ **unfolding** *V_def* **by** *auto*

have (*ball x* ((*infdist x T*)/*2*)) $\cap$ (*ball y* ((*infdist y S*)/*2*)) $= \{\}$ **if** $x \in S$ $y \in T$ **for** x y
proof *auto*
fix z **assume** *H*: *dist x z* $*$ *2* $<$ *infdist x T dist y z* $*$ *2* $<$ *infdist y S*
have *2* $*$ *dist x y* $\leq$ *2* $*$ *dist x z* $+$ *2* $*$ *dist y z*
by *metric*

```
    also have ... < infdist x T + infdist y S
      using H by auto
    finally have dist x y < infdist x T ∨ dist x y < infdist y S
      by auto
    then show False
      using infdist_le[OF ‹x ∈ S›, of y] infdist_le[OF ‹y ∈ T›, of x] by (auto
simp add: dist_commute)
  qed
  then have E: U ∩ V = {}
    unfolding U_def V_def by auto
  show ?thesis
    using A B C D E by blast
 qed
qed

lemma tendsto_infdist [tendsto_intros]:
 assumes f: (f ⟶ l) F
 shows ((λx. infdist (f x) A) ⟶ infdist l A) F
proof (rule tendstoI)
 fix e ::real
 assume e > 0
 from tendstoD[OF f this]
 show eventually (λx. dist (infdist (f x) A) (infdist l A) < e) F
 proof (eventually_elim)
  fix x
  from infdist_triangle[of l A f x] infdist_triangle[of f x A l]
  have dist (infdist (f x) A) (infdist l A) ≤ dist (f x) l
    by (simp add: dist_commute dist_real_def)
  also assume dist (f x) l < e
  finally show dist (infdist (f x) A) (infdist l A) < e .
 qed
qed

lemma continuous_infdist[continuous_intros]:
 assumes continuous F f
 shows continuous F (λx. infdist (f x) A)
 using assms unfolding continuous_def by (rule tendsto_infdist)

lemma continuous_on_infdist [continuous_intros]:
 assumes continuous_on S f
 shows continuous_on S (λx. infdist (f x) A)
using assms unfolding continuous_on by (auto intro: tendsto_infdist)

lemma compact_infdist_le:
 fixes A::'a::heine_borel set
 assumes A ≠ {}
 assumes compact A
 assumes e > 0
 shows compact {x. infdist x A ≤ e}
```

proof −
from *continuous_closed_vimage*[*of* {*0..e*} *λx. infdist x A*]
continuous_infdist[*OF continuous_ident*, *of _ UNIV A*]
have *closed* {*x. infdist x A ≤ e*} **by** (*auto simp*: *vimage_def infdist_nonneg*)
moreover
from *assms* **obtain** *x0 b* **where** *b*: ⋀*x. x ∈ A ⟹ dist x0 x ≤ b closed A*
by (*auto simp*: *compact_eq_bounded_closed bounded_def*)
{
fix *y*
assume *infdist y A ≤ e*
moreover
from *infdist_attains_inf*[*OF* ‹*closed A*› ‹*A ≠ {}*›, *of y*]
obtain *z* **where** *z ∈ A infdist y A = dist y z* **by** *blast*
ultimately
have *dist x0 y ≤ b + e* **using** *b* **by** *metric*
} then
have *bounded* {*x. infdist x A ≤ e*}
by (*auto simp*: *bounded_any_center*[**where** *a=x0*] *intro*!: *exI*[**where** *x=b + e*])
ultimately show *compact* {*x. infdist x A ≤ e*}
by (*simp add*: *compact_eq_bounded_closed*)
qed

3.2.22 Separation between Points and Sets

proposition *separate_point_closed*:
fixes *S* :: *'a*::*heine_borel set*
assumes *closed S* **and** *a ∉ S*
shows ∃ *d*>*0*. ∀ *x*∈*S*. *d ≤ dist a x*
by (*metis assms distance_attains_inf empty_iff gt_ex zero_less_dist_iff*)

proposition *separate_compact_closed*:
fixes *S T* :: *'a*::*heine_borel set*
assumes *compact S*
and *T*: *closed T S ∩ T = {}*
shows ∃ *d*>*0*. ∀ *x*∈*S*. ∀ *y*∈*T*. *d ≤ dist x y*
proof *cases*
assume *S ≠ {} ∧ T ≠ {}*
then have *S ≠ {} T ≠ {}* **by** *auto*
let *?inf = λx. infdist x T*
have *continuous_on S ?inf*
by (*auto intro*!: *continuous_at_imp_continuous_on continuous_infdist continuous_ident*)
then obtain *x* **where** *x*: *x ∈ S* ∀ *y*∈*S*. *?inf x ≤ ?inf y*
using *continuous_attains_inf*[*OF* ‹*compact S*› ‹*S ≠ {}*›] **by** *auto*
then have *0 < ?inf x*
using *T* ‹*T ≠ {}*› *in_closed_iff_infdist_zero* **by** (*auto simp*: *less_le infdist_nonneg*)
moreover have ∀ *x'*∈*S*. ∀ *y*∈*T*. *?inf x ≤ dist x' y*
using *x* **by** (*auto intro*: *order_trans infdist_le*)

ultimately show *?thesis* **by** *auto*
qed (*auto intro*!: *exI*[*of* _ *1*])

proposition *separate_closed_compact*:
fixes *S T* :: *'a*::*heine_borel set*
assumes *S*: *closed S*
and *T*: *compact T*
and *dis*: $S \cap T = \{\}$
shows $\exists\, d{>}0.\ \forall\, x{\in}S.\ \forall\, y{\in}T.\ d \leq \mathit{dist}\ x\ y$
by (*metis separate_compact_closed*[*OF T S*] *dis dist_commute inf_commute*)

proposition *compact_in_open_separated*:
fixes *A*::*'a*::*heine_borel set*
assumes *A*: $A \neq \{\}$ *compact A*
assumes *open B*
assumes $A \subseteq B$
obtains *e* **where** $e > 0$ $\{x.\ \mathit{infdist}\ x\ A \leq e\} \subseteq B$
proof *atomize_elim*
have *closed* $(-B)$ *compact* $A - B \cap A = \{\}$
using *assms* **by** (*auto simp*: *open_Diff compact_eq_bounded_closed*)
from *separate_closed_compact*[*OF this*]
obtain *d'*::*real* **where** *d'*: $d'{>}0$ $\bigwedge x\ y.\ x \notin B \Longrightarrow y \in A \Longrightarrow d' \leq \mathit{dist}\ x\ y$
by *auto*
define *d* **where** $d = d'\ /\ 2$
hence $d{>}0$ $d < d'$ **using** *d'* **by** *auto*
with *d'* **have** *d*: $\bigwedge x\ y.\ x \notin B \Longrightarrow y \in A \Longrightarrow d < \mathit{dist}\ x\ y$
by *force*
show $\exists\, e{>}0.\ \{x.\ \mathit{infdist}\ x\ A \leq e\} \subseteq B$
proof (*rule ccontr*)
assume $\nexists\, e.\ 0 < e \wedge \{x.\ \mathit{infdist}\ x\ A \leq e\} \subseteq B$
with ‹$d > 0$› **obtain** *x* **where** *x*: $\mathit{infdist}\ x\ A \leq d$ $x \notin B$
by *auto*
then show *False*
by (*metis A compact_eq_bounded_closed infdist_attains_inf x d linorder_not_less*)
qed
qed

3.2.23 Uniform Continuity

lemma *uniformly_continuous_onE*:
assumes *uniformly_continuous_on s f* $0 < e$
obtains *d* **where** $d{>}0$ $\bigwedge x\ x'.\ [\![x{\in}s;\ x'{\in}s;\ \mathit{dist}\ x'\ x < d]\!] \Longrightarrow \mathit{dist}\ (f\ x')\ (f\ x) < e$
using *assms*
by (*auto simp*: *uniformly_continuous_on_def*)

lemma *uniformly_continuous_on_sequentially*:
uniformly_continuous_on s f $\longleftrightarrow$ $(\forall\, x\ y.\ (\forall\, n.\ x\ n \in s) \wedge (\forall\, n.\ y\ n \in s) \wedge$
$(\lambda n.\ \mathit{dist}\ (x\ n)\ (y\ n)) \longrightarrow 0 \longrightarrow (\lambda n.\ \mathit{dist}\ (f(x\ n))\ (f(y\ n))) \longrightarrow 0)$ (**is**
?lhs = *?rhs*)

proof
assume *?lhs*
{
fix *x y*
assume *x*: $\forall n.\ x\ n \in s$
and *y*: $\forall n.\ y\ n \in s$
and *xy*: $((\lambda n.\ dist\ (x\ n)\ (y\ n)) \longrightarrow 0)\ sequentially$
{
fix *e* :: *real*
assume $e > 0$
then obtain *d* **where** $d > 0$ **and** *d*: $\forall x{\in}s.\ \forall x'{\in}s.\ dist\ x'\ x < d \longrightarrow dist\ (f\ x')\ (f\ x) < e$
by (*metis* ‹*?lhs*› *uniformly_continuous_onE*)
obtain *N* **where** *N*: $\forall n{\geq}N.\ dist\ (x\ n)\ (y\ n) < d$
using *xy*[*unfolded lim_sequentially dist_norm*] **and** ‹$d{>}0$› **by** *auto*
then have $\exists N.\ \forall n{\geq}N.\ dist\ (f\ (x\ n))\ (f\ (y\ n)) < e$
using *d x y* **by** *blast*
}
then have $((\lambda n.\ dist\ (f(x\ n))\ (f(y\ n))) \longrightarrow 0)\ sequentially$
unfolding *lim_sequentially* **and** *dist_real_def* **by** *auto*
}
then show *?rhs* **by** *auto*
next
assume *?rhs*
{
assume $\neg$ *?lhs*
then obtain *e* **where** $e > 0\ \forall d{>}0.\ \exists x{\in}s.\ \exists x'{\in}s.\ dist\ x'\ x < d \wedge \neg\ dist\ (f\ x')\ (f\ x) < e$
unfolding *uniformly_continuous_on_def* **by** *auto*
then obtain *fa* **where** *fa*:
$\forall x.\ 0 < x \longrightarrow fst\ (fa\ x) \in s \wedge snd\ (fa\ x) \in s \wedge dist\ (fst\ (fa\ x))\ (snd\ (fa\ x)) < x \wedge \neg\ dist\ (f\ (fst\ (fa\ x)))\ (f\ (snd\ (fa\ x))) < e$
using *choice*[*of* $\lambda d\ x.\ d{>}0 \longrightarrow fst\ x \in s \wedge snd\ x \in s \wedge dist\ (snd\ x)\ (fst\ x) < d \wedge \neg\ dist\ (f\ (snd\ x))\ (f\ (fst\ x)) < e$]
by (*auto simp*: *Bex_def dist_commute*)
define *x* **where** $x\ n = fst\ (fa\ (inverse\ (real\ n + 1)))$ **for** *n*
define *y* **where** $y\ n = snd\ (fa\ (inverse\ (real\ n + 1)))$ **for** *n*
have *xyn*: $\forall n.\ x\ n \in s \wedge y\ n \in s$
and *xy0*: $\forall n.\ dist\ (x\ n)\ (y\ n) < inverse\ (real\ n + 1)$
and *fxy*:$\forall n.\ \neg\ dist\ (f\ (x\ n))\ (f\ (y\ n)) < e$
unfolding *x_def* **and** *y_def* **using** *fa*
by *auto*
{
fix *e* :: *real*
assume $e > 0$
then obtain *N* :: *nat* **where** $N \neq 0$ **and** *N*: $0 < inverse\ (real\ N) \wedge inverse\ (real\ N) < e$
unfolding *real_arch_inverse*[*of e*] **by** *auto*
then have $\exists N.\ \forall n{\geq}N.\ dist\ (x\ n)\ (y\ n) < e$

by (*smt* (*verit*, *ccfv_SIG*) *inverse_le_imp_le nat_le_real_less of_nat_le_0_iff xy0*)
}
then have $\forall e{>}0.\ \exists N.\ \forall n{\geq}N.\ dist\ (f\ (x\ n))\ (f\ (y\ n)) < e$
using ‹*?rhs*›[*THEN spec*[**where** *x*=*x*], *THEN spec*[**where** *x*=*y*]] **and** *xyn*
unfolding *lim_sequentially dist_real_def* **by** *auto*
then have *False* **using** *fxy* **and** ‹*e*>*0*› **by** *auto*
}
then show *?lhs*
unfolding *uniformly_continuous_on_def* **by** *blast*
qed

3.2.24 Continuity on a Compact Domain Implies Uniform Continuity

From the proof of the Heine-Borel theorem: Lemma 2 in section 3.7, page 69 of J. C. Burkill and H. Burkill. A Second Course in Mathematical Analysis (CUP, 2002)

lemma *Heine_Borel_lemma*:
assumes *compact S* **and** *Ssub*: $S \subseteq \bigcup \mathcal{G}$ **and** *opn*: $\bigwedge G.\ G \in \mathcal{G} \Longrightarrow open\ G$
obtains *e* **where** $0 < e\ \bigwedge x.\ x \in S \Longrightarrow \exists\, G \in \mathcal{G}.\ ball\ x\ e \subseteq G$
proof −
have *False* **if** *neg*: $\bigwedge e.\ 0 < e \Longrightarrow \exists\, x \in S.\ \forall\, G \in \mathcal{G}.\ \neg\ ball\ x\ e \subseteq G$
proof −
have $\exists\, x \in S.\ \forall\, G \in \mathcal{G}.\ \neg\ ball\ x\ (1\ /\ Suc\ n) \subseteq G$ **for** *n*
using *neg* **by** *simp*
then obtain *f* **where** $\bigwedge n.\ f\ n \in S$ **and** *fG*: $\bigwedge G\ n.\ G \in \mathcal{G} \Longrightarrow \neg\ ball\ (f\ n)\ (1\ /\ Suc\ n) \subseteq G$
by *metis*
then obtain *l r* **where** $l \in S$ *strict_mono r* **and** *to_l*: $(f \circ r) \longrightarrow l$
using ‹*compact S*› *compact_def that* **by** *metis*
then obtain *G* **where** $l \in G\ G \in \mathcal{G}$
using *Ssub* **by** *auto*
then obtain *e* **where** $0 < e$ **and** *e*: $\bigwedge z.\ dist\ z\ l < e \Longrightarrow z \in G$
using *opn open_dist* **by** *blast*
obtain *N1* **where** *N1*: $\bigwedge n.\ n \geq N1 \Longrightarrow dist\ (f\ (r\ n))\ l < e/2$
using *to_l* **apply** (*simp add*: *lim_sequentially*)
using ‹*0 < e*› *half_gt_zero that* **by** *blast*
obtain *N2* **where** *N2*: *of_nat N2 > 2/e*
using *reals_Archimedean2* **by** *blast*
obtain *x* **where** $x \in ball\ (f\ (r\ (max\ N1\ N2)))\ (1\ /\ real\ (Suc\ (r\ (max\ N1\ N2))))$ **and** $x \notin G$
using *fG* [*OF* ‹*G* ∈ $\mathcal{G}$›, *of r* (*max N1 N2*)] **by** *blast*
then have $dist\ (f\ (r\ (max\ N1\ N2)))\ x < 1\ /\ real\ (Suc\ (r\ (max\ N1\ N2)))$
by *simp*
also have ... ≤ $1\ /\ real\ (Suc\ (max\ N1\ N2))$
by (*meson Suc_le_mono* ‹*strict_mono r*› *inverse_of_nat_le nat.discI seq_suble*)
also have ... ≤ $1\ /\ real\ (Suc\ N2)$

```
    by (simp add: field_simps)
  also have ... < e/2
    using N2 ‹0 < e› by (simp add: field_simps)
  finally have dist (f (r (max N1 N2))) x < e/2 .
  moreover have dist (f (r (max N1 N2))) l < e/2
    using N1 max.cobounded1 by blast
  ultimately have dist x l < e
    by metric
  then show ?thesis
    using e ‹x ∉ G› by blast
 qed
 then show ?thesis
  by (meson that)
qed

lemma compact_uniformly_equicontinuous:
 assumes compact S
    and cont: ⋀x e. ⟦x ∈ S; 0 < e⟧
                  ⟹ ∃d. 0 < d ∧
                       (∀f ∈ ℱ. ∀x' ∈ S. dist x' x < d ⟶ dist (f x') (f x) < e)
    and 0 < e
 obtains d where 0 < d
               ⋀f x x'. ⟦f ∈ ℱ; x ∈ S; x' ∈ S; dist x' x < d⟧ ⟹ dist (f x') (f x)
< e
proof −
 obtain d where d_pos: ⋀x e. ⟦x ∈ S; 0 < e⟧ ⟹ 0 < d x e
   and d_dist : ⋀x x' e f. ⟦dist x' x < d x e; x ∈ S; x' ∈ S; 0 < e; f ∈ ℱ⟧ ⟹
dist (f x') (f x) < e
   using cont by metis
 let ?𝒢 = ((λx. ball x (d x (e/2))) ‘ S)
 have Ssub: S ⊆ ⋃ ?𝒢
   using ‹0 < e› d_pos by auto
 then obtain k where 0 < k and k: ⋀x. x ∈ S ⟹ ∃G ∈ ?𝒢. ball x k ⊆ G
   by (rule Heine_Borel_lemma [OF ‹compact S›]) auto
 moreover have dist (f v) (f u) < e if f ∈ ℱ u ∈ S v ∈ S dist v u < k for f u v
 proof −
   obtain G where G ∈ ?𝒢 u ∈ G v ∈ G
     using k that
    by (metis ‹dist v u < k› ‹u ∈ S› ‹0 < k› centre_in_ball subsetD dist_commute
mem_ball)
   then obtain w where w: dist w u < d w (e/2) dist w v < d w (e/2) w ∈ S
     by auto
   with that d_dist have dist (f w) (f v) < e/2
     by (metis ‹0 < e› dist_commute half_gt_zero)
   moreover
   have dist (f w) (f u) < e/2
       using that d_dist w by (metis ‹0 < e› dist_commute divide_pos_pos
zero_less_numeral)
   ultimately show ?thesis
```

using *dist_triangle_half_r* **by** *blast*
qed
ultimately show *?thesis* **using** *that* **by** *blast*
qed

corollary *compact_uniformly_continuous*:
fixes f :: $'a$ :: *metric_space* ⇒ $'b$ :: *metric_space*
assumes f: *continuous_on S f* **and** S: *compact S*
shows *uniformly_continuous_on S f*
using f
unfolding *continuous_on_iff uniformly_continuous_on_def*
by (*force intro*: *compact_uniformly_equicontinuous* [*OF S*, *of* {f}])

3.2.25 Theorems relating continuity and uniform continuity to closures

lemma *continuous_on_closure*:
continuous_on (*closure S*) f ⟷
(∀ x e. x ∈ *closure S* ∧ 0 < e
⟶ (∃ d. 0 < d ∧ (∀ y. y ∈ S ∧ *dist y x* < d ⟶ *dist* (*f y*) (*f x*) < e)))
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs* **then show** *?rhs*
unfolding *continuous_on_iff* **by** (*metis Un_iff closure_def*)
next
assume R [*rule_format*]: *?rhs*
show *?lhs*
proof
fix x **and** e::*real*
assume 0 < e **and** x: x ∈ *closure S*
obtain δ::*real* **where** δ > 0
and δ: ⋀y. ⟦y ∈ S; *dist y x* < δ⟧ ⟹ *dist* (*f y*) (*f x*) < $e/2$
using R [*of x e*/2] ‹0 < e› x **by** *auto*
have *dist* (*f y*) (*f x*) ≤ e **if** y: y ∈ *closure S* **and** *dyx*: *dist y x* < $\delta/2$ **for** y
proof −
obtain δ'::*real* **where** δ' > 0
and δ': ⋀z. ⟦z ∈ S; *dist z y* < δ'⟧ ⟹ *dist* (*f z*) (*f y*) < $e/2$
using R [*of y e*/2] ‹0 < e› y **by** *auto*
obtain z **where** z ∈ S **and** z: *dist z y* < *min* δ' δ / 2
using *closure_approachable y*
by (*metis* ‹0 < δ'› ‹0 < δ› *divide_pos_pos min_less_iff_conj zero_less_numeral*)
have *dist* (*f z*) (*f y*) < $e/2$
using δ' [*OF* ‹z ∈ S›] z ‹0 < δ'› **by** *metric*
moreover have *dist* (*f z*) (*f x*) < $e/2$
using δ[*OF* ‹z ∈ S›] z *dyx* **by** *metric*
ultimately show *?thesis*
by *metric*
qed
then show ∃ d>0. ∀ x'∈*closure S*. *dist* x' x < d ⟶ *dist* (*f* x') (*f x*) ≤ e

```
    by (rule_tac x=δ/2 in exI) (simp add: ‹δ > 0›)
  qed
qed

lemma continuous_on_closure_sequentially:
  fixes f :: 'a::metric_space ⇒ 'b :: metric_space
  shows
   continuous_on (closure S) f ⟷
    (∀ x a. a ∈ closure S ∧ (∀ n. x n ∈ S) ∧ x ⟶⟶ a ⟶ (f ∘ x) ⟶⟶ f a)
   (is ?lhs = ?rhs)
proof −
  have continuous_on (closure S) f ⟷
        (∀ x ∈ closure S. continuous (at x within S) f)
    by (force simp: continuous_on_closure continuous_within_eps_delta)
  also have ... = ?rhs
    by (force simp: continuous_within_sequentially)
  finally show ?thesis .
qed

lemma uniformly_continuous_on_closure:
  fixes f :: 'a::metric_space ⇒ 'b::metric_space
  assumes ucont: uniformly_continuous_on S f
      and cont: continuous_on (closure S) f
    shows uniformly_continuous_on (closure S) f
unfolding uniformly_continuous_on_def
proof (intro allI impI)
  fix e::real
  assume 0 < e
  then obtain d::real
    where d>0
      and d: ⋀x x'. ⟦x∈S; x'∈S; dist x' x < d⟧ ⟹ dist (f x') (f x) < e/3
   using ucont [unfolded uniformly_continuous_on_def, rule_format, of e/3] by
auto
  show ∃ d>0. ∀ x∈closure S. ∀ x'∈closure S. dist x' x < d ⟶ dist (f x') (f x) <
e
  proof (rule exI [where x=d/3], clarsimp simp: ‹d > 0›)
    fix x y
    assume x: x ∈ closure S and y: y ∈ closure S and dyx: dist y x * 3 < d
    obtain d1::real where d1 > 0
          and d1: ⋀w. ⟦w ∈ closure S; dist w x < d1⟧ ⟹ dist (f w) (f x) < e/3
      using cont [unfolded continuous_on_iff, rule_format, of x e/3] ‹0 < e› x
by auto
     obtain x' where x' ∈ S and x': dist x' x < min d1 (d / 3)
       using closure_approachable [of x S]
      by (metis ‹0 < d1› ‹0 < d› divide_pos_pos min_less_iff_conj x zero_less_numeral)
    obtain d2::real where d2 > 0
          and d2: ∀ w ∈ closure S. dist w y < d2 ⟶ dist (f w) (f y) < e/3
      using cont [unfolded continuous_on_iff, rule_format, of y e/3] ‹0 < e› y
by auto
```

```
    obtain y' where y' ∈ S and y': dist y' y < min d2 (d / 3)
      using closure_approachable [of y S]
    by (metis ‹0 < d2› ‹0 < d› divide_pos_pos min_less_iff_conj y zero_less_numeral)
    have dist x' x < d/3 using x' by auto
    then have dist x' y' < d
      using dyx y' by metric
    then have dist (f x') (f y') < e/3
      by (rule d [OF ‹y' ∈ S› ‹x' ∈ S›])
    moreover have dist (f x') (f x) < e/3 using ‹x' ∈ S› closure_subset x' d1
      by (simp add: closure_def)
    moreover have dist (f y') (f y) < e/3 using ‹y' ∈ S› closure_subset y' d2
      by (simp add: closure_def)
    ultimately show dist (f y) (f x) < e by metric
  qed
qed

lemma uniformly_continuous_on_extension_at_closure:
  fixes f::'a::metric_space ⇒ 'b::complete_space
  assumes uc: uniformly_continuous_on X f
  assumes x ∈ closure X
  obtains l where (f ⟶ l) (at x within X)
proof −
  from assms obtain xs where xs: xs ⟶ x ⋀n. xs n ∈ X
    by (auto simp: closure_sequential)

  from uniformly_continuous_on_Cauchy[OF uc LIMSEQ_imp_Cauchy, OF xs]
  obtain l where l: (λn. f (xs n)) ⟶ l
    by atomize_elim (simp only: convergent_eq_Cauchy)

  have (f ⟶ l) (at x within X)
  proof (safe intro!: Lim_within_LIMSEQ)
    fix xs'
    assume ∀ n. xs' n ≠ x ∧ xs' n ∈ X
      and xs': xs' ⟶ x
    then have xs' n ≠ x xs' n ∈ X for n by auto

    from uniformly_continuous_on_Cauchy[OF uc LIMSEQ_imp_Cauchy, OF
‹xs' ⟶ x› ‹xs' _ ∈ X›]
    obtain l' where l': (λn. f (xs' n)) ⟶ l'
      by atomize_elim (simp only: convergent_eq_Cauchy)

    show (λn. f (xs' n)) ⟶ l
    proof (rule tendstoI)
      fix e::real assume e > 0
      define e' where e' ≡ e/2
      have e' > 0 using ‹e > 0› by (simp add: e'_def)

      have ∀F n in sequentially. dist (f (xs n)) l < e'
        by (simp add: ‹0 < e'› l tendstoD)
```

```
    moreover
    from uc[unfolded uniformly_continuous_on_def, rule_format, OF ‹e' > 0›]
    obtain d where d: d > 0 ⋀x x'. x ∈ X ⟹ x' ∈ X ⟹ dist x x' < d ⟹
dist (f x) (f x') < e'
      by auto
    have ∀F n in sequentially. dist (xs n) (xs' n) < d
      by (auto intro!: ‹0 < d› order_tendstoD tendsto_eq_intros xs xs')
    ultimately
    show ∀F n in sequentially. dist (f (xs' n)) l < e
    proof eventually_elim
      case (elim n)
      have dist (f (xs' n)) l ≤ dist (f (xs n)) (f (xs' n)) + dist (f (xs n)) l
        by metric
      also have dist (f (xs n)) (f (xs' n)) < e'
        by (auto intro!: d xs ‹xs' _ ∈ _› elim)
      also note ‹dist (f (xs n)) l < e'›
      also have e' + e' = e by (simp add: e'_def)
      finally show ?case by simp
    qed
  qed
 qed
 thus ?thesis ..
qed

lemma uniformly_continuous_on_extension_on_closure:
 fixes f::'a::metric_space ⇒ 'b::complete_space
 assumes uc: uniformly_continuous_on X f
 obtains g where uniformly_continuous_on (closure X) g ⋀x. x ∈ X ⟹ f x =
g x
   ⋀Y h x. X ⊆ Y ⟹ Y ⊆ closure X ⟹ continuous_on Y h ⟹ (⋀x. x ∈ X
⟹ f x = h x) ⟹ x ∈ Y ⟹ h x = g x
proof −
 from uc have cont_f: continuous_on X f
   by (simp add: uniformly_continuous_imp_continuous)
 obtain y where y: (f ⟶ y x) (at x within X) if x ∈ closure X for x
   using uniformly_continuous_on_extension_at_closure[OF assms] by meson
 let ?g = λx. if x ∈ X then f x else y x

 have uniformly_continuous_on (closure X) ?g
   unfolding uniformly_continuous_on_def
 proof safe
   fix e::real assume e > 0
   define e' where e' ≡ e / 3
   have e' > 0 using ‹e > 0› by (simp add: e'_def)
   from uc[unfolded uniformly_continuous_on_def, rule_format, OF ‹0 < e'›]
   obtain d where d > 0 and d: ⋀x x'. x ∈ X ⟹ x' ∈ X ⟹ dist x' x < d
⟹ dist (f x') (f x) < e'
     by auto
   define d' where d' = d / 3
```

```
    have d' > 0 using ‹d > 0› by (simp add: d'_def)
    show ∃ d>0. ∀ x∈closure X. ∀ x'∈closure X. dist x' x < d ⟶ dist (?g x') (?g
x) < e
    proof (safe intro!: exI[where x=d'] ‹d' > 0›)
      fix x x' assume x: x ∈ closure X and x': x' ∈ closure X and dist: dist x' x
< d'
      then obtain xs xs' where xs: xs ⟶⟶ x ⋀n. xs n ∈ X
        and xs': xs' ⟶⟶ x' ⋀n. xs' n ∈ X
        by (auto simp: closure_sequential)
      have ∀⇩F n in sequentially. dist (xs' n) x' < d'
        and ∀⇩F n in sequentially. dist (xs n) x < d'
        by (auto intro!: ‹0 < d'› order_tendstoD tendsto_eq_intros xs xs')
      moreover
      have (λx. f (xs x)) ⟶⟶ y x if x ∈ closure X x ∉ X xs ⟶⟶ x ⋀n. xs n
∈ X for xs x
        using that not_eventuallyD
        by (force intro!: filterlim_compose[OF y[OF ‹x ∈ closure X›]] simp: filter-
lim_at)
      then have (λx. f (xs' x)) ⟶⟶ ?g x' (λx. f (xs x)) ⟶⟶ ?g x
        using x x'
        by (auto intro!: continuous_on_tendsto_compose[OF cont_f] simp: xs' xs)
      then have ∀⇩F n in sequentially. dist (f (xs' n)) (?g x') < e'
        ∀⇩F n in sequentially. dist (f (xs n)) (?g x) < e'
        by (auto intro!: ‹0 < e'› order_tendstoD tendsto_eq_intros)
      ultimately
      have ∀⇩F n in sequentially. dist (?g x') (?g x) < e
      proof eventually_elim
        case (elim n)
        have dist (?g x') (?g x) ≤
          dist (f (xs' n)) (?g x') + dist (f (xs' n)) (f (xs n)) + dist (f (xs n)) (?g x)
            by (metis add.commute add_le_cancel_left dist_commute dist_triangle
dist_triangle_le)
        also
        from ‹dist (xs' n) x' < d'› ‹dist x' x < d'› ‹dist (xs n) x < d'›
        have dist (xs' n) (xs n) < d unfolding d'_def by metric
        with ‹xs _ ∈ X› ‹xs' _ ∈ X› have dist (f (xs' n)) (f (xs n)) < e'
          by (rule d)
        also note ‹dist (f (xs' n)) (?g x') < e'›
        also note ‹dist (f (xs n)) (?g x) < e'›
        finally show ?case by (simp add: e'_def)
      qed
      then show dist (?g x') (?g x) < e by simp
    qed
  qed
  moreover have f x = ?g x if x ∈ X for x using that by simp
  moreover
  {
    fix Y h x
    assume Y: x ∈ Y X ⊆ Y Y ⊆ closure X and cont_h: continuous_on Y h
```

```
    and extension: (⋀x. x ∈ X ⟹ f x = h x)
  {
    assume x ∉ X
    have x ∈ closure X using Y by auto
    then obtain xs where xs: xs ⟶ x ⋀n. xs n ∈ X
      by (auto simp: closure_sequential)
    from continuous_on_tendsto_compose[OF cont_h xs(1)] xs(2) Y
    have hx: (λx. f (xs x)) ⟶ h x
      by (auto simp: subsetD extension)
    then have (λx. f (xs x)) ⟶ y x
      using ‹x ∉ X› not_eventuallyD xs(2)
      by (force intro!: filterlim_compose[OF y[OF ‹x ∈ closure X›]] simp: filter-
lim_at xs)
    with hx have h x = y x by (rule LIMSEQ_unique)
  } then
  have h x = ?g x
    using extension by auto
 }
 ultimately show ?thesis ..
qed

lemma bounded_uniformly_continuous_image:
 fixes f :: 'a :: heine_borel ⇒ 'b :: heine_borel
 assumes uniformly_continuous_on S f bounded S
 shows bounded(f ‘ S)
  by (metis (no_types, lifting) assms bounded_closure_image compact_closure
compact_continuous_image compact_eq_bounded_closed image_cong uniformly_continuous_imp_continuous
uniformly_continuous_on_extension_on_closure)
```

3.2.26 With Abstract Topology (TODO: move and remove dependency?)

```
lemma openin_contains_ball:
   openin (top_of_set T) S ⟷
    S ⊆ T ∧ (∀x ∈ S. ∃e. 0 < e ∧ ball x e ∩ T ⊆ S)
   (is ?lhs = ?rhs)
proof
 assume ?lhs
 then show ?rhs
  by (metis IntD2 Int_commute Int_lower1 Int_mono inf.idem openE openin_open)
next
 assume ?rhs
 then show ?lhs
  by (smt (verit) open_ball Int_commute Int_iff centre_in_ball in_mono openin_open_Int
openin_subopen)
qed

lemma openin_contains_cball:
   openin (top_of_set T) S ⟷
```

```
      S ⊆ T ∧ (∀x ∈ S. ∃e. 0 < e ∧ cball x e ∩ T ⊆ S)
   (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    by (force simp add: openin_contains_ball intro: exI [where x=_/2])
next
  assume ?rhs
  then show ?lhs
    by (force simp add: openin_contains_ball)
qed
```

3.2.27 Closed Nest

Bounded closed nest property (proof does not use Heine-Borel)

```
lemma bounded_closed_nest:
  fixes S :: nat ⇒ ('a::heine_borel) set
  assumes ⋀n. closed (S n)
      and ⋀n. S n ≠ {}
      and ⋀m n. m ≤ n ⟹ S n ⊆ S m
      and bounded (S 0)
  obtains a where ⋀n. a ∈ S n
proof −
  from assms(2) obtain x where x: ∀n. x n ∈ S n
    using choice[of λn x. x ∈ S n] by auto
  from assms(4,1) have seq_compact (S 0)
    by (simp add: bounded_closed_imp_seq_compact)
  then obtain l r where lr: l ∈ S 0 strict_mono r (x ∘ r) ⟶ l
    using x and assms(3) unfolding seq_compact_def by blast
  have ∀n. l ∈ S n
  proof
    fix n :: nat
    have closed (S n)
      using assms(1) by simp
    moreover have ∀i. (x ∘ r) i ∈ S i
      using x and assms(3) and lr(2) [THEN seq_suble] by auto
    then have ∀i. (x ∘ r) (i + n) ∈ S n
      using assms(3) by (fast intro!: le_add2)
    moreover have (λi. (x ∘ r) (i + n)) ⟶ l
      using lr(3) by (rule LIMSEQ_ignore_initial_segment)
    ultimately show l ∈ S n
      by (metis closed_sequentially)
  qed
  then show ?thesis
    using that by blast
qed
```

Decreasing case does not even need compactness, just completeness.

```
lemma decreasing_closed_nest:
```

```
  fixes S :: nat ⇒ ('a::complete_space) set
  assumes ⋀n. closed (S n)
          ⋀n. S n ≠ {}
          ⋀m n. m ≤ n ⟹ S n ⊆ S m
          ⋀e. e>0 ⟹ ∃n. ∀x∈S n. ∀y∈S n. dist x y < e
  obtains a where ⋀n. a ∈ S n
proof −
  obtain t where t: ∀n. t n ∈ S n
    by (meson assms(2) equals0I)
  {
    fix e :: real
    assume e > 0
    then obtain N where N: ∀x∈S N. ∀y∈S N. dist x y < e
      using assms(4) by blast
    {
      fix m n :: nat
      assume N ≤ m ∧ N ≤ n
      then have t m ∈ S N t n ∈ S N
        using assms(3) t unfolding  subset_eq t by blast+
      then have dist (t m) (t n) < e
        using N by auto
    }
    then have ∃N. ∀m n. N ≤ m ∧ N ≤ n ⟶ dist (t m) (t n) < e
      by auto
  }
  then have Cauchy t
    by (metis metric_CauchyI)
  then obtain l where l:(t ⟶ l) sequentially
    using complete_UNIV unfolding complete_def by auto
  { fix n :: nat
    { fix e :: real
      assume e > 0
      then obtain N :: nat where N: ∀n≥N. dist (t n) l < e
        using l[unfolded lim_sequentially] by auto
      have t (max n N) ∈ S n
        by (meson assms(3) contra_subsetD max.cobounded1 t)
      then have ∃y∈S n. dist y l < e
        using N max.cobounded2 by blast
    }
    then have l ∈ S n
      using closed_approachable[of S n l] assms(1) by auto
  }
  then show ?thesis
    using that by blast
qed
```

Strengthen it to the intersection actually being a singleton.

```
lemma decreasing_closed_nest_sing:
  fixes S :: nat ⇒ 'a::complete_space set
```

```
  assumes ⋀n. closed(S n)
          ⋀n. S n ≠ {}
          ⋀m n. m ≤ n ⟹ S n ⊆ S m
          ⋀e. e>0 ⟹ ∃n. ∀x ∈ (S n). ∀ y∈(S n). dist x y < e
  shows ∃a. ⋂(range S) = {a}
proof −
  obtain a where a: ∀n. a ∈ S n
    using decreasing_closed_nest[of S] using assms by auto
  { fix b
    assume b: b ∈ ⋂(range S)
    { fix e :: real
      assume e > 0
      then have dist a b < e
        using assms(4) and b and a by blast
    }
    then have dist a b = 0
      by (metis dist_eq_0_iff dist_nz less_le)
  }
  with a have ⋂(range S) = {a}
    unfolding image_def by auto
  then show ?thesis ..
qed
```

3.2.28 Making a continuous function avoid some value in a neighbourhood

```
lemma continuous_within_avoid:
  fixes f :: 'a::metric_space ⇒ 'b::t1_space
  assumes continuous (at x within s) f
    and f x ≠ a
  shows ∃e>0. ∀y ∈ s. dist x y < e −−> f y ≠ a
proof −
  obtain U where open U and f x ∈ U and a ∉ U
    using t1_space [OF ‹f x ≠ a›] by fast
  have (f ⟶ f x) (at x within s)
    using assms(1) by (simp add: continuous_within)
  then have eventually (λy. f y ∈ U) (at x within s)
    using ‹open U› and ‹f x ∈ U›
    unfolding tendsto_def by fast
  then have eventually (λy. f y ≠ a) (at x within s)
    using ‹a ∉ U› by (fast elim: eventually_mono)
  then show ?thesis
    using ‹f x ≠ a› by (auto simp: dist_commute eventually_at)
qed

lemma continuous_at_avoid:
  fixes f :: 'a::metric_space ⇒ 'b::t1_space
  assumes continuous (at x) f
    and f x ≠ a
```

shows $\exists e{>}0.\ \forall y.\ dist\ x\ y < e \longrightarrow f\ y \neq a$
using *assms continuous_within_avoid*[*of x UNIV f a*] **by** *simp*

lemma *continuous_on_avoid*:
fixes $f :: 'a{::}metric\_space \Rightarrow 'b{::}t1\_space$
assumes *continuous_on s f*
and $x \in s$
and $f\ x \neq a$
shows $\exists e{>}0.\ \forall y \in s.\ dist\ x\ y < e \longrightarrow f\ y \neq a$
using *assms*(*1*)[*unfolded continuous_on_eq_continuous_within*, *THEN bspec*[**where** *x*=*x*],
OF assms(*2*)] *continuous_within_avoid*[*of x s f a*]
using *assms*(*3*)
by *auto*

lemma *continuous_on_open_avoid*:
fixes $f :: 'a{::}metric\_space \Rightarrow 'b{::}t1\_space$
assumes *continuous_on s f*
and *open s*
and $x \in s$
and $f\ x \neq a$
shows $\exists e{>}0.\ \forall y.\ dist\ x\ y < e \longrightarrow f\ y \neq a$
using *assms*(*1*)[*unfolded continuous_on_eq_continuous_at*[*OF assms*(*2*)], *THEN bspec*[**where** *x*=*x*], *OF assms*(*3*)]
using *continuous_at_avoid*[*of x f a*] *assms*(*4*)
by *auto*

3.2.29 Consequences for Real Numbers

lemma *closed_contains_Inf*:
fixes $S :: real\ set$
shows $S \neq \{\} \Longrightarrow bdd\_below\ S \Longrightarrow closed\ S \Longrightarrow Inf\ S \in S$
by (*metis closure_contains_Inf closure_closed*)

lemma *closed_subset_contains_Inf*:
fixes $A\ C :: real\ set$
shows $closed\ C \Longrightarrow A \subseteq C \Longrightarrow A \neq \{\} \Longrightarrow bdd\_below\ A \Longrightarrow Inf\ A \in C$
by (*metis closure_contains_Inf closure_minimal subset_eq*)

lemma *closed_contains_Sup*:
fixes $S :: real\ set$
shows $S \neq \{\} \Longrightarrow bdd\_above\ S \Longrightarrow closed\ S \Longrightarrow Sup\ S \in S$
by (*subst closure_closed*[*symmetric*], *assumption*, *rule closure_contains_Sup*)

lemma *closed_subset_contains_Sup*:
fixes $A\ C :: real\ set$
shows $closed\ C \Longrightarrow A \subseteq C \Longrightarrow A \neq \{\} \Longrightarrow bdd\_above\ A \Longrightarrow Sup\ A \in C$
by (*metis closure_contains_Sup closure_minimal subset_eq*)

lemma *atLeastAtMost_subset_contains_Inf*:
fixes A :: *real set* **and** a b :: *real*
shows $A \neq \{\} \Longrightarrow a \leq b \Longrightarrow A \subseteq \{a..b\} \Longrightarrow Inf\ A \in \{a..b\}$
by (*rule closed_subset_contains_Inf*)
(*auto intro*: *closed_real_atLeastAtMost intro*!: *bdd_belowI*[*of A a*])

lemma *bounded_real*: *bounded* (S::*real set*) $\longleftrightarrow$ ($\exists\, a.\ \forall\, x{\in}S.\ |x| \leq a$)
by (*simp add*: *bounded_iff*)

lemma *bounded_imp_bdd_above*: *bounded* $S \Longrightarrow$ *bdd_above* (S :: *real set*)
by (*auto simp*: *bounded_def bdd_above_def dist_real_def*)
(*metis abs_le_D1 abs_minus_commute diff_le_eq*)

lemma *bounded_imp_bdd_below*: *bounded* $S \Longrightarrow$ *bdd_below* (S :: *real set*)
by (*auto simp*: *bounded_def bdd_below_def dist_real_def*)
(*metis abs_le_D1 add.commute diff_le_eq*)

lemma *bounded_has_Sup*:
fixes S :: *real set*
assumes *bounded* S
and $S \neq \{\}$
shows $\forall\, x{\in}S.\ x \leq Sup\ S$
and $\forall\, b.\ (\forall\, x{\in}S.\ x \leq b) \longrightarrow Sup\ S \leq b$
proof
show $\forall\, b.\ (\forall\, x{\in}S.\ x \leq b) \longrightarrow Sup\ S \leq b$
using *assms* **by** (*metis cSup_least*)
qed (*metis cSup_upper assms*(*1*) *bounded_imp_bdd_above*)

lemma *Sup_insert*:
fixes S :: *real set*
shows *bounded* $S \Longrightarrow Sup\ (insert\ x\ S) = (if\ S = \{\}\ then\ x\ else\ max\ x\ (Sup\ S))$
by (*auto simp*: *bounded_imp_bdd_above sup_max cSup_insert_If*)

lemma *bounded_has_Inf*:
fixes S :: *real set*
assumes *bounded* S
and $S \neq \{\}$
shows $\forall\, x{\in}S.\ x \geq Inf\ S$
and $\forall\, b.\ (\forall\, x{\in}S.\ x \geq b) \longrightarrow Inf\ S \geq b$
proof
show $\forall\, b.\ (\forall\, x{\in}S.\ x \geq b) \longrightarrow Inf\ S \geq b$
using *assms* **by** (*metis cInf_greatest*)
qed (*metis cInf_lower assms*(*1*) *bounded_imp_bdd_below*)

lemma *Inf_insert*:
fixes S :: *real set*
shows *bounded* $S \Longrightarrow Inf\ (insert\ x\ S) = (if\ S = \{\}\ then\ x\ else\ min\ x\ (Inf\ S))$
by (*auto simp*: *bounded_imp_bdd_below inf_min cInf_insert_If*)

lemma *open_real*:
fixes s :: *real set*
shows *open* $s \longleftrightarrow (\forall x \in s.\ \exists e{>}0.\ \forall x'.\ |x' - x| < e \longrightarrow x' \in s)$
unfolding *open_dist dist_norm* **by** *simp*

lemma *islimpt_approachable_real*:
fixes s :: *real set*
shows x *islimpt* $s \longleftrightarrow (\forall e{>}0.\ \exists x'{\in}\ s.\ x' \neq x \wedge |x' - x| < e)$
unfolding *islimpt_approachable dist_norm* **by** *simp*

lemma *closed_real*:
fixes s :: *real set*
shows *closed* $s \longleftrightarrow (\forall x.\ (\forall e{>}0.\ \ \exists x' \in s.\ x' \neq x \wedge |x' - x| < e) \longrightarrow x \in s)$
unfolding *closed_limpt islimpt_approachable dist_norm* **by** *simp*

lemma *continuous_at_real_range*:
fixes f :: *'a::real_normed_vector* $\Rightarrow$ *real*
shows *continuous* (*at* x) $f \longleftrightarrow (\forall e{>}0.\ \exists d{>}0.\ \forall x'.\ norm(x' - x) < d \longrightarrow |f\ x' - f\ x| < e)$
by (*metis* (*mono_tags*, *opaque_lifting*) *LIM_eq continuous_within norm_eq_zero real_norm_def right_minus_eq*)

lemma *continuous_on_real_range*:
fixes f :: *'a::real_normed_vector* $\Rightarrow$ *real*
shows *continuous_on* s $f \longleftrightarrow$
$(\forall x \in s.\ \forall e{>}0.\ \exists d{>}0.\ (\forall x' \in s.\ norm(x' - x) < d \longrightarrow |f\ x' - f\ x| < e))$
unfolding *continuous_on_iff dist_norm* **by** *simp*

lemma *continuous_on_closed_Collect_le*:
fixes $f\ g$ :: *'a::topological_space* $\Rightarrow$ *real*
assumes f: *continuous_on* s f **and** g: *continuous_on* s g **and** s: *closed* s
shows *closed* $\{x \in s.\ f\ x \leq g\ x\}$
proof −
have *closed* $((\lambda x.\ g\ x - f\ x) -` \{0..\} \cap s)$
using *closed_real_atLeast continuous_on_diff* [*OF* g f]
by (*simp add*: *continuous_on_closed_vimage* [*OF* s])
also have $((\lambda x.\ g\ x - f\ x) -` \{0..\} \cap s) = \{x{\in}s.\ f\ x \leq g\ x\}$
by *auto*
finally show *?thesis* **.**
qed

lemma *continuous_le_on_closure*:
fixes a::*real*
assumes f: *continuous_on* (*closure* s) f
and x: $x \in closure(s)$
and *xlo*: $\bigwedge x.\ x \in s \Longrightarrow f(x) \leq a$
shows $f(x) \leq a$
using *image_closure_subset* [*OF* f, **where** $T{=}\ \{x.\ x \leq a\}$] *assms*
continuous_on_closed_Collect_le[*of UNIV* $\lambda x.\ x$ $\lambda x.\ a$]

by *auto*

lemma *continuous_ge_on_closure*:
fixes *a*::*real*
assumes *f*: *continuous_on* (*closure s*) *f*
and *x*: $x \in closure(s)$
and *xlo*: $\bigwedge x.\ x \in s \Longrightarrow f(x) \geq a$
shows $f(x) \geq a$
using *image_closure_subset* [*OF f*, **where** $T = \{x.\ a \leq x\}$] *assms*
continuous_on_closed_Collect_le[*of UNIV* $\lambda x.\ a$ $\lambda x.\ x$]
by *auto*

3.2.30 The infimum of the distance between two sets

definition *setdist* :: *'a*::*metric_space set* ⇒ *'a set* ⇒ *real* **where**
setdist s t ≡
(*if* $s = \{\} \vee t = \{}$ *then* 0
else Inf $\{dist\ x\ y |\ x\ y.\ x \in s \wedge y \in t\}$)

lemma *setdist_empty1* [*simp*]: *setdist* $\{\}$ $t = 0$
by (*simp add*: *setdist_def*)

lemma *setdist_empty2* [*simp*]: *setdist t* $\{\} = 0$
by (*simp add*: *setdist_def*)

lemma *setdist_pos_le* [*simp*]: $0 \leq setdist\ s\ t$
by (*auto simp*: *setdist_def ex_in_conv* [*symmetric*] *intro*: *cInf_greatest*)

lemma *le_setdistI*:
assumes $s \neq \{\}$ $t \neq \{\}$ $\bigwedge x\ y.\ [\![x \in s;\ y \in t]\!] \Longrightarrow d \leq dist\ x\ y$
shows $d \leq setdist\ s\ t$
using *assms*
by (*auto simp*: *setdist_def Set.ex_in_conv* [*symmetric*] *intro*: *cInf_greatest*)

lemma *setdist_le_dist*: $[\![x \in s;\ y \in t]\!] \Longrightarrow setdist\ s\ t \leq dist\ x\ y$
unfolding *setdist_def*
by (*auto intro*!: *bdd_belowI* [**where** *m*=*0*] *cInf_lower*)

lemma *le_setdist_iff*:
$d \leq setdist\ S\ T \longleftrightarrow$
$(\forall x \in S.\ \forall y \in T.\ d \leq dist\ x\ y) \wedge (S = \{\} \vee T = \{\} \longrightarrow d \leq 0)$
by (*smt* (*verit*) *le_setdistI setdist_def setdist_le_dist*)

lemma *setdist_ltE*:
assumes $setdist\ S\ T < b$ $S \neq \{\}$ $T \neq \{\}$
obtains *x y* **where** $x \in S$ $y \in T$ $dist\ x\ y < b$
using *assms*
by (*auto simp*: *not_le* [*symmetric*] *le_setdist_iff*)

lemma *setdist_refl*: *setdist S S = 0*
by (*metis dist_eq_0_iff ex_in_conv order_antisym setdist_def setdist_le_dist setdist_pos_le*)

lemma *setdist_sym*: *setdist S T = setdist T S*
by (*force simp*: *setdist_def dist_commute intro*!: *arg_cong* [**where** *f=Inf*])

lemma *setdist_triangle*: *setdist S T ≤ setdist S {a} + setdist {a} T*
proof (*cases S = {} ∨ T = {}*)
case *True* **then show** *?thesis*
using *setdist_pos_le* **by** *fastforce*
next
case *False*
then have *⋀x. x ∈ S ⟹ setdist S T − dist x a ≤ setdist {a} T*
using *dist_self dist_triangle3 empty_not_insert le_setdist_iff setdist_le_dist singleton_iff*
by (*smt* (*verit*, *best*) *dist_self dist_triangle3 empty_not_insert le_setdist_iff setdist_le_dist singleton_iff*)
then have *setdist S T − setdist {a} T ≤ setdist S {a}*
using *False* **by** (*fastforce intro*: *le_setdistI*)
then show *?thesis*
by (*simp add*: *algebra_simps*)
qed

lemma *setdist_singletons* [*simp*]: *setdist {x} {y} = dist x y*
by (*simp add*: *setdist_def*)

lemma *setdist_Lipschitz*: *|setdist {x} S − setdist {y} S| ≤ dist x y*
apply (*subst setdist_singletons* [*symmetric*])
by (*metis abs_diff_le_iff diff_le_eq setdist_triangle setdist_sym*)

lemma *continuous_at_setdist* [*continuous_intros*]: *continuous* (*at x*) (*λy.* (*setdist {y} S*))
by (*force simp*: *continuous_at_eps_delta dist_real_def intro*: *le_less_trans* [*OF setdist_Lipschitz*])

lemma *continuous_on_setdist* [*continuous_intros*]: *continuous_on T* (*λy.* (*setdist {y} S*))
by (*metis continuous_at_setdist continuous_at_imp_continuous_on*)

lemma *uniformly_continuous_on_setdist*: *uniformly_continuous_on T* (*λy.* (*setdist {y} S*))
by (*force simp*: *uniformly_continuous_on_def dist_real_def intro*: *le_less_trans* [*OF setdist_Lipschitz*])

lemma *setdist_subset_right*: *⟦T ≠ {}; T ⊆ u⟧ ⟹ setdist S u ≤ setdist S T*
by (*smt* (*verit*, *best*) *in_mono le_setdist_iff*)

lemma *setdist_subset_left*: *⟦S ≠ {}; S ⊆ T⟧ ⟹ setdist T u ≤ setdist S u*

by (*metis setdist_subset_right setdist_sym*)

lemma *setdist_closure_1* [*simp*]: *setdist* (*closure S*) *T* = *setdist S T*
proof (*cases S* = {} ∨ *T* = {})
case *True* **then show** *?thesis* **by** *force*
next
case *False*
{ fix *y*
assume *y* ∈ *T*
have *continuous_on* (*closure S*) (λ*a*. *dist a y*)
by (*auto simp*: *continuous_intros dist_norm*)
then have ∗: ⋀*x*. *x* ∈ *closure S* ⟹ *setdist S T* ≤ *dist x y*
by (*fast intro*: *setdist_le_dist* ‹*y* ∈ *T*› *continuous_ge_on_closure*)
} then
show *?thesis*
by (*metis False antisym closure_eq_empty closure_subset le_setdist_iff setdist_subset_left*)
qed

lemma *setdist_closure_2* [*simp*]: *setdist T* (*closure S*) = *setdist T S*
by (*metis setdist_closure_1 setdist_sym*)

lemma *setdist_eq_0I*: ⟦*x* ∈ *S*; *x* ∈ *T*⟧ ⟹ *setdist S T* = *0*
by (*metis antisym dist_self setdist_le_dist setdist_pos_le*)

lemma *setdist_unique*:
⟦*a* ∈ *S*; *b* ∈ *T*; ⋀*x y*. *x* ∈ *S* ∧ *y* ∈ *T* ==> *dist a b* ≤ *dist x y*⟧
⟹ *setdist S T* = *dist a b*
by (*force simp add*: *setdist_le_dist le_setdist_iff intro*: *antisym*)

lemma *setdist_le_sing*: *x* ∈ *S* ==> *setdist S T* ≤ *setdist* {*x*} *T*
using *setdist_subset_left* **by** *auto*

lemma *infdist_eq_setdist*: *infdist x A* = *setdist* {*x*} *A*
by (*simp add*: *infdist_def setdist_def Setcompr_eq_image*)

lemma *setdist_eq_infdist*: *setdist A B* = (*if A* = {} *then 0 else INF a*∈*A*. *infdist a B*)
proof −
have *Inf* {*dist x y* |*x y*. *x* ∈ *A* ∧ *y* ∈ *B*} = (*INF x*∈*A*. *Inf* (*dist x* ‘ *B*))
if *b* ∈ *B a* ∈ *A* **for** *a b*
proof (*rule order_antisym*)
have *Inf* {*dist x y* |*x y*. *x* ∈ *A* ∧ *y* ∈ *B*} ≤ *Inf* (*dist x* ‘ *B*)
if *b* ∈ *B a* ∈ *A x* ∈ *A* **for** *x*
proof −
have ⋀*b*′. *b*′ ∈ *B* ⟹ *Inf* {*dist x y* |*x y*. *x* ∈ *A* ∧ *y* ∈ *B*} ≤ *dist x b*′
by (*metis* (*mono_tags*, *lifting*) *ex_in_conv setdist_def setdist_le_dist that*(*3*))
then show *?thesis*

```
      by (smt (verit) cINF_greatest ex_in_conv ‹b ∈ B›)
    qed
    then show Inf {dist x y |x y. x ∈ A ∧ y ∈ B} ≤ (INF x∈A. Inf (dist x ‘ B))
      by (metis (mono_tags, lifting) cINF_greatest emptyE that)
  next
    have ∗: ⋀x y. ⟦b ∈ B; a ∈ A; x ∈ A; y ∈ B⟧ ⟹ ∃a∈A. Inf (dist a ‘ B) ≤
dist x y
      by (meson bdd_below_image_dist cINF_lower)
    show (INF x∈A. Inf (dist x ‘ B)) ≤ Inf {dist x y |x y. x ∈ A ∧ y ∈ B}
    proof (rule conditionally_complete_lattice_class.cInf_mono)
      show bdd_below ((λx. Inf (dist x ‘ B)) ‘ A)
      by (metis (no_types, lifting) bdd_belowI2 ex_in_conv infdist_def infdist_nonneg
that(1))
    qed (use that in ‹auto simp: ∗›)
  qed
  then show ?thesis
    by (auto simp: setdist_def infdist_def)
qed

lemma infdist_mono:
  assumes A ⊆ B A ≠ {}
  shows infdist x B ≤ infdist x A
  by (simp add: assms infdist_eq_setdist setdist_subset_right)

lemma infdist_singleton [simp]: infdist x {y} = dist x y
  by (simp add: infdist_eq_setdist)

proposition setdist_attains_inf:
  assumes compact B B ≠ {}
  obtains y where y ∈ B setdist A B = infdist y A
proof (cases A = {})
  case True
  then show thesis
    by (metis assms diameter_compact_attained infdist_def setdist_def that)
next
  case False
  obtain y where y ∈ B and min: ⋀y'. y' ∈ B ⟹ infdist y A ≤ infdist y' A
    by (metis continuous_attains_inf [OF assms continuous_on_infdist] continu-
ous_on_id)
  show thesis
  proof
    have setdist A B = (INF y∈B. infdist y A)
      by (metis ‹B ≠ {}› setdist_eq_infdist setdist_sym)
    also have ... = infdist y A
    proof (rule order_antisym)
      show (INF y∈B. infdist y A) ≤ infdist y A
        by (meson ‹y ∈ B› bdd_belowI2 cInf_lower image_eqI infdist_nonneg)
    next
      show infdist y A ≤ (INF y∈B. infdist y A)
```

```
      by (simp add: ‹B ≠ {}› cINF_greatest min)
    qed
    finally show setdist A B = infdist y A .
  qed (fact ‹y ∈ B›)
qed

end
```

3.3 Elementary Normed Vector Spaces

```
theory Elementary_Normed_Spaces
  imports
  HOL−Library.FuncSet
  Elementary_Metric_Spaces Cartesian_Space
  Connected
begin
```

3.3.1 Orthogonal Transformation of Balls

3.3.2 Various Lemmas Combining Imports

```
lemma open_sums:
  fixes T :: ('b::real_normed_vector) set
  assumes open S ∨ open T
  shows open (⋃x∈ S. ⋃y ∈ T. {x + y})
  using assms
proof
  assume S: open S
  show ?thesis
  proof (clarsimp simp: open_dist)
    fix x y
    assume x ∈ S y ∈ T
    with S obtain e where e > 0 and e: ⋀x'. dist x' x < e ⟹ x' ∈ S
      by (auto simp: open_dist)
    then have ⋀z. dist z (x + y) < e ⟹ ∃ x∈S. ∃ y∈T. z = x + y
      by (metis ‹y ∈ T› diff_add_cancel dist_add_cancel2)
    then show ∃ e>0. ∀ z. dist z (x + y) < e ⟶ (∃ x∈S. ∃ y∈T. z = x + y)
      using ‹0 < e› ‹x ∈ S› by blast
  qed
next
  assume T: open T
  show ?thesis
  proof (clarsimp simp: open_dist)
    fix x y
    assume x ∈ S y ∈ T
    with T obtain e where e > 0 and e: ⋀x'. dist x' y < e ⟹ x' ∈ T
      by (auto simp: open_dist)
    then have ⋀z. dist z (x + y) < e ⟹ ∃ x∈S. ∃ y∈T. z = x + y
    by (metis ‹x ∈ S› add_diff_cancel_left' add_diff_eq diff_diff_add dist_norm)
```

```
    then show ∃e>0. ∀z. dist z (x + y) < e ⟶ (∃x∈S. ∃y∈T. z = x + y)
      using ‹0 < e› ‹y ∈ T› by blast
  qed
qed

lemma image_orthogonal_transformation_ball:
  fixes f :: 'a::euclidean_space ⇒ 'a
  assumes orthogonal_transformation f
  shows f ‘ ball x r = ball (f x) r
proof (intro equalityI subsetI)
  fix y assume y ∈ f ‘ ball x r
  with assms show y ∈ ball (f x) r
    by (auto simp: orthogonal_transformation_isometry)
next
  fix y assume y: y ∈ ball (f x) r
  then obtain z where z: y = f z
    using assms orthogonal_transformation_surj by blast
  with y assms show y ∈ f ‘ ball x r
    by (auto simp: orthogonal_transformation_isometry)
qed

lemma image_orthogonal_transformation_cball:
  fixes f :: 'a::euclidean_space ⇒ 'a
  assumes orthogonal_transformation f
  shows f ‘ cball x r = cball (f x) r
proof (intro equalityI subsetI)
  fix y assume y ∈ f ‘ cball x r
  with assms show y ∈ cball (f x) r
    by (auto simp: orthogonal_transformation_isometry)
next
  fix y assume y: y ∈ cball (f x) r
  then obtain z where z: y = f z
    using assms orthogonal_transformation_surj by blast
  with y assms show y ∈ f ‘ cball x r
    by (auto simp: orthogonal_transformation_isometry)
qed
```

3.3.3 Support

```
definition (in monoid_add) support_on :: 'b set ⇒ ('b ⇒ 'a) ⇒ 'b set
  where support_on S f = {x∈S. f x ≠ 0}

lemma in_support_on: x ∈ support_on S f ⟷ x ∈ S ∧ f x ≠ 0
  by (simp add: support_on_def)

lemma support_on_simps[simp]:
  support_on {} f = {}
  support_on (insert x S) f =
    (if f x = 0 then support_on S f else insert x (support_on S f))
```

support_on $(S \cup T)$ f = *support_on* S f $\cup$ *support_on* T f
support_on $(S \cap T)$ f = *support_on* S f $\cap$ *support_on* T f
support_on $(S - T)$ f = *support_on* S f $-$ *support_on* T f
support_on $(f \text{ ` } S)$ $g$ = $f$ ` (*support_on* S $(g \circ f)$)
unfolding *support_on_def* **by** *auto*

lemma *support_on_cong*:
$(\bigwedge x.\ x \in S \Longrightarrow f\ x = 0 \longleftrightarrow g\ x = 0) \Longrightarrow$ *support_on* S f = *support_on* S g
by (*auto simp*: *support_on_def*)

lemma *support_on_if*: $a \neq 0 \Longrightarrow$ *support_on* A $(\lambda x.\ \textit{if}\ P\ x\ \textit{then}\ a\ \textit{else}\ 0) = \{x{\in}A.\ P\ x\}$
by (*auto simp*: *support_on_def*)

lemma *support_on_if_subset*: *support_on* A $(\lambda x.\ \textit{if}\ P\ x\ \textit{then}\ a\ \textit{else}\ 0) \subseteq \{x \in A.\ P\ x\}$
by (*auto simp*: *support_on_def*)

lemma *finite_support*[*intro*]: *finite* $S \Longrightarrow$ *finite* (*support_on* S f)
unfolding *support_on_def* **by** *auto*

definition (**in** *comm_monoid_add*) *supp_sum* :: $('b \Rightarrow 'a) \Rightarrow 'b\ set \Rightarrow 'a$
where *supp_sum* f S = $(\sum x{\in}$*support_on* S f. $f\ x)$

lemma *supp_sum_empty*[*simp*]: *supp_sum* f $\{\}$ = 0
unfolding *supp_sum_def* **by** *auto*

lemma *supp_sum_insert*[*simp*]:
finite (*support_on* S f) $\Longrightarrow$
supp_sum f (*insert* x S) = (*if* $x \in S$ *then supp_sum* f S *else* $f\ x$ + *supp_sum* f S)
by (*simp add*: *supp_sum_def in_support_on insert_absorb*)

lemma *supp_sum_divide_distrib*: *supp_sum* f A / (r::'a::*field*) = *supp_sum* $(\lambda n.\ f\ n\ /\ r)$ A
by (*cases* $r = 0$)
(*auto simp*: *supp_sum_def sum_divide_distrib intro*!: *sum.cong support_on_cong*)

3.3.4 Intervals

lemma *image_affinity_interval*:
fixes c :: 'a::*ordered_real_vector*
shows $((\lambda x.\ m *_R x + c)$ ` $\{a..b\}) =$
(*if* $\{a..b\}=\{\}$ *then* $\{\}$
else if $0 \leq m$ *then* $\{m *_R a + c\ ..\ m *_R b + c\}$
else $\{m *_R b + c\ ..\ m *_R a + c\}$)
(**is** *?lhs* = *?rhs*)
proof (*cases* $m{=}0$)

```
  case True
  then show ?thesis
    by force
next
  case False
  show ?thesis
  proof
    show ?lhs ⊆ ?rhs
      by (auto simp: scaleR_left_mono scaleR_left_mono_neg)
    show ?rhs ⊆ ?lhs
    proof (clarsimp, intro conjI impI subsetI)
      show ⟦0 ≤ m; a ≤ b; x ∈ {m *R a + c..m *R b + c}⟧
           ⟹ x ∈ (λx. m *R x + c) ` {a..b} for x
        using False
        by (rule_tac x=inverse m *R (x−c) in image_eqI)
          (auto simp: pos_le_divideR_eq pos_divideR_le_eq le_diff_eq diff_le_eq)
      show ⟦¬ 0 ≤ m; a ≤ b;  x ∈ {m *R b + c..m *R a + c}⟧
           ⟹ x ∈ (λx. m *R x + c) ` {a..b} for x
        by (rule_tac x=inverse m *R (x−c) in image_eqI)
            (auto simp add: neg_le_divideR_eq neg_divideR_le_eq le_diff_eq
diff_le_eq)
    qed
  qed
qed
```

3.3.5 Limit Points

```
lemma islimpt_ball:
  fixes x y :: 'a::{real_normed_vector,perfect_space}
  shows y islimpt ball x e ⟷ 0 < e ∧ y ∈ cball x e
  (is ?lhs ⟷ ?rhs)
proof
  show ?rhs if ?lhs
  proof
    {
      assume e ≤ 0
      then have *: ball x e = {}
        using ball_eq_empty[of x e] by auto
      have False using ‹?lhs›
        unfolding * using islimpt_EMPTY[of y] by auto
    }
    then show e > 0 by (metis not_less)
    show y ∈ cball x e
      using closed_cball[of x e] islimpt_subset[of y ball x e cball x e]
        ball_subset_cball[of x e] ‹?lhs›
      unfolding closed_limpt by auto
  qed
  show ?lhs if ?rhs
  proof −
```

```
  from that have e > 0 by auto
  {
   fix d :: real
   assume d > 0
   have ∃x'∈ball x e. x' ≠ y ∧ dist x' y < d
   proof (cases d ≤ dist x y)
    case True
    then show ?thesis
    proof (cases x = y)
     case True
     then have False
      using ‹d ≤ dist x y› ‹d>0› by auto
     then show ?thesis
      by auto
    next
     case False
     have dist x (y − (d / (2 * dist y x)) *R (y − x)) =
       norm (x − y + (d / (2 * norm (y − x))) *R (y − x))
      unfolding mem_cball mem_ball dist_norm diff_diff_eq2 diff_add_eq[symmetric]
      by auto
     also have ... = |− 1 + d / (2 * norm (x − y))| * norm (x − y)
      using scaleR_left_distrib[of − 1 d / (2 * norm (y − x)), symmetric, of
y − x]
      unfolding scaleR_minus_left scaleR_one
      by (auto simp: norm_minus_commute)
     also have ... = |− norm (x − y) + d / 2|
      unfolding abs_mult_pos[of norm (x − y), OF norm_ge_zero[of x − y]]
      unfolding distrib_right using ‹x≠y› by auto
     also have ... ≤ e − d/2 using ‹d ≤ dist x y› and ‹d>0› and ‹?rhs›
      by (auto simp: dist_norm)
     finally have y − (d / (2 * dist y x)) *R (y − x) ∈ ball x e using ‹d>0›
      by auto
     moreover
     have (d / (2*dist y x)) *R (y − x) ≠ 0
      using ‹x≠y›[unfolded dist_nz] ‹d>0› unfolding scaleR_eq_0_iff
      by (auto simp: dist_commute)
     moreover
     have dist (y − (d / (2 * dist y x)) *R (y − x)) y < d
      using ‹0 < d› by (fastforce simp: dist_norm)
     ultimately show ?thesis
      by (rule_tac x = y − (d / (2*dist y x)) *R (y − x) in bexI) auto
    qed
   next
    case False
    then have d > dist x y by auto
    show ∃x' ∈ ball x e. x' ≠ y ∧ dist x' y < d
    proof (cases x = y)
     case True
     obtain z where z: z ≠ y dist z y < min e d
```

```
        using perfect_choose_dist[of min e d y]
        using ‹d > 0› ‹e>0› by auto
      show ?thesis
        by (metis True z dist_commute mem_ball min_less_iff_conj)
    next
      case False
      then show ?thesis
        using ‹d>0› ‹d > dist x y› ‹?rhs› by force
    qed
  qed
 }
 then show ?thesis
  unfolding mem_cball islimpt_approachable mem_ball by auto
 qed
qed

lemma closure_ball_lemma:
 fixes x y :: 'a::real_normed_vector
 assumes x ≠ y
 shows y islimpt ball x (dist x y)
proof (rule islimptI)
 fix T
 assume y ∈ T open T
 then obtain r where 0 < r ∀ z. dist z y < r ⟶ z ∈ T
  unfolding open_dist by fast
 — choose point between x and y, within distance r of y.
 define k where k = min 1 (r / (2 * dist x y))
 define z where z = y + scaleR k (x - y)
 have z_def2: z = x + scaleR (1 - k) (y - x)
  unfolding z_def by (simp add: algebra_simps)
 have dist z y < r
  unfolding z_def k_def using ‹0 < r›
  by (simp add: dist_norm min_def)
 then have z ∈ T
  using ‹∀ z. dist z y < r ⟶ z ∈ T› by simp
 have dist x z < dist x y
  using ‹0 < r› assms by (simp add: z_def2 k_def dist_norm norm_minus_commute)

 then have z ∈ ball x (dist x y)
  by simp
 have z ≠ y
  unfolding z_def k_def using ‹x ≠ y› ‹0 < r›
  by (simp add: min_def)
 show ∃ z∈ball x (dist x y). z ∈ T ∧ z ≠ y
  using ‹z ∈ ball x (dist x y)› ‹z ∈ T› ‹z ≠ y›
  by fast
qed
```

3.3.6 Balls and Spheres in Normed Spaces

lemma *mem_ball_0* [*simp*]: $x \in ball\ 0\ e \longleftrightarrow norm\ x < e$
for $x :: 'a$::*real_normed_vector*
by *simp*

lemma *mem_cball_0* [*simp*]: $x \in cball\ 0\ e \longleftrightarrow norm\ x \leq e$
for $x :: 'a$::*real_normed_vector*
by *simp*

lemma *closure_ball* [*simp*]:
fixes $x :: 'a$::*real_normed_vector*
assumes $0 < e$
shows $closure\ (ball\ x\ e) = cball\ x\ e$
proof
show $closure\ (ball\ x\ e) \subseteq cball\ x\ e$
using *closed_cball closure_minimal* **by** *blast*
have $\bigwedge y.\ dist\ x\ y < e \vee dist\ x\ y = e \Longrightarrow y \in closure\ (ball\ x\ e)$
by (*metis Un_iff assms closure_ball_lemma closure_def dist_eq_0_iff mem_Collect_eq mem_ball*)
then show $cball\ x\ e \subseteq closure\ (ball\ x\ e)$
by *force*
qed

lemma *mem_sphere_0* [*simp*]: $x \in sphere\ 0\ e \longleftrightarrow norm\ x = e$
for $x :: 'a$::*real_normed_vector*
by *simp*

lemma *interior_cball* [*simp*]:
fixes $x :: 'a$::{*real_normed_vector*, *perfect_space*}
shows $interior\ (cball\ x\ e) = ball\ x\ e$
proof (*cases* $e \geq 0$)
case *False* **note** *cs* = *this*
from *cs* **have** *null*: $ball\ x\ e = \{\}$
using *ball_empty*[*of e x*] **by** *auto*
moreover
have $cball\ x\ e = \{\}$
proof (*rule equals0I*)
fix *y*
assume $y \in cball\ x\ e$
then show *False*
by (*metis ball_eq_empty null cs dist_eq_0_iff dist_le_zero_iff empty_subsetI mem_cball*
subset_antisym subset_ball)
qed
then have $interior\ (cball\ x\ e) = \{\}$
using *interior_empty* **by** *auto*
ultimately show *?thesis* **by** *blast*
next

```
  case True note cs = this
  have ball x e ⊆ cball x e
    using ball_subset_cball by auto
  moreover
  {
    fix S y
    assume as: S ⊆ cball x e open S y∈S
    then obtain d where d>0 and d: ∀ x'. dist x' y < d ⟶ x' ∈ S
      unfolding open_dist by blast
    then obtain xa where xa_y: xa ≠ y and xa: dist xa y < d
      using perfect_choose_dist [of d] by auto
    have xa ∈ S
      using d[THEN spec[where x = xa]]
      using xa by (auto simp: dist_commute)
    then have xa_cball: xa ∈ cball x e
      using as(1) by auto
    then have y ∈ ball x e
    proof (cases x = y)
      case True
     then have e > 0 using cs order.order_iff_strict xa_cball xa_y by fastforce
      then show y ∈ ball x e
        using ‹x = y › by simp
    next
      case False
      have dist (y + (d / 2 / dist y x) *R (y − x)) y < d
        unfolding dist_norm
        using ‹d>0› norm_ge_zero[of y − x] ‹x ≠ y› by auto
      then have ∗: y + (d / 2 / dist y x) *R (y − x) ∈ cball x e
        using d as(1)[unfolded subset_eq] by blast
      have y − x ≠ 0 using ‹x ≠ y› by auto
      hence ∗∗:d / (2 ∗ norm (y − x)) > 0
        unfolding zero_less_norm_iff[symmetric] using ‹d>0› by auto
      have dist (y + (d / 2 / dist y x) *R (y − x)) x =
        norm (y + (d / (2 ∗ norm (y − x))) *R y − (d / (2 ∗ norm (y − x))) *R
x − x)
        by (auto simp: dist_norm algebra_simps)
      also have ... = norm ((1 + d / (2 ∗ norm (y − x))) *R (y − x))
        by (auto simp: algebra_simps)
      also have ... = |1 + d / (2 ∗ norm (y − x))| ∗ norm (y − x)
        using ∗∗ by auto
      also have ... = (dist y x) + d/2
        using ∗∗ by (auto simp: distrib_right dist_norm)
      finally have e ≥ dist x y +d/2
        using ∗[unfolded mem_cball] by (auto simp: dist_commute)
      then show y ∈ ball x e
        unfolding mem_ball using ‹d>0› by auto
    qed
  }
  then have ∀ S ⊆ cball x e. open S ⟶ S ⊆ ball x e
```

```
    by auto
  ultimately show ?thesis
    using interior_unique[of ball x e cball x e]
    using open_ball[of x e]
    by auto
qed

lemma frontier_ball [simp]:
  fixes a :: 'a::real_normed_vector
  shows 0 < e ⟹ frontier (ball a e) = sphere a e
  by (force simp: frontier_def)

lemma frontier_cball [simp]:
  fixes a :: 'a::{real_normed_vector, perfect_space}
  shows frontier (cball a e) = sphere a e
  by (force simp: frontier_def)

corollary compact_sphere [simp]:
  fixes a :: 'a::{real_normed_vector,perfect_space,heine_borel}
  shows compact (sphere a r)
using compact_frontier [of cball a r] by simp

corollary bounded_sphere [simp]:
  fixes a :: 'a::{real_normed_vector,perfect_space,heine_borel}
  shows bounded (sphere a r)
by (simp add: compact_imp_bounded)

corollary closed_sphere [simp]:
  fixes a :: 'a::{real_normed_vector,perfect_space,heine_borel}
  shows closed (sphere a r)
by (simp add: compact_imp_closed)

lemma image_add_ball [simp]:
  fixes a :: 'a::real_normed_vector
  shows (+) b ` ball a r = ball (a+b) r
proof −
  { fix x :: 'a
    assume dist (a + b) x < r
    moreover
    have b + (x − b) = x
      by simp
    ultimately have x ∈ (+) b ` ball a r
      by (metis add.commute dist_add_cancel image_eqI mem_ball) }
  then show ?thesis
    by (auto simp: add.commute)
qed

lemma image_add_cball [simp]:
  fixes a :: 'a::real_normed_vector
```

shows $(+)\ b$ ‘ $cball\ a\ r = cball\ (a+b)\ r$
proof −
have $\bigwedge x.\ dist\ (a + b)\ x \leq r \Longrightarrow \exists\, y{\in}cball\ a\ r.\ x = b + y$
by (*metis* (*no_types*) *add.commute diff_add_cancel dist_add_cancel2 mem_cball*)
then show *?thesis*
by (*force simp*: *add.commute*)
qed

3.3.7 Various Lemmas on Normed Algebras

lemma *closed_of_nat_image*: *closed* (*of_nat* ‘ A :: $'a$::*real_normed_algebra_1 set*)
by (*rule discrete_imp_closed*[*of 1*]) (*auto simp*: *dist_of_nat*)

lemma *closed_of_int_image*: *closed* (*of_int* ‘ A :: $'a$::*real_normed_algebra_1 set*)
by (*rule discrete_imp_closed*[*of 1*]) (*auto simp*: *dist_of_int*)

lemma *closed_Nats* [*simp*]: *closed* ($\mathbb{N}$ :: $'a$ :: *real_normed_algebra_1 set*)
unfolding *Nats_def* **by** (*rule closed_of_nat_image*)

lemma *closed_Ints* [*simp*]: *closed* ($\mathbb{Z}$ :: $'a$ :: *real_normed_algebra_1 set*)
unfolding *Ints_def* **by** (*rule closed_of_int_image*)

lemma *closed_subset_Ints*:
fixes A :: $'a$ :: *real_normed_algebra_1 set*
assumes $A \subseteq \mathbb{Z}$
shows *closed A*
proof (*intro discrete_imp_closed*[*OF zero_less_one*] *ballI impI*, *goal_cases*)
case (*1 x y*)
with *assms* **have** $x \in \mathbb{Z}$ **and** $y \in \mathbb{Z}$ **by** *auto*
with ‹*dist y x* < *1*› **show** $y = x$
by (*auto elim*!: *Ints_cases simp*: *dist_of_int*)
qed

3.3.8 Filters

definition *indirection* :: $'a$::*real_normed_vector* ⇒ $'a$ ⇒ $'a$ *filter* (**infixr** *indirection 70*)
where *a indirection v* = *at a within* {*b*. ∃ *c*≥*0*. *b* − *a* = *scaleR c v*}

3.3.9 Trivial Limits

lemma *trivial_limit_at_infinity*:
¬ *trivial_limit* (*at_infinity* :: ($'a$::{*real_normed_vector*,*perfect_space*}) *filter*)
proof −
obtain x::$'a$ **where** $x \neq 0$
by (*meson perfect_choose_dist zero_less_one*)
then have $b \leq norm\ ((b\ /\ norm\ x)\ *_R\ x)$ **for** b
by *simp*

then show *?thesis*
unfolding *trivial_limit_def eventually_at_infinity*
by *blast*
qed

lemma *at_within_ball_bot_iff*:
fixes *x y* :: *'a*::{*real_normed_vector*,*perfect_space*}
shows *at x within ball y r = bot* ⟷ (*r=0* ∨ *x* ∉ *cball y r*)
unfolding *trivial_limit_within*
by (*metis* (*no_types*) *cball_empty equals0D islimpt_ball less_linear*)

3.3.10 Limits

proposition *Lim_at_infinity*: (*f* ⟶ *l*) *at_infinity* ⟷ (∀ *e*>*0*. ∃ *b*. ∀ *x*. *norm* *x* ≥ *b* ⟶ *dist* (*f x*) *l* < *e*)
by (*auto simp*: *tendsto_iff eventually_at_infinity*)

corollary *Lim_at_infinityI* [*intro?*]:
assumes ⋀*e*. *e* > *0* ⟹ ∃ *B*. ∀ *x*. *norm x* ≥ *B* ⟶ *dist* (*f x*) *l* ≤ *e*
shows (*f* ⟶ *l*) *at_infinity*
proof −
have ⋀*e*. *e* > *0* ⟹ ∃ *B*. ∀ *x*. *norm x* ≥ *B* ⟶ *dist* (*f x*) *l* < *e*
by (*meson assms dense le_less_trans*)
then show *?thesis*
using *Lim_at_infinity* **by** *blast*
qed

lemma *Lim_transform_within_set_eq*:
fixes *a* :: *'a*::*metric_space* **and** *l* :: *'b*::*metric_space*
shows *eventually* (λ*x*. *x* ∈ *S* ⟷ *x* ∈ *T*) (*at a*)
⟹ ((*f* ⟶ *l*) (*at a within S*) ⟷ (*f* ⟶ *l*) (*at a within T*))
by (*force intro*: *Lim_transform_within_set elim*: *eventually_mono*)

lemma *Lim_null*:
fixes *f* :: *'a* ⇒ *'b*::*real_normed_vector*
shows (*f* ⟶ *l*) *net* ⟷ ((λ*x*. *f*(*x*) − *l*) ⟶ *0*) *net*
by (*simp add*: *Lim dist_norm*)

lemma *Lim_null_comparison*:
fixes *f* :: *'a* ⇒ *'b*::*real_normed_vector*
assumes *eventually* (λ*x*. *norm* (*f x*) ≤ *g x*) *net* (*g* ⟶ *0*) *net*
shows (*f* ⟶ *0*) *net*
using *assms*(*2*)
proof (*rule metric_tendsto_imp_tendsto*)
show *eventually* (λ*x*. *dist* (*f x*) *0* ≤ *dist* (*g x*) *0*) *net*
using *assms*(*1*) **by** (*rule eventually_mono*) (*simp add*: *dist_norm*)
qed

lemma *Lim_transform_bound*:

fixes $f :: 'a \Rightarrow 'b$::*real_normed_vector*
and $g :: 'a \Rightarrow 'c$::*real_normed_vector*
assumes *eventually* $(\lambda n.\ norm\ (f\ n) \leq norm\ (g\ n))$ *net*
and $(g \longrightarrow 0)$ *net*
shows $(f \longrightarrow 0)$ *net*
using *assms*(1) *tendsto_norm_zero* [*OF assms*(2)]
by (*rule Lim_null_comparison*)

lemma *lim_null_mult_right_bounded*:
fixes $f :: 'a \Rightarrow 'b$::*real_normed_div_algebra*
assumes *f*: $(f \longrightarrow 0)$ *F* **and** *g*: *eventually* $(\lambda x.\ norm(g\ x) \leq B)$ *F*
shows $((\lambda z.\ f\ z * g\ z) \longrightarrow 0)$ *F*
proof −
have $((\lambda x.\ norm\ (f\ x) * norm\ (g\ x)) \longrightarrow 0)$ *F*
proof (*rule Lim_null_comparison*)
show $\forall_F\ x\ in\ F.\ norm\ (norm\ (f\ x) * norm\ (g\ x)) \leq norm\ (f\ x) * B$
by (*simp add*: *eventually_mono* [*OF g*] *mult_left_mono*)
show $((\lambda x.\ norm\ (f\ x) * B) \longrightarrow 0)$ *F*
by (*simp add*: *f tendsto_mult_left_zero tendsto_norm_zero*)
qed
then show *?thesis*
by (*subst tendsto_norm_zero_iff* [*symmetric*]) (*simp add*: *norm_mult*)
qed

lemma *lim_null_mult_left_bounded*:
fixes $f :: 'a \Rightarrow 'b$::*real_normed_div_algebra*
assumes *g*: *eventually* $(\lambda x.\ norm(g\ x) \leq B)$ *F* **and** *f*: $(f \longrightarrow 0)$ *F*
shows $((\lambda z.\ g\ z * f\ z) \longrightarrow 0)$ *F*
proof −
have $((\lambda x.\ norm\ (g\ x) * norm\ (f\ x)) \longrightarrow 0)$ *F*
proof (*rule Lim_null_comparison*)
show $\forall_F\ x\ in\ F.\ norm\ (norm\ (g\ x) * norm\ (f\ x)) \leq B * norm\ (f\ x)$
by (*simp add*: *eventually_mono* [*OF g*] *mult_right_mono*)
show $((\lambda x.\ B * norm\ (f\ x)) \longrightarrow 0)$ *F*
by (*simp add*: *f tendsto_mult_right_zero tendsto_norm_zero*)
qed
then show *?thesis*
by (*subst tendsto_norm_zero_iff* [*symmetric*]) (*simp add*: *norm_mult*)
qed

lemma *lim_null_scaleR_bounded*:
assumes *f*: $(f \longrightarrow 0)$ *net* **and** *gB*: *eventually* $(\lambda a.\ f\ a = 0 \vee norm(g\ a) \leq B)$ *net*
shows $((\lambda n.\ f\ n *_R g\ n) \longrightarrow 0)$ *net*
proof
fix ε::*real*
assume $0 < \varepsilon$
then have *B*: $0 < \varepsilon\ /\ (abs\ B + 1)$ **by** *simp*
have $*$: $|f\ x| * norm\ (g\ x) < \varepsilon$ **if** *f*: $|f\ x| * (|B| + 1) < \varepsilon$ **and** *g*: $norm\ (g\ x) \leq$

B **for** *x*
proof −
have $|f\ x| * norm\ (g\ x) \leq |f\ x| * B$
by (*simp add*: *mult_left_mono g*)
also have $\ldots \leq |f\ x| * (|B| + 1)$
by (*simp add*: *mult_left_mono*)
also have $\ldots < \varepsilon$
by (*rule f*)
finally show *?thesis* **.**
qed
have $\bigwedge x.\ [\![|f\ x| < \varepsilon\ /\ (|B| + 1);\ norm\ (g\ x) \leq B]\!] \Longrightarrow |f\ x| * norm\ (g\ x) < \varepsilon$
by (*simp add*: * *pos_less_divide_eq*)
then show $\forall_F\ x\ in\ net.\ dist\ (f\ x *_R g\ x)\ 0 < \varepsilon$
using ‹*0* < *ε*› **by** (*auto intro*: *eventually_mono* [*OF eventually_conj* [*OF tendstoD* [*OF f B*] *gB*]])
qed

lemma *Lim_norm_ubound*:
fixes *f* :: *'a* ⇒ *'b*::*real_normed_vector*
assumes ¬(*trivial_limit net*) (*f* ⟶ *l*) *net eventually* (λ*x*. *norm*(*f x*) ≤ *e*) *net*
shows *norm*(*l*) ≤ *e*
using *assms* **by** (*fast intro*: *tendsto_le tendsto_intros*)

lemma *Lim_norm_lbound*:
fixes *f* :: *'a* ⇒ *'b*::*real_normed_vector*
assumes ¬ *trivial_limit net*
and (*f* ⟶ *l*) *net*
and *eventually* (λ*x*. *e* ≤ *norm* (*f x*)) *net*
shows *e* ≤ *norm l*
using *assms* **by** (*fast intro*: *tendsto_le tendsto_intros*)

Limit under bilinear function

lemma *Lim_bilinear*:
assumes (*f* ⟶ *l*) *net*
and (*g* ⟶ *m*) *net*
and *bounded_bilinear h*
shows ((λ*x*. *h* (*f x*) (*g x*)) ⟶ (*h l m*)) *net*
using ‹*bounded_bilinear h*› ‹(*f* ⟶ *l*) *net*› ‹(*g* ⟶ *m*) *net*›
by (*rule bounded_bilinear.tendsto*)

lemma *Lim_at_zero*:
fixes *a* :: *'a*::*real_normed_vector*
and *l* :: *'b*::*topological_space*
shows (*f* ⟶ *l*) (*at a*) ⟷ ((λ*x*. *f*(*a* + *x*)) ⟶ *l*) (*at 0*)
using *LIM_offset_zero LIM_offset_zero_cancel* **..**

3.3.11 Limit Point of Filter

lemma *netlimit_at_vector*:

fixes a :: $'a$::*real_normed_vector*
shows *netlimit* (*at* a) = a
proof (*cases* $\exists\, x.\ x \neq a$)
case *True* **then obtain** x **where** x: $x \neq a$ **..**
have $\bigwedge d.\ 0 < d \Longrightarrow \exists\, x.\ x \neq a \wedge$ *norm* $(x - a) < d$
by (*rule_tac* x=a + *scaleR* $(d\ /\ 2)$ (*sgn* $(x - a)$)) **in** *exI*) (*simp add*: *norm_sgn sgn_zero_iff* x)
then have ¬ *trivial_limit* (*at* a)
by (*auto simp*: *trivial_limit_def eventually_at dist_norm*)
then show *?thesis*
by (*rule Lim_ident_at* [*of* a *UNIV*])
qed *simp*

3.3.12 Boundedness

lemma *continuous_on_closure_norm_le*:
fixes f :: $'a$::*metric_space* ⇒ $'b$::*real_normed_vector*
assumes *continuous_on* (*closure* s) f
and $\forall\, y \in s.$ *norm*$(f\ y) \leq b$
and $x \in$ (*closure* s)
shows *norm* $(f\ x) \leq b$
proof −
have ∗: f ‘ $s \subseteq$ *cball* 0 b
using *assms*(2)[*unfolded mem_cball_0*[*symmetric*]] **by** *auto*
show *?thesis*
by (*meson* ∗ *assms*(1) *assms*(3) *closed_cball image_closure_subset image_subset_iff mem_cball_0*)
qed

lemma *bounded_pos*: *bounded* $S \longleftrightarrow (\exists\, b{>}0.\ \forall\, x{\in}\ S.\ \text{norm}\ x \leq b)$
unfolding *bounded_iff*
by (*meson less_imp_le not_le order_trans zero_less_one*)

lemma *bounded_pos_less*: *bounded* $S \longleftrightarrow (\exists\, b{>}0.\ \forall\, x{\in}\ S.\ \text{norm}\ x < b)$
by (*metis bounded_pos le_less_trans less_imp_le linordered_field_no_ub*)

lemma *bounded_normE*:
assumes *bounded* A
obtains B **where** $B > 0$ $\bigwedge z.\ z \in A \Longrightarrow$ *norm* $z \leq B$
by (*meson assms bounded_pos*)

lemma *bounded_normE_less*:
assumes *bounded* A
obtains B **where** $B > 0$ $\bigwedge z.\ z \in A \Longrightarrow$ *norm* $z < B$
by (*meson assms bounded_pos_less*)

lemma *Bseq_eq_bounded*:
fixes f :: *nat* ⇒ $'a$::*real_normed_vector*
shows *Bseq* $f \longleftrightarrow$ *bounded* (*range* f)

```
  unfolding Bseq_def bounded_pos by auto

lemma bounded_linear_image:
  assumes bounded S
    and bounded_linear f
  shows bounded (f ‘ S)
proof −
  from assms(1) obtain b where b > 0 and b: ∀ x∈S. norm x ≤ b
    unfolding bounded_pos by auto
  from assms(2) obtain B where B: B > 0 ∀ x. norm (f x) ≤ B ∗ norm x
    using bounded_linear.pos_bounded by (auto simp: ac_simps)
  show ?thesis
    unfolding bounded_pos
  proof (intro exI, safe)
    show norm (f x) ≤ B ∗ b if x ∈ S for x
      by (meson B b less_imp_le mult_left_mono order_trans that)
  qed (use ‹b > 0› ‹B > 0› in auto)
qed

lemma bounded_scaling:
  fixes S :: 'a::real_normed_vector set
  shows bounded S ⟹ bounded ((λx. c ∗R x) ‘ S)
  by (simp add: bounded_linear_image bounded_linear_scaleR_right)

lemma bounded_scaleR_comp:
  fixes f :: 'a ⇒ 'b::real_normed_vector
  assumes bounded (f ‘ S)
  shows bounded ((λx. r ∗R f x) ‘ S)
  using bounded_scaling[of f ‘ S r] assms
  by (auto simp: image_image)

lemma bounded_translation:
  fixes S :: 'a::real_normed_vector set
  assumes bounded S
  shows bounded ((λx. a + x) ‘ S)
proof −
  from assms obtain b where b: b > 0 ∀ x∈S. norm x ≤ b
    unfolding bounded_pos by auto
  {
    fix x
    assume x ∈ S
    then have norm (a + x) ≤ b + norm a
      using norm_triangle_ineq[of a x] b by auto
  }
  then show ?thesis
    unfolding bounded_pos
    using norm_ge_zero[of a] b(1) and add_strict_increasing[of b 0 norm a]
    by (auto intro!: exI[of _ b + norm a])
qed
```

lemma *bounded_translation_minus*:
fixes $S :: 'a::real\_normed\_vector\ set$
shows $bounded\ S \Longrightarrow bounded\ ((\lambda x.\ x - a)\ `\ S)$
using *bounded_translation* [*of* $S\ -a$] **by** *simp*

lemma *bounded_uminus* [*simp*]:
fixes $X :: 'a::real\_normed\_vector\ set$
shows $bounded\ (uminus\ `\ X) \longleftrightarrow bounded\ X$
by (*auto simp*: *bounded_def dist_norm*; *rule_tac* $x=-x$ **in** *exI*; *force simp*: *add.commute norm_minus_commute*)

lemma *uminus_bounded_comp* [*simp*]:
fixes $f :: 'a \Rightarrow 'b::real\_normed\_vector$
shows $bounded\ ((\lambda x.\ -\ f\ x)\ `\ S) \longleftrightarrow bounded\ (f\ `\ S)$
using *bounded_uminus*[*of* $f\ `\ S$]
by (*auto simp*: *image_image*)

lemma *bounded_plus_comp*:
fixes $f\ g::'a \Rightarrow 'b::real\_normed\_vector$
assumes $bounded\ (f\ `\ S)$
assumes $bounded\ (g\ `\ S)$
shows $bounded\ ((\lambda x.\ f\ x + g\ x)\ `\ S)$
proof −
{
fix $B\ C$
assume $\bigwedge x.\ x \in S \Longrightarrow norm\ (f\ x) \leq B\ \bigwedge x.\ x \in S \Longrightarrow norm\ (g\ x) \leq C$
then have $\bigwedge x.\ x \in S \Longrightarrow norm\ (f\ x + g\ x) \leq B + C$
by (*auto intro*!: *norm_triangle_le add_mono*)
} **then show** *?thesis*
using *assms* **by** (*fastforce simp*: *bounded_iff*)
qed

lemma *bounded_plus*:
fixes $S :: 'a::real\_normed\_vector\ set$
assumes *bounded S bounded T*
shows $bounded\ ((\lambda(x,y).\ x + y)\ `\ (S \times T))$
using *bounded_plus_comp* [*of fst* $S \times T$ *snd*] *assms*
by (*auto simp*: *split_def split*: *if_split_asm*)

lemma *bounded_minus_comp*:
$bounded\ (f\ `\ S) \Longrightarrow bounded\ (g\ `\ S) \Longrightarrow bounded\ ((\lambda x.\ f\ x - g\ x)\ `\ S)$
for $f\ g::'a \Rightarrow 'b::real\_normed\_vector$
using *bounded_plus_comp*[*of f S* $\lambda x.\ -\ g\ x$]
by *auto*

lemma *bounded_minus*:
fixes $S :: 'a::real\_normed\_vector\ set$
assumes *bounded S bounded T*

shows *bounded* ((λ(*x*,*y*). *x* − *y*) ‘ (*S* × *T*))
using *bounded_minus_comp* [*of fst S* × *T snd*] *assms*
by (*auto simp*: *split_def split*: *if_split_asm*)

lemma *bounded_sums*:
fixes *S* :: *'a*::*real_normed_vector set*
assumes *bounded S* **and** *bounded T*
shows *bounded* (⋃*x*∈ *S*. ⋃*y* ∈ *T*. {*x* + *y*})
using *assms* **by** (*simp add*: *bounded_iff*) (*meson norm_triangle_mono*)

lemma *bounded_differences*:
fixes *S* :: *'a*::*real_normed_vector set*
assumes *bounded S* **and** *bounded T*
shows *bounded* (⋃*x*∈ *S*. ⋃*y* ∈ *T*. {*x* − *y*})
using *assms* **by** (*simp add*: *bounded_iff*) (*meson add_mono norm_triangle_le_diff*)

lemma *not_bounded_UNIV*[*simp*]:
¬ *bounded* (*UNIV* :: *'a*::{*real_normed_vector*, *perfect_space*} *set*)
proof (*auto simp*: *bounded_pos not_le*)
obtain *x* :: *'a* **where** *x* ≠ *0*
using *perfect_choose_dist* [*OF zero_less_one*] **by** *fast*
fix *b* :: *real*
assume *b*: *b* >*0*
have *b1*: *b* +*1* ≥ *0*
using *b* **by** *simp*
with ‹*x* ≠ *0*› **have** *b* < *norm* (*scaleR* (*b* + *1*) (*sgn x*))
by (*simp add*: *norm_sgn*)
then show ∃*x*::*'a*. *b* < *norm x* **..**
qed

corollary *cobounded_imp_unbounded*:
fixes *S* :: *'a*::{*real_normed_vector*, *perfect_space*} *set*
shows *bounded* (− *S*) ⟹ ¬ *bounded S*
using *bounded_Un* [*of S* −*S*] **by** (*simp*)

3.3.13 Relations among convergence and absolute convergence for power series

lemma *summable_imp_bounded*:
fixes *f* :: *nat* ⇒ *'a*::*real_normed_vector*
shows *summable f* ⟹ *bounded* (*range f*)
by (*frule summable_LIMSEQ_zero*) (*simp add*: *convergent_imp_bounded*)

lemma *summable_imp_sums_bounded*:
summable f ⟹ *bounded* (*range* (λ*n*. *sum f* {..<*n*}))
by (*auto simp*: *summable_def sums_def dest*: *convergent_imp_bounded*)

lemma *power_series_conv_imp_absconv_weak*:
fixes *a*:: *nat* ⇒ *'a*::{*real_normed_div_algebra*,*banach*} **and** *w* :: *'a*

```
  assumes sum: summable (λn. a n * z ^ n) and no: norm w < norm z
    shows summable (λn. of_real(norm(a n)) * w ^ n)
proof −
  obtain M where M: ⋀x. norm (a x * z ^ x) ≤ M
    using summable_imp_bounded [OF sum] by (force simp: bounded_iff)
  show ?thesis
  proof (rule series_comparison_complex)
    have ⋀n. norm (a n) * norm z ^ n ≤ M
      by (metis (no_types) M norm_mult norm_power)
    then show summable (λn. complex_of_real (norm (a n) * norm w ^ n))
      using Abel_lemma no norm_ge_zero summable_of_real by blast
  qed (auto simp: norm_mult norm_power)
qed
```

3.3.14 Normed spaces with the Heine-Borel property

```
lemma not_compact_UNIV[simp]:
  fixes s :: 'a::{real_normed_vector,perfect_space,heine_borel} set
  shows ¬ compact (UNIV::'a set)
    by (simp add: compact_eq_bounded_closed)

lemma not_compact_space_euclideanreal [simp]: ¬ compact_space euclideanreal
  by (simp add: compact_space_def)
```

Representing sets as the union of a chain of compact sets.

```
lemma closed_Union_compact_subsets:
  fixes S :: 'a::{heine_borel,real_normed_vector} set
  assumes closed S
  obtains F where ⋀n. compact(F n) ⋀n. F n ⊆ S ⋀n. F n ⊆ F(Suc n)
                  (⋃n. F n) = S ⋀K. ⟦compact K; K ⊆ S⟧ ⟹ ∃N. ∀n ≥ N. K ⊆
F n
proof
  show compact (S ∩ cball 0 (of_nat n)) for n
    using assms compact_eq_bounded_closed by auto
next
  show (⋃n. S ∩ cball 0 (real n)) = S
    by (auto simp: real_arch_simple)
next
  fix K :: 'a set
  assume compact K K ⊆ S
  then obtain N where K ⊆ cball 0 N
    by (meson bounded_pos mem_cball_0 compact_imp_bounded subsetI)
  then show ∃N. ∀n≥N. K ⊆ S ∩ cball 0 (real n)
   by (metis of_nat_le_iff Int_subset_iff ‹K ⊆ S› real_arch_simple subset_cball
subset_trans)
qed auto
```

3.3.15 Intersecting chains of compact sets and the Baire property

proposition *bounded_closed_chain*:
fixes $\mathcal{F}$ :: *'a*::*heine_borel set set*
assumes $B \in \mathcal{F}$ *bounded* B **and** $\mathcal{F}$: $\bigwedge S.\ S \in \mathcal{F} \Longrightarrow$ *closed* S **and** $\{\} \notin \mathcal{F}$
and *chain*: $\bigwedge S\ T.\ S \in \mathcal{F} \wedge T \in \mathcal{F} \Longrightarrow S \subseteq T \vee T \subseteq S$
shows $\bigcap \mathcal{F} \neq \{\}$
proof −
have $B \cap \bigcap \mathcal{F} \neq \{\}$
proof (*rule compact_imp_fip*)
show *compact* B $\bigwedge T.\ T \in \mathcal{F} \Longrightarrow$ *closed* T
by (*simp_all add*: *assms compact_eq_bounded_closed*)
show $[\![$*finite* $\mathcal{G}$; $\mathcal{G} \subseteq \mathcal{F}]\!] \Longrightarrow B \cap \bigcap \mathcal{G} \neq \{\}$ **for** $\mathcal{G}$
proof (*induction* $\mathcal{G}$ *rule*: *finite_induct*)
case *empty*
with *assms* **show** *?case* **by** *force*
next
case (*insert* U $\mathcal{G}$)
then have $U \in \mathcal{F}$ **and** *ne*: $B \cap \bigcap \mathcal{G} \neq \{\}$ **by** *auto*
then consider $B \subseteq U \mid U \subseteq B$
using ‹$B \in \mathcal{F}$› *chain* **by** *blast*
then show *?case*
proof *cases*
case *1*
then show *?thesis*
using *Int_left_commute ne* **by** *auto*
next
case *2*
have $U \neq \{\}$
using ‹$U \in \mathcal{F}$› ‹$\{\} \notin \mathcal{F}$› **by** *blast*
moreover
have *False* **if** $\bigwedge x.\ x \in U \Longrightarrow \exists\, Y{\in}\mathcal{G}.\ x \notin Y$
proof −
have $\bigwedge x.\ x \in U \Longrightarrow \exists\, Y{\in}\mathcal{G}.\ Y \subseteq U$
by (*metis chain contra_subsetD insert.prems insert_subset that*)
then obtain Y **where** $Y \in \mathcal{G}$ $Y \subseteq U$
by (*metis all_not_in_conv* ‹$U \neq \{\}$›)
moreover obtain x **where** $x \in \bigcap \mathcal{G}$
by (*metis Int_emptyI ne*)
ultimately show *?thesis*
by (*metis Inf_lower subset_eq that*)
qed
with *2* **show** *?thesis*
by *blast*
qed
qed
qed
then show *?thesis* **by** *blast*
qed

corollary *compact_chain*:
fixes $\mathcal{F}$ :: *'a::heine_borel set set*
assumes $\bigwedge S.\ S \in \mathcal{F} \Longrightarrow$ *compact* S $\{\} \notin \mathcal{F}$
$\bigwedge S\ T.\ S \in \mathcal{F} \wedge T \in \mathcal{F} \Longrightarrow S \subseteq T \vee T \subseteq S$
shows $\bigcap \mathcal{F} \neq \{\}$
proof (*cases* $\mathcal{F} = \{\}$)
case *True*
then show *?thesis* **by** *auto*
next
case *False*
show *?thesis*
by (*metis False all_not_in_conv assms compact_imp_bounded compact_imp_closed bounded_closed_chain*)
qed

lemma *compact_nest*:
fixes F :: *'a::linorder* $\Rightarrow$ *'b::heine_borel set*
assumes F: $\bigwedge n.$ *compact*$(F\ n)$ $\bigwedge n.\ F\ n \neq \{\}$ **and** *mono*: $\bigwedge m\ n.\ m \leq n \Longrightarrow F\ n \subseteq F\ m$
shows $\bigcap$(*range* F) $\neq \{\}$
proof −
have $*$: $\bigwedge S\ T.\ S \in$ *range* $F \wedge T \in$ *range* $F \Longrightarrow S \subseteq T \vee T \subseteq S$
by (*metis mono image_iff le_cases*)
show *?thesis*
using F **by** (*intro compact_chain* [*OF* _ _ $*$]; *blast dest*: $*$)
qed

The Baire property of dense sets

theorem *Baire*:
fixes S::*'a*::{*real_normed_vector*,*heine_borel*} *set*
assumes *closed* S *countable* $\mathcal{G}$
and *ope*: $\bigwedge T.\ T \in \mathcal{G} \Longrightarrow$ *openin* (*top_of_set* S) $T \wedge S \subseteq$ *closure* T
shows $S \subseteq$ *closure*$(\bigcap \mathcal{G})$
proof (*cases* $\mathcal{G} = \{\}$)
case *True*
then show *?thesis*
using *closure_subset* **by** *auto*
next
let *?g* = *from_nat_into* $\mathcal{G}$
case *False*
then have *gin*: *?g* $n \in \mathcal{G}$ **for** n
by (*simp add*: *from_nat_into*)
show *?thesis*
proof (*clarsimp simp*: *closure_approachable*)
fix x **and** *e*::*real*
assume $x \in S\ 0 < e$
obtain *TF* **where** *openF*: $\bigwedge n.$ *openin* (*top_of_set* S) (*TF* n)
and *ne*: $\bigwedge n.$ *TF* $n \neq \{\}$

```
            and subg: ⋀n. S ∩ closure(TF n) ⊆ ?g n
            and subball: ⋀n. closure(TF n) ⊆ ball x e
            and decr: ⋀n. TF(Suc n) ⊆ TF n
    proof −
      have ∗: ∃ Y. (openin (top_of_set S) Y ∧ Y ≠ {} ∧
                S ∩ closure Y ⊆ ?g n ∧ closure Y ⊆ ball x e) ∧ Y ⊆ U
        if opeU: openin (top_of_set S) U and U ≠ {} and cloU: closure U ⊆ ball
x e for U n
      proof −
        obtain T where T: open T U = T ∩ S
          using ‹openin (top_of_set S) U› by (auto simp: openin_subtopology)
        with ‹U ≠ {}› have T ∩ closure (?g n) ≠ {}
          using gin ope by fastforce
        then have T ∩ ?g n ≠ {}
          using ‹open T› open_Int_closure_eq_empty by blast
        then obtain y where y ∈ U y ∈ ?g n
          using T ope [of ?g n, OF gin] by (blast dest: openin_imp_subset)
        moreover have openin (top_of_set S) (U ∩ ?g n)
          using gin ope opeU by blast
        ultimately obtain d where U: U ∩ ?g n ⊆ S and d > 0 and d: ball y d
∩ S ⊆ U ∩ ?g n
          by (force simp: openin_contains_ball)
        show ?thesis
        proof (intro exI conjI)
          show openin (top_of_set S) (S ∩ ball y (d/2))
            by (simp add: openin_open_Int)
          show S ∩ ball y (d/2) ≠ {}
            using ‹0 < d› ‹y ∈ U› opeU openin_imp_subset by fastforce
          have S ∩ closure (S ∩ ball y (d/2)) ⊆ S ∩ closure (ball y (d/2))
            using closure_mono by blast
          also have ... ⊆ ?g n
            using ‹d > 0› d by force
          finally show S ∩ closure (S ∩ ball y (d/2)) ⊆ ?g n .
          have closure (S ∩ ball y (d/2)) ⊆ S ∩ ball y d
          proof −
            have closure (ball y (d/2)) ⊆ ball y d
              using ‹d > 0› by auto
            then have closure (S ∩ ball y (d/2)) ⊆ ball y d
              by (meson closure_mono inf.cobounded2 subset_trans)
            then show ?thesis
              by (simp add: ‹closed S› closure_minimal)
          qed
          also have ... ⊆ ball x e
            using cloU closure_subset d by blast
          finally show closure (S ∩ ball y (d/2)) ⊆ ball x e .
          show S ∩ ball y (d/2) ⊆ U
            using ball_divide_subset_numeral d by blast
        qed
      qed
```

```
    let ?Φ = λn X. openin (top_of_set S) X ∧ X ≠ {} ∧
                  S ∩ closure X ⊆ ?g n ∧ closure X ⊆ ball x e
    have closure (S ∩ ball x (e/2)) ⊆ closure(ball x (e/2))
      by (simp add: closure_mono)
    also have ... ⊆ ball x e
      using ‹e > 0› by auto
    finally have closure (S ∩ ball x (e/2)) ⊆ ball x e .
    moreover haveopenin (top_of_set S) (S ∩ ball x (e/2)) S ∩ ball x (e/2) ≠
{}
      using ‹0 < e› ‹x ∈ S› by auto
    ultimately obtain Y where Y: ?Φ 0 Y ∧ Y ⊆ S ∩ ball x (e/2)
        using ∗ [of S ∩ ball x (e/2) 0] by metis
    show thesis
    proof (rule exE [OF dependent_nat_choice])
      show ∃x. ?Φ 0 x
        using Y by auto
      show ∃Y. ?Φ (Suc n) Y ∧ Y ⊆ X if ?Φ n X for X n
        using that by (blast intro: ∗)
    qed (use that in metis)
  qed
  have (⋂n. S ∩ closure (TF n)) ≠ {}
  proof (rule compact_nest)
    show ⋀n. compact (S ∩ closure (TF n))
    by (metis closed_closure subball bounded_subset_ballI compact_eq_bounded_closed
closed_Int_compact [OF ‹closed S›])
    show ⋀n. S ∩ closure (TF n) ≠ {}
      by (metis Int_absorb1 opeF ‹closed S› closure_eq_empty closure_minimal
ne openin_imp_subset)
    show ⋀m n. m ≤ n ⟹ S ∩ closure (TF n) ⊆ S ∩ closure (TF m)
    by (meson closure_mono decr dual_order.refl inf_mono lift_Suc_antimono_le)
  qed
  moreover have (⋂n. S ∩ closure (TF n)) ⊆ {y ∈ ⋂G. dist y x < e}
  proof (clarsimp, intro conjI)
    fix y
    assume y ∈ S and y: ∀n. y ∈ closure (TF n)
    then show ∀T∈G. y ∈ T
      by (metis Int_iff from_nat_into_surj [OF ‹countable G›] subsetD subg)
    show dist y x < e
      by (metis y dist_commute mem_ball subball subsetCE)
  qed
  ultimately show ∃y ∈ ⋂G. dist y x < e
    by auto
 qed
qed
```

3.3.16 Continuity

Structural rules for uniform continuity

lemma (**in** *bounded_linear*) *uniformly_continuous_on*[*continuous_intros*]:

fixes g :: _::*metric_space* ⇒ _
assumes *uniformly_continuous_on s g*
shows *uniformly_continuous_on s* (λ*x*. *f* (*g x*))
using *assms* **unfolding** *uniformly_continuous_on_sequentially*
unfolding *dist_norm tendsto_norm_zero_iff diff*[*symmetric*]
by (*auto intro*: *tendsto_zero*)

lemma *uniformly_continuous_on_dist*[*continuous_intros*]:
fixes *f g* :: *'a*::*metric_space* ⇒ *'b*::*metric_space*
assumes *uniformly_continuous_on s f*
and *uniformly_continuous_on s g*
shows *uniformly_continuous_on s* (λ*x*. *dist* (*f x*) (*g x*))
proof −
{
fix *a b c d* :: *'b*
have |*dist a b* − *dist c d*| ≤ *dist a c* + *dist b d*
using *dist_triangle2* [*of a b c*] *dist_triangle2* [*of b c d*]
using *dist_triangle3* [*of c d a*] *dist_triangle* [*of a d b*]
by *arith*
} **note** *le* = *this*
{
fix *x y*
assume *f*: (λ*n*. *dist* (*f* (*x n*)) (*f* (*y n*))) ⟶ *0*
assume *g*: (λ*n*. *dist* (*g* (*x n*)) (*g* (*y n*))) ⟶ *0*
have (λ*n*. |*dist* (*f* (*x n*)) (*g* (*x n*)) − *dist* (*f* (*y n*)) (*g* (*y n*))|) ⟶ *0*
by (*rule Lim_transform_bound* [*OF* _ *tendsto_add_zero* [*OF f g*]],
simp add: *le*)
}
then show *?thesis*
using *assms* **unfolding** *uniformly_continuous_on_sequentially*
unfolding *dist_real_def* **by** *simp*
qed

lemma *uniformly_continuous_on_cmul_right* [*continuous_intros*]:
fixes *f* :: *'a*::*real_normed_vector* ⇒ *'b*::*real_normed_algebra*
shows *uniformly_continuous_on s f* ⟹ *uniformly_continuous_on s* (λ*x*. *f x* ∗ *c*)
using *bounded_linear.uniformly_continuous_on*[*OF bounded_linear_mult_left*]
.

lemma *uniformly_continuous_on_cmul_left*[*continuous_intros*]:
fixes *f* :: *'a*::*real_normed_vector* ⇒ *'b*::*real_normed_algebra*
assumes *uniformly_continuous_on s f*
shows *uniformly_continuous_on s* (λ*x*. *c* ∗ *f x*)
by (*metis assms bounded_linear.uniformly_continuous_on bounded_linear_mult_right*)

lemma *uniformly_continuous_on_norm*[*continuous_intros*]:
fixes *f* :: *'a* :: *metric_space* ⇒ *'b* :: *real_normed_vector*
assumes *uniformly_continuous_on s f*

shows *uniformly_continuous_on s* (λ*x*. *norm* (*f x*))
unfolding *norm_conv_dist* **using** *assms*
by (*intro uniformly_continuous_on_dist uniformly_continuous_on_const*)

lemma *uniformly_continuous_on_cmul*[*continuous_intros*]:
fixes *f* :: *'a*::*metric_space* ⇒ *'b*::*real_normed_vector*
assumes *uniformly_continuous_on s f*
shows *uniformly_continuous_on s* (λ*x*. *c* ∗*R* *f*(*x*))
using *bounded_linear_scaleR_right assms*
by (*rule bounded_linear.uniformly_continuous_on*)

lemma *dist_minus*:
fixes *x y* :: *'a*::*real_normed_vector*
shows *dist* (− *x*) (− *y*) = *dist x y*
unfolding *dist_norm minus_diff_minus norm_minus_cancel* **..**

lemma *uniformly_continuous_on_minus*[*continuous_intros*]:
fixes *f* :: *'a*::*metric_space* ⇒ *'b*::*real_normed_vector*
shows *uniformly_continuous_on s f* ⟹ *uniformly_continuous_on s* (λ*x*. − *f x*)
unfolding *uniformly_continuous_on_def dist_minus* **.**

lemma *uniformly_continuous_on_add*[*continuous_intros*]:
fixes *f g* :: *'a*::*metric_space* ⇒ *'b*::*real_normed_vector*
assumes *uniformly_continuous_on s f*
and *uniformly_continuous_on s g*
shows *uniformly_continuous_on s* (λ*x*. *f x* + *g x*)
using *assms*
unfolding *uniformly_continuous_on_sequentially*
unfolding *dist_norm tendsto_norm_zero_iff add_diff_add*
by (*auto intro*: *tendsto_add_zero*)

lemma *uniformly_continuous_on_diff*[*continuous_intros*]:
fixes *f* :: *'a*::*metric_space* ⇒ *'b*::*real_normed_vector*
assumes *uniformly_continuous_on s f*
and *uniformly_continuous_on s g*
shows *uniformly_continuous_on s* (λ*x*. *f x* − *g x*)
using *assms uniformly_continuous_on_add* [*of s f* − *g*]
by (*simp add*: *fun_Compl_def uniformly_continuous_on_minus*)

lemma *uniformly_continuous_on_sum* [*continuous_intros*]:
fixes *f* :: *'a* ⇒ *'b*::*metric_space* ⇒ *'c*::*real_normed_vector*
shows (⋀*i*. *i* ∈ *I* ⟹ *uniformly_continuous_on S* (*f i*)) ⟹ *uniformly_continuous_on S* (λ*x*. ∑ *i*∈*I*. *f i x*)
by (*induction I rule*: *infinite_finite_induct*)
(*auto simp*: *uniformly_continuous_on_add uniformly_continuous_on_const*)

3.3.17 Arithmetic Preserves Topological Properties

lemma *open_scaling*[*intro*]:
fixes *s* :: *'a*::*real_normed_vector set*
assumes *c* ≠ *0*
and *open s*
shows *open*((λ*x*. *c* ∗*R* *x*) ‘ *s*)
proof −
{
fix *x*
assume *x* ∈ *s*
then obtain *e* **where** *e*>*0*
and *e*:∀ *x'*. *dist x' x* < *e* ⟶ *x'* ∈ *s* **using** *assms*(*2*)[*unfolded open_dist*, *THEN bspec*[**where** *x*=*x*]]
by *auto*
have *e* ∗ |*c*| > *0*
using *assms*(*1*)[*unfolded zero_less_abs_iff*[*symmetric*]] ‹*e*>*0*› **by** *auto*
moreover
{
fix *y*
assume *dist y* (*c* ∗*R* *x*) < *e* ∗ |*c*|
then have *norm* (*c* ∗*R* ((*1* / *c*) ∗*R* *y* − *x*)) < *e* ∗ *norm c*
by (*simp add*: ‹*c* ≠ *0*› *dist_norm scale_right_diff_distrib*)
then have *norm* ((*1* / *c*) ∗*R* *y* − *x*) < *e*
by (*simp add*: ‹*c* ≠ *0*›)
then have *y* ∈ (∗*R*) *c* ‘ *s*
using *rev_image_eqI*[*of* (*1* / *c*) ∗*R* *y s y* (∗*R*) *c*]
by (*simp add*: ‹*c* ≠ *0*› *dist_norm e*)
}
ultimately have ∃ *e*>*0*. ∀ *x'*. *dist x'* (*c* ∗*R* *x*) < *e* ⟶ *x'* ∈ (∗*R*) *c* ‘ *s*
by (*rule_tac x*=*e* ∗ |*c*| **in** *exI*, *auto*)
}
then show *?thesis* **unfolding** *open_dist* **by** *auto*
qed

lemma *minus_image_eq_vimage*:
fixes *A* :: *'a*::*ab_group_add set*
shows (λ*x*. − *x*) ‘ *A* = (λ*x*. − *x*) −‘ *A*
by (*auto intro*!: *image_eqI* [**where** *f*=λ*x*. − *x*])

lemma *open_negations*:
fixes *S* :: *'a*::*real_normed_vector set*
shows *open S* ⟹ *open* ((λ*x*. − *x*) ‘ *S*)
using *open_scaling* [*of* − *1 S*] **by** *simp*

lemma *open_translation*:
fixes *S* :: *'a*::*real_normed_vector set*
assumes *open S*
shows *open*((λ*x*. *a* + *x*) ‘ *S*)
proof −

```
  {
    fix x
    have continuous (at x) (λx. x − a)
      by (intro continuous_diff continuous_ident continuous_const)
  }
  moreover have {x. x − a ∈ S} = (+) a ‘ S
    by force
  ultimately show ?thesis
    by (metis assms continuous_open_vimage vimage_def)
qed

lemma open_translation_subtract:
  fixes S :: 'a::real_normed_vector set
  assumes open S
  shows open ((λx. x − a) ‘ S)
  using assms open_translation [of S − a] by (simp cong: image_cong_simp)

lemma open_neg_translation:
  fixes S :: 'a::real_normed_vector set
  assumes open S
  shows open((λx. a − x) ‘ S)
  using open_translation[OF open_negations[OF assms], of a]
  by (auto simp: image_image)

lemma open_affinity:
  fixes S :: 'a::real_normed_vector set
  assumes open S  c ≠ 0
  shows open ((λx. a + c *R x) ‘ S)
proof −
  have *: (λx. a + c *R x) = (λx. a + x) ∘ (λx. c *R x)
    unfolding o_def ..
  have (+) a ‘ (*R) c ‘ S = ((+) a ∘ (*R) c) ‘ S
    by auto
  then show ?thesis
    using assms open_translation[of (*R) c ‘ S a]
    unfolding *
    by auto
qed

lemma interior_translation:
  interior ((+) a ‘ S) = (+) a ‘ (interior S) for S :: 'a::real_normed_vector set
proof (rule set_eqI, rule)
  fix x
  assume x ∈ interior ((+) a ‘ S)
  then obtain e where e > 0 and e: ball x e ⊆ (+) a ‘ S
    unfolding mem_interior by auto
  then have ball (x − a) e ⊆ S
    unfolding subset_eq Ball_def mem_ball dist_norm
    by (auto simp: diff_diff_eq)
```

```
  then show x ∈ (+) a ‘ interior S
    unfolding image_iff
    by (metis ‹0 < e› add.commute diff_add_cancel mem_interior)
next
  fix x
  assume x ∈ (+) a ‘ interior S
  then obtain y e where e > 0 and e: ball y e ⊆ S and y: x = a + y
    unfolding image_iff Bex_def mem_interior by auto
  {
    fix z
    have *: a + y − z = y + a − z by auto
    assume z ∈ ball x e
    then have z − a ∈ S
      using e[unfolded subset_eq, THEN bspec[where x=z − a]]
      unfolding mem_ball dist_norm y group_add_class.diff_diff_eq2 *
      by auto
    then have z ∈ (+) a ‘ S
      unfolding image_iff by (auto intro!: bexI[where x=z − a])
  }
  then have ball x e ⊆ (+) a ‘ S
    unfolding subset_eq by auto
  then show x ∈ interior ((+) a ‘ S)
    unfolding mem_interior using ‹e > 0› by auto
qed

lemma interior_translation_subtract:
  interior ((λx. x − a) ‘ S) = (λx. x − a) ‘ interior S for S :: 'a::real_normed_vector
set
  using interior_translation [of − a] by (simp cong: image_cong_simp)


lemma compact_scaling:
  fixes s :: 'a::real_normed_vector set
  assumes compact s
  shows compact ((λx. c *R x) ‘ s)
proof −
  let ?f = λx. scaleR c x
  have *: bounded_linear ?f by (rule bounded_linear_scaleR_right)
  show ?thesis
   using compact_continuous_image[of s ?f] continuous_at_imp_continuous_on[of
s ?f]
    using linear_continuous_at[OF *] assms
    by auto
qed

lemma compact_negations:
  fixes s :: 'a::real_normed_vector set
  assumes compact s
  shows compact ((λx. − x) ‘ s)
```

using *compact_scaling* [*OF assms*, *of* − *1*] **by** *auto*

lemma *compact_sums*:
fixes *s t* :: *'a*::*real_normed_vector set*
assumes *compact s*
and *compact t*
shows *compact* {*x* + *y* | *x y*. *x* ∈ *s* ∧ *y* ∈ *t*}
proof −
have ∗: {*x* + *y* | *x y*. *x* ∈ *s* ∧ *y* ∈ *t*} = (λ*z*. *fst z* + *snd z*) ‘ (*s* × *t*)
by (*fastforce simp*: *image_iff*)
have *continuous_on* (*s* × *t*) (λ*z*. *fst z* + *snd z*)
unfolding *continuous_on* **by** (*rule ballI*) (*intro tendsto_intros*)
then show *?thesis*
unfolding ∗ **using** *compact_continuous_image compact_Times* [*OF assms*]
by *auto*
qed

lemma *compact_differences*:
fixes *s t* :: *'a*::*real_normed_vector set*
assumes *compact s*
and *compact t*
shows *compact* {*x* − *y* | *x y*. *x* ∈ *s* ∧ *y* ∈ *t*}
proof−
have {*x* − *y* | *x y*. *x*∈*s* ∧ *y* ∈ *t*} = {*x* + *y* | *x y*. *x* ∈ *s* ∧ *y* ∈ (*uminus* ‘ *t*)}
using *diff_conv_add_uminus* **by** *force*
then show *?thesis*
using *compact_sums*[*OF assms*(*1*) *compact_negations*[*OF assms*(*2*)]] **by** *auto*
qed

lemma *compact_sums′*:
fixes *S* :: *'a*::*real_normed_vector set*
assumes *compact S* **and** *compact T*
shows *compact* (⋃*x*∈ *S*. ⋃*y* ∈ *T*. {*x* + *y*})
proof −
have (⋃*x*∈*S*. ⋃*y*∈*T*. {*x* + *y*}) = {*x* + *y* |*x y*. *x* ∈ *S* ∧ *y* ∈ *T*}
by *blast*
then show *?thesis*
using *compact_sums* [*OF assms*] **by** *simp*
qed

lemma *compact_differences′*:
fixes *S* :: *'a*::*real_normed_vector set*
assumes *compact S* **and** *compact T*
shows *compact* (⋃*x*∈ *S*. ⋃*y* ∈ *T*. {*x* − *y*})
proof −
have (⋃*x*∈*S*. ⋃*y*∈*T*. {*x* − *y*}) = {*x* − *y* |*x y*. *x* ∈ *S* ∧ *y* ∈ *T*}
by *blast*
then show *?thesis*
using *compact_differences* [*OF assms*] **by** *simp*

qed

lemma *compact_translation*:
  compact ((+) *a* ‘ *s*) **if** *compact s* **for** *s* :: *'a*::*real_normed_vector set*
proof −
  have {*x* + *y* |*x y*. *x* ∈ *s* ∧ *y* ∈ {*a*}} = (λ*x*. *a* + *x*) ‘ *s*
    by *auto*
  then show *?thesis*
    using *compact_sums* [*OF that compact_sing* [*of a*]] **by** *auto*
qed

lemma *compact_translation_subtract*:
  compact ((λ*x*. *x* − *a*) ‘ *s*) **if** *compact s* **for** *s* :: *'a*::*real_normed_vector set*
  using *that compact_translation* [*of s* − *a*] **by** (*simp cong*: *image_cong_simp*)

lemma *compact_affinity*:
  fixes *s* :: *'a*::*real_normed_vector set*
  assumes *compact s*
  shows *compact* ((λ*x*. *a* + *c* ∗*R* *x*) ‘ *s*)
proof −
  have (+) *a* ‘ (∗*R*) *c* ‘ *s* = (λ*x*. *a* + *c* ∗*R* *x*) ‘ *s*
    by *auto*
  then show *?thesis*
    using *compact_translation*[*OF compact_scaling*[*OF assms*], *of a c*] **by** *auto*
qed

lemma *closed_scaling*:
  fixes *S* :: *'a*::*real_normed_vector set*
  assumes *closed S*
  shows *closed* ((λ*x*. *c* ∗*R* *x*) ‘ *S*)
proof (*cases c* = *0*)
  case *True* **then show** *?thesis*
    by (*auto simp*: *image_constant_conv*)
next
  case *False*
  from *assms* **have** *closed* ((λ*x*. *inverse c* ∗*R* *x*) −‘ *S*)
    by (*simp add*: *continuous_closed_vimage*)
  also have (λ*x*. *inverse c* ∗*R* *x*) −‘ *S* = (λ*x*. *c* ∗*R* *x*) ‘ *S*
    using ‹*c* ≠ *0*› **by** (*auto elim*: *image_eqI* [*rotated*])
  finally show *?thesis* **.**
qed

lemma *closed_negations*:
  fixes *S* :: *'a*::*real_normed_vector set*
  assumes *closed S*
  shows *closed* ((λ*x*. −*x*) ‘ *S*)
  using *closed_scaling*[*OF assms*, *of* − *1*] **by** *simp*

lemma *compact_closed_sums*:

```
  fixes S :: 'a::real_normed_vector set
  assumes compact S and closed T
  shows closed (⋃x∈ S. ⋃y ∈ T. {x + y})
proof −
  let ?S = {x + y |x y. x ∈ S ∧ y ∈ T}
  {
    fix x l
    assume as: ∀ n. x n ∈ ?S  (x ⟶ l) sequentially
    from as(1) obtain f where f: ∀ n. x n = fst (f n) + snd (f n)  ∀ n. fst (f n)
∈ S  ∀ n. snd (f n) ∈ T
      using choice[of λn y. x n = (fst y) + (snd y) ∧ fst y ∈ S ∧ snd y ∈ T] by
auto
    obtain l' r where l'∈S and r: strict_mono r and lr: (((λn. fst (f n)) ∘ r)
⟶ l') sequentially
      using assms(1)[unfolded compact_def, THEN spec[where x=λ n. fst (f n)]]
using f(2) by auto
    have ((λn. snd (f (r n))) ⟶ l − l') sequentially
      using tendsto_diff[OF LIMSEQ_subseq_LIMSEQ[OF as(2) r] lr] and f(1)
      unfolding o_def
      by auto
    then have l − l' ∈ T
      using assms(2)[unfolded closed_sequential_limits,
        THEN spec[where x=λ n. snd (f (r n))],
        THEN spec[where x=l − l']]
      using f(3)
      by auto
    then have l ∈ ?S
      using ‹l' ∈ S› by force
  }
  moreover have ?S = (⋃x∈ S. ⋃y ∈ T. {x + y})
    by force
  ultimately show ?thesis
    unfolding closed_sequential_limits
    by (metis (no_types, lifting))
qed

lemma closed_compact_sums:
  fixes S T :: 'a::real_normed_vector set
  assumes closed S compact T
  shows closed (⋃x∈ S. ⋃y ∈ T. {x + y})
proof −
  have (⋃x∈ T. ⋃y ∈ S. {x + y}) = (⋃x∈ S. ⋃y ∈ T. {x + y})
    by auto
  then show ?thesis
    using compact_closed_sums[OF assms(2,1)] by simp
qed

lemma compact_closed_differences:
  fixes S T :: 'a::real_normed_vector set
```

assumes *compact S closed T*
shows *closed* $(\bigcup x\in S.\ \bigcup y \in T.\ \{x - y\})$
proof −
have $(\bigcup x\in S.\ \bigcup y \in$ *uminus* ‘ $T.\ \{x + y\}) = (\bigcup x\in S.\ \bigcup y \in T.\ \{x - y\})$
by *force*
then show *?thesis*
by (*metis assms closed_negations compact_closed_sums*)
qed

lemma *closed_compact_differences*:
fixes *S T* :: *'a*::*real_normed_vector set*
assumes *closed S compact T*
shows *closed* $(\bigcup x\in S.\ \bigcup y \in T.\ \{x - y\})$
proof −
have $(\bigcup x\in S.\ \bigcup y \in$ *uminus* ‘ $T.\ \{x + y\}) = \{x - y\ |x\ y.\ x \in S \wedge y \in T\}$
by *auto*
then show *?thesis*
using *closed_compact_sums*[*OF assms*(*1*) *compact_negations*[*OF assms*(*2*)]] **by** *simp*
qed

lemma *closed_translation*:
closed ((+) *a* ‘ *S*) **if** *closed S* **for** *a* :: *'a*::*real_normed_vector*
proof −
have $(\bigcup x\in \{a\}.\ \bigcup y \in S.\ \{x + y\}) =$ ((+) *a* ‘ *S*) **by** *auto*
then show *?thesis*
using *compact_closed_sums* [*OF compact_sing* [*of a*] *that*] **by** *auto*
qed

lemma *closed_translation_subtract*:
closed $((\lambda x.\ x - a)$ ‘ *S*) **if** *closed S* **for** *a* :: *'a*::*real_normed_vector*
using *that closed_translation* [*of S* − *a*] **by** (*simp cong*: *image_cong_simp*)

lemma *closure_translation*:
closure ((+) *a* ‘ *s*) = (+) *a* ‘ *closure s* **for** *a* :: *'a*::*real_normed_vector*
proof −
have ∗: (+) *a* ‘ (− *s*) = − (+) *a* ‘ *s*
by (*auto intro*!: *image_eqI* [**where** *x* = *x* − *a* **for** *x*])
show *?thesis*
using *interior_translation* [*of a* − *s*, *symmetric*]
by (*simp add*: *closure_interior translation_Compl* ∗)
qed

lemma *closure_translation_subtract*:
closure $((\lambda x.\ x - a)$ ‘ *s*) = $(\lambda x.\ x - a)$ ‘ *closure s* **for** *a* :: *'a*::*real_normed_vector*
using *closure_translation* [*of* − *a s*] **by** (*simp cong*: *image_cong_simp*)

lemma *frontier_translation*:
frontier ((+) *a* ‘ *s*) = (+) *a* ‘ *frontier s* **for** *a* :: *'a*::*real_normed_vector*

by (*auto simp add*: *frontier_def translation_diff interior_translation closure_translation*)

lemma *frontier_translation_subtract*:
frontier ((+) *a* ‘ *s*) = (+) *a* ‘ *frontier s* **for** *a* :: *'a*::*real_normed_vector*
by (*auto simp add*: *frontier_def translation_diff interior_translation closure_translation*)

lemma *sphere_translation*:
sphere (*a* + *c*) *r* = (+) *a* ‘ *sphere c r* **for** *a* :: *'n*::*real_normed_vector*
by (*auto simp*: *dist_norm algebra_simps intro*!: *image_eqI* [**where** *x* = *x* − *a* **for** *x*])

lemma *sphere_translation_subtract*:
sphere (*c* − *a*) *r* = (λ*x*. *x* − *a*) ‘ *sphere c r* **for** *a* :: *'n*::*real_normed_vector*
using *sphere_translation* [*of* − *a c*] **by** (*simp cong*: *image_cong_simp*)

lemma *cball_translation*:
cball (*a* + *c*) *r* = (+) *a* ‘ *cball c r* **for** *a* :: *'n*::*real_normed_vector*
by (*auto simp*: *dist_norm algebra_simps intro*!: *image_eqI* [**where** *x* = *x* − *a* **for** *x*])

lemma *cball_translation_subtract*:
cball (*c* − *a*) *r* = (λ*x*. *x* − *a*) ‘ *cball c r* **for** *a* :: *'n*::*real_normed_vector*
using *cball_translation* [*of* − *a c*] **by** (*simp cong*: *image_cong_simp*)

lemma *ball_translation*:
ball (*a* + *c*) *r* = (+) *a* ‘ *ball c r* **for** *a* :: *'n*::*real_normed_vector*
by (*auto simp*: *dist_norm algebra_simps intro*!: *image_eqI* [**where** *x* = *x* − *a* **for** *x*])

lemma *ball_translation_subtract*:
ball (*c* − *a*) *r* = (λ*x*. *x* − *a*) ‘ *ball c r* **for** *a* :: *'n*::*real_normed_vector*
using *ball_translation* [*of* − *a c*] **by** (*simp cong*: *image_cong_simp*)

3.3.18 Homeomorphisms

lemma *homeomorphic_scaling*:
fixes *S* :: *'a*::*real_normed_vector set*
assumes *c* ≠ *0*
shows *S homeomorphic* ((λ*x*. *c* ∗*R* *x*) ‘ *S*)
unfolding *homeomorphic_minimal*
apply (*rule_tac x*=λ*x*. *c* ∗*R* *x* **in** *exI*)
apply (*rule_tac x*=λ*x*. (*1* / *c*) ∗*R* *x* **in** *exI*)
using *assms* **by** (*auto simp*: *continuous_intros*)

lemma *homeomorphic_translation*:
fixes *S* :: *'a*::*real_normed_vector set*
shows *S homeomorphic* ((λ*x*. *a* + *x*) ‘ *S*)
unfolding *homeomorphic_minimal*
apply (*rule_tac x*=λ*x*. *a* + *x* **in** *exI*)

apply (*rule_tac x*=$\lambda x.\ -a + x$ **in** *exI*)
by (*auto simp*: *continuous_intros*)

lemma *homeomorphic_affinity*:
fixes *S* :: *'a*::*real_normed_vector set*
assumes $c \neq 0$
shows *S homeomorphic* ((λ*x*. *a* + *c* *$_R$ *x*) ‘ *S*)
proof −
have *: (+) *a* ‘ (*$_R$) *c* ‘ *S* = (λ*x*. *a* + *c* *$_R$ *x*) ‘ *S* **by** *auto*
show *?thesis*
by (*metis* * *assms homeomorphic_scaling homeomorphic_trans homeomorphic_translation*)
qed

lemma *homeomorphic_balls*:
fixes *a b* ::*'a*::*real_normed_vector*
assumes $0 < d$ $0 < e$
shows (*ball a d*) *homeomorphic* (*ball b e*) (**is** *?th*)
and (*cball a d*) *homeomorphic* (*cball b e*) (**is** *?cth*)
proof −
show *?th* **unfolding** *homeomorphic_minimal*
apply(*rule_tac x*=$\lambda x.\ b + (e/d)$ *$_R$ $(x - a)$ **in** *exI*)
apply(*rule_tac x*=$\lambda x.\ a + (d/e)$ *$_R$ $(x - b)$ **in** *exI*)
using *assms*
by (*auto intro*!: *continuous_intros simp*: *dist_commute dist_norm pos_divide_less_eq*)
show *?cth* **unfolding** *homeomorphic_minimal*
apply(*rule_tac x*=$\lambda x.\ b + (e/d)$ *$_R$ $(x - a)$ **in** *exI*)
apply(*rule_tac x*=$\lambda x.\ a + (d/e)$ *$_R$ $(x - b)$ **in** *exI*)
using *assms*
by (*auto intro*!: *continuous_intros simp*: *dist_commute dist_norm pos_divide_le_eq*)
qed

lemma *homeomorphic_spheres*:
fixes *a b* ::*'a*::*real_normed_vector*
assumes $0 < d$ $0 < e$
shows (*sphere a d*) *homeomorphic* (*sphere b e*)
unfolding *homeomorphic_minimal*
apply(*rule_tac x*=$\lambda x.\ b + (e/d)$ *$_R$ $(x - a)$ **in** *exI*)
apply(*rule_tac x*=$\lambda x.\ a + (d/e)$ *$_R$ $(x - b)$ **in** *exI*)
using *assms*
by (*auto intro*!: *continuous_intros simp*: *dist_commute dist_norm pos_divide_less_eq*)

lemma *homeomorphic_ball01_UNIV*:
ball (*0*::*'a*::*real_normed_vector*) *1 homeomorphic* (*UNIV*:: *'a set*)
(**is** *?B homeomorphic ?U*)
proof
have *x* ∈ (λ*z*. *z* /$_R$ (*1* − *norm z*)) ‘ *ball 0 1* **for** *x*::*'a*
apply (*rule_tac x*=*x* /$_R$ (*1* + *norm x*) **in** *image_eqI*)
apply (*auto simp*: *field_split_simps*)

```
    using norm_ge_zero [of x] apply linarith+
    done
  then show (λz::'a. z /R (1 − norm z)) ‘ ?B = ?U
    by blast
  have x ∈ range (λz. (1 / (1 + norm z)) ∗R z) if norm x < 1 for x::'a
    using that
   by (rule_tac x=x /R (1 − norm x) in image_eqI) (auto simp: field_split_simps)
  then show (λz::'a. z /R (1 + norm z)) ‘ ?U = ?B
    by (force simp: field_split_simps dest: add_less_zeroD)
  show continuous_on (ball 0 1) (λz. z /R (1 − norm z))
    by (rule continuous_intros | force)+
  have 0: ⋀z. 1 + norm z ≠ 0
    by (metis (no_types) le_add_same_cancel1 norm_ge_zero not_one_le_zero)
  then show continuous_on UNIV (λz. z /R (1 + norm z))
    by (auto intro!: continuous_intros)
  show ⋀x. x ∈ ball 0 1 ⟹
        x /R (1 − norm x) /R (1 + norm (x /R (1 − norm x))) = x
    by (auto simp: field_split_simps)
  show ⋀y. y /R (1 + norm y) /R (1 − norm (y /R (1 + norm y))) = y
    using 0 by (auto simp: field_split_simps)
qed

proposition homeomorphic_ball_UNIV:
  fixes a ::'a::real_normed_vector
  assumes 0 < r shows ball a r homeomorphic (UNIV:: 'a set)
   using assms homeomorphic_ball01_UNIV homeomorphic_balls(1) homeomor-
phic_trans zero_less_one by blast
```

3.3.19 Discrete

```
lemma finite_implies_discrete:
  fixes S :: 'a::topological_space set
  assumes finite (f ‘ S)
  shows (∀x ∈ S. ∃e>0. ∀y. y ∈ S ∧ f y ≠ f x ⟶ e ≤ norm (f y − f x))
proof −
  have ∃e>0. ∀y. y ∈ S ∧ f y ≠ f x ⟶ e ≤ norm (f y − f x) if x ∈ S for x
  proof (cases f ‘ S − {f x} = {})
    case True
    with zero_less_numeral show ?thesis
      by (fastforce simp add: Set.image_subset_iff cong: conj_cong)
  next
    case False
    then obtain z where z ∈ S f z ≠ f x
      by blast
    moreover have finn: finite {norm (z − f x) |z. z ∈ f ‘ S − {f x}}
      using assms by simp
    ultimately have ∗: 0 < Inf{norm(z − f x) | z. z ∈ f ‘ S − {f x}}
      by (force intro: finite_imp_less_Inf)
    show ?thesis
```

```
    by (force intro!: * cInf_le_finite [OF finn])
  qed
  with assms show ?thesis
    by blast
qed
```

3.3.20 Completeness of "Isometry" (up to constant bounds)

```
lemma cauchy_isometric:— TODO: rename lemma to Cauchy_isometric
  assumes e: e > 0
    and s: subspace s
    and f: bounded_linear f
    and normf: ∀ x∈s. norm (f x) ≥ e * norm x
    and xs: ∀ n. x n ∈ s
    and cf: Cauchy (f ∘ x)
  shows Cauchy x
proof −
  interpret f: bounded_linear f by fact
  have ∃ N. ∀ n≥N. norm (x n − x N) < d if d > 0 for d :: real
  proof −
    from that obtain N where N: ∀ n≥N. norm (f (x n) − f (x N)) < e * d
      using cf[unfolded Cauchy_def o_def dist_norm, THEN spec[where x=e*d]]
e
      by auto
    have norm (x n − x N) < d if n ≥ N for n
    proof −
      have e * norm (x n − x N) ≤ norm (f (x n − x N))
        using subspace_diff[OF s, of x n x N]
        using xs[THEN spec[where x=N]] and xs[THEN spec[where x=n]]
        using normf[THEN bspec[where x=x n − x N]]
        by auto
      also have norm (f (x n − x N)) < e * d
        using ‹N ≤ n› N unfolding f.diff[symmetric] by auto
      finally show ?thesis
        using ‹e>0› by simp
    qed
    then show ?thesis by auto
  qed
  then show ?thesis
    by (simp add: Cauchy_altdef2 dist_norm)
qed

lemma complete_isometric_image:
  assumes 0 < e
    and s: subspace s
    and f: bounded_linear f
    and normf: ∀ x∈s. norm(f x) ≥ e * norm(x)
    and cs: complete s
  shows complete (f ‘ s)
```

proof −
have ∃ *l*∈*f* ‘ *s*. (*g* ⟶ *l*) *sequentially*
if *as*:∀ *n*::*nat*. *g n* ∈ *f* ‘ *s* **and** *cfg*:*Cauchy g* **for** *g*
proof −
from *that* **obtain** *x* **where** ∀ *n*. *x n* ∈ *s* ∧ *g n* = *f* (*x n*)
using *choice*[*of* λ *n xa*. *xa* ∈ *s* ∧ *g n* = *f xa*] **by** *auto*
then have *x*: ∀ *n*. *x n* ∈ *s* ∀ *n*. *g n* = *f* (*x n*) **by** *auto*
then have *f* ∘ *x* = *g* **by** (*simp add*: *fun_eq_iff*)
then obtain *l* **where** *l*∈*s* **and** *l*:(*x* ⟶ *l*) *sequentially*
using *cs*[*unfolded complete_def*, *THEN spec*[**where** *x*=*x*]]
using *cauchy_isometric*[*OF* ‹*0* < *e*› *s f normf*] **and** *cfg* **and** *x*(*1*)
by *auto*
then show *?thesis*
using *linear_continuous_at*[*OF f*, *unfolded continuous_at_sequentially*, *THEN spec*[**where** *x*=*x*], *of l*]
by (*auto simp*: ‹*f* ∘ *x* = *g*›)
qed
then show *?thesis*
unfolding *complete_def* **by** *auto*
qed

3.3.21 Connected Normed Spaces

lemma *compact_components*:
fixes *s* :: *'a*::*heine_borel set*
shows ⟦*compact s*; *c* ∈ *components s*⟧ ⟹ *compact c*
by (*meson bounded_subset closed_components in_components_subset compact_eq_bounded_closed*)

lemma *discrete_subset_disconnected*:
fixes *S* :: *'a*::*topological_space set*
fixes *t* :: *'b*::*real_normed_vector set*
assumes *conf*: *continuous_on S f*
and *no*: ⋀*x*. *x* ∈ *S* ⟹ ∃ *e*>*0*. ∀ *y*. *y* ∈ *S* ∧ *f y* ≠ *f x* ⟶ *e* ≤ *norm* (*f y* − *f x*)
shows *f* ‘ *S* ⊆ {*y*. *connected_component_set* (*f* ‘ *S*) *y* = {*y*}}
proof −
{ fix *x* **assume** *x*: *x* ∈ *S*
then obtain *e* **where** *e*>*0* **and** *ele*: ⋀*y*. ⟦*y* ∈ *S*; *f y* ≠ *f x*⟧ ⟹ *e* ≤ *norm* (*f y* − *f x*)
using *conf no* [*OF x*] **by** *auto*
then have *e2*: *0* ≤ *e*/*2*
by *simp*
define *F* **where** *F* ≡ *connected_component_set* (*f* ‘ *S*) (*f x*)
have *False* **if** *y* ∈ *S* **and** *ccs*: *f y* ∈ *F* **and** *not*: *f y* ≠ *f x* **for** *y*
proof −
define *C* **where** *C* ≡ *cball* (*f x*) (*e*/*2*)
define *D* **where** *D* ≡ − *ball* (*f x*) *e*
have *disj*: *C* ∩ *D* = {}
unfolding *C_def D_def* **using** ‹*0* < *e*› **by** *fastforce*

```
    moreover have FCD: F ⊆ C ∪ D
    proof −
      have t ∈ C ∨ t ∈ D if t ∈ F for t
      proof −
        obtain y where y ∈ S t = f y
          using F_def ‹t ∈ F› connected_component_in by blast
        then show ?thesis
            by (metis C_def ComplI D_def centre_in_cball dist_norm e2 ele
mem_ball norm_minus_commute not_le)
      qed
      then show ?thesis
        by auto
    qed
    ultimately have C ∩ F = {} ∨ D ∩ F = {}
      using connected_closed [of F] ‹e>0› not
      unfolding C_def D_def
       by (metis Elementary_Metric_Spaces.open_ball F_def closed_cball con-
nected_connected_component inf_bot_left open_closed)
    moreover have C ∩ F ≠ {}
      unfolding disjoint_iff
       by (metis FCD ComplD image_eqI mem_Collect_eq subsetD x  D_def
F_def Un_iff ‹0 < e› centre_in_ball connected_component_refl_eq)
    moreover have D ∩ F ≠ {}
      unfolding disjoint_iff
    by (metis ComplI D_def ccs dist_norm ele mem_ball norm_minus_commute
not not_le that(1))
    ultimately show ?thesis by metis
  qed
  moreover have connected_component_set (f ‘ S) (f x) ⊆ f ‘ S
    by (auto simp: connected_component_in)
  ultimately have connected_component_set (f ‘ S) (f x) = {f x}
    by (auto simp: x F_def)
 }
 with assms show ?thesis
   by blast
qed

lemma continuous_disconnected_range_constant_eq:
    (connected S ⟷
        (∀ f::'a::topological_space ⇒ 'b::real_normed_algebra_1.
        ∀ t. continuous_on S f ∧ f ‘ S ⊆ t ∧ (∀ y ∈ t. connected_component_set
t y = {y})
          ⟶ f constant_on S)) (is ?thesis1)
 and continuous_discrete_range_constant_eq:
    (connected S ⟷
      (∀ f::'a::topological_space ⇒ 'b::real_normed_algebra_1.
       continuous_on S f ∧
       (∀ x ∈ S. ∃ e. 0 < e ∧ (∀ y. y ∈ S ∧ (f y ≠ f x) ⟶ e ≤ norm(f y − f x)))
       ⟶ f constant_on S)) (is ?thesis2)
```

```
  and continuous_finite_range_constant_eq:
      (connected S ⟷
        (∀ f::'a::topological_space ⇒ 'b::real_normed_algebra_1.
         continuous_on S f ∧ finite (f ‘ S)
         ⟶ f constant_on S)) (is ?thesis3)
proof −
  have ∗: ⋀s t u v. ⟦s ⟹ t; t ⟹ u; u ⟹ v; v ⟹ s⟧
    ⟹ (s ⟷ t) ∧ (s ⟷ u) ∧ (s ⟷ v)
    by blast
  have ?thesis1 ∧ ?thesis2 ∧ ?thesis3
    apply (rule ∗)
    using continuous_disconnected_range_constant
    apply (metis image_subset_iff_funcset)
   apply (smt (verit, best) discrete_subset_disconnected mem_Collect_eq subsetD
subsetI)
    apply (blast dest: finite_implies_discrete)
    apply (blast intro!: finite_range_constant_imp_connected)
    done
  then show ?thesis1 ?thesis2 ?thesis3
    by blast+
qed

lemma continuous_discrete_range_constant:
  fixes f :: 'a::topological_space ⇒ 'b::real_normed_algebra_1
  assumes S: connected S
      and continuous_on S f
      and ⋀x. x ∈ S ⟹ ∃ e>0. ∀ y. y ∈ S ∧ f y ≠ f x ⟶ e ≤ norm (f y − f x)
    shows f constant_on S
  using continuous_discrete_range_constant_eq [THEN iffD1, OF S] assms by
blast

lemma continuous_finite_range_constant:
  fixes f :: 'a::topological_space ⇒ 'b::real_normed_algebra_1
  assumes connected S
      and continuous_on S f
      and finite (f ‘ S)
    shows f constant_on S
  using assms continuous_finite_range_constant_eq  by blast

end
```

3.4 Linear Decision Procedure for Normed Spaces

```
theory Norm_Arith
imports HOL−Library.Sum_of_Squares
begin


lemma sum_sqs_eq:
```

fixes $x::'a::idom$ **shows** $x * x + y * y = x * (y * 2) \Longrightarrow y = x$
by *algebra*

lemma *norm_cmul_rule_thm*:
fixes $x :: 'a::real\_normed\_vector$
shows $b \geq norm\ x \Longrightarrow |c| * b \geq norm\ (scaleR\ c\ x)$
unfolding *norm_scaleR*
apply (*erule mult_left_mono*)
apply *simp*
done

lemma *norm_add_rule_thm*:
fixes *x1 x2* $:: 'a::real\_normed\_vector$
shows $norm\ x1 \leq b1 \Longrightarrow norm\ x2 \leq b2 \Longrightarrow norm\ (x1 + x2) \leq b1 + b2$
by (*rule order_trans* [*OF norm_triangle_ineq add_mono*])

lemma *ge_iff_diff_ge_0*:
fixes $a :: 'a::linordered\_ring$
shows $a \geq b \equiv a - b \geq 0$
by (*simp add*: *field_simps*)

lemma *pth_1*:
fixes $x :: 'a::real\_normed\_vector$
shows $x \equiv scaleR\ 1\ x$ **by** *simp*

lemma *pth_2*:
fixes $x :: 'a::real\_normed\_vector$
shows $x - y \equiv x + -y$
by (*atomize* (*full*)) *simp*

lemma *pth_3*:
fixes $x :: 'a::real\_normed\_vector$
shows $- x \equiv scaleR\ (-1)\ x$
by *simp*

lemma *pth_4*:
fixes $x :: 'a::real\_normed\_vector$
shows $scaleR\ 0\ x \equiv 0$
and $scaleR\ c\ 0 = (0::'a)$
by *simp_all*

lemma *pth_5*:
fixes $x :: 'a::real\_normed\_vector$
shows $scaleR\ c\ (scaleR\ d\ x) \equiv scaleR\ (c * d)\ x$
by *simp*

lemma *pth_6*:
fixes $x :: 'a::real\_normed\_vector$

shows *scaleR c (x + y) ≡ scaleR c x + scaleR c y*
by (*simp add*: *scaleR_right_distrib*)

lemma *pth_7*:
fixes *x* :: *'a*::*real_normed_vector*
shows *0 + x ≡ x*
and *x + 0 ≡ x*
by *simp_all*

lemma *pth_8*:
fixes *x* :: *'a*::*real_normed_vector*
shows *scaleR c x + scaleR d x ≡ scaleR (c + d) x*
by (*simp add*: *scaleR_left_distrib*)

lemma *pth_9*:
fixes *x* :: *'a*::*real_normed_vector*
shows *(scaleR c x + z) + scaleR d x ≡ scaleR (c + d) x + z*
and *scaleR c x + (scaleR d x + z) ≡ scaleR (c + d) x + z*
and *(scaleR c x + w) + (scaleR d x + z) ≡ scaleR (c + d) x + (w + z)*
by (*simp_all add*: *algebra_simps*)

lemma *pth_a*:
fixes *x* :: *'a*::*real_normed_vector*
shows *scaleR 0 x + y ≡ y*
by *simp*

lemma *pth_b*:
fixes *x* :: *'a*::*real_normed_vector*
shows *scaleR c x + scaleR d y ≡ scaleR c x + scaleR d y*
and *(scaleR c x + z) + scaleR d y ≡ scaleR c x + (z + scaleR d y)*
and *scaleR c x + (scaleR d y + z) ≡ scaleR c x + (scaleR d y + z)*
and *(scaleR c x + w) + (scaleR d y + z) ≡ scaleR c x + (w + (scaleR d y + z))*
by (*simp_all add*: *algebra_simps*)

lemma *pth_c*:
fixes *x* :: *'a*::*real_normed_vector*
shows *scaleR c x + scaleR d y ≡ scaleR d y + scaleR c x*
and *(scaleR c x + z) + scaleR d y ≡ scaleR d y + (scaleR c x + z)*
and *scaleR c x + (scaleR d y + z) ≡ scaleR d y + (scaleR c x + z)*
and *(scaleR c x + w) + (scaleR d y + z) ≡ scaleR d y + ((scaleR c x + w) + z)*
by (*simp_all add*: *algebra_simps*)

lemma *pth_d*:
fixes *x* :: *'a*::*real_normed_vector*
shows *x + 0 ≡ x*
by *simp*

lemma *norm_imp_pos_and_ge*:
fixes x :: *'a*::*real_normed_vector*
shows $norm\ x \equiv n \Longrightarrow norm\ x \geq 0 \wedge n \geq norm\ x$
by *atomize auto*

lemma *real_eq_0_iff_le_ge_0*:
fixes x :: *real*
shows $x = 0 \equiv x \geq 0 \wedge - x \geq 0$
by *arith*

lemma *norm_pths*:
fixes x :: *'a*::*real_normed_vector*
shows $x = y \longleftrightarrow norm\ (x - y) \leq 0$
and $x \neq y \longleftrightarrow \neg\ (norm\ (x - y) \leq 0)$
using *norm_ge_zero*[*of* $x - y$] **by** *auto*

lemmas *arithmetic_simps* =
arith_simps
add_numeral_special
add_neg_numeral_special
mult_1_left
mult_1_right

ML_file ‹*normarith.ML*›

method_setup *norm* = ‹
Scan.succeed (*SIMPLE_METHOD' o NormArith.norm_arith_tac*)
› *prove simple linear statements about vector norms*

Hence more metric properties.

proposition *dist_triangle_add*:
fixes $x\ y\ x'\ y'$ :: *'a*::*real_normed_vector*
shows $dist\ (x + y)\ (x' + y') \leq dist\ x\ x' + dist\ y\ y'$
by *norm*

lemma *dist_triangle_add_half*:
fixes $x\ x'\ y\ y'$ :: *'a*::*real_normed_vector*
shows $dist\ x\ x' < e\ /\ 2 \Longrightarrow dist\ y\ y' < e\ /\ 2 \Longrightarrow dist(x + y)\ (x' + y') < e$
by *norm*

end

Chapter 4

Vector Analysis

theory *Topology_Euclidean_Space*
imports
Elementary_Normed_Spaces
Linear_Algebra
Norm_Arith
begin

4.1 Elementary Topology in Euclidean Space

lemma *euclidean_dist_l2*:
fixes *x y* :: *'a* :: *euclidean_space*
shows *dist x y* = *L2_set* (λ*i*. *dist* (*x* · *i*) (*y* · *i*)) *Basis*
unfolding *dist_norm norm_eq_sqrt_inner L2_set_def*
by (*subst euclidean_inner*) (*simp add*: *power2_eq_square inner_diff_left*)

lemma *norm_nth_le*: *norm* (*x* · *i*) ≤ *norm x* **if** *i* ∈ *Basis*
proof −
have (*x* · *i*)^2 = (∑ *i*∈{*i*}. (*x* · *i*)^2)
by *simp*
also have ... ≤ (∑ *i*∈*Basis*. (*x* · *i*)^2)
by (*intro sum_mono2*) (*auto simp*: *that*)
finally show *?thesis*
unfolding *norm_conv_dist euclidean_dist_l2*[*of x*] *L2_set_def*
by (*auto intro*!: *real_le_rsqrt*)
qed

4.1.1 Continuity of the representation WRT an orthogonal basis

lemma *orthogonal_Basis*: *pairwise orthogonal Basis*
by (*simp add*: *inner_not_same_Basis orthogonal_def pairwise_def*)

lemma *representation_bound*:
fixes *B* :: *'N*::*real_inner set*

assumes *finite B independent B b ∈ B* **and** *orth*: *pairwise orthogonal B*
obtains *m* **where** *m > 0* $\bigwedge x.\ x \in span\ B \Longrightarrow |representation\ B\ x\ b| \leq m * norm\ x$
proof
fix *x*
assume *x*: *x ∈ span B*
have *b ≠ 0*
using ‹*independent B*› ‹*b ∈ B*› *dependent_zero* **by** *blast*
have [*simp*]: $b \cdot b' = (if\ b' = b\ then\ (norm\ b)^2\ else\ 0)$
if *b ∈ B b′ ∈ B* **for** *b b′*
using *orth* **by** (*simp add*: *orthogonal_def pairwise_def norm_eq_sqrt_inner that*)
have $norm\ x = norm\ (\sum b{\in}B.\ representation\ B\ x\ b *_R b)$
using *real_vector.sum_representation_eq* [*OF* ‹*independent B*› *x* ‹*finite B*› *order_refl*]
by *simp*
also have $\ldots = sqrt\ ((\sum b{\in}B.\ representation\ B\ x\ b *_R b) \cdot (\sum b{\in}B.\ representation\ B\ x\ b *_R b))$
by (*simp add*: *norm_eq_sqrt_inner*)
also have $\ldots = sqrt\ (\sum b{\in}B.\ (representation\ B\ x\ b *_R b) \cdot (representation\ B\ x\ b *_R b))$
using ‹*finite B*›
by (*simp add*: *inner_sum_left inner_sum_right if_distrib* [*of* $\lambda x.\ \_ * x$] *cong*: *if_cong sum.cong_simp*)
also have $\ldots = sqrt\ (\sum b{\in}B.\ (norm\ (representation\ B\ x\ b *_R b))^2)$
by (*simp add*: *mult.commute mult.left_commute power2_eq_square*)
also have $\ldots = sqrt\ (\sum b{\in}B.\ (representation\ B\ x\ b)^2 * (norm\ b)^2)$
by (*simp add*: *norm_mult power_mult_distrib*)
finally have $norm\ x = sqrt\ (\sum b{\in}B.\ (representation\ B\ x\ b)^2 * (norm\ b)^2)$.
moreover
have $sqrt\ ((representation\ B\ x\ b)^2 * (norm\ b)^2) \leq sqrt\ (\sum b{\in}B.\ (representation\ B\ x\ b)^2 * (norm\ b)^2)$
using ‹*b ∈ B*› ‹*finite B*› **by** (*auto intro*: *member_le_sum*)
then have $|representation\ B\ x\ b| \leq (1\ /\ norm\ b) * sqrt\ (\sum b{\in}B.\ (representation\ B\ x\ b)^2 * (norm\ b)^2)$
using ‹*b ≠ 0*› **by** (*simp add*: *field_split_simps real_sqrt_mult del*: *real_sqrt_le_iff*)
ultimately show $|representation\ B\ x\ b| \leq (1\ /\ norm\ b) * norm\ x$
by *simp*
next
show *0 < 1 / norm b*
using ‹*independent B*› ‹*b ∈ B*› *dependent_zero* **by** *auto*
qed

lemma *continuous_on_representation*:
fixes *B* :: *'N*::*euclidean_space set*
assumes *finite B independent B b ∈ B pairwise orthogonal B*
shows *continuous_on* (*span B*) ($\lambda x.$ *representation B x b*)
proof
show $\exists\, d{>}0.\ \forall\, x'{\in}span\ B.\ dist\ x'\ x < d \longrightarrow dist\ (representation\ B\ x'\ b)\ (representation$

```
B x b) ≤ e
    if e > 0 x ∈ span B for x e
  proof −
    obtain m where m > 0 and m: ⋀x. x ∈ span B ⟹ |representation B x b| ≤
m * norm x
      using assms representation_bound by blast
    show ?thesis
      unfolding dist_norm
    proof (intro exI conjI ballI impI)
      show e/m > 0
        by (simp add: ‹e > 0› ‹m > 0›)
      show norm (representation B x' b − representation B x b) ≤ e
        if x': x' ∈ span B and less: norm (x'−x) < e/m for x'
      proof −
        have |representation B (x'−x) b| ≤ m * norm (x'−x)
          using m [of x'−x] ‹x ∈ span B› span_diff x' by blast
        also have ... < e
          by (metis ‹m > 0› less mult.commute pos_less_divide_eq)
        finally have |representation B (x'−x) b| ≤ e by simp
        then show ?thesis
          by (simp add: ‹x ∈ span B› ‹independent B› representation_diff x')
      qed
    qed
  qed
qed
```

4.1.2 Balls in Euclidean Space

```
lemma cball_subset_cball_iff:
  fixes a :: 'a :: euclidean_space
  shows cball a r ⊆ cball a' r' ⟷ dist a a' + r ≤ r' ∨ r < 0
    (is ?lhs ⟷ ?rhs)
proof
  assume ?lhs
  then show ?rhs
  proof (cases r < 0)
    case True
    then show ?rhs by simp
  next
    case False
    then have [simp]: r ≥ 0 by simp
    have norm (a − a') + r ≤ r'
    proof (cases a = a')
      case True
      then show ?thesis
        using subsetD [where c = a + r *R (SOME i. i ∈ Basis), OF ‹?lhs›]
subsetD [where c = a, OF ‹?lhs›]
        by (force simp: SOME_Basis dist_norm)
    next
```

```
    case False
    have norm (a' - (a + (r / norm (a - a')) *R (a - a'))) = norm ((-1 -
(r / norm (a - a'))) *R (a - a'))
      by (simp add: algebra_simps)
    also from ‹a ≠ a'› have ... = |- norm (a - a') - r|
      by (simp add: divide_simps)
    finally have [simp]: norm (a' - (a + (r / norm (a - a')) *R (a - a'))) =
|norm (a - a') + r|
      by linarith
    from ‹a ≠ a'› show ?thesis
      using subsetD [where c = a' + (1 + r / norm(a - a')) *R (a - a'), OF
‹?lhs›]
      by (simp add: dist_norm scaleR_add_left)
  qed
  then show ?rhs
    by (simp add: dist_norm)
 qed
qed metric

lemma cball_subset_ball_iff: cball a r ⊆ ball a' r' ⟷ dist a a' + r < r' ∨ r <
0
 (is ?lhs ⟷ ?rhs)
 for a :: 'a::euclidean_space
proof
 assume ?lhs
 then show ?rhs
 proof (cases r < 0)
   case True then
   show ?rhs by simp
 next
   case False
   then have [simp]: r ≥ 0 by simp
   have norm (a - a') + r < r'
   proof (cases a = a')
     case True
     then show ?thesis
       using subsetD [where c = a + r *R (SOME i. i ∈ Basis), OF ‹?lhs›]
subsetD [where c = a, OF ‹?lhs›]
      by (force simp: SOME_Basis dist_norm)
   next
     case False
     have False if norm (a - a') + r ≥ r'
     proof -
       from that have |r' - norm (a - a')| ≤ r
       by (smt (verit, best) ‹0 ≤ r› ‹?lhs› ball_subset_cball cball_subset_cball_iff
dist_norm order_trans)
       then show ?thesis
        using subsetD [where c = a + (r' / norm(a - a') - 1) *R (a - a'), OF
‹?lhs›] ‹a ≠ a'›
```

```
        apply (simp add: dist_norm)
        apply (simp add: scaleR_left_diff_distrib)
        apply (simp add: field_simps)
        done
    qed
    then show ?thesis by force
  qed
  then show ?rhs by (simp add: dist_norm)
 qed
next
 assume ?rhs
 then show ?lhs
  by metric
qed

lemma ball_subset_cball_iff: ball a r ⊆ cball a' r' ⟷ dist a a' + r ≤ r' ∨ r ≤
0
 (is ?lhs = ?rhs)
 for a :: 'a::euclidean_space
proof (cases r ≤ 0)
 case True
 then show ?thesis
  by metric
next
 case False
 show ?thesis
 proof
  assume ?lhs
  then have (cball a r ⊆ cball a' r')
  by (metis False closed_cball closure_ball closure_closed closure_mono not_less)
  with False show ?rhs
   by (fastforce iff: cball_subset_cball_iff)
 next
  assume ?rhs
  with False show ?lhs
   by metric
 qed
qed

lemma ball_subset_ball_iff:
 fixes a :: 'a :: euclidean_space
 shows ball a r ⊆ ball a' r' ⟷ dist a a' + r ≤ r' ∨ r ≤ 0
      (is ?lhs = ?rhs)
proof (cases r ≤ 0)
 case True then show ?thesis
  by metric
next
 case False show ?thesis
 proof
```

assume *?lhs*
then have $0 < r'$
using *False* **by** *metric*
then have *cball a r* $\subseteq$ *cball a' r'*
by (*metis False* ‹*?lhs*› *closure_ball closure_mono not_less*)
then show *?rhs*
using *False cball_subset_cball_iff* **by** *fastforce*
qed *metric*
qed

lemma *ball_eq_ball_iff*:
fixes $x :: 'a :: euclidean\_space$
shows *ball x d* = *ball y e* $\longleftrightarrow$ $d \leq 0 \wedge e \leq 0 \vee x{=}y \wedge d{=}e$
by (*smt* (*verit*, *del_insts*) *ball_empty ball_subset_cball_iff dist_norm norm_pths*(*2*))

lemma *cball_eq_cball_iff*:
fixes $x :: 'a :: euclidean\_space$
shows *cball x d* = *cball y e* $\longleftrightarrow$ $d < 0 \wedge e < 0 \vee x{=}y \wedge d{=}e$
by (*smt* (*verit*, *ccfv_SIG*) *cball_empty cball_subset_cball_iff dist_norm norm_pths*(*2*) *zero_le_dist*)

lemma *ball_eq_cball_iff*:
fixes $x :: 'a :: euclidean\_space$
shows *ball x d* = *cball y e* $\longleftrightarrow$ $d \leq 0 \wedge e < 0$ (**is** *?lhs* = *?rhs*)
by (*smt* (*verit*) *ball_eq_empty ball_subset_cball_iff cball_eq_empty cball_subset_ball_iff order.refl*)

lemma *cball_eq_ball_iff*:
fixes $x :: 'a :: euclidean\_space$
shows *cball x d* = *ball y e* $\longleftrightarrow$ $d < 0 \wedge e \leq 0$
using *ball_eq_cball_iff* **by** *blast*

lemma *finite_ball_avoid*:
fixes $S :: 'a :: euclidean\_space\ set$
assumes *open S finite X p* $\in S$
shows $\exists\, e{>}0.\ \forall\, w{\in}ball\ p\ e.\ w{\in}S \wedge (w{\neq}p \longrightarrow w{\notin}X)$
proof −
obtain *e1* **where** $0 < e1$ **and** *e1_b*:*ball p e1* $\subseteq S$
using *open_contains_ball_eq*[*OF* ‹*open S*›] *assms* **by** *auto*
obtain *e2* **where** $0 < e2$ **and** $\forall\, x{\in}X.\ x \neq p \longrightarrow e2 \leq dist\ p\ x$
using *finite_set_avoid*[*OF* ‹*finite X*›,*of p*] **by** *auto*
hence $\forall\, w{\in}ball\ p\ (min\ e1\ e2).\ w{\in}S \wedge (w{\neq}p \longrightarrow w{\notin}X)$ **using** *e1_b* **by** *auto*
thus $\exists\, e{>}0.\ \forall\, w{\in}ball\ p\ e.\ w \in S \wedge (w \neq p \longrightarrow w \notin X)$
using ‹*e2*>*0*› ‹*e1*>*0*› **by** (*rule_tac x*=*min e1 e2* **in** *exI*) *auto*
qed

lemma *finite_cball_avoid*:
fixes $S :: 'a :: euclidean\_space\ set$

assumes *open S finite X* $p \in S$
shows $\exists e{>}0.\ \forall w{\in}cball\ p\ e.\ w{\in}S \wedge (w{\neq}p \longrightarrow w{\notin}X)$
proof −
obtain *e1* **where** $e1{>}0$ **and** $e1{:}\ \forall w{\in}ball\ p\ e1.\ w{\in}S \wedge (w{\neq}p \longrightarrow w{\notin}X)$
using *finite_ball_avoid*[*OF assms*] **by** *auto*
define *e2* **where** $e2 \equiv e1/2$
have $e2{>}0$ **and** $e2 < e1$ **unfolding** *e2_def* **using** ‹$e1{>}0$› **by** *auto*
then have *cball p e2* $\subseteq$ *ball p e1* **by** (*subst cball_subset_ball_iff*,*auto*)
then show $\exists e{>}0.\ \forall w{\in}cball\ p\ e.\ w \in S \wedge (w \neq p \longrightarrow w \notin X)$ **using** ‹$e2{>}0$› *e1* **by** *auto*
qed

lemma *dim_cball*:
assumes $e > 0$
shows *dim* (*cball* (0 :: 'n::*euclidean_space*) *e*) = *DIM*('n)
proof −
{
fix *x* :: 'n::*euclidean_space*
define *y* **where** $y = (e\ /\ norm\ x) *_R x$
then have $y \in cball\ 0\ e$
using *assms* **by** *auto*
moreover have ∗: $x = (norm\ x\ /\ e) *_R y$
using *y_def assms* **by** *simp*
moreover from ∗ **have** $x = (norm\ x/e) *_R y$
by *auto*
ultimately have $x \in span\ (cball\ 0\ e)$
using *span_scale*[*of y cball 0 e norm x*/*e*]
span_superset[*of cball 0 e*]
by (*simp add*: *span_base*)
}
then have *span* (*cball 0 e*) = (*UNIV* :: 'n::*euclidean_space set*)
by *auto*
then show *?thesis*
using *dim_span*[*of cball* (0 :: 'n::*euclidean_space*) *e*] **by** (*auto*)
qed

4.1.3 Boxes

abbreviation *One* :: 'a::*euclidean_space* **where**
$One \equiv \sum Basis$

lemma *One_non_0*: **assumes** *One* = (0::'a::*euclidean_space*) **shows** *False*
proof −
have *dependent* (*Basis* :: 'a *set*)
apply (*simp add*: *dependent_finite*)
apply (*rule_tac* $x{=}\lambda i.\ 1$ **in** *exI*)
using *SOME_Basis* **apply** (*auto simp*: *assms*)
done
with *independent_Basis* **show** *False* **by** *force*

qed

corollary *One_neq_0*[*iff*]: $One \neq 0$
by (*metis One_non_0*)

corollary *Zero_neq_One*[*iff*]: $0 \neq One$
by (*metis One_non_0*)

definition (**in** *euclidean_space*) *eucl_less* (**infix** $<e$ *50*) **where**
$eucl\_less\ a\ b \longleftrightarrow (\forall i{\in}Basis.\ a \cdot i < b \cdot i)$

definition *box_eucl_less*: $box\ a\ b = \{x.\ a <e\ x \wedge x <e\ b\}$
definition $cbox\ a\ b = \{x.\ \forall i{\in}Basis.\ a \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i\}$

lemma *box_def*: $box\ a\ b = \{x.\ \forall i{\in}Basis.\ a \cdot i < x \cdot i \wedge x \cdot i < b \cdot i\}$
and *in_box_eucl_less*: $x \in box\ a\ b \longleftrightarrow a <e\ x \wedge x <e\ b$
and *mem_box*: $x \in box\ a\ b \longleftrightarrow (\forall i{\in}Basis.\ a \cdot i < x \cdot i \wedge x \cdot i < b \cdot i)$
$x \in cbox\ a\ b \longleftrightarrow (\forall i{\in}Basis.\ a \cdot i \leq x \cdot i \wedge x \cdot i \leq b \cdot i)$
by (*auto simp*: *box_eucl_less eucl_less_def cbox_def*)

lemma *cbox_Pair_eq*: $cbox\ (a,\ c)\ (b,\ d) = cbox\ a\ b \times cbox\ c\ d$
by (*force simp*: *cbox_def Basis_prod_def*)

lemma *cbox_Pair_iff* [*iff*]: $(x,\ y) \in cbox\ (a,\ c)\ (b,\ d) \longleftrightarrow x \in cbox\ a\ b \wedge y \in cbox\ c\ d$
by (*force simp*: *cbox_Pair_eq*)

lemma *cbox_Complex_eq*: $cbox\ (Complex\ a\ c)\ (Complex\ b\ d) = (\lambda(x,y).\ Complex\ x\ y)\ `\ (cbox\ a\ b \times cbox\ c\ d)$
by (*force simp*: *cbox_def Basis_complex_def*)

lemma *cbox_Pair_eq_0*: $cbox\ (a,\ c)\ (b,\ d) = \{\} \longleftrightarrow cbox\ a\ b = \{\} \vee cbox\ c\ d = \{\}$
by (*force simp*: *cbox_Pair_eq*)

lemma *swap_cbox_Pair* [*simp*]: $prod.swap\ `\ cbox\ (c,\ a)\ (d,\ b) = cbox\ (a,c)\ (b,d)$
by *auto*

lemma *mem_box_real*[*simp*]:
$(x{::}real) \in box\ a\ b \longleftrightarrow a < x \wedge x < b$
$(x{::}real) \in cbox\ a\ b \longleftrightarrow a \leq x \wedge x \leq b$
by (*auto simp*: *mem_box*)

lemma *box_real*[*simp*]:
fixes *a b*:: *real*
shows $box\ a\ b = \{a <..< b\}\ cbox\ a\ b = \{a\ ..\ b\}$
by *auto*

lemma *box_Int_box*:

```
  fixes a :: 'a::euclidean_space
  shows box a b ∩ box c d =
    box (∑ i∈Basis. max (a•i) (c•i) *R i) (∑ i∈Basis. min (b•i) (d•i) *R i)
  unfolding set_eq_iff and Int_iff and mem_box by auto

lemma rational_boxes:
  fixes x :: 'a::euclidean_space
  assumes e > 0
  shows ∃ a b. (∀ i∈Basis. a • i ∈ ℚ ∧ b • i ∈ ℚ) ∧ x ∈ box a b ∧ box a b ⊆ ball x
e
proof −
  define e' where e' = e / (2 * sqrt (real (DIM ('a))))
  then have e: e' > 0
    using assms by (auto)
  have ∃ y. y ∈ ℚ ∧ y < x • i ∧ x • i − y < e' for i
    using Rats_dense_in_real[of x • i − e' x • i] e by force
  then obtain a where
    a: ⋀u. a u ∈ ℚ ∧ a u < x • u ∧ x • u − a u < e' by metis
  have ∃ y. y ∈ ℚ ∧ x • i < y ∧ y − x • i < e' for i
    using Rats_dense_in_real[of x • i x • i + e'] e by force
  then obtain b where
    b: ⋀u. b u ∈ ℚ ∧ x • u < b u ∧ b u − x • u < e' by metis
  let ?a = ∑ i∈Basis. a i *R i and ?b = ∑ i∈Basis. b i *R i
  show ?thesis
  proof (rule exI[of _ ?a], rule exI[of _ ?b], safe)
    fix y :: 'a
    assume *: y ∈ box ?a ?b
    have dist x y = sqrt (∑ i∈Basis. (dist (x • i) (y • i))^2)
      unfolding L2_set_def[symmetric] by (rule euclidean_dist_l2)
    also have ... < sqrt (∑ (i::'a)∈Basis. e^2 / real (DIM('a)))
    proof (rule real_sqrt_less_mono, rule sum_strict_mono)
      fix i :: 'a
      assume i: i ∈ Basis
      have a i < y•i ∧ y•i < b i
        using * i by (auto simp: box_def)
      moreover have a i < x•i x•i − a i < e' x•i < b i b i − x•i < e'
        using a b by auto
      ultimately have |x•i − y•i| < 2 * e'
        by auto
      then have dist (x • i) (y • i) < e/sqrt (real (DIM('a)))
        unfolding e'_def by (auto simp: dist_real_def)
      then have (dist (x • i) (y • i))^2 < (e/sqrt (real (DIM('a))))^2
        by (rule power_strict_mono) auto
      then show (dist (x • i) (y • i))^2 < e^2 / real DIM('a)
        by (simp add: power_divide)
    qed auto
    also have ... = e
      using ‹0 < e› by simp
    finally show y ∈ ball x e
```

```
    by (auto simp: ball_def)
  qed (use a b in ‹auto simp: box_def›)
qed

lemma open_UNION_box:
  fixes M :: 'a::euclidean_space set
  assumes open M
  defines a' ≡ λf :: 'a ⇒ real × real. (∑(i::'a)∈Basis. fst (f i) *R i)
  defines b' ≡ λf :: 'a ⇒ real × real. (∑(i::'a)∈Basis. snd (f i) *R i)
  defines I ≡ {f∈Basis →E ℚ × ℚ. box (a' f) (b' f) ⊆ M}
  shows M = (⋃f∈I. box (a' f) (b' f))
proof −
  have x ∈ (⋃f∈I. box (a' f) (b' f)) if x ∈ M for x
  proof −
    obtain e where e: e > 0 ball x e ⊆ M
      using openE[OF ‹open M› ‹x ∈ M›] by auto
    moreover obtain a b where ab:
      x ∈ box a b
      ∀ i ∈ Basis. a · i ∈ ℚ
      ∀ i∈Basis. b · i ∈ ℚ
      box a b ⊆ ball x e
      using rational_boxes[OF e(1)] by metis
    ultimately show ?thesis
      by (intro UN_I[of λi∈Basis. (a · i, b · i)])
        (auto simp: euclidean_representation I_def a'_def b'_def)
  qed
  then show ?thesis by (auto simp: I_def)
qed

corollary open_countable_Union_open_box:
  fixes S :: 'a :: euclidean_space set
  assumes open S
  obtains 𝒟 where countable 𝒟 𝒟 ⊆ Pow S ⋀X. X ∈ 𝒟 ⟹ ∃ a b. X = box a b
⋃𝒟 = S
proof −
  let ?a = λf. (∑(i::'a)∈Basis. fst (f i) *R i)
  let ?b = λf. (∑(i::'a)∈Basis. snd (f i) *R i)
  let ?I = {f∈Basis →E ℚ × ℚ. box (?a f) (?b f) ⊆ S}
  let ?𝒟 = (λf. box (?a f) (?b f)) ‘ ?I
  show ?thesis
  proof
    have countable ?I
      by (simp add: countable_PiE countable_rat)
    then show countable ?𝒟
      by blast
    show ⋃?𝒟 = S
      using open_UNION_box [OF assms] by metis
  qed auto
qed
```

lemma *rational_cboxes*:
fixes $x :: 'a::euclidean\_space$
assumes $e > 0$
shows $\exists a\ b.\ (\forall i{\in}Basis.\ a \cdot i \in \mathbb{Q} \wedge b \cdot i \in \mathbb{Q}) \wedge x \in cbox\ a\ b \wedge cbox\ a\ b \subseteq ball\ x\ e$
proof −
define e' **where** $e' = e\ /\ (2 * sqrt\ (real\ (DIM\ ('a))))$
then have e': $e' > 0$
using *assms* **by** *auto*
have $\exists y.\ y \in \mathbb{Q} \wedge y < x \cdot i \wedge x \cdot i - y < e'$ **for** i
using *Rats_dense_in_real*[*of* $x \cdot i - e'\ x \cdot i$] e **by** *force*
then obtain a **where**
a: $\forall u.\ a\ u \in \mathbb{Q} \wedge a\ u < x \cdot u \wedge x \cdot u - a\ u < e'$ **by** *metis*
have $\exists y.\ y \in \mathbb{Q} \wedge x \cdot i < y \wedge y - x \cdot i < e'$ **for** i
using *Rats_dense_in_real*[*of* $x \cdot i\ x \cdot i + e'$] e **by** *force*
then obtain b **where**
b: $\forall u.\ b\ u \in \mathbb{Q} \wedge x \cdot u < b\ u \wedge b\ u - x \cdot u < e'$ **by** *metis*
let $?a = \sum i{\in}Basis.\ a\ i *_R i$ **and** $?b = \sum i{\in}Basis.\ b\ i *_R i$
show *?thesis*
proof (*rule exI*[*of* _ *?a*], *rule exI*[*of* _ *?b*], *safe*)
fix $y :: 'a$
assume $*$: $y \in cbox\ ?a\ ?b$
have $dist\ x\ y = sqrt\ (\sum i{\in}Basis.\ (dist\ (x \cdot i)\ (y \cdot i))^2)$
unfolding *L2_set_def*[*symmetric*] **by** (*rule euclidean_dist_l2*)
also have $\ldots < sqrt\ (\sum (i::'a){\in}Basis.\ e\hat{}2\ /\ real\ (DIM('a)))$
proof (*rule real_sqrt_less_mono*, *rule sum_strict_mono*)
fix $i :: 'a$
assume i: $i \in Basis$
have $a\ i \le y{\cdot}i \wedge y{\cdot}i \le b\ i$
using $*$ i **by** (*auto simp*: *cbox_def*)
moreover have $a\ i < x{\cdot}i\ x{\cdot}i - a\ i < e'\ x{\cdot}i < b\ i\ b\ i - x{\cdot}i < e'$
using a b **by** *auto*
ultimately have $|x{\cdot}i - y{\cdot}i| < 2 * e'$
by *auto*
then have $dist\ (x \cdot i)\ (y \cdot i) < e/sqrt\ (real\ (DIM('a)))$
unfolding e'*_def* **by** (*auto simp*: *dist_real_def*)
then have $(dist\ (x \cdot i)\ (y \cdot i))^2 < (e/sqrt\ (real\ (DIM('a))))^2$
by (*rule power_strict_mono*) *auto*
then show $(dist\ (x \cdot i)\ (y \cdot i))^2 < e^2\ /\ real\ DIM('a)$
by (*simp add*: *power_divide*)
qed *auto*
also have $\ldots = e$
using ‹$0 < e$› **by** *simp*
finally show $y \in ball\ x\ e$
by (*auto simp*: *ball_def*)
next
show $x \in cbox\ (\sum i{\in}Basis.\ a\ i *_R i)\ (\sum i{\in}Basis.\ b\ i *_R i)$
using a b *less_imp_le* **by** (*auto simp*: *cbox_def*)

```
  qed (use a b cbox_def in auto)
qed

lemma open_UNION_cbox:
  fixes M :: 'a::euclidean_space set
  assumes open M
  defines a' ≡ λf. (∑(i::'a)∈Basis. fst (f i) *R i)
  defines b' ≡ λf. (∑(i::'a)∈Basis. snd (f i) *R i)
  defines I ≡ {f∈Basis →E ℚ × ℚ. cbox (a' f) (b' f) ⊆ M}
  shows M = (⋃f∈I. cbox (a' f) (b' f))
proof −
  have x ∈ (⋃f∈I. cbox (a' f) (b' f)) if x ∈ M for x
  proof −
    obtain e where e: e > 0 ball x e ⊆ M
      using openE[OF ‹open M› ‹x ∈ M›] by auto
    moreover obtain a b where ab: x ∈ cbox a b ∀ i ∈ Basis. a · i ∈ ℚ
                                  ∀ i ∈ Basis. b · i ∈ ℚ cbox a b ⊆ ball x e
      using rational_cboxes[OF e(1)] by metis
    ultimately show ?thesis
      by (intro UN_I[of λi∈Basis. (a · i, b · i)])
         (auto simp: euclidean_representation I_def a'_def b'_def)
  qed
  then show ?thesis by (auto simp: I_def)
qed

corollary open_countable_Union_open_cbox:
  fixes S :: 'a :: euclidean_space set
  assumes open S
  obtains 𝒟 where countable 𝒟 𝒟 ⊆ Pow S ⋀X. X ∈ 𝒟 ⟹ ∃ a b. X = cbox a
b ⋃𝒟 = S
proof −
  let ?a = λf. (∑(i::'a)∈Basis. fst (f i) *R i)
  let ?b = λf. (∑(i::'a)∈Basis. snd (f i) *R i)
  let ?I = {f∈Basis →E ℚ × ℚ. cbox (?a f) (?b f) ⊆ S}
  let ?𝒟 = (λf. cbox (?a f) (?b f)) ‘ ?I
  show ?thesis
  proof
    have countable ?I
      by (simp add: countable_PiE countable_rat)
    then show countable ?𝒟
      by blast
    show ⋃?𝒟 = S
      using open_UNION_cbox [OF assms] by metis
  qed auto
qed

lemma box_eq_empty:
  fixes a :: 'a::euclidean_space
  shows (box a b = {} ⟷ (∃ i∈Basis. b·i ≤ a·i)) (is ?th1)
```

```
    and (cbox a b = {} ⟷ (∃ i∈Basis. b•i < a•i)) (is ?th2)
proof −
  have False if i ∈ Basis and b•i ≤ a•i and x ∈ box a b for i x
    by (smt (verit, ccfv_SIG) mem_box(1) that)
  moreover
  { assume as: ∀ i∈Basis. ¬ (b•i ≤ a•i)
    let ?x = (1/2) *R (a + b)
    { fix i :: 'a
      assume i: i ∈ Basis
      have a•i < b•i
        using as i by fastforce
      then have a•i < ((1/2) *R (a+b)) • i ((1/2) *R (a+b)) • i < b•i
        by (auto simp: inner_add_left)
    }
    then have box a b ≠ {}
      by (metis (no_types, opaque_lifting) emptyE mem_box(1))
  }
  ultimately show ?th1 by blast

  have False if i∈Basis and b•i < a•i and x ∈ cbox a b for i x
    using mem_box(2) that by force
  moreover
  have cbox a b ≠ {} if ∀ i∈Basis. ¬ (b•i < a•i)
    by (metis emptyE linorder_linear mem_box(2) order.strict_iff_not that)
  ultimately show ?th2 by blast
qed

lemma box_ne_empty:
  fixes a :: 'a::euclidean_space
  shows cbox a b ≠ {} ⟷ (∀ i∈Basis. a•i ≤ b•i)
  and box a b ≠ {} ⟷ (∀ i∈Basis. a•i < b•i)
  unfolding box_eq_empty[of a b] by fastforce+

lemma
  fixes a :: 'a::euclidean_space
  shows cbox_idem [simp]: cbox a a = {a}
    and box_idem [simp]: box a a = {}
  unfolding set_eq_iff mem_box eq_iff [symmetric] using euclidean_eq_iff by
fastforce+

lemma subset_box_imp:
  fixes a :: 'a::euclidean_space
  shows (∀ i∈Basis. a•i ≤ c•i ∧ d•i ≤ b•i) ⟹ cbox c d ⊆ cbox a b
    and (∀ i∈Basis. a•i < c•i ∧ d•i < b•i) ⟹ cbox c d ⊆ box a b
    and (∀ i∈Basis. a•i ≤ c•i ∧ d•i ≤ b•i) ⟹ box c d ⊆ cbox a b
     and (∀ i∈Basis. a•i ≤ c•i ∧ d•i ≤ b•i) ⟹ box c d ⊆ box a b
  unfolding subset_eq[unfolded Ball_def] unfolding mem_box
  by (best intro: order_trans less_le_trans le_less_trans less_imp_le)+
```

lemma *box_subset_cbox*:
fixes *a* :: *'a*::*euclidean_space*
shows *box a b* ⊆ *cbox a b*
unfolding *subset_eq* [*unfolded Ball_def*] *mem_box*
by (*fast intro*: *less_imp_le*)

lemma *subset_box*:
fixes *a* :: *'a*::*euclidean_space*
shows *cbox c d* ⊆ *cbox a b* ⟷ (∀ *i*∈*Basis*. *c*•*i* ≤ *d*•*i*) ⟶ (∀ *i*∈*Basis*. *a*•*i* ≤ *c*•*i* ∧ *d*•*i* ≤ *b*•*i*) (**is** *?th1*)
and *cbox c d* ⊆ *box a b* ⟷ (∀ *i*∈*Basis*. *c*•*i* ≤ *d*•*i*) ⟶ (∀ *i*∈*Basis*. *a*•*i* < *c*•*i* ∧ *d*•*i* < *b*•*i*) (**is** *?th2*)
and *box c d* ⊆ *cbox a b* ⟷ (∀ *i*∈*Basis*. *c*•*i* < *d*•*i*) ⟶ (∀ *i*∈*Basis*. *a*•*i* ≤ *c*•*i* ∧ *d*•*i* ≤ *b*•*i*) (**is** *?th3*)
and *box c d* ⊆ *box a b* ⟷ (∀ *i*∈*Basis*. *c*•*i* < *d*•*i*) ⟶ (∀ *i*∈*Basis*. *a*•*i* ≤ *c*•*i* ∧ *d*•*i* ≤ *b*•*i*) (**is** *?th4*)
proof −
let *?lesscd* = ∀ *i*∈*Basis*. *c*•*i* < *d*•*i*
let *?lerhs* = ∀ *i*∈*Basis*. *a*•*i* ≤ *c*•*i* ∧ *d*•*i* ≤ *b*•*i*
show *?th1 ?th2*
by (*fastforce simp*: *mem_box*)+
have *acdb*: *a*•*i* ≤ *c*•*i* ∧ *d*•*i* ≤ *b*•*i*
if *i*: *i* ∈ *Basis* **and** *box*: *box c d* ⊆ *cbox a b* **and** *cd*: ⋀*i*. *i* ∈ *Basis* ⟹ *c*•*i* < *d*•*i* **for** *i*
proof −
have *box c d* ≠ {}
using *that*
unfolding *box_eq_empty* **by** *force*
{ let *?x* = (∑ *j*∈*Basis*. (*if j*=*i then* ((*min* (*a*•*j*) (*d*•*j*))+*c*•*j*)/*2 else* (*c*•*j*+*d*•*j*)/*2*) ∗*R* *j*)::*'a*
assume ∗: *a*•*i* > *c*•*i*
then have *c* • *j* < *?x* • *j* ∧ *?x* • *j* < *d* • *j* **if** *j* ∈ *Basis* **for** *j*
using *cd that* **by** (*fastforce simp add*: *i* ∗)
then have *?x* ∈ *box c d*
unfolding *mem_box* **by** *auto*
moreover have *?x* ∉ *cbox a b*
using *i cd* ∗ **by** (*force simp*: *mem_box*)
ultimately have *False* **using** *box* **by** *auto*
}
then have *a*•*i* ≤ *c*•*i* **by** *force*
moreover
{ let *?x* = (∑ *j*∈*Basis*. (*if j*=*i then* ((*max* (*b*•*j*) (*c*•*j*))+*d*•*j*)/*2 else* (*c*•*j*+*d*•*j*)/*2*) ∗*R* *j*)::*'a*
assume ∗: *b*•*i* < *d*•*i*
then have *d* • *j* > *?x* • *j* ∧ *?x* • *j* > *c* • *j* **if** *j* ∈ *Basis* **for** *j*
using *cd that* **by** (*fastforce simp add*: *i* ∗)
then have *?x* ∈ *box c d*
unfolding *mem_box* **by** *auto*
moreover have *?x* ∉ *cbox a b*

```
      using i cd * by (force simp: mem_box)
    ultimately have False using box by auto
  }
  then have b·i ≥ d·i by (rule ccontr) auto
  ultimately show ?thesis by auto
 qed
 show ?th3
  using acdb by (fastforce simp add: mem_box)
 have acdb': a·i ≤ c·i ∧ d·i ≤ b·i
  if i ∈ Basis box c d ⊆ box a b ⋀i. i ∈ Basis ⟹ c·i < d·i for i
   using box_subset_cbox[of a b] that acdb by auto
 show ?th4
  using acdb' by (fastforce simp add: mem_box)
qed

lemma eq_cbox: cbox a b = cbox c d ⟷ cbox a b = {} ∧ cbox c d = {} ∨ a =
c ∧ b = d
    (is ?lhs = ?rhs)
proof
 assume ?lhs
 then have cbox a b ⊆ cbox c d cbox c d ⊆ cbox a b
  by auto
 then show ?rhs
  by (force simp: subset_box box_eq_empty intro: antisym euclidean_eqI)
qed auto

lemma eq_cbox_box [simp]: cbox a b = box c d ⟷ cbox a b = {} ∧ box c d =
{}
 (is ?lhs ⟷ ?rhs)
proof
 assume L: ?lhs
 then have cbox a b ⊆ box c d box c d ⊆ cbox a b
  by auto
 with L subset_box show ?rhs
  by (smt (verit) SOME_Basis box_ne_empty(1))
qed force

lemma eq_box_cbox [simp]: box a b = cbox c d ⟷ box a b = {} ∧ cbox c d =
{}
 by (metis eq_cbox_box)

lemma eq_box: box a b = box c d ⟷ box a b = {} ∧ box c d = {} ∨ a = c ∧ b
= d
 (is ?lhs ⟷ ?rhs)
proof
 assume L: ?lhs
 then have box a b ⊆ box c d box c d ⊆ box a b
  by auto
 then show ?rhs
```

unfolding *subset_box* **by** (*smt* (*verit*) *box_ne_empty*(2) *euclidean_eq_iff*)+
qed *force*

lemma *subset_box_complex*:
cbox a b ⊆ *cbox c d* ⟷
(*Re a* ≤ *Re b* ∧ *Im a* ≤ *Im b*) ⟶ *Re a* ≥ *Re c* ∧ *Im a* ≥ *Im c* ∧ *Re b* ≤ *Re d* ∧ *Im b* ≤ *Im d*
cbox a b ⊆ *box c d* ⟷
(*Re a* ≤ *Re b* ∧ *Im a* ≤ *Im b*) ⟶ *Re a* > *Re c* ∧ *Im a* > *Im c* ∧ *Re b* < *Re d* ∧ *Im b* < *Im d*
box a b ⊆ *cbox c d* ⟷
(*Re a* < *Re b* ∧ *Im a* < *Im b*) ⟶ *Re a* ≥ *Re c* ∧ *Im a* ≥ *Im c* ∧ *Re b* ≤ *Re d* ∧ *Im b* ≤ *Im d*
box a b ⊆ *box c d* ⟷
(*Re a* < *Re b* ∧ *Im a* < *Im b*) ⟶ *Re a* ≥ *Re c* ∧ *Im a* ≥ *Im c* ∧ *Re b* ≤ *Re d* ∧ *Im b* ≤ *Im d*
by (*subst subset_box*; *force simp*: *Basis_complex_def*)+

lemma *in_cbox_complex_iff*:
x ∈ *cbox a b* ⟷ *Re x* ∈ {*Re a*..*Re b*} ∧ *Im x* ∈ {*Im a*..*Im b*}
by (*cases x*; *cases a*; *cases b*) (*auto simp*: *cbox_Complex_eq*)

lemma *cbox_complex_of_real*: *cbox* (*complex_of_real x*) (*complex_of_real y*) = *complex_of_real* ‘ {*x*..*y*}
proof −
have (*x* ≤ *Re z* ∧ *Re z* ≤ *y* ∧ *Im z* = *0*) = (*z* ∈ *complex_of_real* ‘ {*x*..*y*}) **for** *z*
by (*cases z*) (*simp add*: *complex_eq_cancel_iff2 image_iff*)
then show *?thesis*
by (*auto simp*: *in_cbox_complex_iff*)
qed

lemma *box_Complex_eq*:
box (*Complex a c*) (*Complex b d*) = (λ(*x*,*y*). *Complex x y*) ‘ (*box a b* × *box c d*)
by (*auto simp*: *box_def Basis_complex_def image_iff complex_eq_iff*)

lemma *in_box_complex_iff*:
x ∈ *box a b* ⟷ *Re x* ∈ {*Re a*<..<*Re b*} ∧ *Im x* ∈ {*Im a*<..<*Im b*}
by (*cases x*; *cases a*; *cases b*) (*auto simp*: *box_Complex_eq*)

lemma *box_complex_of_real* [*simp*]: *box* (*complex_of_real x*) (*complex_of_real y*) = {}
by (*auto simp*: *in_box_complex_iff*)

lemma *Int_interval*:
fixes *a* :: *'a*::*euclidean_space*
shows *cbox a b* ∩ *cbox c d* =
cbox (∑ *i*∈*Basis*. *max* (*a*•*i*) (*c*•*i*) ∗$_R$ *i*) (∑ *i*∈*Basis*. *min* (*b*•*i*) (*d*•*i*) ∗$_R$ *i*)
unfolding *set_eq_iff* **and** *Int_iff* **and** *mem_box*
by *auto*

lemma *disjoint_interval*:
fixes *a*::'*a*::*euclidean_space*
shows *cbox a b* ∩ *cbox c d* = {} ⟷ (∃ *i*∈*Basis*. (*b*•*i* < *a*•*i* ∨ *d*•*i* < *c*•*i* ∨ *b*•*i* < *c*•*i* ∨ *d*•*i* < *a*•*i*)) (**is** *?th1*)
and *cbox a b* ∩ *box c d* = {} ⟷ (∃ *i*∈*Basis*. (*b*•*i* < *a*•*i* ∨ *d*•*i* ≤ *c*•*i* ∨ *b*•*i* ≤ *c*•*i* ∨ *d*•*i* ≤ *a*•*i*)) (**is** *?th2*)
and *box a b* ∩ *cbox c d* = {} ⟷ (∃ *i*∈*Basis*. (*b*•*i* ≤ *a*•*i* ∨ *d*•*i* < *c*•*i* ∨ *b*•*i* ≤ *c*•*i* ∨ *d*•*i* ≤ *a*•*i*)) (**is** *?th3*)
and *box a b* ∩ *box c d* = {} ⟷ (∃ *i*∈*Basis*. (*b*•*i* ≤ *a*•*i* ∨ *d*•*i* ≤ *c*•*i* ∨ *b*•*i* ≤ *c*•*i* ∨ *d*•*i* ≤ *a*•*i*)) (**is** *?th4*)
proof −
let *?z* = (∑ *i*∈*Basis*. (((*max* (*a*•*i*) (*c*•*i*)) + (*min* (*b*•*i*) (*d*•*i*))) / *2*) *R* *i*)::'*a*
have **: ⋀*P Q*. (⋀*i* :: '*a*. *i* ∈ *Basis* ⟹ *Q ?z i* ⟹ *P i*) ⟹
(⋀*i x* :: '*a*. *i* ∈ *Basis* ⟹ *P i* ⟹ *Q x i*) ⟹ (∀ *x*. ∃ *i*∈*Basis*. *Q x i*) ⟷ (∃ *i*∈*Basis*. *P i*)
by *blast*
note * = *set_eq_iff Int_iff empty_iff mem_box ball_conj_distrib*[*symmetric*] *eq_False ball_simps*(*10*)
show *?th1* **unfolding** * **by** (*intro* **) *auto*
show *?th2* **unfolding** * **by** (*intro* **) *auto*
show *?th3* **unfolding** * **by** (*intro* **) *auto*
show *?th4* **unfolding** * **by** (*intro* **) *auto*
qed

lemma *UN_box_eq_UNIV*: (⋃ *i*::*nat*. *box* (− (*real i R One*)) (*real i R One*)) = *UNIV*
proof −
have |*x* • *b*| < *real_of_int* (⌈*Max* ((λ*b*. |*x* • *b*|) ‘*Basis*)⌉ + *1*)
if [*simp*]: *b* ∈ *Basis* **for** *x b* :: '*a*
proof −
have |*x* • *b*| ≤ *real_of_int* ⌈|*x* • *b*|⌉
by (*rule le_of_int_ceiling*)
also have ... ≤ *real_of_int* ⌈*Max* ((λ*b*. |*x* • *b*|) ‘*Basis*)⌉
by (*auto intro*!: *ceiling_mono*)
also have ... < *real_of_int* (⌈*Max* ((λ*b*. |*x* • *b*|) ‘*Basis*)⌉ + *1*)
by *simp*
finally show *?thesis* .
qed
then have ∃ *n*::*nat*. ∀ *b*∈*Basis*. |*x* • *b*| < *real n* **for** *x* :: '*a*
by (*metis order.strict_trans reals_Archimedean2*)
moreover have ⋀*x b*::'*a*. ⋀*n*::*nat*. |*x* • *b*| < *real n* ⟷ − *real n* < *x* • *b* ∧ *x* • *b* < *real n*
by *auto*
ultimately show *?thesis*
by (*auto simp*: *box_def inner_sum_left inner_Basis sum.If_cases*)
qed

lemma *image_affinity_cbox*: **fixes** *m*::*real*

```
  fixes a b c :: 'a::euclidean_space
  shows (λx. m *R x + c) ‘ cbox a b =
    (if cbox a b = {} then {}
     else (if 0 ≤ m then cbox (m *R a + c) (m *R b + c)
     else cbox (m *R b + c) (m *R a + c)))
proof (cases m = 0)
  case True
  {
    fix x
    assume ∀ i∈Basis. x · i ≤ c · i ∀ i∈Basis. c · i ≤ x · i
    then have x = c
      by (simp add: dual_order.antisym euclidean_eqI)
  }
  moreover have c ∈ cbox (m *R a + c) (m *R b + c)
    unfolding True by auto
  ultimately show ?thesis using True by (auto simp: cbox_def)
next
  case False
  {
    fix y
    assume ∀ i∈Basis. a · i ≤ y · i ∀ i∈Basis. y · i ≤ b · i m > 0
    then have ∀ i∈Basis. (m *R a + c) · i ≤ (m *R y + c) · i
          and ∀ i∈Basis. (m *R y + c) · i ≤ (m *R b + c) · i
      by (auto simp: inner_distrib)
  }
  moreover
  {
    fix y
    assume ∀ i∈Basis. a · i ≤ y · i ∀ i∈Basis. y · i ≤ b · i m < 0
    then have ∀ i∈Basis. (m *R b + c) · i ≤ (m *R y + c) · i
         and  ∀ i∈Basis. (m *R y + c) · i ≤ (m *R a + c) · i
      by (auto simp: mult_left_mono_neg inner_distrib)
  }
  moreover
  {
    fix y
    assume m > 0 and ∀ i∈Basis. (m *R a + c) · i ≤ y · i
      and  ∀ i∈Basis. y · i ≤ (m *R b + c) · i
    then have y ∈ (λx. m *R x + c) ‘ cbox a b
      unfolding image_iff Bex_def mem_box
      apply (intro exI[where x=(1 / m) *R (y − c)])
       apply (auto simp: pos_le_divide_eq pos_divide_le_eq mult.commute in-
ner_distrib inner_diff_left)
      done
  }
  moreover
  {
    fix y
    assume ∀ i∈Basis. (m *R b + c) · i ≤ y · i ∀ i∈Basis. y · i ≤ (m *R a + c) ·
```

i m < 0
then have *y* ∈ (λ*x*. *m* ∗*R* *x* + *c*) ‘ *cbox a b*
unfolding *image_iff Bex_def mem_box*
apply (*intro exI*[**where** *x*=(*1* / *m*) ∗*R* (*y* − *c*)])
apply (*auto simp*: *neg_le_divide_eq neg_divide_le_eq mult.commute inner_distrib inner_diff_left*)
done
}
ultimately show *?thesis* **using** *False* **by** (*auto simp*: *cbox_def*)
qed

lemma *image_smult_cbox*:(λ*x*. *m* ∗*R* (*x*::_::*euclidean_space*)) ‘ *cbox a b* =
(*if cbox a b* = {} *then* {} *else if* *0* ≤ *m then cbox* (*m* ∗*R* *a*) (*m* ∗*R* *b*) *else cbox* (*m* ∗*R* *b*) (*m* ∗*R* *a*))
using *image_affinity_cbox*[*of m 0 a b*] **by** *auto*

lemma *swap_continuous*:
assumes *continuous_on* (*cbox* (*a*,*c*) (*b*,*d*)) (λ(*x*,*y*). *f x y*)
shows *continuous_on* (*cbox* (*c*,*a*) (*d*,*b*)) (λ(*x*, *y*). *f y x*)
proof −
have (λ(*x*, *y*). *f y x*) = (λ(*x*, *y*). *f x y*) ∘ *prod.swap*
by *auto*
then show *?thesis*
by (*metis assms continuous_on_compose continuous_on_swap swap_cbox_Pair*)
qed

lemma *open_contains_cbox*:
fixes *x* :: ′*a* :: *euclidean_space*
assumes *open A x* ∈ *A*
obtains *a b* **where** *cbox a b* ⊆ *A x* ∈ *box a b* ∀ *i*∈*Basis*. *a* ∙ *i* < *b* ∙ *i*
proof −
from *assms* **obtain** *R* **where** *R*: *R* > *0 ball x R* ⊆ *A*
by (*auto simp*: *open_contains_ball*)
define *r* :: *real* **where** *r* = *R* / (*2* ∗ *sqrt DIM*(′*a*))
from ‹*R* > *0*› **have** [*simp*]: *r* > *0* **by** (*auto simp*: *r_def*)
define *d* :: ′*a* **where** *d* = *r* ∗*R* *Topology_Euclidean_Space.One*
have *cbox* (*x* − *d*) (*x* + *d*) ⊆ *A*
proof *safe*
fix *y* **assume** *y*: *y* ∈ *cbox* (*x* − *d*) (*x* + *d*)
have *dist x y* = *sqrt* (∑ *i*∈*Basis*. (*dist* (*x* ∙ *i*) (*y* ∙ *i*))2)
by (*subst euclidean_dist_l2*) (*auto simp*: *L2_set_def*)
also from *y* **have** *sqrt* (∑ *i*∈*Basis*. (*dist* (*x* ∙ *i*) (*y* ∙ *i*))2) ≤ *sqrt* (∑ *i*∈(*Basis*::′*a set*). *r*2)
by (*intro real_sqrt_le_mono sum_mono power_mono*)
(*auto simp*: *dist_norm d_def cbox_def algebra_simps*)
also have ... = *sqrt* (*DIM*(′*a*) ∗ *r*2) **by** *simp*
also have *DIM*(′*a*) ∗ *r*2 = (*R* / *2*) ^ *2*
by (*simp add*: *r_def power_divide*)
also have *sqrt* ... = *R* / *2*

```
    using ‹R > 0› by simp
  also from ‹R > 0› have ... < R by simp
  finally have y ∈ ball x R by simp
  with R show y ∈ A by blast
 qed
 thus ?thesis
  using that[of x − d x + d] by (auto simp: algebra_simps d_def box_def)
qed

lemma open_contains_box:
 fixes x :: 'a :: euclidean_space
 assumes open A x ∈ A
 obtains a b where box a b ⊆ A x ∈ box a b ∀ i∈Basis. a • i < b • i
 by (meson assms box_subset_cbox dual_order.trans open_contains_cbox)

lemma inner_image_box:
 assumes (i :: 'a :: euclidean_space) ∈ Basis
 assumes ∀ i∈Basis. a • i < b • i
 shows   (λx. x • i) ‘ box a b = {a • i<..<b • i}
proof safe
 fix x assume x: x ∈ {a • i<..<b • i}
 let ?y = (∑ j∈Basis. (if i = j then x else (a + b) • j / 2) *R j)
 from x assms have ?y • i ∈ (λx. x • i) ‘ box a b
  by (intro imageI) (auto simp: box_def algebra_simps)
 also have ?y • i = (∑ j∈Basis. (if i = j then x else (a + b) • j / 2) * (j • i))
  by (simp add: inner_sum_left)
 also have ... = (∑ j∈Basis. if i = j then x else 0)
  by (intro sum.cong) (auto simp: inner_not_same_Basis assms)
 also have ... = x using assms by simp
 finally show x ∈ (λx. x • i) ‘ box a b  .
qed (insert assms, auto simp: box_def)

lemma inner_image_cbox:
 assumes (i :: 'a :: euclidean_space) ∈ Basis
 assumes ∀ i∈Basis. a • i ≤ b • i
 shows   (λx. x • i) ‘ cbox a b = {a • i..b • i}
proof safe
 fix x assume x: x ∈ {a • i..b • i}
 let ?y = (∑ j∈Basis. (if i = j then x else a • j) *R j)
 from x assms have ?y • i ∈ (λx. x • i) ‘ cbox a b
  by (intro imageI) (auto simp: cbox_def)
 also have ?y • i = (∑ j∈Basis. (if i = j then x else a • j) * (j • i))
  by (simp add: inner_sum_left)
 also have ... = (∑ j∈Basis. if i = j then x else 0)
  by (intro sum.cong) (auto simp: inner_not_same_Basis assms)
 also have ... = x using assms by simp
 finally show x ∈ (λx. x • i) ‘ cbox a b  .
qed (insert assms, auto simp: cbox_def)
```

4.1.4 General Intervals

definition *is_interval* (*s*::('*a*::*euclidean_space*) *set*) ⟷
(∀ *a*∈*s*. ∀ *b*∈*s*. ∀ *x*. (∀ *i*∈*Basis*. ((*a*•*i* ≤ *x*•*i* ∧ *x*•*i* ≤ *b*•*i*) ∨ (*b*•*i* ≤ *x*•*i* ∧ *x*•*i* ≤ *a*•*i*))) ⟶ *x* ∈ *s*)

lemma *is_interval_1*:
is_interval (*s*::*real set*) ⟷ (∀ *a*∈*s*. ∀ *b*∈*s*. ∀ *x*. *a* ≤ *x* ∧ *x* ≤ *b* ⟶ *x* ∈ *s*)
unfolding *is_interval_def* **by** *auto*

lemma *is_interval_Int*: *is_interval X* ⟹ *is_interval Y* ⟹ *is_interval* (*X* ∩ *Y*)
unfolding *is_interval_def*
by *blast*

lemma *is_interval_cbox* [*simp*]: *is_interval* (*cbox a* (*b*::'*a*::*euclidean_space*)) (**is** *?th1*)
and *is_interval_box* [*simp*]: *is_interval* (*box a b*) (**is** *?th2*)
unfolding *is_interval_def mem_box Ball_def atLeastAtMost_iff*
by (*meson order_trans le_less_trans less_le_trans less_trans*)+

lemma *is_interval_empty* [*iff*]: *is_interval* {}
unfolding *is_interval_def* **by** *simp*

lemma *is_interval_univ* [*iff*]: *is_interval UNIV*
unfolding *is_interval_def* **by** *simp*

lemma *mem_is_intervalI*:
assumes *is_interval S*
and *a* ∈ *S b* ∈ *S*
and ⋀*i*. *i* ∈ *Basis* ⟹ *a* • *i* ≤ *x* • *i* ∧ *x* • *i* ≤ *b* • *i* ∨ *b* • *i* ≤ *x* • *i* ∧ *x* • *i* ≤ *a* • *i*
shows *x* ∈ *S*
using *assms is_interval_def* **by** *force*

lemma *interval_subst*:
fixes *S*::'*a*::*euclidean_space set*
assumes *is_interval S*
and *x* ∈ *S y j* ∈ *S*
and *j* ∈ *Basis*
shows (∑ *i*∈*Basis*. (*if i* = *j then y i* • *i else x* • *i*) ∗$_R$ *i*) ∈ *S*
by (*rule mem_is_intervalI*[*OF assms*(1,2)]) (*auto simp*: *assms*)

lemma *mem_box_componentwiseI*:
fixes *S*::'*a*::*euclidean_space set*
assumes *is_interval S*
assumes ⋀*i*. *i* ∈ *Basis* ⟹ *x* • *i* ∈ ((λ*x*. *x* • *i*) ‘ *S*)
shows *x* ∈ *S*
proof −
from *assms* **have** ∀ *i* ∈ *Basis*. ∃ *s* ∈ *S*. *x* • *i* = *s* • *i*

```
    by auto
  with finite_Basis obtain s and bs::'a list
    where s: ⋀i. i ∈ Basis ⟹ x • i = s i • i ⋀i. i ∈ Basis ⟹ s i ∈ S
      and bs: set bs = Basis distinct bs
    by (metis finite_distinct_list)
  from nonempty_Basis s obtain j where j: j ∈ Basis s j ∈ S
    by blast
  define y where
    y = rec_list (s j) (λj _ Y. (∑ i∈Basis. (if i = j then s i • i else Y • i) *R i))
  have x = (∑ i∈Basis. (if i ∈ set bs then s i • i else s j • i) *R i)
    using bs by (auto simp: s(1)[symmetric] euclidean_representation)
  also have [symmetric]: y bs = ...
    using bs(2) bs(1)[THEN equalityD1]
   by (induct bs) (auto simp: y_def euclidean_representation intro!: euclidean_eqI[where
'a='a])
  also have y bs ∈ S
    using bs(1)[THEN equalityD1]
  proof (induction bs)
    case Nil
    then show ?case
      by (simp add: j y_def)
  next
    case (Cons a bs)
    then show ?case
      using interval_subset[OF assms(1)] s by (simp add: y_def)
  qed
  finally show ?thesis .
qed

lemma cbox01_nonempty [simp]: cbox 0 One ≠ {}
  by (simp add: box_ne_empty inner_Basis inner_sum_left sum_nonneg)

lemma box01_nonempty [simp]: box 0 One ≠ {}
  by (simp add: box_ne_empty inner_Basis inner_sum_left)

lemma empty_as_interval: {} = cbox One (0::'a::euclidean_space)
   using nonempty_Basis box01_nonempty box_eq_empty(1) box_ne_empty(1)
by blast

lemma interval_subset_is_interval:
  assumes is_interval S
  shows cbox a b ⊆ S ⟷ cbox a b = {} ∨ a ∈ S ∧ b ∈ S (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs  using box_ne_empty(1) mem_box(2) by fastforce
next
  assume ?rhs
  have cbox a b ⊆ S if a ∈ S b ∈ S
    using assms that
```

```
    by (force simp: mem_box intro: mem_is_intervalI)
  with ‹?rhs› show ?lhs
    by blast
qed

lemma is_real_interval_union:
  is_interval (X ∪ Y)
  if X: is_interval X and Y: is_interval Y and I: (X ≠ {} ⟹ Y ≠ {} ⟹ X
∩ Y ≠ {})
  for X Y::real set
proof −
  consider X ≠ {} Y ≠ {} | X = {} | Y = {} by blast
  then show ?thesis
  proof cases
    case 1
    then obtain r where r ∈ X ∨ X ∩ Y = {} r ∈ Y ∨ X ∩ Y = {}
      by blast
    then show ?thesis
      using I 1 X Y unfolding is_interval_1
      by (metis (full_types) Un_iff le_cases)
  qed (use that in auto)
qed

lemma is_interval_translationI:
  assumes is_interval X
  shows is_interval ((+) x ‘ X)
  unfolding is_interval_def
proof safe
  fix b d e
  assume b ∈ X d ∈ X
    ∀ i∈Basis. (x + b) · i ≤ e · i ∧ e · i ≤ (x + d) · i ∨
       (x + d) · i ≤ e · i ∧ e · i ≤ (x + b) · i
  hence e − x ∈ X
    by (intro mem_is_intervalI[OF assms ‹b ∈ X› ‹d ∈ X›, of e − x])
      (auto simp: algebra_simps)
  thus e ∈ (+) x ‘ X by force
qed

lemma is_interval_uminusI:
  assumes is_interval X
  shows is_interval (uminus ‘ X)
  unfolding is_interval_def
proof safe
  fix b d e
  assume b ∈ X d ∈ X
    ∀ i∈Basis. (− b) · i ≤ e · i ∧ e · i ≤ (− d) · i ∨
       (− d) · i ≤ e · i ∧ e · i ≤ (− b) · i
  hence − e ∈ X
    by (smt (verit, ccfv_threshold) assms inner_minus_left mem_is_intervalI)
```

thus $e \in \mathit{uminus}\ `\ X$ **by** *force*
qed

lemma *is_interval_uminus*[*simp*]: *is_interval* (*uminus* ` *x*) = *is_interval x*
using *is_interval_uminusI*[*of x*] *is_interval_uminusI*[*of uminus* ` *x*]
by (*auto simp*: *image_image*)

lemma *is_interval_neg_translationI*:
assumes *is_interval X*
shows *is_interval* ((−) *x* ` *X*)
proof −
have (−) *x* ` *X* = (+) *x* ` *uminus* ` *X*
by (*force simp*: *algebra_simps*)
also have *is_interval* ...
by (*metis is_interval_uminusI is_interval_translationI assms*)
finally show *?thesis* **.**
qed

lemma *is_interval_translation*[*simp*]:
is_interval ((+) *x* ` *X*) = *is_interval X*
using *is_interval_neg_translationI*[*of* (+) *x* ` *X x*]
by (*auto intro*!: *is_interval_translationI simp*: *image_image*)

lemma *is_interval_minus_translation*[*simp*]:
shows *is_interval* ((−) *x* ` *X*) = *is_interval X*
proof −
have (−) *x* ` *X* = (+) *x* ` *uminus* ` *X*
by (*force simp*: *algebra_simps*)
also have *is_interval* ... = *is_interval X*
by *simp*
finally show *?thesis* **.**
qed

lemma *is_interval_minus_translation'*[*simp*]:
shows *is_interval* ((λ*x*. *x* − *c*) ` *X*) = *is_interval X*
using *is_interval_translation*[*of* −*c X*]
by (*metis image_cong uminus_add_conv_diff*)

lemma *is_interval_cball_1*[*intro*, *simp*]: *is_interval* (*cball a b*) **for** *a b*::*real*
by (*simp add*: *cball_eq_atLeastAtMost is_interval_def*)

lemma *is_interval_ball_real*: *is_interval* (*ball a b*) **for** *a b*::*real*
by (*simp add*: *ball_eq_greaterThanLessThan is_interval_def*)

4.1.5 Bounded Projections

lemma *bounded_inner_imp_bdd_above*:
assumes *bounded s*
shows *bdd_above* ((λ*x*. *x* · *a*) ` *s*)

by (*simp add*: *assms bounded_imp_bdd_above bounded_linear_image bounded_linear_inner_left*)

lemma *bounded_inner_imp_bdd_below*:
assumes *bounded s*
shows *bdd_below* ((λ*x*. *x* ⋅ *a*) ‘ *s*)
by (*simp add*: *assms bounded_imp_bdd_below bounded_linear_image bounded_linear_inner_left*)

4.1.6 Structural rules for pointwise continuity

lemma *continuous_infnorm*[*continuous_intros*]:
continuous F f ⟹ *continuous F* (λ*x*. *infnorm* (*f x*))
unfolding *continuous_def* **by** (*rule tendsto_infnorm*)

lemma *continuous_inner*[*continuous_intros*]:
assumes *continuous F f*
and *continuous F g*
shows *continuous F* (λ*x*. *inner* (*f x*) (*g x*))
using *assms* **unfolding** *continuous_def* **by** (*rule tendsto_inner*)

4.1.7 Structural rules for setwise continuity

lemma *continuous_on_infnorm*[*continuous_intros*]:
continuous_on s f ⟹ *continuous_on s* (λ*x*. *infnorm* (*f x*))
unfolding *continuous_on* **by** (*fast intro*: *tendsto_infnorm*)

lemma *continuous_on_inner*[*continuous_intros*]:
fixes *g* :: *'a*::*topological_space* ⇒ *'b*::*real_inner*
assumes *continuous_on s f*
and *continuous_on s g*
shows *continuous_on s* (λ*x*. *inner* (*f x*) (*g x*))
using *bounded_bilinear_inner assms*
by (*rule bounded_bilinear.continuous_on*)

4.1.8 Openness of halfspaces.

lemma *open_halfspace_lt*: *open* {*x*. *inner a x* < *b*}
by (*simp add*: *open_Collect_less continuous_on_inner*)

lemma *open_halfspace_gt*: *open* {*x*. *inner a x* > *b*}
by (*simp add*: *open_Collect_less continuous_on_inner*)

lemma *open_halfspace_component_lt*: *open* {*x*::*'a*::*euclidean_space*. *x*•*i* < *a*}
by (*simp add*: *open_Collect_less continuous_on_inner*)

lemma *open_halfspace_component_gt*: *open* {*x*::*'a*::*euclidean_space*. *x*•*i* > *a*}
by (*simp add*: *open_Collect_less continuous_on_inner*)

lemma *eucl_less_eq_halfspaces*:
fixes *a* :: *'a*::*euclidean_space*
shows {*x*. *x* <*e* *a*} = (⋂ *i*∈*Basis*. {*x*. *x* ⋅ *i* < *a* ⋅ *i*})

$\{x.\ a <e\ x\} = (\bigcap i{\in}Basis.\ \{x.\ a \cdot i < x \cdot i\})$
by (*auto simp*: *eucl_less_def*)

lemma *open_Collect_eucl_less*[*simp*, *intro*]:
fixes $a :: 'a{::}euclidean\_space$
shows *open* $\{x.\ x <e\ a\}$ *open* $\{x.\ a <e\ x\}$
by (*auto simp*: *eucl_less_eq_halfspaces open_halfspace_component_lt open_halfspace_component_gt*)

4.1.9 Closure and Interior of halfspaces and hyperplanes

lemma *continuous_at_inner*: *continuous* (*at* x) (*inner* a)
unfolding *continuous_at* **by** (*intro tendsto_intros*)

lemma *closed_halfspace_le*: *closed* $\{x.\ inner\ a\ x \leq b\}$
by (*simp add*: *closed_Collect_le continuous_on_inner*)

lemma *closed_halfspace_ge*: *closed* $\{x.\ inner\ a\ x \geq b\}$
by (*simp add*: *closed_Collect_le continuous_on_inner*)

lemma *closed_hyperplane*: *closed* $\{x.\ inner\ a\ x = b\}$
by (*simp add*: *closed_Collect_eq continuous_on_inner*)

lemma *closed_halfspace_component_le*: *closed* $\{x{::}'a{::}euclidean\_space.\ x{\cdot}i \leq a\}$
by (*simp add*: *closed_Collect_le continuous_on_inner*)

lemma *closed_halfspace_component_ge*: *closed* $\{x{::}'a{::}euclidean\_space.\ x{\cdot}i \geq a\}$
by (*simp add*: *closed_Collect_le continuous_on_inner*)

lemma *closed_interval_left*:
fixes $b :: 'a{::}euclidean\_space$
shows *closed* $\{x{::}'a.\ \forall i{\in}Basis.\ x{\cdot}i \leq b{\cdot}i\}$
by (*simp add*: *Collect_ball_eq closed_INT closed_Collect_le continuous_on_inner*)

lemma *closed_interval_right*:
fixes $a :: 'a{::}euclidean\_space$
shows *closed* $\{x{::}'a.\ \forall i{\in}Basis.\ a{\cdot}i \leq x{\cdot}i\}$
by (*simp add*: *Collect_ball_eq closed_INT closed_Collect_le continuous_on_inner*)

lemma *interior_halfspace_le* [*simp*]:
assumes $a \neq 0$
shows *interior* $\{x.\ a \cdot x \leq b\} = \{x.\ a \cdot x < b\}$
proof −
have $*$: $a \cdot x < b$ **if** x: $x \in S$ **and** S: $S \subseteq \{x.\ a \cdot x \leq b\}$ **and** *open* S **for** $S\ x$
proof −
obtain e **where** $e{>}0$ **and** e: *cball* $x\ e \subseteq S$
using ‹*open* S› *open_contains_cball* x **by** *blast*
then have $x + (e\ /\ norm\ a) *_R\ a \in cball\ x\ e$
by (*simp add*: *dist_norm*)
then have $x + (e\ /\ norm\ a) *_R\ a \in S$

```
    using e by blast
  then have x + (e / norm a) *R a ∈ {x. a • x ≤ b}
    using S by blast
  moreover have e * (a • a) / norm a > 0
    by (simp add: ‹0 < e› assms)
  ultimately show ?thesis
    by (simp add: algebra_simps)
 qed
 show ?thesis
  by (rule interior_unique) (auto simp: open_halfspace_lt *)
qed

lemma interior_halfspace_ge [simp]:
  a ≠ 0 ⟹ interior {x. a • x ≥ b} = {x. a • x > b}
using interior_halfspace_le [of −a −b] by simp

lemma closure_halfspace_lt [simp]:
 assumes a ≠ 0
  shows closure {x. a • x < b} = {x. a • x ≤ b}
proof −
 have [simp]: −{x. a • x < b} = {x. a • x ≥ b}
  by force
 then show ?thesis
  using interior_halfspace_ge [of a b] assms
  by (force simp: closure_interior)
qed

lemma closure_halfspace_gt [simp]:
  a ≠ 0 ⟹ closure {x. a • x > b} = {x. a • x ≥ b}
using closure_halfspace_lt [of −a −b] by simp

lemma interior_hyperplane [simp]:
 assumes a ≠ 0
  shows interior {x. a • x = b} = {}
proof −
 have [simp]: {x. a • x = b} = {x. a • x ≤ b} ∩ {x. a • x ≥ b}
  by force
 then show ?thesis
  by (auto simp: assms)
qed

lemma frontier_halfspace_le:
 assumes a ≠ 0 ∨ b ≠ 0
  shows frontier {x. a • x ≤ b} = {x. a • x = b}
proof (cases a = 0)
 case True with assms show ?thesis by simp
next
 case False then show ?thesis
  by (force simp: frontier_def closed_halfspace_le)
```

qed

lemma *frontier_halfspace_ge*:
assumes $a \neq 0 \vee b \neq 0$
shows *frontier* $\{x.\ a \cdot x \geq b\} = \{x.\ a \cdot x = b\}$
proof (*cases* $a = 0$)
case *True* **with** *assms* **show** *?thesis* **by** *simp*
next
case *False* **then show** *?thesis*
by (*force simp*: *frontier_def closed_halfspace_ge*)
qed

lemma *frontier_halfspace_lt*:
assumes $a \neq 0 \vee b \neq 0$
shows *frontier* $\{x.\ a \cdot x < b\} = \{x.\ a \cdot x = b\}$
proof (*cases* $a = 0$)
case *True* **with** *assms* **show** *?thesis* **by** *simp*
next
case *False* **then show** *?thesis*
by (*force simp*: *frontier_def interior_open open_halfspace_lt*)
qed

lemma *frontier_halfspace_gt*:
assumes $a \neq 0 \vee b \neq 0$
shows *frontier* $\{x.\ a \cdot x > b\} = \{x.\ a \cdot x = b\}$
proof (*cases* $a = 0$)
case *True* **with** *assms* **show** *?thesis* **by** *simp*
next
case *False* **then show** *?thesis*
by (*force simp*: *frontier_def interior_open open_halfspace_gt*)
qed

4.1.10 Some more convenient intermediate-value theorem formulations

lemma *connected_ivt_hyperplane*:
assumes *connected* S **and** *xy*: $x \in S$ $y \in S$ **and** b: *inner* a $x \leq b$ $b \leq$ *inner* a y
shows $\exists z \in S.$ *inner* a $z = b$
proof (*rule ccontr*)
assume *as*:$\neg$ $(\exists z{\in}S.$ *inner* a $z = b)$
let *?A* $= \{x.$ *inner* a $x < b\}$
let *?B* $= \{x.$ *inner* a $x > b\}$
have *open ?A open ?B*
using *open_halfspace_lt* **and** *open_halfspace_gt* **by** *auto*
moreover have *?A* $\cap$ *?B* $= \{\}$ **by** *auto*
moreover have $S \subseteq$ *?A* $\cup$ *?B* **using** *as* **by** *auto*
ultimately show *False*
using ‹*connected S*› **unfolding** *connected_def*
by (*smt* (*verit*, *del_insts*) *as b disjoint_iff empty_iff mem_Collect_eq xy*)

qed

lemma *connected_ivt_component*:
fixes *x*::*'a*::*euclidean_space*
shows *connected S* $\Longrightarrow$ $x \in S \Longrightarrow y \in S \Longrightarrow x{\cdot}k \leq a \Longrightarrow a \leq y{\cdot}k \Longrightarrow (\exists z{\in}S.\ z{\cdot}k = a)$
using *connected_ivt_hyperplane*[*of S x y k*::*'a a*]
by (*auto simp*: *inner_commute*)

4.1.11 Limit Component Bounds

lemma *Lim_component_le*:
fixes *f* :: *'a* $\Rightarrow$ *'b*::*euclidean_space*
assumes (*f* $\longrightarrow$ *l*) *net*
and $\neg$ (*trivial_limit net*)
and *eventually* ($\lambda x.\ f(x){\cdot}i \leq b$) *net*
shows $l{\cdot}i \leq b$
by (*rule tendsto_le*[*OF assms*(*2*) *tendsto_const tendsto_inner*[*OF assms*(*1*) *tendsto_const*] *assms*(*3*)])

lemma *Lim_component_ge*:
fixes *f* :: *'a* $\Rightarrow$ *'b*::*euclidean_space*
assumes (*f* $\longrightarrow$ *l*) *net*
and $\neg$ (*trivial_limit net*)
and *eventually* ($\lambda x.\ b \leq (f\ x){\cdot}i$) *net*
shows $b \leq l{\cdot}i$
by (*rule tendsto_le*[*OF assms*(*2*) *tendsto_inner*[*OF assms*(*1*) *tendsto_const*] *tendsto_const assms*(*3*)])

lemma *Lim_component_eq*:
fixes *f* :: *'a* $\Rightarrow$ *'b*::*euclidean_space*
assumes *net*: (*f* $\longrightarrow$ *l*) *net* $\neg$ *trivial_limit net*
and *ev*:*eventually* ($\lambda x.\ f(x){\cdot}i = b$) *net*
shows $l{\cdot}i = b$
using *ev*[*unfolded order_eq_iff eventually_conj_iff*]
using *Lim_component_ge*[*OF net*, *of b i*]
using *Lim_component_le*[*OF net*, *of i b*]
by *auto*

lemma *open_box*[*intro*]: *open* (*box a b*)
proof −
have *open* ($\bigcap i{\in}Basis.\ ((\cdot)\ i) -` \{a \cdot i <..< b \cdot i\}$)
by (*auto intro*!: *continuous_open_vimage continuous_inner continuous_ident continuous_const*)
also have ($\bigcap i{\in}Basis.\ ((\cdot)\ i) -` \{a \cdot i <..< b \cdot i\}$) = *box a b*
by (*auto simp*: *box_def inner_commute*)
finally show *?thesis* **.**
qed

lemma *closed_cbox*[*intro*]:
fixes *a b* :: *'a*::*euclidean_space*
shows *closed* (*cbox a b*)
proof −
have *closed* (⋂ *i*∈*Basis*. (λ*x*. *x*•*i*) −‘ {*a*•*i* .. *b*•*i*})
by (*intro closed_INT ballI continuous_closed_vimage allI*
linear_continuous_at closed_real_atLeastAtMost finite_Basis bounded_linear_inner_left)
also have (⋂ *i*∈*Basis*. (λ*x*. *x*•*i*) −‘ {*a*•*i* .. *b*•*i*}) = *cbox a b*
by (*auto simp*: *cbox_def*)
finally show *closed* (*cbox a b*) **.**
qed

lemma *interior_cbox* [*simp*]:
fixes *a b* :: *'a*::*euclidean_space*
shows *interior* (*cbox a b*) = *box a b* (**is** *?L* = *?R*)
proof(*rule subset_antisym*)
show *?R* ⊆ *?L*
using *box_subset_cbox open_box*
by (*rule interior_maximal*)
{
fix *x*
assume *x* ∈ *interior* (*cbox a b*)
then obtain *s* **where** *s*: *open s x* ∈ *s s* ⊆ *cbox a b* **..**
then obtain *e* **where** *e*>*0* **and** *e*:∀ *x'*. *dist x' x* < *e* ⟶ *x'* ∈ *cbox a b*
unfolding *open_dist* **and** *subset_eq* **by** *auto*
{
fix *i* :: *'a*
assume *i*: *i* ∈ *Basis*
have *dist* (*x* − (*e* / *2*) ∗*R i*) *x* < *e*
and *dist* (*x* + (*e* / *2*) ∗*R i*) *x* < *e*
using *norm_Basis*[*OF i*] ‹*e*>*0*› **by** (*auto simp*: *dist_norm*)
then have *a* • *i* ≤ (*x* − (*e* / *2*) ∗*R i*) • *i* **and** (*x* + (*e* / *2*) ∗*R i*) • *i* ≤ *b* • *i*
using *e*[*THEN spec*[**where** *x*=*x* − (*e*/*2*) ∗*R i*]]
and *e*[*THEN spec*[**where** *x*=*x* + (*e*/*2*) ∗*R i*]]
unfolding *mem_box* **using** *i* **by** *blast*+
then have *a* • *i* < *x* • *i* **and** *x* • *i* < *b* • *i*
using ‹*e*>*0*› *i*
by (*auto simp*: *inner_diff_left inner_Basis inner_add_left*)
}
then have *x* ∈ *box a b*
unfolding *mem_box* **by** *auto*
}
then show *?L* ⊆ *?R* **..**
qed

lemma *bounded_cbox* [*simp*]:
fixes *a* :: *'a*::*euclidean_space*
shows *bounded* (*cbox a b*)
proof −

```
  let ?b = ∑ i∈Basis. |a•i| + |b•i|
  {
    fix x :: 'a
    assume ⋀i. i∈Basis ⟹ a • i ≤ x • i ∧ x • i ≤ b • i
    then have (∑ i∈Basis. |x • i|) ≤ ?b
      by (force simp: intro!: sum_mono)
    then have norm x ≤ ?b
      using norm_le_l1[of x] by auto
  }
  then show ?thesis
    unfolding cbox_def bounded_iff by force
qed

lemma bounded_box [simp]:
  fixes a :: 'a::euclidean_space
  shows bounded (box a b)
  by (metis bounded_cbox bounded_interior interior_cbox)

lemma not_interval_UNIV [simp]:
  fixes a :: 'a::euclidean_space
  shows cbox a b ≠ UNIV box a b ≠ UNIV
  using bounded_box[of a b] bounded_cbox[of a b] by force+

lemma not_interval_UNIV2 [simp]:
  fixes a :: 'a::euclidean_space
  shows UNIV ≠ cbox a b UNIV ≠ box a b
  using bounded_box[of a b] bounded_cbox[of a b] by force+

lemma box_midpoint:
  fixes a :: 'a::euclidean_space
  assumes box a b ≠ {}
  shows ((1/2) *R (a + b)) ∈ box a b
proof −
  have a • i < ((1 / 2) *R (a + b)) • i ∧ ((1 / 2) *R (a + b)) • i < b • i if i ∈
Basis for i
    using assms that by (auto simp: inner_add_left box_ne_empty)
  then show ?thesis unfolding mem_box by auto
qed

lemma open_cbox_convex:
  fixes x :: 'a::euclidean_space
  assumes x: x ∈ box a b
    and y: y ∈ cbox a b
    and e: 0 < e e ≤ 1
  shows (e *R x + (1 − e) *R y) ∈ box a b
proof −
  {
    fix i :: 'a
    assume i: i ∈ Basis
```

```
    have a · i = e * (a · i) + (1 − e) * (a · i)
      unfolding left_diff_distrib by simp
    also have ... < e * (x · i) + (1 − e) * (y · i)
      by (smt (verit, best) e i mem_box mult_le_cancel_left_pos mult_left_mono
x y)
    finally have a · i < (e *R x + (1 − e) *R y) · i
      unfolding inner_simps by auto
    moreover
    {
      have b · i = e * (b·i) + (1 − e) * (b·i)
        unfolding left_diff_distrib by simp
      also have ... > e * (x · i) + (1 − e) * (y · i)
       by (smt (verit, best) e i mem_box mult_le_cancel_left_pos mult_left_mono
x y)
      finally have (e *R x + (1 − e) *R y) · i < b · i
        unfolding inner_simps by auto
    }
    ultimately have a · i < (e *R x + (1 − e) *R y) · i ∧ (e *R x + (1 − e) *R
y) · i < b · i
      by auto
  }
  then show ?thesis
    unfolding mem_box by auto
qed

lemma closure_cbox [simp]: closure (cbox a b) = cbox a b
  by (simp add: closed_cbox)

lemma closure_box [simp]:
  fixes a :: 'a::euclidean_space
   assumes box a b ≠ {}
  shows closure (box a b) = cbox a b
proof −
  have ab: a <e b
    using assms by (simp add: eucl_less_def box_ne_empty)
  let ?c = (1 / 2) *R (a + b)
  {
    fix x
    assume as: x ∈ cbox a b
    define f where [abs_def]: f n = x + (inverse (real n + 1)) *R (?c − x) for n
    {
      fix n
      assume fn: f n <e b ⟶ a <e f n ⟶ f n = x and xc: x ≠ ?c
      have *: 0 < inverse (real n + 1) inverse (real n + 1) ≤ 1
        unfolding inverse_le_1_iff by auto
      have (inverse (real n + 1)) *R ((1 / 2) *R (a + b)) + (1 − inverse (real n
+ 1)) *R x =
        x + (inverse (real n + 1)) *R (((1 / 2) *R (a + b)) − x)
        by (auto simp: algebra_simps)
```

```
    then have f n <e b and a <e f n
      using open_cbox_convex[OF box_midpoint[OF assms] as *]
      unfolding f_def by (auto simp: box_def eucl_less_def)
    then have False
      using fn unfolding f_def using xc by auto
  }
  moreover
  {
    have ∃N::nat. ∀n≥N. inverse (real n + 1) < ε if ε > 0 for ε
      using reals_Archimedean [of ε] that
      by (metis inverse_inverse_eq inverse_less_imp_less nat_le_real_less
order_less_trans
          reals_Archimedean2)
    then have (λn. inverse (real n + 1)) ⟶ 0
      unfolding lim_sequentially by(auto simp: dist_norm)
    then have f ⟶ x
      unfolding f_def
      using tendsto_add[OF tendsto_const, of λn. (inverse (real n + 1)) *R ((1
/ 2) *R (a + b) - x) 0 sequentially x]
      using tendsto_scaleR [OF _ tendsto_const, of λn. inverse (real n + 1) 0
sequentially ((1 / 2) *R (a + b) - x)]
      by auto
  }
  ultimately have x ∈ closure (box a b)
    using as box_midpoint[OF assms]
    unfolding closure_def islimpt_sequential
    by (cases x=?c) (auto simp: in_box_eucl_less)
 }
 then show ?thesis
  using closure_minimal[OF box_subset_cbox, of a b] by blast
qed

lemma bounded_subset_box_symmetric:
 fixes S :: ('a::euclidean_space) set
 assumes bounded S
 obtains a where S ⊆ box (-a) a
proof -
 obtain b where b>0 and b: ∀x∈S. norm x ≤ b
  using assms[unfolded bounded_pos] by auto
 define a :: 'a where a = (∑ i∈Basis. (b + 1) *R i)
 have (-a)•i < x•i and x•i < a•i if x ∈ S and i: i ∈ Basis for x i
  using b Basis_le_norm[OF i, of x] that by (auto simp: a_def)
 then have S ⊆ box (-a) a
  by (auto simp: simp add: box_def)
 then show ?thesis ..
qed

lemma bounded_subset_cbox_symmetric:
 fixes S :: ('a::euclidean_space) set
```

assumes *bounded S*
obtains *a* **where** $S \subseteq$ *cbox* $(-a)$ *a*
by (*meson assms bounded_subset_box_symmetric box_subset_cbox order.trans*)

lemma *frontier_cbox*:
fixes *a b* :: *'a*::*euclidean_space*
shows *frontier* (*cbox a b*) = *cbox a b* − *box a b*
unfolding *frontier_def* **unfolding** *interior_cbox* **and** *closure_closed*[*OF closed_cbox*]
..

lemma *frontier_box*:
fixes *a b* :: *'a*::*euclidean_space*
shows *frontier* (*box a b*) = (*if box a b* = {} *then* {} *else cbox a b* − *box a b*)
by (*simp add*: *frontier_def interior_open open_box*)

lemma *Int_interval_mixed_eq_empty*:
fixes *a* :: *'a*::*euclidean_space*
assumes *box c d* ≠ {}
shows *box a b* ∩ *cbox c d* = {} ⟷ *box a b* ∩ *box c d* = {}
unfolding *closure_box*[*OF assms*, *symmetric*]
unfolding *open_Int_closure_eq_empty*[*OF open_box*] **..**

4.1.12 Class Instances

lemma *compact_lemma*:
fixes *f* :: *nat* ⇒ *'a*::*euclidean_space*
assumes *bounded* (*range f*)
shows ∀ *d*⊆*Basis*. ∃ *l*::*'a*. ∃ *r*.
strict_mono r ∧ (∀ *e*>*0*. *eventually* (λ*n*. ∀ *i*∈*d*. *dist* (*f* (*r n*) · *i*) (*l* · *i*) < *e*) *sequentially*)
by (*rule compact_lemma_general*[**where** *unproj*=λ*e*. $\sum$ *i*∈*Basis*. *e i* ∗$_R$ *i*])
(*auto intro*!: *assms bounded_linear_inner_left bounded_linear_image*
simp: *euclidean_representation*)

instance *euclidean_space* ⊆ *heine_borel*
proof
fix *f* :: *nat* ⇒ *'a*
assume *f*: *bounded* (*range f*)
then obtain *l*::*'a* **and** *r* **where** *r*: *strict_mono r*
and *l*: ∀ *e*>*0*. *eventually* (λ*n*. ∀ *i*∈*Basis*. *dist* (*f* (*r n*) · *i*) (*l* · *i*) < *e*) *sequentially*
using *compact_lemma* [*OF f*] **by** *blast*
{
fix *e*::*real*
assume *e* > *0*
hence *e* / *real_of_nat DIM*(*'a*) > *0* **by** (*simp*)
with *l* **have** *eventually* (λ*n*. ∀ *i*∈*Basis*. *dist* (*f* (*r n*) · *i*) (*l* · *i*) < *e* / (*real_of_nat DIM*(*'a*))) *sequentially*
by *simp*
moreover

```
    { fix n
      assume n: ∀ i∈Basis. dist (f (r n) · i) (l · i) < e / (real_of_nat DIM('a))
      have dist (f (r n)) l ≤ (∑ i∈Basis. dist (f (r n) · i) (l · i))
        using L2_set_le_sum [OF zero_le_dist] by (subst euclidean_dist_l2)
      also have ... < (∑ i∈(Basis::'a set). e / (real_of_nat DIM('a)))
        by (meson eucl.finite_Basis n nonempty_Basis sum_strict_mono)
      finally have dist (f (r n)) l < e
        by auto
    }
    ultimately have ∀F n in sequentially. dist (f (r n)) l < e
      by (rule eventually_mono)
  }
  then have *: (f ∘ r) ⟶ l
    unfolding o_def tendsto_iff by simp
  with r show ∃ l r. strict_mono r ∧ (f ∘ r) ⟶ l
    by auto
qed

instance euclidean_space ⊆ banach ..

instance euclidean_space ⊆ second_countable_topology
proof
  define a where a f = (∑ i∈Basis. fst (f i) *R i) for f :: 'a ⇒ real × real
  then have a: ⋀f. (∑ i∈Basis. fst (f i) *R i) = a f
    by simp
  define b where b f = (∑ i∈Basis. snd (f i) *R i) for f :: 'a ⇒ real × real
  then have b: ⋀f. (∑ i∈Basis. snd (f i) *R i) = b f
    by simp
  define B where B = (λf. box (a f) (b f)) ` (Basis →E (ℚ × ℚ))

  have Ball B open by (simp add: B_def open_box)
  moreover have (∀ A. open A ⟶ (∃ B'⊆B. ⋃ B' = A))
  proof safe
    fix A::'a set
    assume open A
    show ∃ B'⊆B. ⋃ B' = A
      using open_UNION_box[OF ‹open A›]
     by (smt (verit, ccfv_threshold) B_def a b image_iff mem_Collect_eq subsetI)
  qed
  ultimately
  have topological_basis B
    unfolding topological_basis_def by blast
  moreover
  have countable B
    unfolding B_def
   by (intro countable_image countable_PiE finite_Basis countable_SIGMA count-
able_rat)
  ultimately show ∃ B::'a set set. countable B ∧ open = generate_topology B
    by (blast intro: topological_basis_imp_subbasis)
```

qed

instance *euclidean_space* ⊆ *polish_space* **..**

4.1.13 Compact Boxes

lemma *compact_cbox* [*simp*]:
fixes *a* :: *'a*::*euclidean_space*
shows *compact* (*cbox a b*)
using *bounded_closed_imp_seq_compact*[*of cbox a b*] **using** *bounded_cbox*[*of a b*]
by (*auto simp*: *compact_eq_seq_compact_metric*)

proposition *is_interval_compact*:
is_interval S ∧ *compact S* ⟷ (∃ *a b*. *S* = *cbox a b*) (**is** *?lhs* = *?rhs*)
proof (*cases S* = {})
case *True*
with *empty_as_interval* **show** *?thesis* **by** *auto*
next
case *False*
show *?thesis*
proof
assume *L*: *?lhs*
then have *is_interval S compact S* **by** *auto*
define *a* **where** *a* ≡ ∑ *i*∈*Basis*. (*INF x*∈*S*. *x* ⋅ *i*) *∗*$_R$ *i*
define *b* **where** *b* ≡ ∑ *i*∈*Basis*. (*SUP x*∈*S*. *x* ⋅ *i*) *∗*$_R$ *i*
have *1*: ⋀*x i*. ⟦*x* ∈ *S*; *i* ∈ *Basis*⟧ ⟹ (*INF x*∈*S*. *x* ⋅ *i*) ≤ *x* ⋅ *i*
by (*simp add*: *cInf_lower bounded_inner_imp_bdd_below compact_imp_bounded L*)
have *2*: ⋀*x i*. ⟦*x* ∈ *S*; *i* ∈ *Basis*⟧ ⟹ *x* ⋅ *i* ≤ (*SUP x*∈*S*. *x* ⋅ *i*)
by (*simp add*: *cSup_upper bounded_inner_imp_bdd_above compact_imp_bounded L*)
have *3*: *x* ∈ *S* **if** *inf*: ⋀*i*. *i* ∈ *Basis* ⟹ (*INF x*∈*S*. *x* ⋅ *i*) ≤ *x* ⋅ *i*
and *sup*: ⋀*i*. *i* ∈ *Basis* ⟹ *x* ⋅ *i* ≤ (*SUP x*∈*S*. *x* ⋅ *i*) **for** *x*
proof (*rule mem_box_componentwiseI* [*OF* ‹*is_interval S*›])
fix *i*::*'a*
assume *i*: *i* ∈ *Basis*
have *cont*: *continuous_on S* (λ*x*. *x* ⋅ *i*)
by (*intro continuous_intros*)
obtain *a* **where** *a* ∈ *S* **and** *a*: ⋀*y*. *y*∈*S* ⟹ *a* ⋅ *i* ≤ *y* ⋅ *i*
using *continuous_attains_inf* [*OF* ‹*compact S*› *False cont*] **by** *blast*
obtain *b* **where** *b* ∈ *S* **and** *b*: ⋀*y*. *y*∈*S* ⟹ *y* ⋅ *i* ≤ *b* ⋅ *i*
using *continuous_attains_sup* [*OF* ‹*compact S*› *False cont*] **by** *blast*
have *a* ⋅ *i* ≤ (*INF x*∈*S*. *x* ⋅ *i*)
by (*simp add*: *False a cINF_greatest*)
also have ... ≤ *x* ⋅ *i*
by (*simp add*: *i inf*)
finally have *ai*: *a* ⋅ *i* ≤ *x* ⋅ *i* **.**
have *x* ⋅ *i* ≤ (*SUP x*∈*S*. *x* ⋅ *i*)

```
      by (simp add: i sup)
    also have (SUP x∈S. x · i) ≤ b · i
      by (simp add: False b cSUP_least)
    finally have bi: x · i ≤ b · i .
    show x · i ∈ (λx. x · i) ‘ S
      apply (rule_tac x=∑j∈Basis. (((·)a)(i := x · j))j *R j in image_eqI)
      apply (simp add: i)
      apply (rule mem_is_intervalI [OF ‹is_interval S› ‹a ∈ S› ‹b ∈ S›])
      using i ai bi
      apply force
      done
  qed
  have S = cbox a b
    by (auto simp: a_def b_def mem_box intro: 1 2 3)
  then show ?rhs
    by blast
next
  assume R: ?rhs
  then show ?lhs
    using compact_cbox is_interval_cbox by blast
qed
qed
```

4.1.14 Componentwise limits and continuity

But is the premise really necessary? Need to generalise *dist ?x ?y = L2_set* (*λi. dist* (*?x · i*) (*?y · i*)) *Basis*

```
lemma Euclidean_dist_upper: i ∈ Basis ⟹ dist (x · i) (y · i) ≤ dist x y
  by (metis (no_types) member_le_L2_set euclidean_dist_l2 finite_Basis)
```

But is the premise *i ∈ Basis* really necessary?

```
lemma open_preimage_inner:
  assumes open S i ∈ Basis
    shows open {x. x · i ∈ S}
proof (rule openI, simp)
  fix x
  assume x: x · i ∈ S
  with assms obtain e where 0 < e and e: ball (x · i) e ⊆ S
    by (auto simp: open_contains_ball_eq)
  have ∃e>0. ball (y · i) e ⊆ S if dxy: dist x y < e / 2 for y
  proof (intro exI conjI)
    have dist (x · i) (y · i) < e / 2
      by (meson ‹i ∈ Basis› dual_order.trans Euclidean_dist_upper not_le that)
    then have dist (x · i) z < e if dist (y · i) z < e / 2 for z
      by (metis dist_commute dist_triangle_half_l that)
    then have ball (y · i) (e / 2) ⊆ ball (x · i) e
      using mem_ball by blast
      with e show ball (y · i) (e / 2) ⊆ S
```

```
      by (metis order_trans)
  qed (simp add: ‹0 < e›)
  then show ∃ e>0. ball x e ⊆ {s. s · i ∈ S}
   by (metis (no_types, lifting) ‹0 < e› ‹open S› half_gt_zero_iff mem_Collect_eq
mem_ball open_contains_ball_eq subsetI)
qed

proposition tendsto_componentwise_iff:
  fixes f :: _ ⇒ 'b::euclidean_space
  shows (f ⟶ l) F ⟷ (∀ i ∈ Basis. ((λx. (f x · i)) ⟶ (l · i)) F)
        (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    unfolding tendsto_def
    by (smt (verit) eventually_elim2 mem_Collect_eq open_preimage_inner)
next
  assume R: ?rhs
  then have ⋀e. e > 0 ⟹ ∀ i∈Basis. ∀ F x in F. dist (f x · i) (l · i) < e
    unfolding tendsto_iff by blast
  then have R': ⋀e. e > 0 ⟹ ∀ F x in F. ∀ i∈Basis. dist (f x · i) (l · i) < e
     by (simp add: eventually_ball_finite_distrib [symmetric])
  show ?lhs
  unfolding tendsto_iff
  proof clarify
    fix e::real
    assume 0 < e
    have ∗: L2_set (λi. dist (f x · i) (l · i)) Basis < e
           if ∀ i∈Basis. dist (f x · i) (l · i) < e / real DIM('b) for x
    proof −
      have L2_set (λi. dist (f x · i) (l · i)) Basis ≤ sum (λi. dist (f x · i) (l · i))
Basis
        by (simp add: L2_set_le_sum)
      also have ... < DIM('b) ∗ (e / real DIM('b))
        by (meson DIM_positive sum_bounded_above_strict that)
      also have ... = e
        by (simp add: field_simps)
      finally show L2_set (λi. dist (f x · i) (l · i)) Basis < e .
    qed
    have ∀ F x in F. ∀ i∈Basis. dist (f x · i) (l · i) < e / DIM('b)
      by (simp add: R' ‹0 < e›)
    then show ∀ F x in F. dist (f x) l < e
      by eventually_elim (metis (full_types) ∗ euclidean_dist_l2)
  qed
qed


corollary continuous_componentwise:
   continuous F f ⟷ (∀ i ∈ Basis. continuous F (λx. (f x · i)))
```

by (*simp add*: *continuous_def tendsto_componentwise_iff* [*symmetric*])

corollary *continuous_on_componentwise*:
fixes $S :: 'a :: t2\_space\ set$
shows *continuous_on S f* $\longleftrightarrow$ $(\forall i \in Basis.$ *continuous_on S* $(\lambda x.\ (f\ x \cdot i)))$
by (*metis continuous_componentwise continuous_on_eq_continuous_within*)

lemma *linear_componentwise_iff*:
linear f' $\longleftrightarrow$ $(\forall i{\in}Basis.$ *linear* $(\lambda x.\ f'\ x \cdot i))$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
show *?lhs* $\Longrightarrow$ *?rhs*
by (*simp add*: *Real_Vector_Spaces.linear_iff inner_left_distrib*)
show *?rhs* $\Longrightarrow$ *?lhs*
by (*simp add*: *linear_iff*) (*metis euclidean_eqI inner_left_distrib inner_scaleR_left*)
qed

lemma *bounded_linear_componentwise_iff*:
(*bounded_linear f'*) $\longleftrightarrow$ $(\forall i{\in}Basis.$ *bounded_linear* $(\lambda x.\ f'\ x \cdot i))$
(**is** *?lhs* = *?rhs*)
proof
assume *?rhs*
then have $(\forall i{\in}Basis.\ \exists K.\ \forall x.\ |f'\ x \cdot i| \le norm\ x * K)$ *linear f'*
by (*auto simp*: *bounded_linear_def bounded_linear_axioms_def linear_componentwise_iff* [*symmetric*] *ball_conj_distrib*)
then obtain *F* **where** *F*: $\bigwedge i\ x.\ i \in Basis \Longrightarrow |f'\ x \cdot i| \le norm\ x * F\ i$
by *metis*
have *norm* $(f'\ x) \le norm\ x * sum\ F\ Basis$ **for** *x*
proof −
have *norm* $(f'\ x) \le (\sum i{\in}Basis.\ |f'\ x \cdot i|)$
by (*rule norm_le_l1*)
also have ... $\le (\sum i{\in}Basis.\ norm\ x * F\ i)$
by (*metis F sum_mono*)
also have ... = *norm x* $*$ *sum F Basis*
by (*simp add*: *sum_distrib_left*)
finally show *?thesis* **.**
qed
then show *?lhs*
by (*force simp*: *bounded_linear_def bounded_linear_axioms_def* ‹*linear f'*›)
qed (*simp add*: *bounded_linear_inner_left_comp*)

4.1.15 Continuous Extension

definition *clamp* :: $'a{::}euclidean\_space \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$ **where**
clamp a b x = (*if* $(\forall i{\in}Basis.\ a \cdot i \le b \cdot i)$
then $(\sum i{\in}Basis.$ (*if* $x{\cdot}i < a{\cdot}i$ *then* $a{\cdot}i$ *else if* $x{\cdot}i \le b{\cdot}i$ *then* $x{\cdot}i$ *else* $b{\cdot}i) *_R i)$
else a)

lemma *clamp_in_interval*[*simp*]:
assumes $\bigwedge i.\ i \in Basis \Longrightarrow a \cdot i \le b \cdot i$

```
  shows clamp a b x ∈ cbox a b
  unfolding clamp_def
  using box_ne_empty(1)[of a b] assms by (auto simp: cbox_def)

lemma clamp_cancel_cbox[simp]:
  fixes x a b :: 'a::euclidean_space
  assumes x: x ∈ cbox a b
  shows clamp a b x = x
  using assms
  by (auto simp: clamp_def mem_box intro!: euclidean_eqI[where 'a='a])

lemma clamp_empty_interval:
  assumes i ∈ Basis a · i > b · i
  shows clamp a b = (λ_. a)
  using assms
  by (force simp: clamp_def[abs_def] split: if_splits intro!: ext)

lemma dist_clamps_le_dist_args:
  fixes x :: 'a::euclidean_space
  shows dist (clamp a b y) (clamp a b x) ≤ dist y x
proof cases
  assume le: (∀ i∈Basis. a · i ≤ b · i)
  then have (∑ i∈Basis. (dist (clamp a b y · i) (clamp a b x · i))^2) ≤
    (∑ i∈Basis. (dist (y · i) (x · i))^2)
   by (auto intro!: sum_mono simp: clamp_def dist_real_def abs_le_square_iff[symmetric])
  then show ?thesis
    by (auto intro: real_sqrt_le_mono
      simp: euclidean_dist_l2[where y=x] euclidean_dist_l2[where y=clamp a b
x] L2_set_def)
qed (auto simp: clamp_def)

lemma clamp_continuous_at:
  fixes f :: 'a::euclidean_space ⇒ 'b::metric_space
    and x :: 'a
  assumes f_cont: continuous_on (cbox a b) f
  shows continuous (at x) (λx. f (clamp a b x))
proof cases
  assume le: (∀ i∈Basis. a · i ≤ b · i)
  show ?thesis
    unfolding continuous_at_eps_delta
  proof safe
    fix x :: 'a
    fix e :: real
    assume e > 0
    moreover have clamp a b x ∈ cbox a b
      by (simp add: le)
    moreover note f_cont[simplified continuous_on_iff]
    ultimately
    obtain d where d: 0 < d
```

⋀x'. $x' \in cbox\ a\ b \Longrightarrow dist\ x'\ (clamp\ a\ b\ x) < d \Longrightarrow dist\ (f\ x')\ (f\ (clamp\ a\ b\ x)) < e$
by *force*
show $\exists\, d{>}0.\ \forall\, x'.\ dist\ x'\ x < d \longrightarrow dist\ (f\ (clamp\ a\ b\ x'))\ (f\ (clamp\ a\ b\ x)) < e$
using *le*
by (*auto intro*!: *d clamp_in_interval dist_clamps_le_dist_args*[*THEN le_less_trans*])
qed
qed (*auto simp*: *clamp_empty_interval*)

lemma *clamp_continuous_on*:
fixes f :: $'a$::*euclidean_space* ⇒ $'b$::*metric_space*
assumes *f_cont*: *continuous_on* (*cbox a b*) *f*
shows *continuous_on S* (λ*x*. *f* (*clamp a b x*))
using *assms*
by (*auto intro*: *continuous_at_imp_continuous_on clamp_continuous_at*)

lemma *clamp_bounded*:
fixes f :: $'a$::*euclidean_space* ⇒ $'b$::*metric_space*
assumes *bounded*: *bounded* (*f* ‘ (*cbox a b*))
shows *bounded* (*range* (λ*x*. *f* (*clamp a b x*)))
proof *cases*
assume *le*: (∀ *i*∈*Basis*. *a* · *i* ≤ *b* · *i*)
from *bounded* **obtain** *c* **where** *f_bound*: ∀ *x*∈*f* ‘ *cbox a b*. *dist undefined x* ≤ *c*
by (*auto simp*: *bounded_any_center*[**where** *a*=*undefined*])
then show *?thesis*
by (*metis bounded bounded_subset clamp_in_interval image_mono image_subsetI le range_composition*)
qed (*auto simp*: *clamp_empty_interval image_def*)

definition *ext_cont* :: (*'a*::*euclidean_space* ⇒ *'b*::*metric_space*) ⇒ *'a* ⇒ *'a* ⇒ *'a* ⇒ *'b*
where *ext_cont f a b* = (λ*x*. *f* (*clamp a b x*))

lemma *ext_cont_cancel_cbox*[*simp*]:
fixes *x a b* :: $'a$::*euclidean_space*
assumes *x*: *x* ∈ *cbox a b*
shows *ext_cont f a b x* = *f x*
using *assms* **by** (*simp add*: *ext_cont_def*)

lemma *continuous_on_ext_cont*[*continuous_intros*]:
continuous_on (*cbox a b*) *f* ⟹ *continuous_on S* (*ext_cont f a b*)
by (*auto intro*!: *clamp_continuous_on simp*: *ext_cont_def*)

4.1.16 Separability

lemma *univ_second_countable_sequence*:
obtains *B* :: *nat* ⇒ $'a$::*euclidean_space set*

```
    where inj B ⋀n. open(B n) ⋀S. open S ⟹ ∃k. S = ⋃{B n |n. n ∈ k}
proof −
  obtain ℬ :: 'a set set
  where countable ℬ
    and opn: ⋀C. C ∈ ℬ ⟹ open C
    and Un: ⋀S. open S ⟹ ∃U. U ⊆ ℬ ∧ S = ⋃U
    using univ_second_countable by blast
  have ∗: infinite (range (λn. ball (0::'a) (inverse(Suc n))))
    by (simp add: inj_on_def ball_eq_ball_iff Infinite_Set.range_inj_infinite)
  have infinite ℬ
  proof
    assume finite ℬ
    then have finite (Union ‘ (Pow ℬ))
      by simp
    moreover have range (λn. ball 0 (inverse (real (Suc n)))) ⊆ ⋃ ‘ Pow ℬ
      by (metis (no_types, lifting) PowI image_eqI image_subset_iff Un [OF
open_ball])
    ultimately show False
      by (metis finite_subset ∗)
  qed
  obtain f :: nat ⇒ 'a set where ℬ = range f inj f
    by (blast intro: countable_as_injective_image [OF ‹countable ℬ› ‹infinite ℬ›])
  have ∗: ∃k. S = ⋃{f n |n. n ∈ k} if open S for S
    using Un [OF that]
    apply clarify
    apply (rule_tac x=f−‘U in exI)
    using ‹inj f› ‹ℬ = range f› apply force
    done
  show ?thesis
    using ∗ ‹ℬ = range f› ‹inj f› opn that by force
qed

proposition separable:
  fixes S :: 'a::{metric_space, second_countable_topology} set
  obtains T where countable T T ⊆ S S ⊆ closure T
proof −
  obtain ℬ :: 'a set set
    where countable ℬ
      and {} ∉ ℬ
      and ope: ⋀C. C ∈ ℬ ⟹ openin(top_of_set S) C
      and if_ope: ⋀T. openin(top_of_set S) T ⟹ ∃𝒰. 𝒰 ⊆ ℬ ∧ T = ⋃𝒰
    by (meson subset_second_countable)
  then obtain f where f: ⋀C. C ∈ ℬ ⟹ f C ∈ C
    by (metis equals0I)
  show ?thesis
  proof
    show countable (f ‘ ℬ)
      by (simp add: ‹countable ℬ›)
    show f ‘ ℬ ⊆ S
```

```
    using ope f openin_imp_subset by blast
  show S ⊆ closure (f ` ℬ)
  proof (clarsimp simp: closure_approachable)
    fix x and e::real
    assume x ∈ S 0 < e
    have openin (top_of_set S) (S ∩ ball x e)
      by (simp add: openin_Int_open)
    with if_ope obtain 𝒰 where 𝒰: 𝒰 ⊆ ℬ S ∩ ball x e = ⋃𝒰
      by meson
    show ∃ C ∈ ℬ. dist (f C) x < e
    proof (cases 𝒰 = {})
      case True
      then show ?thesis
        using ‹0 < e› 𝒰 ‹x ∈ S› by auto
    next
      case False
      then show ?thesis
        by (metis IntI Union_iff 𝒰 ‹0 < e› ‹x ∈ S› dist_commute dist_self f
inf_le2 mem_ball subset_eq)
    qed
  qed
qed
```

4.1.17 Diameter

```
lemma diameter_cball [simp]:
  fixes a :: 'a::euclidean_space
  shows diameter(cball a r) = (if r < 0 then 0 else 2*r)
proof −
  have diameter(cball a r) = 2*r if r ≥ 0
  proof (rule order_antisym)
    show diameter (cball a r) ≤ 2*r
    proof (rule diameter_le)
      fix x y assume x ∈ cball a r y ∈ cball a r
      then have norm (x − a) ≤ r norm (a − y) ≤ r
        by (auto simp: dist_norm norm_minus_commute)
      then have norm (x − y) ≤ r+r
        using norm_diff_triangle_le by blast
      then show norm (x − y) ≤ 2*r by simp
    qed (simp add: that)
    have 2*r = dist (a + r *R (SOME i. i ∈ Basis)) (a − r *R (SOME i. i ∈
Basis))
      using ‹0 ≤ r› that by (simp add: dist_norm flip: scaleR_2)
    also have ... ≤ diameter (cball a r)
      apply (rule diameter_bounded_bound)
      using that by (auto simp: dist_norm)
    finally show 2*r ≤ diameter (cball a r) .
  qed
```

then show *?thesis* **by** *simp*
qed

lemma *diameter_ball* [*simp*]:
fixes *a* :: *'a*::*euclidean_space*
shows *diameter*(*ball a r*) = (*if r* < *0 then 0 else 2*∗*r*)
proof −
have *diameter*(*ball a r*) = *2*∗*r* **if** *r* > *0*
by (*metis bounded_ball diameter_closure closure_ball diameter_cball less_eq_real_def linorder_not_less that*)
then show *?thesis*
by (*simp add*: *diameter_def*)
qed

lemma *diameter_closed_interval* [*simp*]: *diameter* {*a*..*b*} = (*if b* < *a then 0 else b*−*a*)
proof −
have {*a*..*b*} = *cball* ((*a*+*b*)/*2*) ((*b*−*a*)/*2*)
using *atLeastAtMost_eq_cball* **by** *blast*
then show *?thesis*
by *simp*
qed

lemma *diameter_open_interval* [*simp*]: *diameter* {*a*<..<*b*} = (*if b* < *a then 0 else b*−*a*)
proof −
have {*a* <..< *b*} = *ball* ((*a*+*b*)/*2*) ((*b*−*a*)/*2*)
using *greaterThanLessThan_eq_ball* **by** *blast*
then show *?thesis*
by *simp*
qed

lemma *diameter_cbox*:
fixes *a b*::*'a*::*euclidean_space*
shows (∀ *i* ∈ *Basis*. *a* · *i* ≤ *b* · *i*) ⟹ *diameter* (*cbox a b*) = *dist a b*
by (*force simp*: *diameter_def intro*!: *cSup_eq_maximum L2_set_mono simp*: *euclidean_dist_l2*[**where** *'a*=*'a*] *cbox_def dist_norm*)

4.1.18 Relating linear images to open/closed/interior/closure/connected

proposition *open_surjective_linear_image*:
fixes *f* :: *'a*::*real_normed_vector* ⇒ *'b*::*euclidean_space*
assumes *open A linear f surj f*
shows *open*(*f* ‘ *A*)
unfolding *open_dist*
proof *clarify*
fix *x*
assume *x* ∈ *A*

```
  have bounded (inv f ‘ Basis)
    by (simp add: finite_imp_bounded)
  with bounded_pos obtain B where B > 0 and B: ⋀x. x ∈ inv f ‘ Basis ⟹
norm x ≤ B
    by metis
  obtain e where e > 0 and e: ⋀z. dist z x < e ⟹ z ∈ A
    by (metis open_dist ‹x ∈ A› ‹open A›)
  define δ where δ ≡ e / B / DIM('b)
  show ∃e>0. ∀y. dist y (f x) < e ⟶ y ∈ f ‘ A
  proof (intro exI conjI)
    show δ > 0
      using ‹e > 0› ‹B > 0›  by (simp add: δ_def field_split_simps)
    have y ∈ f ‘ A if dist y (f x) * (B * real DIM('b)) < e for y
    proof −
      define u where u ≡ y − f x
      show ?thesis
      proof (rule image_eqI)
        show y = f (x + (∑i∈Basis. (u • i) *R inv f i))
       apply (simp add: linear_add linear_sum linear.scaleR ‹linear f› surj_f_inv_f
‹surj f›)
          apply (simp add: euclidean_representation u_def)
          done
        have dist (x + (∑i∈Basis. (u • i) *R inv f i)) x ≤ (∑i∈Basis. norm ((u
• i) *R inv f i))
          by (simp add: dist_norm sum_norm_le)
        also have ... = (∑i∈Basis. |u • i| * norm (inv f i))
          by simp
        also have ... ≤ (∑i∈Basis. |u • i|) * B
          by (simp add: B sum_distrib_right sum_mono mult_left_mono)
        also have ... ≤ DIM('b) * dist y (f x) * B
          apply (rule mult_right_mono [OF sum_bounded_above])
          using ‹0 < B› by (auto simp: Basis_le_norm dist_norm u_def)
        also have ... < e
          by (metis mult.commute mult.left_commute that)
        finally show x + (∑i∈Basis. (u • i) *R inv f i) ∈ A
          by (rule e)
      qed
    qed
    then show ∀y. dist y (f x) < δ ⟶ y ∈ f ‘ A
      using ‹e > 0› ‹B > 0›
      by (auto simp: δ_def field_split_simps)
  qed
qed

corollary open_bijective_linear_image_eq:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes linear f bij f
    shows open(f ‘ A) ⟷ open A
proof
```

assume *open*(*f* ‘ *A*)
then show *open A*
by (*metis assms bij_is_inj continuous_open_vimage inj_vimage_image_eq linear_continuous_at linear_linear*)
next
assume *open A*
then show *open*(*f* ‘ *A*)
by (*simp add*: *assms bij_is_surj open_surjective_linear_image*)
qed

corollary *interior_bijective_linear_image*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *linear f bij f*
shows *interior* (*f* ‘ *S*) = *f* ‘ *interior S*
by (*smt* (*verit*) *assms bij_is_inj inj_image_subset_iff interior_maximal interior_subset*
open_bijective_linear_image_eq open_interior subset_antisym subset_imageE)

lemma *interior_injective_linear_image*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'a*::*euclidean_space*
assumes *linear f inj f*
shows *interior*(*f* ‘ *S*) = *f* ‘ (*interior S*)
by (*simp add*: *linear_injective_imp_surjective assms bijI interior_bijective_linear_image*)

lemma *interior_surjective_linear_image*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'a*::*euclidean_space*
assumes *linear f surj f*
shows *interior*(*f* ‘ *S*) = *f* ‘ (*interior S*)
by (*simp add*: *assms interior_injective_linear_image linear_surjective_imp_injective*)

lemma *interior_negations*:
fixes *S* :: *'a*::*euclidean_space set*
shows *interior*(*uminus* ‘ *S*) = *image uminus* (*interior S*)
by (*simp add*: *bij_uminus interior_bijective_linear_image linear_uminus*)

lemma *connected_linear_image*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*real_normed_vector*
assumes *linear f* **and** *connected s*
shows *connected* (*f* ‘ *s*)
using *connected_continuous_image assms linear_continuous_on linear_conv_bounded_linear*
by *blast*

4.1.19 "Isometry" (up to constant bounds) of Injective Linear Map

proposition *injective_imp_isometric*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *s*: *closed s subspace s*
and *f*: *bounded_linear f* ∀ *x*∈*s*. *f x* = *0* ⟶ *x* = *0*

```
  shows ∃ e>0. ∀ x∈s. norm (f x) ≥ e * norm x
proof (cases s ⊆ {0::'a})
  case True
  have norm x ≤ norm (f x) if x ∈ s for x
  proof −
    from True that have x = 0 by auto
    then show ?thesis by simp
  qed
  then show ?thesis
    by (auto intro!: exI[where x=1])
next
  case False
  interpret f: bounded_linear f by fact
  from False obtain a where a: a ≠ 0 a ∈ s
    by auto
  from False have s ≠ {}
    by auto
  let ?S = {f x| x. x ∈ s ∧ norm x = norm a}
  let ?S' = {x::'a. x∈s ∧ norm x = norm a}
  let ?S'' = {x::'a. norm x = norm a}

  have ?S'' = frontier (cball 0 (norm a))
    by (simp add: sphere_def dist_norm)
  then have compact ?S'' by (metis compact_cball compact_frontier)
  moreover have ?S' = s ∩ ?S'' by auto
  ultimately have compact ?S'
    using closed_Int_compact[of s ?S''] using s(1) by auto
  moreover have *:f ` ?S' = ?S by auto
  ultimately have compact ?S
    using compact_continuous_image[OF linear_continuous_on[OF f(1)], of ?S']
by auto
  then have closed ?S
    using compact_imp_closed by auto
  moreover from a have ?S ≠ {} by auto
  ultimately obtain b' where b'∈?S ∀ y∈?S. norm b' ≤ norm y
    using distance_attains_inf[of ?S 0] unfolding dist_0_norm by auto
  then obtain b where b∈s
    and ba: norm b = norm a
    and b: ∀ x∈{x ∈ s. norm x = norm a}. norm (f b) ≤ norm (f x)
    unfolding *[symmetric] unfolding image_iff by auto

  let ?e = norm (f b) / norm b
  have norm b > 0
    using ba and a and norm_ge_zero by auto
  moreover have norm (f b) > 0
    using f(2)[THEN bspec[where x=b], OF ‹b∈s›]
    using ‹norm b >0› by simp
  ultimately have 0 < norm (f b) / norm b by simp
  moreover
```

```
  have norm (f b) / norm b * norm x ≤ norm (f x) if x∈s for x
  proof (cases x = 0)
    case True
    then show norm (f b) / norm b * norm x ≤ norm (f x)
      by auto
  next
    case False
    with ‹a ≠ 0› have *: 0 < norm a / norm x
      unfolding zero_less_norm_iff[symmetric] by simp
    have ∀ x∈s. c *R x ∈ s for c
      using s[unfolded subspace_def] by simp
   with ‹x ∈ s› ‹x ≠ 0› have (norm a / norm x) *R x ∈ {x ∈ s. norm x = norm
a}
      by simp
    with ‹x ≠ 0› ‹a ≠ 0› show norm (f b) / norm b * norm x ≤ norm (f x)
      using b[THEN bspec[where x=(norm a / norm x) *R x]]
      unfolding f.scaleR and ba
      by (auto simp: mult.commute pos_le_divide_eq pos_divide_le_eq)
  qed
  ultimately show ?thesis by auto
qed

proposition closed_injective_image_subspace:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes subspace s bounded_linear f ∀ x∈s. f x = 0 ⟶ x = 0 closed s
  shows closed(f ‘ s)
proof −
  obtain e where e > 0 and e: ∀ x∈s. e * norm x ≤ norm (f x)
    using assms injective_imp_isometric by blast
  with assms show ?thesis
    by (meson complete_eq_closed complete_isometric_image)
qed


lemma closure_bounded_linear_image_subset:
  assumes f: bounded_linear f
  shows f ‘ closure S ⊆ closure (f ‘ S)
  using linear_continuous_on [OF f] closed_closure closure_subset
  by (rule image_closure_subset)

lemma closure_linear_image_subset:
  fixes f :: 'm::euclidean_space ⇒ 'n::real_normed_vector
  assumes linear f
  shows f ‘ (closure S) ⊆ closure (f ‘ S)
  using assms unfolding linear_conv_bounded_linear
  by (rule closure_bounded_linear_image_subset)

lemma closed_injective_linear_image:
    fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
```

```
    assumes S: closed S and f: linear f inj f
    shows closed (f ‘ S)
proof −
  obtain g where g: linear g g ∘ f = id
    using linear_injective_left_inverse [OF f] by blast
  then have confg: continuous_on (range f) g
    using linear_continuous_on linear_conv_bounded_linear by blast
  have [simp]: g ‘ f ‘ S = S
    using g by (simp add: image_comp)
  have cgf: closed (g ‘ f ‘ S)
    by (simp add: ‹g ∘ f = id› S image_comp)
  have [simp]: (range f ∩ g −‘ S) = f ‘ S
    using g unfolding o_def id_def image_def by auto metis+
  show ?thesis
  proof (rule closedin_closed_trans [of range f])
    show closedin (top_of_set (range f)) (f ‘ S)
      using continuous_closedin_preimage [OF confg cgf] by simp
    show closed (range f)
      using closed_injective_image_subspace f linear_conv_bounded_linear
        linear_injective_0 subspace_UNIV by blast
  qed
qed

lemma closed_injective_linear_image_eq:
    fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
    assumes f: linear f inj f
      shows (closed(image f s) ⟷ closed s)
 by (metis closed_injective_linear_image closure_eq closure_linear_image_subset
closure_subset_eq f(1) f(2) inj_image_subset_iff)

lemma closure_injective_linear_image:
    fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
    shows ⟦linear f; inj f⟧ ⟹ f ‘ (closure S) = closure (f ‘ S)
  by (simp add: closed_injective_linear_image closure_linear_image_subset
       closure_minimal closure_subset image_mono subset_antisym)

lemma closure_bounded_linear_image:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes linear f bounded S
    shows f ‘ (closure S) = closure (f ‘ S)  (is ?lhs = ?rhs)
proof
  show ?lhs ⊆ ?rhs
    using assms closure_linear_image_subset by blast
  show ?rhs ⊆ ?lhs
   using assms by (meson closure_minimal closure_subset compact_closure com-
pact_eq_bounded_closed
                  compact_continuous_image image_mono linear_continuous_on
linear_linear)
qed
```

lemma *closure_scaleR*:
fixes $S :: 'a::real\_normed\_vector\ set$
shows $((*_R)\ c)\ `\ (closure\ S) = closure\ (((*_R)\ c)\ `\ S)$ (**is** *?lhs = ?rhs*)
proof
show *?lhs* $\subseteq$ *?rhs*
using *bounded_linear_scaleR_right* **by** (*rule closure_bounded_linear_image_subset*)
show *?rhs* $\subseteq$ *?lhs*
by (*intro closure_minimal image_mono closure_subset closed_scaling closed_closure*)
qed

4.1.20 Some properties of a canonical subspace

lemma *closed_substandard*: *closed* $\{x::'a::euclidean\_space.\ \forall i{\in}Basis.\ P\ i \longrightarrow x{\cdot}i = 0\}$
(**is** *closed ?A*)
proof −
let *?D* $= \{i{\in}Basis.\ P\ i\}$
have *closed* $(\bigcap i{\in}?D.\ \{x::'a.\ x{\cdot}i = 0\})$
by (*simp add*: *closed_INT closed_Collect_eq continuous_on_inner*)
also have $(\bigcap i{\in}?D.\ \{x::'a.\ x{\cdot}i = 0\}) = ?A$
by *auto*
finally show *closed ?A* **.**
qed

lemma *closed_subspace*:
fixes $S :: 'a::euclidean\_space\ set$
assumes *subspace S*
shows *closed S*
proof −
have *dim S* $\leq$ *card* $(Basis :: 'a\ set)$
using *dim_subset_UNIV* **by** *auto*
with *obtain_subset_with_card_n*
obtain $d :: 'a\ set$ **where** *cd*: *card d* $=$ *dim S* **and** *d*: $d \subseteq Basis$
by *metis*
let *?t* $= \{x::'a.\ \forall i{\in}Basis.\ i \notin d \longrightarrow x{\cdot}i = 0\}$
have $\exists f.\ linear\ f \wedge f\ `\ \{x::'a.\ \forall i{\in}Basis.\ i \notin d \longrightarrow x \cdot i = 0\} = S \wedge$
inj_on f $\{x::'a.\ \forall i{\in}Basis.\ i \notin d \longrightarrow x \cdot i = 0\}$
using *dim_substandard*[*of d*] *cd d assms*
by (*intro subspace_isomorphism*[*OF subspace_substandard*[*of* $\lambda i.\ i \notin d$]]) (*auto simp*: *inner_Basis*)
then obtain *f* **where** *f*:
linear f
$f\ `\ \{x.\ \forall i{\in}Basis.\ i \notin d \longrightarrow x \cdot i = 0\} = S$
inj_on f $\{x.\ \forall i{\in}Basis.\ i \notin d \longrightarrow x \cdot i = 0\}$
by *blast*
interpret *f*: *bounded_linear f*
using *f* **by** (*simp add*: *linear_conv_bounded_linear*)
have $x \in ?t \Longrightarrow f\ x = 0 \Longrightarrow x = 0$ **for** *x*

using *f.zero d f(3)*[*THEN inj_onD*, *of x 0*] **by** *auto*
then show *?thesis*
using *closed_injective_image_subspace*[*of ?t f*] *closed_substandard subspace_substandard*
using *f(2) f.bounded_linear_axioms* **by** *force*
qed

lemma *complete_subspace*: *subspace S* $\Longrightarrow$ *complete S*
for *S* :: *'a*::*euclidean_space set*
using *complete_eq_closed closed_subspace* **by** *auto*

lemma *closed_span* [*iff*]: *closed* (*span S*)
for *S* :: *'a*::*euclidean_space set*
by (*simp add*: *closed_subspace*)

lemma *dim_closure* [*simp*]: *dim* (*closure S*) = *dim S* (**is** *?dc* = *?d*)
for *S* :: *'a*::*euclidean_space set*
by (*metis closed_span closure_minimal closure_subset dim_eq_span span_eq_dim span_superset subset_le_dim*)

4.1.21 Set Distance

lemma *setdist_compact_closed*:
fixes *A* :: *'a*::*heine_borel set*
assumes *compact A closed B*
and *A* $\neq$ {} *B* $\neq$ {}
shows $\exists x \in A.\ \exists y \in B.$ *dist x y* = *setdist A B*
by (*metis assms infdist_attains_inf setdist_attains_inf setdist_sym*)

lemma *setdist_closed_compact*:
fixes *S* :: *'a*::*heine_borel set*
assumes *S*: *closed S* **and** *T*: *compact T*
and *S* $\neq$ {} *T* $\neq$ {}
shows $\exists x \in S.\ \exists y \in T.$ *dist x y* = *setdist S T*
using *setdist_compact_closed* [*OF T S* ‹*T* $\neq$ {}› ‹*S* $\neq$ {}›]
by (*metis dist_commute setdist_sym*)

lemma *setdist_eq_0_compact_closed*:
assumes *S*: *compact S* **and** *T*: *closed T*
shows *setdist S T* = *0* $\longleftrightarrow$ *S* = {} $\vee$ *T* = {} $\vee$ *S* $\cap$ *T* $\neq$ {}
proof (*cases S* = {} $\vee$ *T* = {})
case *False*
then show *?thesis*
by (*metis S T disjoint_iff in_closed_iff_infdist_zero setdist_attains_inf setdist_eq_0I setdist_sym*)
qed *auto*

corollary *setdist_gt_0_compact_closed*:
assumes *S*: *compact S* **and** *T*: *closed T*
shows *setdist S T* > *0* $\longleftrightarrow$ (*S* $\neq$ {} $\wedge$ *T* $\neq$ {} $\wedge$ *S* $\cap$ *T* = {})

using *setdist_pos_le* [*of S T*] *setdist_eq_0_compact_closed* [*OF assms*] **by** *linarith*

lemma *setdist_eq_0_closed_compact*:
assumes *S*: *closed S* **and** *T*: *compact T*
shows *setdist S T = 0* $\longleftrightarrow$ *S = {} ∨ T = {} ∨ S ∩ T ≠ {}*
using *setdist_eq_0_compact_closed* [*OF T S*]
by (*metis Int_commute setdist_sym*)

lemma *setdist_eq_0_bounded*:
fixes *S* :: *'a*::*heine_borel set*
assumes *bounded S ∨ bounded T*
shows *setdist S T = 0* $\longleftrightarrow$ *S = {} ∨ T = {} ∨ closure S ∩ closure T ≠ {}*
proof (*cases S = {} ∨ T = {}*)
case *False*
then show *?thesis*
using *setdist_eq_0_compact_closed* [*of closure S closure T*]
setdist_eq_0_closed_compact [*of closure S closure T*] *assms*
by (*force simp*: *bounded_closure compact_eq_bounded_closed*)
qed *force*

lemma *setdist_eq_0_sing_1*:
setdist {x} S = 0 $\longleftrightarrow$ *S = {} ∨ x ∈ closure S*
by (*metis in_closure_iff_infdist_zero infdist_def infdist_eq_setdist*)

lemma *setdist_eq_0_sing_2*:
setdist S {x} = 0 $\longleftrightarrow$ *S = {} ∨ x ∈ closure S*
by (*metis setdist_eq_0_sing_1 setdist_sym*)

lemma *setdist_neq_0_sing_1*:
⟦*setdist {x} S = a*; *a ≠ 0*⟧ $\Longrightarrow$ *S ≠ {} ∧ x ∉ closure S*
by (*metis setdist_closure_2 setdist_empty2 setdist_eq_0I singletonI*)

lemma *setdist_neq_0_sing_2*:
⟦*setdist S {x} = a*; *a ≠ 0*⟧ $\Longrightarrow$ *S ≠ {} ∧ x ∉ closure S*
by (*simp add*: *setdist_neq_0_sing_1 setdist_sym*)

lemma *setdist_sing_in_set*:
x ∈ S $\Longrightarrow$ *setdist {x} S = 0*
by (*simp add*: *setdist_eq_0I*)

lemma *setdist_eq_0_closed*:
closed S $\Longrightarrow$ (*setdist {x} S = 0* $\longleftrightarrow$ *S = {} ∨ x ∈ S*)
by (*simp add*: *setdist_eq_0_sing_1*)

lemma *setdist_eq_0_closedin*:
shows ⟦*closedin* (*top_of_set U*) *S*; *x ∈ U*⟧
$\Longrightarrow$ (*setdist {x} S = 0* $\longleftrightarrow$ *S = {} ∨ x ∈ S*)
by (*auto simp*: *closedin_limpt setdist_eq_0_sing_1 closure_def*)

lemma *setdist_gt_0_closedin*:
shows ⟦*closedin* (*top_of_set U*) *S*; *x* ∈ *U*; *S* ≠ {}; *x* ∉ *S*⟧
⟹ *setdist* {*x*} *S* > *0*
using *less_eq_real_def setdist_eq_0_closedin* **by** *fastforce*

no_notation
eucl_less (**infix** <*e* *50*)

end

4.2 Convex Sets and Functions on (Normed) Euclidean Spaces

theory *Convex_Euclidean_Space*
imports
Convex
Topology_Euclidean_Space
begin

4.2.1 Topological Properties of Convex Sets and Functions

lemma *aff_dim_cball*:
fixes *a* :: *'n*::*euclidean_space*
assumes *e* > *0*
shows *aff_dim* (*cball a e*) = *int* (*DIM*(*'n*))
proof −
have (λ*x*. *a* + *x*) ‘ (*cball 0 e*) ⊆ *cball a e*
unfolding *cball_def dist_norm* **by** *auto*
then have *aff_dim* (*cball* (*0* :: *'n*::*euclidean_space*) *e*) ≤ *aff_dim* (*cball a e*)
using *aff_dim_translation_eq*[*of a cball 0 e*]
aff_dim_subset[*of* (+) *a* ‘ *cball 0 e cball a e*]
by *auto*
moreover have *aff_dim* (*cball* (*0* :: *'n*::*euclidean_space*) *e*) = *int* (*DIM*(*'n*))
using *hull_inc*[*of* (*0* :: *'n*::*euclidean_space*) *cball 0 e*]
centre_in_cball[*of* (*0* :: *'n*::*euclidean_space*)] *assms*
by (*simp add*: *dim_cball*[*of e*] *aff_dim_zero*[*of cball 0 e*])
ultimately show *?thesis*
using *aff_dim_le_DIM*[*of cball a e*] **by** *auto*
qed

lemma *aff_dim_open*:
fixes *S* :: *'n*::*euclidean_space set*
assumes *open S*
and *S* ≠ {}
shows *aff_dim S* = *int* (*DIM*(*'n*))
proof −
obtain *x* **where** *x* ∈ *S*

```
    using assms by auto
  then obtain e where e: e > 0 cball x e ⊆ S
    using open_contains_cball[of S] assms by auto
  then have aff_dim (cball x e) ≤ aff_dim S
    using aff_dim_subset by auto
  with e show ?thesis
    using aff_dim_cball[of e x] aff_dim_le_DIM[of S] by auto
qed

lemma low_dim_interior:
  fixes S :: 'n::euclidean_space set
  assumes ¬ aff_dim S = int (DIM('n))
  shows interior S = {}
proof −
  have aff_dim(interior S) ≤ aff_dim S
    using interior_subset aff_dim_subset[of interior S S] by auto
  then show ?thesis
    using aff_dim_open[of interior S] aff_dim_le_DIM[of S] assms by auto
qed

corollary empty_interior_lowdim:
  fixes S :: 'n::euclidean_space set
  shows dim S < DIM ('n) ⟹ interior S = {}
by (metis low_dim_interior affine_hull_UNIV dim_affine_hull less_not_refl dim_UNIV)

corollary aff_dim_nonempty_interior:
  fixes S :: 'a::euclidean_space set
  shows interior S ≠ {} ⟹ aff_dim S = DIM('a)
by (metis low_dim_interior)
```

4.2.2 Relative interior of a set

```
definition rel_interior S =
  {x. ∃ T. openin (top_of_set (affine hull S)) T ∧ x ∈ T ∧ T ⊆ S}

lemma rel_interior_mono:
   ⟦S ⊆ T; affine hull S = affine hull T⟧
   ⟹ (rel_interior S) ⊆ (rel_interior T)
  by (auto simp: rel_interior_def)

lemma rel_interior_maximal:
   ⟦T ⊆ S; openin(top_of_set (affine hull S)) T⟧ ⟹ T ⊆ (rel_interior S)
  by (auto simp: rel_interior_def)

lemma rel_interior: rel_interior S = {x ∈ S. ∃ T. open T ∧ x ∈ T ∧ T ∩ affine
hull S ⊆ S}
      (is ?lhs = ?rhs)
proof
  show ?lhs ⊆ ?rhs
```

by (*force simp add*: *rel_interior_def openin_open*)
{ fix *x T*
assume ∗: *x* ∈ *S open T x* ∈ *T T* ∩ *affine hull S* ⊆ *S*
then have ∗∗: *x* ∈ *T* ∩ *affine hull S*
using *hull_inc* **by** *auto*
with ∗ **have** ∃ *Tb*. (∃ *Ta*. *open Ta* ∧ *Tb* = *affine hull S* ∩ *Ta*) ∧ *x* ∈ *Tb* ∧ *Tb* ⊆ *S*
by (*rule_tac x* = *T* ∩ (*affine hull S*) **in** *exI*) *auto*
}
then show *?rhs* ⊆ *?lhs*
by (*force simp add*: *rel_interior_def openin_open*)
qed

lemma *mem_rel_interior*: *x* ∈ *rel_interior S* ⟷ (∃ *T*. *open T* ∧ *x* ∈ *T* ∩ *S* ∧ *T* ∩ *affine hull S* ⊆ *S*)
by (*auto simp*: *rel_interior*)

lemma *mem_rel_interior_ball*:
x ∈ *rel_interior S* ⟷ *x* ∈ *S* ∧ (∃ *e*. *e* > *0* ∧ *ball x e* ∩ *affine hull S* ⊆ *S*)
(**is** *?lhs* = *?rhs*)
proof
assume *?rhs* **then show** *?lhs*
by (*simp add*: *rel_interior*) (*meson Elementary_Metric_Spaces.open_ball centre_in_ball*)
qed (*force simp*: *rel_interior open_contains_ball*)

lemma *rel_interior_ball*:
rel_interior S = {*x* ∈ *S*. ∃ *e*. *e* > *0* ∧ *ball x e* ∩ *affine hull S* ⊆ *S*}
using *mem_rel_interior_ball* [*of* _ *S*] **by** *auto*

lemma *mem_rel_interior_cball*:
x ∈ *rel_interior S* ⟷ *x* ∈ *S* ∧ (∃ *e*. *e* > *0* ∧ *cball x e* ∩ *affine hull S* ⊆ *S*)
(**is** *?lhs* = *?rhs*)
proof
assume *?rhs* **then obtain** *e* **where** *x* ∈ *S e* > *0 cball x e* ∩ *affine hull S* ⊆ *S*
by (*auto simp*: *rel_interior*)
then have *ball x e* ∩ *affine hull S* ⊆ *S*
by *auto*
then show *?lhs*
using ‹*0* < *e*› ‹*x* ∈ *S*› *rel_interior_ball* **by** *auto*
qed (*force simp*: *rel_interior open_contains_cball*)

lemma *rel_interior_cball*:
rel_interior S = {*x* ∈ *S*. ∃ *e*. *e* > *0* ∧ *cball x e* ∩ *affine hull S* ⊆ *S*}
using *mem_rel_interior_cball* [*of* _ *S*] **by** *auto*

lemma *rel_interior_empty* [*simp*]: *rel_interior* {} = {}
by (*auto simp*: *rel_interior_def*)

lemma *affine_hull_sing* [*simp*]: *affine hull* {*a* :: *'n*::*euclidean_space*} = {*a*}
by (*metis affine_hull_eq affine_sing*)

lemma *rel_interior_sing* [*simp*]:
fixes *a* :: *'n*::*euclidean_space* **shows** *rel_interior* {*a*} = {*a*}
proof −
have ∃ *x*::*real*. *0* < *x*
using *zero_less_one* **by** *blast*
then show *?thesis*
by (*auto simp*: *rel_interior_ball*)
qed

lemma *subset_rel_interior*:
fixes *S T* :: *'n*::*euclidean_space set*
assumes *S* ⊆ *T*
and *affine hull S* = *affine hull T*
shows *rel_interior S* ⊆ *rel_interior T*
using *assms* **by** (*auto simp*: *rel_interior_def*)

lemma *rel_interior_subset*: *rel_interior S* ⊆ *S*
by (*auto simp*: *rel_interior_def*)

lemma *rel_interior_subset_closure*: *rel_interior S* ⊆ *closure S*
using *rel_interior_subset* **by** (*auto simp*: *closure_def*)

lemma *interior_subset_rel_interior*: *interior S* ⊆ *rel_interior S*
by (*auto simp*: *rel_interior interior_def*)

lemma *interior_rel_interior*:
fixes *S* :: *'n*::*euclidean_space set*
assumes *aff_dim S* = *int*(*DIM*(*'n*))
shows *rel_interior S* = *interior S*
proof −
have *affine hull S* = *UNIV*
using *assms affine_hull_UNIV*[*of S*] **by** *auto*
then show *?thesis*
unfolding *rel_interior interior_def* **by** *auto*
qed

lemma *rel_interior_interior*:
fixes *S* :: *'n*::*euclidean_space set*
assumes *affine hull S* = *UNIV*
shows *rel_interior S* = *interior S*
using *assms* **unfolding** *rel_interior interior_def* **by** *auto*

lemma *rel_interior_open*:
fixes *S* :: *'n*::*euclidean_space set*
assumes *open S*
shows *rel_interior S* = *S*

by (*metis assms interior_eq interior_subset_rel_interior rel_interior_subset set_eq_subset*)

lemma *interior_rel_interior_gen*:
fixes S :: *'n::euclidean_space set*
shows *interior* S = (*if aff_dim* S = *int*(*DIM*(*'n*)) *then rel_interior* S *else* {})
by (*metis interior_rel_interior low_dim_interior*)

lemma *rel_interior_nonempty_interior*:
fixes S :: *'n::euclidean_space set*
shows *interior* $S \neq \{\} \Longrightarrow$ *rel_interior* S = *interior* S
by (*metis interior_rel_interior_gen*)

lemma *affine_hull_nonempty_interior*:
fixes S :: *'n::euclidean_space set*
shows *interior* $S \neq \{\} \Longrightarrow$ *affine hull* S = *UNIV*
by (*metis affine_hull_UNIV interior_rel_interior_gen*)

lemma *rel_interior_affine_hull* [*simp*]:
fixes S :: *'n::euclidean_space set*
shows *rel_interior* (*affine hull* S) = *affine hull* S
proof −
have ∗: *rel_interior* (*affine hull* S) ⊆ *affine hull* S
using *rel_interior_subset* **by** *auto*
{
fix x
assume x: $x \in$ *affine hull* S
define e :: *real* **where** e = *1*
then have $e > 0$ *ball* x $e \cap$ *affine hull* (*affine hull* S) ⊆ *affine hull* S
using *hull_hull*[*of* _ S] **by** *auto*
then have $x \in$ *rel_interior* (*affine hull* S)
using x *rel_interior_ball*[*of affine hull* S] **by** *auto*
}
then show *?thesis* **using** ∗ **by** *auto*
qed

lemma *rel_interior_UNIV* [*simp*]: *rel_interior* (*UNIV* :: (*'n::euclidean_space*) *set*) = *UNIV*
by (*metis open_UNIV rel_interior_open*)

lemma *rel_interior_convex_shrink*:
fixes S :: *'a::euclidean_space set*
assumes *convex* S
and $c \in$ *rel_interior* S
and $x \in S$
and $0 < e$
and $e \leq 1$
shows $x - e *_R (x - c) \in$ *rel_interior* S
proof −

```
  obtain d where d > 0 and d: ball c d ∩ affine hull S ⊆ S
    using assms(2) unfolding  mem_rel_interior_ball by auto
  {
    fix y
    assume as: dist (x − e *R (x − c)) y < e * d y ∈ affine hull S
    have *: y = (1 − (1 − e)) *R ((1 / e) *R y − ((1 − e) / e) *R x) + (1 − e)
*R x
      using ‹e > 0› by (auto simp: scaleR_left_diff_distrib scaleR_right_diff_distrib)
    have x ∈ affine hull S
      using assms hull_subset[of S] by auto
    moreover have 1 / e + − ((1 − e) / e) = 1
      using ‹e > 0› left_diff_distrib[of 1 (1−e) 1/e] by auto
    ultimately have **: (1 / e) *R y − ((1 − e) / e) *R x ∈ affine hull S
     using as affine_affine_hull[of S] mem_affine[of affine hull S y x (1 / e) −((1
− e) / e)]
      by (simp add: algebra_simps)
    have c − ((1 / e) *R y − ((1 − e) / e) *R x) = (1 / e) *R (e *R c − y + (1
− e) *R x)
      using ‹e > 0›
      by (auto simp: euclidean_eq_iff[where 'a='a] field_simps inner_simps)
    then have dist c ((1 / e) *R y − ((1 − e) / e) *R x) = |1/e| * norm (e *R c
− y + (1 − e) *R x)
      unfolding dist_norm norm_scaleR[symmetric] by auto
    also have ... = |1/e| * norm (x − e *R (x − c) − y)
      by (auto intro!:arg_cong[where f=norm] simp add: algebra_simps)
    also have ... < d
      using as[unfolded dist_norm] and ‹e > 0›
      by (auto simp:pos_divide_less_eq[OF ‹e > 0›] mult.commute)
    finally have (1 / e) *R y − ((1 − e) / e) *R x ∈ S
      using **  d by auto
    then have y ∈ S
      using * convexD [OF ‹convex S›] assms(3−5)
     by (metis diff_add_cancel diff_ge_0_iff_ge le_add_same_cancel1 less_eq_real_def)
  }
  then have ball (x − e *R (x − c)) (e*d) ∩ affine hull S ⊆ S
    by auto
  moreover have e * d > 0
    using ‹e > 0› ‹d > 0› by simp
  moreover have c: c ∈ S
    using assms rel_interior_subset by auto
  moreover from c have x − e *R (x − c) ∈ S
    using convexD_alt[of S x c e] assms
   by (metis  diff_add_eq diff_diff_eq2 less_eq_real_def scaleR_diff_left scaleR_one
scale_right_diff_distrib)
  ultimately show ?thesis
    using mem_rel_interior_ball[of x − e *R (x − c) S] ‹e > 0› by auto
qed

lemma interior_real_atLeast [simp]:
```

```
  fixes a :: real
  shows interior {a..} = {a<..}
proof −
  {
    fix y
    have ball y (y − a) ⊆ {a..}
      by (auto simp: dist_norm)
    moreover assume a < y
    ultimately have y ∈ interior {a..}
      by (force simp add: mem_interior)
  }
  moreover
  {
    fix y
    assume y ∈ interior {a..}
    then obtain e where e: e > 0 cball y e ⊆ {a..}
      using mem_interior_cball[of y {a..}] by auto
    moreover from e have y − e ∈ cball y e
      by (auto simp: cball_def dist_norm)
    ultimately have a ≤ y − e by blast
    then have a < y using e by auto
  }
  ultimately show ?thesis by auto
qed

lemma continuous_ge_on_Ioo:
   assumes continuous_on {c..d} g ⋀x. x ∈ {c<..<d} ⟹ g x ≥ a c < d x ∈
{c..d}
  shows g (x::real) ≥ (a::real)
proof−
 from assms(3) have {c..d} = closure {c<..<d} by (rule closure_greaterThanLessThan[symmetric])
  also from assms(2) have {c<..<d} ⊆ (g −‘ {a..} ∩ {c..d}) by auto
  hence closure {c<..<d} ⊆ closure (g −‘ {a..} ∩ {c..d}) by (rule closure_mono)
  also from assms(1) have closed (g −‘ {a..} ∩ {c..d})
    by (auto simp: continuous_on_closed_vimage)
  hence closure (g −‘ {a..} ∩ {c..d}) = g −‘ {a..} ∩ {c..d} by simp
  finally show ?thesis using ‹x ∈ {c..d}› by auto
qed

lemma interior_real_atMost [simp]:
  fixes a :: real
  shows interior {..a} = {..<a}
proof −
  {
    fix y
    have ball y (a − y) ⊆ {..a}
      by (auto simp: dist_norm)
    moreover assume a > y
    ultimately have y ∈ interior {..a}
```

```
    by (force simp add: mem_interior)
  }
  moreover
  {
    fix y
    assume y ∈ interior {..a}
    then obtain e where e: e > 0 cball y e ⊆ {..a}
      using mem_interior_cball[of y {..a}] by auto
    moreover from e have y + e ∈ cball y e
      by (auto simp: cball_def dist_norm)
    ultimately have a ≥ y + e by auto
    then have a > y using e by auto
  }
  ultimately show ?thesis by auto
qed

lemma interior_atLeastAtMost_real [simp]: interior {a..b} = {a<..<b :: real}
proof−
  have {a..b} = {a..} ∩ {..b} by auto
  also have interior ... = {a<..} ∩ {..<b}
    by (simp)
  also have ... = {a<..<b} by auto
  finally show ?thesis .
qed

lemma interior_atLeastLessThan [simp]:
  fixes a::real shows interior {a..<b} = {a<..<b}
 by (metis atLeastLessThan_def greaterThanLessThan_def interior_atLeastAtMost_real
interior_Int interior_interior interior_real_atLeast)

lemma interior_lessThanAtMost [simp]:
  fixes a::real shows interior {a<..b} = {a<..<b}
 by (metis atLeastAtMost_def greaterThanAtMost_def interior_atLeastAtMost_real
interior_Int
          interior_interior interior_real_atLeast)

lemma interior_greaterThanLessThan_real [simp]: interior {a<..<b} = {a<..<b
:: real}
  by (metis interior_atLeastAtMost_real interior_interior)

lemma frontier_real_atMost [simp]:
  fixes a :: real
  shows frontier {..a} = {a}
  unfolding frontier_def by auto

lemma frontier_real_atLeast [simp]: frontier {a..} = {a::real}
  by (auto simp: frontier_def)

lemma frontier_real_greaterThan [simp]: frontier {a<..} = {a::real}
```

by (*auto simp*: *interior_open frontier_def*)

lemma *frontier_real_lessThan* [*simp*]: *frontier* {..<*a*} = {*a*::*real*}
by (*auto simp*: *interior_open frontier_def*)

lemma *rel_interior_real_box* [*simp*]:
fixes *a b* :: *real*
assumes *a* < *b*
shows *rel_interior* {*a* .. *b*} = {*a* <..< *b*}
proof −
have *box a b* ≠ {}
using *assms*
unfolding *set_eq_iff*
by (*auto intro*!: *exI*[*of* _ (*a* + *b*) / *2*] *simp*: *box_def*)
then show *?thesis*
using *interior_rel_interior_gen*[*of cbox a b*, *symmetric*]
by (*simp split*: *if_split_asm del*: *box_real add*: *box_real*[*symmetric*])
qed

lemma *rel_interior_real_semiline* [*simp*]:
fixes *a* :: *real*
shows *rel_interior* {*a*..} = {*a*<..}
proof −
have ∗: {*a*<..} ≠ {}
unfolding *set_eq_iff* **by** (*auto intro*!: *exI*[*of* _ *a* + *1*])
then show *?thesis* **using** *interior_real_atLeast interior_rel_interior_gen*[*of* {*a*..}]
by (*auto split*: *if_split_asm*)
qed

Relative open sets

definition *rel_open S* ⟷ *rel_interior S* = *S*

lemma *rel_open*: *rel_open S* ⟷ *openin* (*top_of_set* (*affine hull S*)) *S* (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
unfolding *rel_open_def rel_interior_def*
using *openin_subopen*[*of top_of_set* (*affine hull S*) *S*] **by** *auto*
qed (*auto simp*: *rel_open_def rel_interior_def*)

lemma *openin_rel_interior*: *openin* (*top_of_set* (*affine hull S*)) (*rel_interior S*)
using *openin_subopen* **by** (*fastforce simp add*: *rel_interior_def*)

lemma *openin_set_rel_interior*:
openin (*top_of_set S*) (*rel_interior S*)
by (*rule openin_subset_trans* [*OF openin_rel_interior rel_interior_subset hull_subset*])

```
lemma affine_rel_open:
  fixes S :: 'n::euclidean_space set
  assumes affine S
  shows rel_open S
  unfolding rel_open_def
  using assms rel_interior_affine_hull[of S] affine_hull_eq[of S]
  by metis

lemma affine_closed:
  fixes S :: 'n::euclidean_space set
  assumes affine S
  shows closed S
proof -
  {
    assume S ≠ {}
    then obtain L where L: subspace L affine_parallel S L
      using assms affine_parallel_subspace[of S] by auto
    then obtain a where a: S = ((+) a ` L)
      using affine_parallel_def[of L S] affine_parallel_commut by auto
    from L have closed L using closed_subspace by auto
    then have closed S
      using closed_translation a by auto
  }
  then show ?thesis by auto
qed

lemma closure_affine_hull:
  fixes S :: 'n::euclidean_space set
  shows closure S ⊆ affine hull S
  by (intro closure_minimal hull_subset affine_closed affine_affine_hull)

lemma closed_affine_hull [iff]:
  fixes S :: 'n::euclidean_space set
  shows closed (affine hull S)
  by (metis affine_affine_hull affine_closed)

lemma closure_same_affine_hull [simp]:
  fixes S :: 'n::euclidean_space set
  shows affine hull (closure S) = affine hull S
proof -
  have affine hull (closure S) ⊆ affine hull S
    using hull_mono[of closure S affine hull S affine]
      closure_affine_hull[of S] hull_hull[of affine S]
    by auto
  moreover have affine hull (closure S) ⊇ affine hull S
    using hull_mono[of S closure S affine] closure_subset by auto
  ultimately show ?thesis by auto
qed
```

```
lemma closure_aff_dim [simp]:
  fixes S :: 'n::euclidean_space set
  shows aff_dim (closure S) = aff_dim S
proof −
  have aff_dim S ≤ aff_dim (closure S)
    using aff_dim_subset closure_subset by auto
  moreover have aff_dim (closure S) ≤ aff_dim (affine hull S)
    using aff_dim_subset closure_affine_hull by blast
  moreover have aff_dim (affine hull S) = aff_dim S
    using aff_dim_affine_hull by auto
  ultimately show ?thesis by auto
qed

lemma rel_interior_closure_convex_shrink:
  fixes S :: _::euclidean_space set
  assumes convex S
    and c ∈ rel_interior S
    and x ∈ closure S
    and e > 0
    and e ≤ 1
  shows x − e *R (x − c) ∈ rel_interior S
proof −
  obtain d where d > 0 and d: ball c d ∩ affine hull S ⊆ S
    using assms(2) unfolding mem_rel_interior_ball by auto
  have ∃y ∈ S. norm (y − x) * (1 − e) < e * d
  proof (cases x ∈ S)
    case True
    then show ?thesis using ‹e > 0› ‹d > 0› by force
  next
    case False
    then have x: x islimpt S
      using assms(3)[unfolded closure_def] by auto
    show ?thesis
    proof (cases e = 1)
      case True
      obtain y where y ∈ S y ≠ x dist y x < 1
        using x[unfolded islimpt_approachable,THEN spec[where x=1]] by auto
      then show ?thesis
        unfolding True using ‹d > 0› by (force simp add: )
    next
      case False
      then have 0 < e * d / (1 − e) and *: 1 − e > 0
        using ‹e ≤ 1› ‹e > 0› ‹d > 0› by auto
      then obtain y where y ∈ S y ≠ x dist y x < e * d / (1 − e)
          using x[unfolded islimpt_approachable,THEN spec[where x=e*d / (1 −
e)]] by auto
      then show ?thesis
        unfolding dist_norm using pos_less_divide_eq[OF *] by force
```

```
    qed
  qed
  then obtain y where y ∈ S and y: norm (y − x) ∗ (1 − e) < e ∗ d
    by auto
  define z where z = c + ((1 − e) / e) ∗R (x − y)
  have ∗: x − e ∗R (x − c) = y − e ∗R (y − z)
    unfolding z_def using ‹e > 0›
   by (auto simp: scaleR_right_diff_distrib scaleR_right_distrib scaleR_left_diff_distrib)
  have zball: z ∈ ball c d
    using mem_ball z_def dist_norm[of c]
    using y and assms(4,5)
    by (simp add: norm_minus_commute) (simp add: field_simps)
  have x ∈ affine hull S
    using closure_affine_hull assms by auto
  moreover have y ∈ affine hull S
    using ‹y ∈ S› hull_subset[of S] by auto
  moreover have c ∈ affine hull S
    using assms rel_interior_subset hull_subset[of S] by auto
  ultimately have z ∈ affine hull S
    using z_def affine_affine_hull[of S]
      mem_affine_3_minus [of affine hull S c x y (1 − e) / e]
      assms
    by simp
  then have z ∈ S using d zball by auto
  obtain d1 where d1 > 0 and d1: ball z d1 ≤ ball c d
    using zball open_ball[of c d] openE[of ball c d z] by auto
  then have ball z d1 ∩ affine hull S ⊆ ball c d ∩ affine hull S
    by auto
  then have ball z d1 ∩ affine hull S ⊆ S
    using d by auto
  then have z ∈ rel_interior S
    using mem_rel_interior_ball using ‹d1 > 0› ‹z ∈ S› by auto
  then have y − e ∗R (y − z) ∈ rel_interior S
    using rel_interior_convex_shrink[of S z y e] assms ‹y ∈ S› by auto
  then show ?thesis using ∗ by auto
qed

lemma rel_interior_eq:
   rel_interior s = s ⟷ openin(top_of_set (affine hull s)) s
using rel_open rel_open_def by blast

lemma rel_interior_openin:
   openin(top_of_set (affine hull s)) s ⟹ rel_interior s = s
by (simp add: rel_interior_eq)

lemma rel_interior_affine:
  fixes S :: 'n::euclidean_space set
  shows  affine S ⟹ rel_interior S = S
using affine_rel_open rel_open_def by auto
```

lemma *rel_interior_eq_closure*:
fixes S :: *'n::euclidean_space set*
shows *rel_interior* S = *closure* $S \longleftrightarrow$ *affine* S
proof (*cases* $S = \{\}$)
case *True*
then show *?thesis*
by *auto*
next
case *False* **show** *?thesis*
proof
assume *eq*: *rel_interior* S = *closure* S
have *openin* (*top_of_set* (*affine hull* S)) S
by (*metis eq closure_subset openin_rel_interior rel_interior_subset subset_antisym*)
moreover have *closedin* (*top_of_set* (*affine hull* S)) S
by (*metis closed_subset closure_subset_eq eq hull_subset rel_interior_subset*)
ultimately have $S = \{\} \vee S$ = *affine hull* S
using *convex_connected connected_clopen convex_affine_hull* **by** *metis*
with *False* **have** *affine hull* S = S
by *auto*
then show *affine* S
by (*metis affine_hull_eq*)
next
assume *affine* S
then show *rel_interior* S = *closure* S
by (*simp add*: *rel_interior_affine affine_closed*)
qed
qed

Relative interior preserves under linear transformations

lemma *rel_interior_translation_aux*:
fixes a :: *'n::euclidean_space*
shows $((\lambda x.\ a + x)$ ‘ *rel_interior* $S) \subseteq$ *rel_interior* $((\lambda x.\ a + x)$ ‘ $S)$
proof −
{
fix x
assume x: $x \in$ *rel_interior* S
then obtain T **where** *open* T $x \in T \cap S$ $T \cap$ *affine hull* $S \subseteq S$
using *mem_rel_interior*[*of* x S] **by** *auto*
then have *open* $((\lambda x.\ a + x)$ ‘ $T)$
and $a + x \in ((\lambda x.\ a + x)$ ‘ $T) \cap ((\lambda x.\ a + x)$ ‘ $S)$
and $((\lambda x.\ a + x)$ ‘ $T) \cap$ *affine hull* $((\lambda x.\ a + x)$ ‘ $S) \subseteq (\lambda x.\ a + x)$ ‘ S
using *affine_hull_translation*[*of* a S] *open_translation*[*of* T a] x **by** *auto*
then have $a + x \in$ *rel_interior* $((\lambda x.\ a + x)$ ‘ $S)$
using *mem_rel_interior*[*of* a+x $((\lambda x.\ a + x)$ ‘ $S)$] **by** *auto*
}
then show *?thesis* **by** *auto*

qed

lemma *rel_interior_translation*:
fixes $a :: 'n::euclidean\_space$
shows $rel\_interior\ ((\lambda x.\ a + x)\ `\ S) = (\lambda x.\ a + x)\ `\ rel\_interior\ S$
proof −
have $(\lambda x.\ (-a) + x)\ `\ rel\_interior\ ((\lambda x.\ a + x)\ `\ S) \subseteq rel\_interior\ S$
using *rel_interior_translation_aux*[*of* $-a$ $(\lambda x.\ a + x)\ `\ S$]
translation_assoc[*of* $-a$ *a*]
by *auto*
then have $((\lambda x.\ a + x)\ `\ rel\_interior\ S) \supseteq rel\_interior\ ((\lambda x.\ a + x)\ `\ S)$
using *translation_inverse_subset*[*of a rel_interior* $((+)\ a\ `\ S)$ *rel_interior S*]
by *auto*
then show *?thesis*
using *rel_interior_translation_aux*[*of a S*] **by** *auto*
qed

lemma *affine_hull_linear_image*:
assumes *bounded_linear f*
shows $f\ `\ (affine\ hull\ s) = affine\ hull\ f\ `\ s$
proof −
interpret *f*: *bounded_linear f* **by** *fact*
have *affine* $\{x.\ f\ x \in affine\ hull\ f\ `\ s\}$
unfolding *affine_def*
by (*auto simp*: *f.scaleR f.add affine_affine_hull*[*unfolded affine_def*, *rule_format*])
moreover have *affine* $\{x.\ x \in f\ `\ (affine\ hull\ s)\}$
using *affine_affine_hull*[*unfolded affine_def*, *of s*]
unfolding *affine_def* **by** (*auto simp*: *f.scaleR* [*symmetric*] *f.add* [*symmetric*])
ultimately show *?thesis*
by (*auto simp*: *hull_inc elim*!: *hull_induct*)
qed

lemma *rel_interior_injective_on_span_linear_image*:
fixes $f :: 'm::euclidean\_space \Rightarrow 'n::euclidean\_space$
and $S :: 'm::euclidean\_space\ set$
assumes *bounded_linear f*
and *inj_on f* (*span S*)
shows $rel\_interior\ (f\ `\ S) = f\ `\ (rel\_interior\ S)$
proof −
{
fix *z*
assume *z*: $z \in rel\_interior\ (f\ `\ S)$
then have $z \in f\ `\ S$
using *rel_interior_subset*[*of f* ‘ *S*] **by** *auto*
then obtain *x* **where** *x*: $x \in S\ f\ x = z$ **by** *auto*
obtain *e2* **where** *e2*: $e2 > 0\ cball\ z\ e2 \cap affine\ hull\ (f\ `\ S) \subseteq (f\ `\ S)$
using *z rel_interior_cball*[*of f* ‘ *S*] **by** *auto*

```
  obtain K where K: K > 0 ⋀x. norm (f x) ≤ norm x * K
   using assms Real_Vector_Spaces.bounded_linear.pos_bounded[of f] by auto
  define e1 where e1 = 1 / K
  then have e1: e1 > 0 ⋀x. e1 * norm (f x) ≤ norm x
    using K pos_le_divide_eq[of e1] by auto
  define e where e = e1 * e2
  then have e > 0 using e1 e2 by auto
  {
    fix y
    assume y: y ∈ cball x e ∩ affine hull S
    then have h1: f y ∈ affine hull (f ‘ S)
      using affine_hull_linear_image[of f S] assms by auto
    from y have norm (x−y) ≤ e1 * e2
      using cball_def[of x e] dist_norm[of x y] e_def by auto
    moreover have f x − f y = f (x − y)
     using assms linear_diff[of f x y] linear_conv_bounded_linear[of f] by auto
    moreover have e1 * norm (f (x−y)) ≤ norm (x − y)
      using e1 by auto
    ultimately have e1 * norm ((f x)−(f y)) ≤ e1 * e2
      by auto
    then have f y ∈ cball z e2
      using cball_def[of f x e2] dist_norm[of f x f y] e1 x by auto
    then have f y ∈ f ‘ S
      using y e2 h1 by auto
    then have y ∈ S
      using assms y hull_subset[of S] affine_hull_subset_span
        inj_on_image_mem_iff [OF ‹inj_on f (span S)›]
      by (metis Int_iff span_superset subsetCE)
  }
  then have z ∈ f ‘ (rel_interior S)
    using mem_rel_interior_cball[of x S] ‹e > 0› x by auto
}
moreover
{
  fix x
  assume x: x ∈ rel_interior S
  then obtain e2 where e2: e2 > 0 cball x e2 ∩ affine hull S ⊆ S
    using rel_interior_cball[of S] by auto
  have x ∈ S using x rel_interior_subset by auto
  then have ∗: f x ∈ f ‘ S by auto
  have ∀ x∈span S. f x = 0 ⟶ x = 0
    using assms subspace_span linear_conv_bounded_linear[of f]
      linear_injective_on_subspace_0[of f span S]
    by auto
  then obtain e1 where e1: e1 > 0 ∀ x ∈ span S. e1 * norm x ≤ norm (f x)
    using assms injective_imp_isometric[of span S f]
      subspace_span[of S] closed_subspace[of span S]
    by auto
  define e where e = e1 * e2
```

```
    hence e > 0 using e1 e2 by auto
    {
      fix y
      assume y: y ∈ cball (f x) e ∩ affine hull (f ‘ S)
      then have y ∈ f ‘ (affine hull S)
        using affine_hull_linear_image[of f S] assms by auto
      then obtain xy where xy: xy ∈ affine hull S f xy = y by auto
      with y have norm (f x − f xy) ≤ e1 ∗ e2
        using cball_def[of f x e] dist_norm[of f x y] e_def by auto
      moreover have f x − f xy = f (x − xy)
       using assms linear_diff[of f x xy] linear_conv_bounded_linear[of f] by auto
      moreover have ∗: x − xy ∈ span S
        using subspace_diff[of span S x xy] subspace_span ‹x ∈ S› xy
          affine_hull_subset_span[of S] span_superset
        by auto
      moreover from ∗ have e1 ∗ norm (x − xy) ≤ norm (f (x − xy))
        using e1 by auto
      ultimately have e1 ∗ norm (x − xy) ≤ e1 ∗ e2
        by auto
      then have xy ∈ cball x e2
        using cball_def[of x e2] dist_norm[of x xy] e1 by auto
      then have y ∈ f ‘ S
        using xy e2 by auto
    }
    then have f x ∈ rel_interior (f ‘ S)
      using mem_rel_interior_cball[of (f x) (f ‘ S)] ∗ ‹e > 0› by auto
  }
  ultimately show ?thesis by auto
qed

lemma rel_interior_injective_linear_image:
  fixes f :: 'm::euclidean_space ⇒ 'n::euclidean_space
  assumes bounded_linear f
    and inj f
  shows rel_interior (f ‘ S) = f ‘ (rel_interior S)
  using assms rel_interior_injective_on_span_linear_image[of f S]
    subset_inj_on[of f UNIV span S]
  by auto
```

4.2.3 Openness and compactness are preserved by convex hull operation

```
lemma open_convex_hull[intro]:
  fixes S :: 'a::real_normed_vector set
  assumes open S
  shows open (convex hull S)
proof (clarsimp simp: open_contains_cball convex_hull_explicit)
  fix T and u :: 'a⇒real
  assume obt: finite T T⊆S ∀ x∈T. 0 ≤ u x sum u T = 1
```

```
  from assms[unfolded open_contains_cball] obtain b
    where b: ⋀x. x∈S ⟹ 0 < b x ∧ cball x (b x) ⊆ S by metis
  have b ‘ T ≠ {}
    using obt by auto
  define i where i = b ‘ T
  let ?Φ = λy. ∃ F. finite F ∧ F ⊆ S ∧ (∃ u. (∀ x∈F. 0 ≤ u x) ∧ sum u F = 1 ∧
(∑ v∈F. u v *R v) = y)
  let ?a = ∑ v∈T. u v *R v
  show ∃ e > 0. cball ?a e ⊆ {y. ?Φ y}
  proof (intro exI subsetI conjI)
    show 0 < Min i
      unfolding i_def and Min_gr_iff[OF finite_imageI[OF obt(1)] ‹b ‘ T≠{}›]
      using b ‹T⊆S› by auto
  next
    fix y
    assume y ∈ cball ?a (Min i)
    then have y: norm (?a − y) ≤ Min i
      unfolding dist_norm[symmetric] by auto
    { fix x
      assume x ∈ T
      then have Min i ≤ b x
        by (simp add: i_def obt(1))
      then have x + (y − ?a) ∈ cball x (b x)
        using y unfolding mem_cball dist_norm by auto
      moreover have x ∈ S
        using ‹x∈T› ‹T⊆S› by auto
      ultimately have x + (y − ?a) ∈ S
        using y b by blast
    }
    moreover
    have *: inj_on (λv. v + (y − ?a)) T
      unfolding inj_on_def by auto
    have (∑ v∈(λv. v + (y − ?a)) ‘ T. u (v − (y − ?a)) *R v) = y
      unfolding sum.reindex[OF *] o_def using obt(4)
    by (simp add: sum.distrib sum_subtractf scaleR_left.sum[symmetric] scaleR_right_distrib)
    ultimately show y ∈ {y. ?Φ y}
    proof (intro CollectI exI conjI)
      show finite ((λv. v + (y − ?a)) ‘ T)
        by (simp add: obt(1))
      show sum (λv. u (v − (y − ?a))) ((λv. v + (y − ?a)) ‘ T) = 1
        unfolding sum.reindex[OF *] o_def using obt(4) by auto
    qed (use obt(1, 3) in auto)
  qed
qed

lemma compact_convex_combinations:
  fixes S T :: 'a::real_normed_vector set
  assumes compact S compact T
```

```
  shows compact { (1 − u) *R x + u *R y | x y u. 0 ≤ u ∧ u ≤ 1 ∧ x ∈ S ∧ y
∈ T}
proof −
  let ?X = {0..1} × S × T
  let ?h = (λz. (1 − fst z) *R fst (snd z) + fst z *R snd (snd z))
  have *: { (1 − u) *R x + u *R y | x y u. 0 ≤ u ∧ u ≤ 1 ∧ x ∈ S ∧ y ∈ T} =
?h ‘ ?X
    by force
  have continuous_on ?X (λz. (1 − fst z) *R fst (snd z) + fst z *R snd (snd z))
    unfolding continuous_on by (rule ballI) (intro tendsto_intros)
  with assms show ?thesis
    by (simp add: * compact_Times compact_continuous_image)
qed

lemma finite_imp_compact_convex_hull:
  fixes S :: 'a::real_normed_vector set
  assumes finite S
  shows compact (convex hull S)
proof (cases S = {})
  case True
  then show ?thesis by simp
next
  case False
  with assms show ?thesis
  proof (induct rule: finite_ne_induct)
    case (singleton x)
    show ?case by simp
  next
    case (insert x A)
    let ?f = λ(u, y::'a). u *R x + (1 − u) *R y
    let ?T = {0..1::real} × (convex hull A)
    have continuous_on ?T ?f
      unfolding split_def continuous_on by (intro ballI tendsto_intros)
    moreover have compact ?T
      by (intro compact_Times compact_Icc insert)
    ultimately have compact (?f ‘ ?T)
      by (rule compact_continuous_image)
    also have ?f ‘ ?T = convex hull (insert x A)
      unfolding convex_hull_insert [OF ‹A ≠ {}›]
      apply safe
      apply (rule_tac x=a in exI, simp)
      apply (rule_tac x=1 − a in exI, simp, fast)
      apply (rule_tac x=(u, b) in image_eqI, simp_all)
      done
    finally show compact (convex hull (insert x A)) .
  qed
qed

lemma compact_convex_hull:
```

```
  fixes S :: 'a::euclidean_space set
  assumes compact S
  shows compact (convex hull S)
proof (cases S = {})
  case True
  then show ?thesis using compact_empty by simp
next
  case False
  then obtain w where w ∈ S by auto
  show ?thesis
    unfolding caratheodory[of S]
  proof (induct (DIM('a) + 1))
    case 0
    have ∗: {x.∃ sa. finite sa ∧ sa ⊆ S ∧ card sa ≤ 0 ∧ x ∈ convex hull sa} = {}
      using compact_empty by auto
    from 0 show ?case unfolding ∗ by simp
  next
    case (Suc n)
    show ?case
    proof (cases n = 0)
      case True
      have {x. ∃ T. finite T ∧ T ⊆ S ∧ card T ≤ Suc n ∧ x ∈ convex hull T} = S
        unfolding set_eq_iff and mem_Collect_eq
      proof (rule, rule)
        fix x
        assume ∃ T. finite T ∧ T ⊆ S ∧ card T ≤ Suc n ∧ x ∈ convex hull T
        then obtain T where T: finite T T ⊆ S card T ≤ Suc n x ∈ convex hull
T
          by auto
        show x ∈ S
        proof (cases card T = 0)
          case True
          then show ?thesis
            using T(4) unfolding card_0_eq[OF T(1)] by simp
        next
          case False
          then have card T = Suc 0 using T(3) ‹n=0› by auto
          then obtain a where T = {a} unfolding card_Suc_eq by auto
          then show ?thesis using T(2,4) by simp
        qed
      next
        fix x assume x∈S
        then show ∃ T. finite T ∧ T ⊆ S ∧ card T ≤ Suc n ∧ x ∈ convex hull T
          by (rule_tac x={x} in exI) (use convex_hull_singleton in auto)
      qed
      then show ?thesis using assms by simp
    next
      case False
      have {x. ∃ T. finite T ∧ T ⊆ S ∧ card T ≤ Suc n ∧ x ∈ convex hull T} =
```

```
    {(1 − u) *R x + u *R y | x y u.
      0 ≤ u ∧ u ≤ 1 ∧ x ∈ S ∧ y ∈ {x. ∃ T. finite T ∧ T ⊆ S ∧ card T ≤ n
∧ x ∈ convex hull T}}
    unfolding set_eq_iff and mem_Collect_eq
  proof (rule, rule)
    fix x
    assume ∃ u v c. x = (1 − c) *R u + c *R v ∧
      0 ≤ c ∧ c ≤ 1 ∧ u ∈ S ∧ (∃ T. finite T ∧ T ⊆ S ∧ card T ≤ n ∧ v ∈
convex hull T)
    then obtain u v c T where obt: x = (1 − c) *R u + c *R v
      0 ≤ c ∧ c ≤ 1 u ∈ S finite T T ⊆ S card T ≤ n  v ∈ convex hull T
      by auto
    moreover have (1 − c) *R u + c *R v ∈ convex hull insert u T
    by (meson convexD_alt convex_convex_hull hull_inc hull_mono in_mono
insertCI obt(2) obt(7) subset_insertI)
    ultimately show ∃ T. finite T ∧ T ⊆ S ∧ card T ≤ Suc n ∧ x ∈ convex
hull T
      by (rule_tac x=insert u T in exI) (auto simp: card_insert_if)
  next
    fix x
    assume ∃ T. finite T ∧ T ⊆ S ∧ card T ≤ Suc n ∧ x ∈ convex hull T
    then obtain T where T: finite T T ⊆ S card T ≤ Suc n x ∈ convex hull
T
      by auto
    show ∃ u v c. x = (1 − c) *R u + c *R v ∧
      0 ≤ c ∧ c ≤ 1 ∧ u ∈ S ∧ (∃ T. finite T ∧ T ⊆ S ∧ card T ≤ n ∧ v ∈
convex hull T)
    proof (cases card T = Suc n)
      case False
      then have card T ≤ n using T(3) by auto
      then show ?thesis
        using ‹w∈S› and T
        by (rule_tac x=w in exI, rule_tac x=x in exI, rule_tac x=1 in exI)
auto
    next
      case True
      then obtain a u where au: T = insert a u a∉u
        by (metis card_le_Suc_iff order_refl)
      show ?thesis
      proof (cases u = {})
        case True
        then have x = a using T(4)[unfolded au] by auto
        show ?thesis unfolding ‹x = a›
          using T ‹n ≠ 0› unfolding au
          by (rule_tac x=a in exI, rule_tac x=a in exI, rule_tac x=1 in exI)
force
      next
        case False
        obtain ux vx b where obt: ux≥0 vx≥0 ux + vx = 1
```

```
            b ∈ convex hull u x = ux *R a + vx *R b
            using T(4)[unfolded au convex_hull_insert[OF False]]
            by auto
          have *: 1 − vx = ux using obt(3) by auto
          show ?thesis
           using obt T(1−3) card_insert_disjoint[OF _ au(2)] unfolding au *
            by (rule_tac x=a in exI, rule_tac x=b in exI, rule_tac x=vx in exI)
force
        qed
      qed
    qed
    then show ?thesis
      using compact_convex_combinations[OF assms Suc] by simp
  qed
 qed
qed
```

4.2.4 Extremal points of a simplex are some vertices

```
lemma dist_increases_online:
 fixes a b d :: 'a::real_inner
 assumes d ≠ 0
 shows dist a (b + d) > dist a b ∨ dist a (b − d) > dist a b
proof (cases inner a d − inner b d > 0)
 case True
 then have 0 < inner d d + (inner a d * 2 − inner b d * 2)
   using assms
   by (intro add_pos_pos) auto
 then show ?thesis
   unfolding dist_norm and norm_eq_sqrt_inner and real_sqrt_less_iff
   by (simp add: algebra_simps inner_commute)
next
 case False
 then have 0 < inner d d + (inner b d * 2 − inner a d * 2)
   using assms
   by (intro add_pos_nonneg) auto
 then show ?thesis
   unfolding dist_norm and norm_eq_sqrt_inner and real_sqrt_less_iff
   by (simp add: algebra_simps inner_commute)
qed

lemma norm_increases_online:
 fixes d :: 'a::real_inner
 shows d ≠ 0 ⟹ norm (a + d) > norm a ∨ norm(a − d) > norm a
 using dist_increases_online[of d a 0] unfolding dist_norm by auto

lemma simplex_furthest_lt:
 fixes S :: 'a::real_inner set
 assumes finite S
```

shows $\forall x \in$ *convex hull S*. $x \notin S \longrightarrow (\exists y \in$ *convex hull S*. *norm* $(x - a) <$ *norm*$(y - a))$
using *assms*
proof *induct*
fix *x S*
assume *as*: *finite S* $x \notin S$ $\forall x \in$*convex hull S*. $x \notin S \longrightarrow (\exists y \in$*convex hull S*. *norm* $(x - a) <$ *norm* $(y - a))$
show $\forall xa \in$*convex hull insert x S*. $xa \notin$ *insert x S* $\longrightarrow$
$(\exists y \in$*convex hull insert x S*. *norm* $(xa - a) <$ *norm* $(y - a))$
proof (*intro impI ballI*, *cases* $S = \{\}$)
case *False*
fix *y*
assume *y*: $y \in$ *convex hull insert x S* $y \notin$ *insert x S*
obtain *u v b* **where** *obt*: $u \geq 0$ $v \geq 0$ $u + v = 1$ $b \in$ *convex hull S* $y = u *_R x + v *_R b$
using *y*(*1*)[*unfolded convex_hull_insert*[*OF False*]] **by** *auto*
show $\exists z \in$*convex hull insert x S*. *norm* $(y - a) <$ *norm* $(z - a)$
proof (*cases* $y \in$ *convex hull S*)
case *True*
then obtain *z* **where** $z \in$ *convex hull S* *norm* $(y - a) <$ *norm* $(z - a)$
using *as*(*3*)[*THEN bspec*[**where** *x*=*y*]] **and** *y*(*2*) **by** *auto*
then show *?thesis*
by (*meson hull_mono subsetD subset_insertI*)
next
case *False*
show *?thesis*
proof (*cases* $u = 0 \vee v = 0$)
case *True*
with *False* **show** *?thesis*
using *obt y* **by** *auto*
next
case *False*
then obtain *w* **where** *w*: $w > 0$ $w < u$ $w < v$
using *field_lbound_gt_zero*[*of u v*] **and** *obt*(*1*,*2*) **by** *auto*
have $x \neq b$
proof
assume $x = b$
then have $y = b$ **unfolding** *obt*(*5*)
using *obt*(*3*) **by** (*auto simp*: *scaleR_left_distrib*[*symmetric*])
then show *False* **using** *obt*(*4*) **and** *False*
using ‹$x = b$› *y*(*2*) **by** *blast*
qed
then have $*$: $w *_R (x - b) \neq 0$ **using** *w*(*1*) **by** *auto*
show *?thesis*
using *dist_increases_online*[*OF* $*$, *of a y*]
proof (*elim disjE*)
assume *dist a y* $<$ *dist a* $(y + w *_R (x - b))$
then have *norm* $(y - a) <$ *norm* $((u + w) *_R x + (v - w) *_R b - a)$
unfolding *dist_commute*[*of a*]

```
        unfolding dist_norm obt(5)
        by (simp add: algebra_simps)
      moreover have (u + w) *R x + (v − w) *R b ∈ convex hull insert x S
        unfolding convex_hull_insert[OF ‹S≠{}›]
      proof (intro CollectI conjI exI)
        show u + w ≥ 0 v − w ≥ 0
          using obt(1) w by auto
      qed (use obt in auto)
      ultimately show ?thesis by auto
    next
      assume dist a y < dist a (y − w *R (x − b))
      then have norm (y − a) < norm ((u − w) *R x + (v + w) *R b − a)
        unfolding dist_commute[of a]
        unfolding dist_norm obt(5)
        by (simp add: algebra_simps)
      moreover have (u − w) *R x + (v + w) *R b ∈ convex hull insert x S
        unfolding convex_hull_insert[OF ‹S≠{}›]
      proof (intro CollectI conjI exI)
        show u − w ≥ 0 v + w ≥ 0
          using obt(1) w by auto
      qed (use obt in auto)
      ultimately show ?thesis by auto
    qed
  qed
 qed
qed auto
qed (auto simp: assms)

lemma simplex_furthest_le:
 fixes S :: 'a::real_inner set
 assumes finite S
   and S ≠ {}
 shows ∃ y∈S. ∀ x∈ convex hull S. norm (x − a) ≤ norm (y − a)
proof −
 have convex hull S ≠ {}
   using hull_subset[of S convex] and assms(2) by auto
 then obtain x where x: x ∈ convex hull S ∀ y∈convex hull S. norm (y − a) ≤
norm (x − a)
    using distance_attains_sup[OF finite_imp_compact_convex_hull[OF ‹finite
S›], of a]
   unfolding dist_commute[of a]
   unfolding dist_norm
   by auto
 show ?thesis
 proof (cases x ∈ S)
   case False
   then obtain y where y ∈ convex hull S norm (x − a) < norm (y − a)
    using simplex_furthest_lt[OF assms(1), THEN bspec[where x=x]] and x(1)
     by auto
```

```
    then show ?thesis
      using x(2)[THEN bspec[where x=y]] by auto
  next
    case True
    with x show ?thesis by auto
  qed
qed

lemma simplex_furthest_le_exists:
  fixes S :: ('a::real_inner) set
  shows finite S ⟹ ∀ x∈(convex hull S). ∃ y∈S. norm (x − a) ≤ norm (y − a)
  using simplex_furthest_le[of S] by (cases S = {}) auto

lemma simplex_extremal_le:
  fixes S :: 'a::real_inner set
  assumes finite S
    and S ≠ {}
  shows ∃ u∈S. ∃ v∈S. ∀ x∈convex hull S. ∀ y ∈ convex hull S. norm (x − y) ≤
norm (u − v)
proof −
  have convex hull S ≠ {}
    using hull_subset[of S convex] and assms(2) by auto
  then obtain u v where obt: u ∈ convex hull S v ∈ convex hull S
    ∀ x∈convex hull S. ∀ y∈convex hull S. norm (x − y) ≤ norm (u − v)
      using compact_sup_maxdistance[OF finite_imp_compact_convex_hull[OF
assms(1)]]
    by (auto simp: dist_norm)
  then show ?thesis
  proof (cases u∉S ∨ v∉S, elim disjE)
    assume u ∉ S
    then obtain y where y ∈ convex hull S norm (u − v) < norm (y − v)
        using simplex_furthest_lt[OF assms(1), THEN bspec[where x=u]] and
obt(1)
      by auto
    then show ?thesis
     using obt(3)[THEN bspec[where x=y], THEN bspec[where x=v]] and obt(2)
      by auto
  next
    assume v ∉ S
    then obtain y where y ∈ convex hull S norm (v − u) < norm (y − u)
        using simplex_furthest_lt[OF assms(1), THEN bspec[where x=v]] and
obt(2)
      by auto
    then show ?thesis
     using obt(3)[THEN bspec[where x=u], THEN bspec[where x=y]] and obt(1)
      by (auto simp: norm_minus_commute)
  qed auto
qed
```

lemma *simplex_extremal_le_exists*:
fixes $S :: 'a::real\_inner\ set$
shows *finite* $S \Longrightarrow x \in$ *convex hull* $S \Longrightarrow y \in$ *convex hull* $S \Longrightarrow$
$\exists u{\in}S.\ \exists v{\in}S.\ norm\ (x - y) \leq norm\ (u - v)$
using *convex_hull_empty simplex_extremal_le*[*of* S]
by(*cases* $S = \{\}$) *auto*

4.2.5 Closest point of a convex set is unique, with a continuous projection

definition *closest_point* :: $'a::\{real\_inner,heine\_borel\}\ set \Rightarrow 'a \Rightarrow 'a$
where *closest_point* $S\ a = (SOME\ x.\ x \in S \wedge (\forall\, y{\in}S.\ dist\ a\ x \leq dist\ a\ y))$

lemma *closest_point_exists*:
assumes *closed* S
and $S \neq \{\}$
shows *closest_point_in_set*: *closest_point* $S\ a \in S$
and $\forall\, y{\in}S.\ dist\ a\ (closest\_point\ S\ a) \leq dist\ a\ y$
unfolding *closest_point_def*
by (*rule_tac someI2_ex*, *auto intro*: *distance_attains_inf*[*OF assms*(1,2), *of a*])+

lemma *closest_point_le*: *closed* $S \Longrightarrow x \in S \Longrightarrow dist\ a\ (closest\_point\ S\ a) \leq dist\ a\ x$
using *closest_point_exists*[*of* S] **by** *auto*

lemma *closest_point_self*:
assumes $x \in S$
shows *closest_point* $S\ x = x$
unfolding *closest_point_def*
by (*rule some1_equality*, *rule ex1I*[*of* _ x]) (*use assms* **in** *auto*)

lemma *closest_point_refl*: *closed* $S \Longrightarrow S \neq \{\} \Longrightarrow closest\_point\ S\ x = x \longleftrightarrow x \in S$
using *closest_point_in_set*[*of* $S\ x$] *closest_point_self*[*of* $x\ S$]
by *auto*

lemma *closer_points_lemma*:
assumes *inner* $y\ z > 0$
shows $\exists\, u{>}0.\ \forall\, v{>}0.\ v \leq u \longrightarrow norm(v *_R z - y) < norm\ y$
proof −
have z: *inner* $z\ z > 0$
unfolding *inner_gt_zero_iff* **using** *assms* **by** *auto*
have $norm\ (v *_R z - y) < norm\ y$
if $0 < v$ **and** $v \leq inner\ y\ z\ /\ inner\ z\ z$ **for** v
unfolding *norm_lt* **using** z *assms that*
by (*simp add*: *field_simps inner_diff inner_commute mult_strict_left_mono*[*OF* _ ‹0<v›])
then show *?thesis*

using *assms z*
by (*rule_tac x = inner y z / inner z z* **in** *exI*) *auto*
qed

lemma *closer_point_lemma*:
assumes *inner* $(y - x)$ $(z - x) > 0$
shows $\exists u{>}0.\ u \leq 1 \wedge dist\ (x + u *_R (z - x))\ y < dist\ x\ y$
proof −
obtain *u* **where** $u > 0$
and *u*: $\bigwedge v.\ [\![0{<}v;\ v \leq u]\!] \Longrightarrow norm\ (v *_R (z - x) - (y - x)) < norm\ (y - x)$
using *closer_points_lemma*[*OF assms*] **by** *auto*
show *?thesis*
using *u*[*of min u 1*] **and** ‹$u > 0$›
by (*metis diff_diff_add dist_commute dist_norm less_eq_real_def not_less u zero_less_one*)
qed

lemma *any_closest_point_dot*:
assumes *convex S closed S* $x \in S$ $y \in S$ $\forall z{\in}S.\ dist\ a\ x \leq dist\ a\ z$
shows *inner* $(a - x)$ $(y - x) \leq 0$
proof (*rule ccontr*)
assume ¬ *?thesis*
then obtain *u* **where** *u*: $u{>}0$ $u{\leq}1$ $dist\ (x + u *_R (y - x))\ a < dist\ x\ a$
using *closer_point_lemma*[*of a x y*] **by** *auto*
let *?z* $= (1 - u) *_R x + u *_R y$
have $?z \in S$
using *convexD_alt*[*OF assms*(*1*,*3*,*4*), *of u*] **using** *u* **by** *auto*
then show *False*
using *assms*(*5*)[*THEN bspec*[**where** *x*=*?z*]] **and** *u*(*3*)
by (*auto simp*: *dist_commute algebra_simps*)
qed

lemma *any_closest_point_unique*:
fixes *x* :: *'a*::*real_inner*
assumes *convex S closed S* $x \in S$ $y \in S$
$\forall z{\in}S.\ dist\ a\ x \leq dist\ a\ z\ \forall z{\in}S.\ dist\ a\ y \leq dist\ a\ z$
shows $x = y$
using *any_closest_point_dot*[*OF assms*(*1*−*4*,*5*)] **and** *any_closest_point_dot*[*OF assms*(*1*−*2*,*4*,*3*,*6*)]
unfolding *norm_pths*(*1*) **and** *norm_le_square*
by (*auto simp*: *algebra_simps*)

lemma *closest_point_unique*:
assumes *convex S closed S* $x \in S$ $\forall z{\in}S.\ dist\ a\ x \leq dist\ a\ z$
shows $x =$ *closest_point S a*
using *any_closest_point_unique*[*OF assms*(*1*−*3*) _ *assms*(*4*), *of closest_point S a*]
using *closest_point_exists*[*OF assms*(*2*)] **and** *assms*(*3*) **by** *auto*

lemma *closest_point_dot*:
assumes *convex S closed S x* ∈ *S*
shows *inner* (*a* − *closest_point S a*) (*x* − *closest_point S a*) ≤ *0*
using *any_closest_point_dot*[*OF assms*(*1*,*2*) _ *assms*(*3*)]
by (*metis assms*(*2*) *assms*(*3*) *closest_point_in_set closest_point_le empty_iff*)

lemma *closest_point_lt*:
assumes *convex S closed S x* ∈ *S x* ≠ *closest_point S a*
shows *dist a* (*closest_point S a*) < *dist a x*
using *closest_point_unique*[**where** *a*=*a*] *closest_point_le*[**where** *a*=*a*] *assms*
by *fastforce*

lemma *setdist_closest_point*:
⟦*closed S*; *S* ≠ {}⟧ ⟹ *setdist* {*a*} *S* = *dist a* (*closest_point S a*)
by (*metis closest_point_exists*(*2*) *closest_point_in_set emptyE insert_iff setdist_unique*)

lemma *closest_point_lipschitz*:
assumes *convex S*
and *closed S S* ≠ {}
shows *dist* (*closest_point S x*) (*closest_point S y*) ≤ *dist x y*
proof −
have *inner* (*x* − *closest_point S x*) (*closest_point S y* − *closest_point S x*) ≤ *0*
and *inner* (*y* − *closest_point S y*) (*closest_point S x* − *closest_point S y*) ≤ *0*
by (*simp_all add*: *assms closest_point_dot closest_point_in_set*)
then show *?thesis* **unfolding** *dist_norm* **and** *norm_le*
using *inner_ge_zero*[*of* (*x* − *closest_point S x*) − (*y* − *closest_point S y*)]
by (*simp add*: *inner_add inner_diff inner_commute*)
qed

lemma *continuous_at_closest_point*:
assumes *convex S*
and *closed S*
and *S* ≠ {}
shows *continuous* (*at x*) (*closest_point S*)
unfolding *continuous_at_eps_delta*
using *le_less_trans*[*OF closest_point_lipschitz*[*OF assms*]] **by** *auto*

lemma *continuous_on_closest_point*:
assumes *convex S*
and *closed S*
and *S* ≠ {}
shows *continuous_on t* (*closest_point S*)
by (*metis continuous_at_imp_continuous_on continuous_at_closest_point*[*OF assms*])

proposition *closest_point_in_rel_interior*:
assumes *closed S S* ≠ {} **and** *x*: *x* ∈ *affine hull S*
shows *closest_point S x* ∈ *rel_interior S* ⟷ *x* ∈ *rel_interior S*

proof (*cases* $x \in S$)
case *True*
then show *?thesis*
by (*simp add*: *closest_point_self*)
next
case *False*
then have *False* **if** *asm*: *closest_point S x* $\in$ *rel_interior S*
proof −
obtain *e* **where** $e > 0$ **and** *clox*: *closest_point S x* $\in S$
and *e*: *cball* (*closest_point S x*) *e* $\cap$ *affine hull S* $\subseteq S$
using *asm mem_rel_interior_cball* **by** *blast*
then have *clo_notx*: *closest_point S x* $\neq x$
using ‹$x \notin S$› **by** *auto*
define *y* **where** $y \equiv$ *closest_point S x* −
(*min 1* (*e* / *norm*(*closest_point S x* − *x*))) $*_R$ (*closest_point S x* − *x*)
have $x - y = (1 - min\ 1\ (e\ /\ norm\ (closest\_point\ S\ x - x))) *_R (x - closest\_point\ S\ x)$
by (*simp add*: *y_def algebra_simps*)
then have *norm* (*x* − *y*) = *abs* ((*1* − *min 1* (*e* / *norm* (*closest_point S x* − *x*)))) ∗ *norm*(*x* − *closest_point S x*)
by *simp*
also have ... < *norm*(*x* − *closest_point S x*)
using *clo_notx* ‹*e* > *0*›
by (*auto simp*: *mult_less_cancel_right2 field_split_simps*)
finally have *no_less*: *norm* (*x* − *y*) < *norm* (*x* − *closest_point S x*) **.**
have *y* ∈ *affine hull S*
unfolding *y_def*
by (*meson affine_affine_hull clox hull_subset mem_affine_3_minus2 subsetD x*)
moreover have *dist* (*closest_point S x*) *y* ≤ *e*
using ‹*e* > *0*› **by** (*auto simp*: *y_def min_mult_distrib_right*)
ultimately have *y* ∈ *S*
using *subsetD* [*OF e*] **by** *simp*
then have *dist x* (*closest_point S x*) ≤ *dist x y*
by (*simp add*: *closest_point_le* ‹*closed S*›)
with *no_less* **show** *False*
by (*simp add*: *dist_norm*)
qed
moreover have *x* ∉ *rel_interior S*
using *rel_interior_subset False* **by** *blast*
ultimately show *?thesis* **by** *blast*
qed

Various point-to-set separating/supporting hyperplane theorems

lemma *supporting_hyperplane_closed_point*:
fixes *z* :: *'a*::{*real_inner*,*heine_borel*}
assumes *convex S*

```
    and closed S
    and S ≠ {}
    and z ∉ S
  shows ∃ a b. ∃ y∈S. inner a z < b ∧ inner a y = b ∧ (∀ x∈S. inner a x ≥ b)
proof −
  obtain y where y ∈ S and y: ∀ x∈S. dist z y ≤ dist z x
    by (metis distance_attains_inf[OF assms(2−3)])
  show ?thesis
  proof (intro exI bexI conjI ballI)
    show (y − z) · z < (y − z) · y
     by (metis ‹y ∈ S› assms(4) diff_gt_0_iff_gt inner_commute inner_diff_left
inner_gt_zero_iff right_minus_eq)
    show (y − z) · y ≤ (y − z) · x if x ∈ S for x
    proof (rule ccontr)
      have ∗: ⋀u. 0 ≤ u ∧ u ≤ 1 ⟶ dist z y ≤ dist z ((1 − u) ∗R y + u ∗R x)
       using assms(1)[unfolded convex_alt] and y and ‹x∈S› and ‹y∈S› by auto
      assume ¬ (y − z) · y ≤ (y − z) · x
      then obtain v where v > 0 v ≤ 1 dist (y + v ∗R (x − y)) z < dist y z
       using closer_point_lemma[of z y x] by (auto simp: inner_diff)
      then show False
       using ∗[of v] by (auto simp: dist_commute algebra_simps)
    qed
  qed (use ‹y ∈ S› in auto)
qed

lemma separating_hyperplane_closed_point:
  fixes z :: 'a::{real_inner,heine_borel}
  assumes convex S
    and closed S
    and z ∉ S
  shows ∃ a b. inner a z < b ∧ (∀ x∈S. inner a x > b)
proof (cases S = {})
  case True
  then show ?thesis
    by (simp add: gt_ex)
next
  case False
  obtain y where y ∈ S and y: ⋀x. x ∈ S ⟹ dist z y ≤ dist z x
    by (metis distance_attains_inf[OF assms(2) False])
  show ?thesis
  proof (intro exI conjI ballI)
    show (y − z) · z < inner (y − z) z + (norm (y − z))^2 / 2
      using ‹y∈S› ‹z∉S› by auto
  next
    fix x
    assume x ∈ S
    have False if ∗: 0 < inner (z − y) (x − y)
    proof −
      obtain u where u > 0 u ≤ 1 dist (y + u ∗R (x − y)) z < dist y z
```

```
      using * closer_point_lemma by blast
    then show False using y[of y + u *_R (x - y)] convexD_alt [OF ‹convex S›]
      using ‹x∈S› ‹y∈S› by (auto simp: dist_commute algebra_simps)
  qed
  moreover have 0 < (norm (y - z))^2
    using ‹y∈S› ‹z∉S› by auto
  then have 0 < inner (y - z) (y - z)
    unfolding power2_norm_eq_inner by simp
  ultimately show (y - z) · z + (norm (y - z))^2 / 2 < (y - z) · x
      by (force simp: field_simps power2_norm_eq_inner inner_commute inner_diff)
 qed
qed

lemma separating_hyperplane_closed_0:
 assumes convex (S::('a::euclidean_space) set)
   and closed S
   and 0 ∉ S
 shows ∃a b. a ≠ 0 ∧ 0 < b ∧ (∀x∈S. inner a x > b)
proof (cases S = {})
 case True
 have (SOME i. i∈Basis) ≠ (0::'a)
   by (metis Basis_zero SOME_Basis)
 then show ?thesis
   using True zero_less_one by blast
next
 case False
 then show ?thesis
   using False using separating_hyperplane_closed_point[OF assms]
  by (metis all_not_in_conv inner_zero_left inner_zero_right less_eq_real_def not_le)
qed
```

Now set-to-set for closed/compact sets

```
lemma separating_hyperplane_closed_compact:
 fixes S :: 'a::euclidean_space set
 assumes convex S
   and closed S
   and convex T
   and compact T
   and T ≠ {}
   and S ∩ T = {}
 shows ∃a b. (∀x∈S. inner a x < b) ∧ (∀x∈T. inner a x > b)
proof (cases S = {})
 case True
 obtain b where b: b > 0 ∀x∈T. norm x ≤ b
   using compact_imp_bounded[OF assms(4)] unfolding bounded_pos by auto
 obtain z :: 'a where z: norm z = b + 1
```

```
    using vector_choose_size[of b + 1] and b(1) by auto
  then have z ∉ T using b(2)[THEN bspec[where x=z]] by auto
  then obtain a b where ab: inner a z < b ∀x∈T. b < inner a x
   using separating_hyperplane_closed_point[OF assms(3) compact_imp_closed[OF
assms(4)], of z]
    by auto
  then show ?thesis
    using True by auto
next
  case False
  then obtain y where y ∈ S by auto
  obtain a b where 0 < b and §: ⋀x. x ∈ (⋃x∈ S. ⋃y ∈ T. {x − y}) ⟹ b <
inner a x
   using separating_hyperplane_closed_point[OF convex_differences[OF assms(1,3)],
of 0]
    using closed_compact_differences assms by fastforce
  have ab: b + inner a y < inner a x if x∈S y∈T for x y
    using § [of x−y] that by (auto simp add: inner_diff_right less_diff_eq)
  define k where k = (SUP x∈T. a • x)
  have k + b / 2 < a • x if x ∈ S for x
  proof −
    have k ≤ inner a x − b
      unfolding k_def
      using ‹T ≠ {}› ab that by (fastforce intro: cSUP_least)
    then show ?thesis
      using ‹0 < b› by auto
  qed
  moreover
  have − (k + b / 2) < − a • x if x ∈ T for x
  proof −
    have inner a x − b / 2 < k
      unfolding k_def
    proof (subst less_cSUP_iff)
      show T ≠ {} by fact
      show bdd_above ((•) a ‘ T)
        using ab[rule_format, of y] ‹y ∈ S›
        by (intro bdd_aboveI2[where M=inner a y − b]) (auto simp: field_simps
intro: less_imp_le)
      show ∃y∈T. a • x − b / 2 < a • y
        using ‹0 < b› that by force
    qed
    then show ?thesis
      by auto
  qed
  ultimately show ?thesis
    by (metis inner_minus_left neg_less_iff_less)
qed

lemma separating_hyperplane_compact_closed:
```

fixes $S :: 'a::euclidean\_space\ set$
assumes *convex* S
and *compact* S
and $S \neq \{\}$
and *convex* T
and *closed* T
and $S \cap T = \{\}$
shows $\exists\, a\ b.\ (\forall\, x{\in}S.\ inner\ a\ x < b) \wedge (\forall\, x{\in}T.\ inner\ a\ x > b)$
proof −
obtain $a\ b$ **where** $(\forall\, x{\in}T.\ inner\ a\ x < b) \wedge (\forall\, x{\in}S.\ b < inner\ a\ x)$
by (*metis disjoint_iff_not_equal separating_hyperplane_closed_compact assms*)
then show *?thesis*
by (*metis inner_minus_left neg_less_iff_less*)
qed

General case without assuming closure and getting non-strict separation

lemma *separating_hyperplane_set_0*:
assumes *convex* S $(0::'a::euclidean\_space) \notin S$
shows $\exists\, a.\ a \neq 0 \wedge (\forall\, x{\in}S.\ 0 \leq inner\ a\ x)$
proof −
let *?k* $= \lambda c.\ \{x::'a.\ 0 \leq inner\ c\ x\}$
have $*$: *frontier* (*cball 0 1*) $\cap \bigcap f \neq \{\}$ **if** *as*: $f \subseteq$ *?k* ' S *finite* f **for** f
proof −
obtain c **where** *c*: $f =$ *?k* ' c $c \subseteq S$ *finite* c
using *finite_subset_image*[*OF as*(*2*,*1*)] **by** *auto*
then obtain $a\ b$ **where** *ab*: $a \neq 0$ $0 < b$ $\forall\, x{\in}convex\ hull\ c.\ b < inner\ a\ x$
using *separating_hyperplane_closed_0*[*OF convex_convex_hull*, *of c*]
using *finite_imp_compact_convex_hull*[*OF c*(*3*), *THEN compact_imp_closed*]
and *assms*(*2*)
using *subset_hull*[*of convex*, *OF assms*(*1*), *symmetric*, *of c*]
by *force*
have *norm* $(a\ /_R\ norm\ a) = 1$
by (*simp add*: *ab*(*1*))
moreover have $(\forall\, y{\in}c.\ 0 \leq y \cdot (a\ /_R\ norm\ a))$
using *hull_subset*[*of c convex*] *ab* **by** (*force simp*: *inner_commute*)
ultimately have $\exists\, x.\ norm\ x = 1 \wedge (\forall\, y{\in}c.\ 0 \leq inner\ y\ x)$
by *blast*
then show *frontier* (*cball 0 1*) $\cap \bigcap f \neq \{\}$
unfolding *c*(*1*) *frontier_cball sphere_def dist_norm* **by** *auto*
qed
have *frontier* (*cball 0 1*) $\cap$ $(\bigcap$ (*?k* ' S)) $\neq \{\}$
by (*rule compact_imp_fip*) (*use* $*$ *closed_halfspace_ge* **in** *auto*)
then obtain x **where** *norm* $x = 1$ $\forall\, y{\in}S.\ x{\in}$*?k* y
unfolding *frontier_cball dist_norm sphere_def* **by** *auto*
then show *?thesis*
by (*metis inner_commute mem_Collect_eq norm_eq_zero zero_neq_one*)
qed

lemma *separating_hyperplane_sets*:
fixes S T :: *'a::euclidean_space set*
assumes *convex* S
and *convex* T
and $S \neq \{\}$
and $T \neq \{\}$
and $S \cap T = \{\}$
shows $\exists\, a\ b.\ a \neq 0 \wedge (\forall x{\in}S.\ \mathit{inner}\ a\ x \leq b) \wedge (\forall x{\in}T.\ \mathit{inner}\ a\ x \geq b)$
proof −
from *separating_hyperplane_set_0*[*OF convex_differences*[*OF assms*(2,1)]]
obtain a **where** $a \neq 0\ \forall x{\in}\{x - y\ |x\ y.\ x \in T \wedge y \in S\}.\ 0 \leq \mathit{inner}\ a\ x$
using *assms*(3−5) **by** *force*
then have $*$: $\bigwedge x\ y.\ x \in T \Longrightarrow y \in S \Longrightarrow \mathit{inner}\ a\ y \leq \mathit{inner}\ a\ x$
by (*force simp*: *inner_diff*)
then have *bdd*: *bdd_above* (((·) *a*) ‘*S*)
using ‹$T \neq \{\}$› **by** (*auto intro*: *bdd_aboveI2*[*OF* $*$])
show *?thesis*
using ‹$a{\neq}0$›
by (*intro exI*[*of* _ *a*] *exI*[*of* _ *SUP* $x{\in}S.\ a \cdot x$])
(*auto intro*!: *cSUP_upper bdd cSUP_least* ‹$a \neq 0$› ‹$S \neq \{\}$› $*$)
qed

4.2.6 More convexity generalities

lemma *convex_closure* [*intro*,*simp*]:
fixes S :: *'a::real_normed_vector set*
assumes *convex* S
shows *convex* (*closure* S)
apply (*rule convexI*)
unfolding *closure_sequential*
apply (*elim exE*)
subgoal for $x\ y\ u\ v\ f\ g$
by (*rule_tac* $x{=}\lambda n.\ u *_R f\ n + v *_R g\ n$ **in** *exI*) (*force intro*: *tendsto_intros dest*: *convexD* [*OF assms*])
done

lemma *convex_interior* [*intro*,*simp*]:
fixes S :: *'a::real_normed_vector set*
assumes *convex* S
shows *convex* (*interior* S)
unfolding *convex_alt Ball_def mem_interior*
proof *clarify*
fix $x\ y\ u$
assume *u*: $0 \leq u\ u \leq (1{::}\mathit{real})$
fix $e\ d$
assume *ed*: *ball* $x\ e \subseteq S$ *ball* $y\ d \subseteq S\ 0{<}d\ 0{<}e$
show $\exists\, e{>}0.\ \mathit{ball}\ ((1 - u) *_R x + u *_R y)\ e \subseteq S$
proof (*intro exI conjI subsetI*)

```
    fix z
    assume z: z ∈ ball ((1 − u) *R x + u *R y) (min d e)
    have (1− u) *R (z − u *R (y − x)) + u *R (z + (1 − u) *R (y − x)) ∈ S
    proof (rule_tac assms[unfolded convex_alt, rule_format])
      show z − u *R (y − x) ∈ S z + (1 − u) *R (y − x) ∈ S
        using ed z u by (auto simp add: algebra_simps dist_norm)
    qed (use u in auto)
    then show z ∈ S
      using u by (auto simp: algebra_simps)
  qed(use u ed in auto)
qed

lemma convex_hull_eq_empty[simp]: convex hull S = {} ⟷ S = {}
  using hull_subset[of S convex] convex_hull_empty by auto
```

4.2.7 Convex set as intersection of halfspaces

```
lemma convex_halfspace_intersection:
  fixes S :: ('a::euclidean_space) set
  assumes closed S convex S
  shows S = ⋂{h. S ⊆ h ∧ (∃ a b. h = {x. inner a x ≤ b})}
proof −
  { fix z
    assume ∀ T. S ⊆ T ∧ (∃ a b. T = {x. inner a x ≤ b}) ⟶ z ∈ T  z ∉ S
    then have §: ⋀a b. S ⊆ {x. inner a x ≤ b} ⟹ z ∈ {x. inner a x ≤ b}
      by blast
    obtain a b where inner a z < b (∀ x∈S. inner a x > b)
      using ‹z ∉ S› assms separating_hyperplane_closed_point by blast
    then have False
      using § [of −a −b] by fastforce
  }
  then show ?thesis
    by force
qed
```

4.2.8 Convexity of general and special intervals

```
lemma is_interval_convex:
  fixes S :: 'a::euclidean_space set
  assumes is_interval S
  shows convex S
proof (rule convexI)
  fix x y and u v :: real
  assume x ∈ S y ∈ S and uv: 0 ≤ u 0 ≤ v u + v = 1
  then have ∗: u = 1 − v 1 − v ≥ 0 and ∗∗: v = 1 − u 1 − u ≥ 0
    by auto
  {
    fix a b
    assume ¬ b ≤ u ∗ a + v ∗ b
    then have u ∗ a < (1 − v) ∗ b
```

```
    unfolding not_le using ‹0 ≤ v›by (auto simp: field_simps)
   then have a < b
    using ∗(1) less_eq_real_def uv(1) by auto
   then have a ≤ u ∗ a + v ∗ b
    unfolding ∗ using ‹0 ≤ v›
    by (auto simp: field_simps intro!:mult_right_mono)
  }
  moreover
  {
   fix a b
   assume ¬ u ∗ a + v ∗ b ≤ a
   then have v ∗ b > (1 − u) ∗ a
    unfolding not_le using ‹0 ≤ v› by (auto simp: field_simps)
   then have a < b
    unfolding ∗ using ‹0 ≤ v›
    by (rule_tac mult_left_less_imp_less) (auto simp: field_simps)
   then have u ∗ a + v ∗ b ≤ b
    unfolding ∗∗
    using ∗∗(2) ‹0 ≤ u› by (auto simp: algebra_simps mult_right_mono)
  }
  ultimately show u ∗R x + v ∗R y ∈ S
   using DIM_positive[where 'a='a]
    by (intro mem_is_intervalI [OF assms ‹x ∈ S› ‹y ∈ S›]) (auto simp: in-
ner_simps)
qed

lemma is_interval_connected:
  fixes S :: 'a::euclidean_space set
  shows is_interval S ⟹ connected S
  using is_interval_convex convex_connected by auto

lemma convex_box [simp]: convex (cbox a b) convex (box a (b::'a::euclidean_space))
  by (auto simp add: is_interval_convex)
```

A non-singleton connected set is perfect (i.e. has no isolated points).

```
lemma connected_imp_perfect:
  fixes a :: 'a::metric_space
  assumes connected S a ∈ S and S: ⋀x. S ≠ {x}
  shows a islimpt S
proof −
  have False if a ∈ T open T ⋀y. ⟦y ∈ S; y ∈ T⟧ ⟹ y = a for T
  proof −
   obtain e where e > 0 and e: cball a e ⊆ T
    using ‹open T› ‹a ∈ T› by (auto simp: open_contains_cball)
   have openin (top_of_set S) {a}
    unfolding openin_open using that ‹a ∈ S› by blast
   moreover have closedin (top_of_set S) {a}
    by (simp add: assms)
   ultimately show False
```

using ‹*connected S*› *connected_clopen S* **by** *blast*
qed
then show *?thesis*
unfolding *islimpt_def* **by** *blast*
qed

lemma *islimpt_Ioc* [*simp*]:
fixes *a* :: *real*
assumes *a*<*b*
shows *x islimpt* {*a*<..*b*} ⟷ *x* ∈ {*a*..*b*} (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⟹ *?rhs*
by (*metis assms closed_atLeastAtMost closed_limpt closure_greaterThanAtMost closure_subset islimpt_subset*)
next
assume *?rhs*
then have *x* ∈ *closure* {*a*<..<*b*}
using *assms closure_greaterThanLessThan* **by** *blast*
then show *?lhs*
by (*metis* (*no_types*) *Diff_empty Diff_insert0 interior_lessThanAtMost interior_limit_point interior_subset islimpt_in_closure islimpt_subset*)
qed

lemma *islimpt_Ico* [*simp*]:
fixes *a* :: *real*
assumes *a*<*b* **shows** *x islimpt* {*a*..<*b*} ⟷ *x* ∈ {*a*..*b*}
by (*metis assms closure_atLeastLessThan closure_greaterThanAtMost islimpt_Ioc limpt_of_closure*)

lemma *islimpt_Icc* [*simp*]:
fixes *a* :: *real*
assumes *a*<*b* **shows** *x islimpt* {*a*..*b*} ⟷ *x* ∈ {*a*..*b*}
by (*metis assms closure_atLeastLessThan islimpt_Ico limpt_of_closure*)

lemma *connected_imp_perfect_aff_dim*:
⟦*connected S*; *aff_dim S* ≠ *0*; *a* ∈ *S*⟧ ⟹ *a islimpt S*
using *aff_dim_sing connected_imp_perfect* **by** *blast*

4.2.9 On *real*, *is_interval*, *convex* and *connected* are all equivalent

lemma *mem_is_interval_1_I*:
fixes *a b c*::*real*
assumes *is_interval S*
assumes *a* ∈ *S c* ∈ *S*
assumes *a* ≤ *b b* ≤ *c*
shows *b* ∈ *S*
using *assms is_interval_1* **by** *blast*

lemma *is_interval_connected_1*:
fixes S :: *real set*
shows *is_interval* $S \longleftrightarrow$ *connected* S
by (*meson connected_iff_interval is_interval_1*)

lemma *is_interval_convex_1*:
fixes S :: *real set*
shows *is_interval* $S \longleftrightarrow$ *convex* S
by (*metis is_interval_convex convex_connected is_interval_connected_1*)

lemma *connected_compact_interval_1*:
connected $S \wedge$ *compact* $S \longleftrightarrow (\exists\, a\ b.\ S = \{a..b::real\})$
by (*auto simp*: *is_interval_connected_1* [*symmetric*] *is_interval_compact*)

lemma *connected_convex_1*:
fixes S :: *real set*
shows *connected* $S \longleftrightarrow$ *convex* S
by (*metis is_interval_convex convex_connected is_interval_connected_1*)

lemma *connected_space_iff_is_interval_1* [*iff*]:
fixes S :: *real set*
shows *connected_space* (*top_of_set* S) $\longleftrightarrow$ *is_interval* S
using *connectedin_topspace is_interval_connected_1* **by** *force*

lemma *connected_convex_1_gen*:
fixes S :: $'a$ :: *euclidean_space set*
assumes $DIM('a) = 1$
shows *connected* $S \longleftrightarrow$ *convex* S
proof −
obtain f:: $'a \Rightarrow$ *real* **where** *linf*: *linear* f **and** *inj* f
using *subspace_isomorphism*[*OF subspace_UNIV subspace_UNIV*,
where $'a='a$ **and** $'b=real$]
unfolding *Euclidean_Space.dim_UNIV*
by (*auto simp*: *assms*)
then have $f -` (f ` S) = S$
by (*simp add*: *inj_vimage_image_eq*)
then show *?thesis*
by (*metis connected_convex_1 convex_linear_vimage linf convex_connected connected_linear_image*)
qed

lemma [*simp*]:
fixes $r\ s$::*real*
shows *is_interval_io*: *is_interval* $\{..<r\}$
and *is_interval_oi*: *is_interval* $\{r<..\}$
and *is_interval_oo*: *is_interval* $\{r<..<s\}$
and *is_interval_oc*: *is_interval* $\{r<..s\}$
and *is_interval_co*: *is_interval* $\{r..<s\}$
by (*simp_all add*: *is_interval_convex_1*)

4.2.10 Another intermediate value theorem formulation

lemma *ivt_increasing_component_on_1*:
fixes $f :: real \Rightarrow 'a::euclidean\_space$
assumes $a \leq b$
and *continuous_on* $\{a..b\}$ f
and $(f\ a)\cdot k \leq y$ $y \leq (f\ b)\cdot k$
shows $\exists x{\in}\{a..b\}.\ (f\ x)\cdot k = y$
proof −
have $f\ a \in f\ `\ cbox\ a\ b$ $f\ b \in f\ `\ cbox\ a\ b$
using $\langle a \leq b \rangle$ **by** *auto*
then show *?thesis*
using *connected_ivt_component*[*of* $f\ `\ cbox\ a\ b$ $f\ a$ $f\ b$ k y]
by (*simp add*: *connected_continuous_image assms*)
qed

lemma *ivt_increasing_component_1*:
fixes $f :: real \Rightarrow 'a::euclidean\_space$
shows $a \leq b \Longrightarrow \forall x{\in}\{a..b\}.\ continuous\ (at\ x)\ f \Longrightarrow$
$f\ a\cdot k \leq y \Longrightarrow y \leq f\ b\cdot k \Longrightarrow \exists x{\in}\{a..b\}.\ (f\ x)\cdot k = y$
by (*rule ivt_increasing_component_on_1*) (*auto simp*: *continuous_at_imp_continuous_on*)

lemma *ivt_decreasing_component_on_1*:
fixes $f :: real \Rightarrow 'a::euclidean\_space$
assumes $a \leq b$
and *continuous_on* $\{a..b\}$ f
and $(f\ b)\cdot k \leq y$
and $y \leq (f\ a)\cdot k$
shows $\exists x{\in}\{a..b\}.\ (f\ x)\cdot k = y$
using *ivt_increasing_component_on_1*[*of* a b $\lambda x.\ -f\ x\ k - y$] *neg_equal_iff_equal*
using *assms continuous_on_minus* **by** *force*

lemma *ivt_decreasing_component_1*:
fixes $f :: real \Rightarrow 'a::euclidean\_space$
shows $a \leq b \Longrightarrow \forall x{\in}\{a..b\}.\ continuous\ (at\ x)\ f \Longrightarrow$
$f\ b\cdot k \leq y \Longrightarrow y \leq f\ a\cdot k \Longrightarrow \exists x{\in}\{a..b\}.\ (f\ x)\cdot k = y$
by (*rule ivt_decreasing_component_on_1*) (*auto simp*: *continuous_at_imp_continuous_on*)

4.2.11 A bound within an interval

lemma *convex_hull_eq_real_cbox*:
fixes $x\ y :: real$ **assumes** $x \leq y$
shows *convex hull* $\{x, y\} = cbox\ x\ y$
proof (*rule hull_unique*)
show $\{x, y\} \subseteq cbox\ x\ y$ **using** $\langle x \leq y \rangle$ **by** *auto*
show *convex* (*cbox x y*)
by (*rule convex_box*)
next
fix S **assume** $\{x, y\} \subseteq S$ **and** *convex* S
then show $cbox\ x\ y \subseteq S$

```
    unfolding is_interval_convex_1 [symmetric] is_interval_def Basis_real_def
    by − (clarify, simp (no_asm_use), fast)
qed

lemma unit_interval_convex_hull:
  cbox (0::'a::euclidean_space) One = convex hull {x. ∀i∈Basis. (x·i = 0) ∨ (x·i
= 1)}
  (is ?int = convex hull ?points)
proof −
  have One[simp]: ⋀i. i ∈ Basis ⟹ One · i = 1
    by (simp add: inner_sum_left sum.If_cases inner_Basis)
  have ?int = {x. ∀i∈Basis. x · i ∈ cbox 0 1}
    by (auto simp: cbox_def)
  also have ... = (∑ i∈Basis. (λx. x *R i) ` cbox 0 1)
    by (simp only: box_eq_set_sum_Basis)
  also have ... = (∑ i∈Basis. (λx. x *R i) ` (convex hull {0, 1}))
    by (simp only: convex_hull_eq_real_cbox zero_le_one)
  also have ... = (∑ i∈Basis. convex hull ((λx. x *R i) ` {0, 1}))
    by (simp add: convex_hull_linear_image)
  also have ... = convex hull (∑ i∈Basis. (λx. x *R i) ` {0, 1})
    by (simp only: convex_hull_set_sum)
  also have ... = convex hull {x. ∀i∈Basis. x·i ∈ {0, 1}}
    by (simp only: box_eq_set_sum_Basis)
  also have convex hull {x. ∀i∈Basis. x·i ∈ {0, 1}} = convex hull ?points
    by simp
  finally show ?thesis .
qed
```

And this is a finite set of vertices.

```
lemma unit_cube_convex_hull:
  obtains S :: 'a::euclidean_space set
  where finite S and cbox 0 (∑ Basis) = convex hull S
proof
  show finite {x::'a. ∀i∈Basis. x · i = 0 ∨ x · i = 1}
  proof (rule finite_subset, clarify)
    show finite ((λS. ∑ i∈Basis. (if i ∈ S then 1 else 0) *R i) ` Pow Basis)
      using finite_Basis by blast
    fix x :: 'a
    assume x: ∀i∈Basis. x · i = 0 ∨ x · i = 1
    show x ∈ (λS. ∑ i∈Basis. (if i∈S then 1 else 0) *R i) ` Pow Basis
      apply (rule image_eqI[where x={i. i ∈ Basis ∧ x·i = 1}])
      using x
      by (subst euclidean_eq_iff, auto)
  qed
  show cbox 0 One = convex hull {x. ∀i∈Basis. x · i = 0 ∨ x · i = 1}
    using unit_interval_convex_hull by blast
qed
```

Hence any cube (could do any nonempty interval).

lemma *cube_convex_hull*:
assumes $d > 0$
obtains $S :: 'a::euclidean\_space\ set$ **where**
finite S **and** $cbox\ (x - (\sum i{\in}Basis.\ d{*_R}i))\ (x + (\sum i{\in}Basis.\ d{*_R}i)) = convex\ hull\ S$
proof −
let $?d = (\sum i{\in}Basis.\ d *_R i)::'a$
have $*$: $cbox\ (x - ?d)\ (x + ?d) = (\lambda y.\ x - ?d + (2 * d) *_R y)\ `\ cbox\ 0\ (\sum Basis)$
proof (*intro set_eqI iffI*)
fix y
assume $y \in cbox\ (x - ?d)\ (x + ?d)$
then have $inverse\ (2 * d) *_R (y - (x - ?d)) \in cbox\ 0\ (\sum Basis)$
using *assms* **by** (*simp add*: *mem_box inner_simps*) (*simp add*: *field_simps*)
with $\langle 0 < d \rangle$ **show** $y \in (\lambda y.\ x - sum\ ((*_R)\ d)\ Basis + (2 * d) *_R y)\ `\ cbox\ 0\ One$
by (*auto intro*: *image_eqI*[**where** $x{=}\ inverse\ (2 * d) *_R (y - (x - ?d))$])
next
fix y
assume $y \in (\lambda y.\ x - ?d + (2 * d) *_R y)\ `\ cbox\ 0\ One$
then obtain z **where** z: $z \in cbox\ 0\ One\ y = x - ?d + (2{*}d) *_R z$
by *auto*
then show $y \in cbox\ (x - ?d)\ (x + ?d)$
using z *assms* **by** (*auto simp*: *mem_box inner_simps*)
qed
obtain S **where** *finite* S $cbox\ 0\ (\sum Basis::'a) = convex\ hull\ S$
using *unit_cube_convex_hull* **by** *auto*
then show *?thesis*
by (*rule_tac that*[*of* $(\lambda y.\ x - ?d + (2 * d) *_R y)`\ S$]) (*auto simp*: *convex_hull_affinity* $*$)
qed

4.2.12 Representation of any interval as a finite convex hull

lemma *image_stretch_interval*:
$(\lambda x.\ \sum k{\in}Basis.\ (m\ k * (x{\cdot}k)) *_R k)\ `\ cbox\ a\ (b::'a::euclidean\_space) =$
(*if* $(cbox\ a\ b) = \{\}$ *then* $\{\}$ *else*
$cbox\ (\sum k{\in}Basis.\ (min\ (m\ k * (a{\cdot}k))\ (m\ k * (b{\cdot}k))) *_R k::'a)$
$(\sum k{\in}Basis.\ (max\ (m\ k * (a{\cdot}k))\ (m\ k * (b{\cdot}k))) *_R k))$
proof *cases*
assume $*$: $cbox\ a\ b \neq \{\}$
show *?thesis*
unfolding *box_ne_empty if_not_P*[*OF* $*$]
apply (*simp add*: *cbox_def image_Collect set_eq_iff euclidean_eq_iff*[**where** $'a='a$] *ball_conj_distrib*[*symmetric*])
apply (*subst choice_Basis_iff*[*symmetric*])
proof (*intro allI ball_cong refl*)
fix $x\ i :: 'a$ **assume** $i \in Basis$
with $*$ **have** a_le_b: $a \cdot i \leq b \cdot i$
unfolding *box_ne_empty* **by** *auto*

```
    show (∃xa. x · i = m i * xa ∧ a · i ≤ xa ∧ xa ≤ b · i) ⟷
        min (m i * (a · i)) (m i * (b · i)) ≤ x · i ∧ x · i ≤ max (m i * (a · i)) (m
i * (b · i))
    proof (cases m i = 0)
      case True
      with a_le_b show ?thesis by auto
    next
      case False
      then have *: ⋀a b. a = m i * b ⟷ b = a / m i
        by (auto simp: field_simps)
      from False have
          min (m i * (a · i)) (m i * (b · i)) = (if 0 < m i then m i * (a · i) else m
i * (b · i))
          max (m i * (a · i)) (m i * (b · i)) = (if 0 < m i then m i * (b · i) else m
i * (a · i))
        using a_le_b by (auto simp: min_def max_def mult_le_cancel_left)
      with False show ?thesis using a_le_b *
        by (simp add: le_divide_eq divide_le_eq) (simp add: ac_simps)
    qed
  qed
qed simp

lemma interval_image_stretch_interval:
  ∃u v. (λx. ∑k∈Basis. (m k * (x·k))*R k) ' cbox a (b::'a::euclidean_space) =
cbox u (v::'a::euclidean_space)
  unfolding image_stretch_interval by auto

lemma cbox_translation: cbox (c + a) (c + b) = image (λx. c + x) (cbox a b)
  using image_affinity_cbox [of 1 c a b]
  using box_ne_empty [of a+c b+c]  box_ne_empty [of a b]
  by (auto simp: inner_left_distrib add.commute)

lemma cbox_image_unit_interval:
  fixes a :: 'a::euclidean_space
  assumes cbox a b ≠ {}
    shows cbox a b =
          (+) a ' (λx. ∑k∈Basis. ((b · k − a · k) * (x · k)) *R k) ' cbox 0 One
using assms
apply (simp add: box_ne_empty image_stretch_interval cbox_translation [symmetric])
apply (simp add: min_def max_def algebra_simps sum_subtractf euclidean_representation)
done

lemma closed_interval_as_convex_hull:
  fixes a :: 'a::euclidean_space
  obtains S where finite S cbox a b = convex hull S
proof (cases cbox a b = {})
  case True with convex_hull_empty that show ?thesis
    by blast
next
```

case *False*
obtain *S*::*'a set* **where** *finite S* **and** *eq*: *cbox 0 One = convex hull S*
by (*blast intro*: *unit_cube_convex_hull*)
let *?S* = ((+) *a* ‘ (λ*x*. ∑ *k*∈*Basis*. ((*b* · *k* − *a* · *k*) ∗ (*x* · *k*)) ∗$_R$ *k*) ‘ *S*)
show *thesis*
proof
show *finite ?S*
by (*simp add*: ‹*finite S*›)
have *lin*: *linear* (λ*x*. ∑ *k*∈*Basis*. ((*b* · *k* − *a* · *k*) ∗ (*x* · *k*)) ∗$_R$ *k*)
by (*rule linear_compose_sum*) (*auto simp*: *algebra_simps linearI*)
show *cbox a b = convex hull ?S*
using *convex_hull_linear_image* [*OF lin*]
by (*simp add*: *convex_hull_translation eq cbox_image_unit_interval* [*OF False*])
qed
qed

4.2.13 Bounded convex function on open set is continuous

lemma *convex_on_bounded_continuous*:
fixes *S* :: (*'a*::*real_normed_vector*) *set*
assumes *open S*
and *convex_on S f*
and ∀ *x*∈*S*. |*f x*| ≤ *b*
shows *continuous_on S f*
proof −
have ∃ *d*>*0*. ∀ *x'*. *norm* (*x'* − *x*) < *d* ⟶ |*f x'* − *f x*| < *e* **if** *x* ∈ *S e* > *0* **for** *x* **and** *e* :: *real*
proof −
define *B* **where** *B* = |*b*| + *1*
then have *B*: *0* < *B*⋀*x*. *x*∈*S* ⟹ |*f x*| ≤ *B*
using *assms*(*3*) **by** *auto*
obtain *k* **where** *k* > *0* **and** *k*: *cball x k* ⊆ *S*
using ‹*x* ∈ *S*› *assms*(*1*) *open_contains_cball_eq* **by** *blast*
show ∃ *d*>*0*. ∀ *x'*. *norm* (*x'* − *x*) < *d* ⟶ |*f x'* − *f x*| < *e*
proof (*intro exI conjI allI impI*)
fix *y*
assume *as*: *norm* (*y* − *x*) < *min* (*k* / *2*) (*e* / (*2* ∗ *B*) ∗ *k*)
show |*f y* − *f x*| < *e*
proof (*cases y* = *x*)
case *False*
define *t* **where** *t* = *k* / *norm* (*y* − *x*)
have *2* < *t* *0*<*t*
unfolding *t_def* **using** *as False* **and** ‹*k*>*0*›
by (*auto simp*:*field_simps*)
have *y* ∈ *S*
apply (*rule k*[*THEN subsetD*])
unfolding *mem_cball dist_norm*
apply (*rule order_trans*[*of* _ *2* ∗ *norm* (*x* − *y*)])

```
        using as
        by (auto simp: field_simps norm_minus_commute)
      {
        define w where w = x + t *R (y - x)
        have w ∈ S
          using ‹k>0› by (auto simp: dist_norm t_def w_def k[THEN subsetD])
        have (1 / t) *R x + - x + ((t - 1) / t) *R x = (1 / t - 1 + (t - 1) /
t) *R x
          by (auto simp: algebra_simps)
        also have ... = 0
          using ‹t > 0› by (auto simp:field_simps)
        finally have w: (1 / t) *R w + ((t - 1) / t) *R x = y
          unfolding w_def using False and ‹t > 0›
          by (auto simp: algebra_simps)
        have 2: 2 * B < e * t
          unfolding t_def using ‹0 < e› ‹0 < k› ‹B > 0› and as and False
          by (auto simp:field_simps)
        have f y - f x ≤ (f w - f x) / t
          using assms(2)[unfolded convex_on_def,rule_format,of w x 1/t (t -
1)/t, unfolded w]
          using ‹0 < t› ‹2 < t› and ‹x ∈ S› ‹w ∈ S›
          by (auto simp:field_simps)
        also have ... < e
          using B(2)[OF ‹w∈S›] and B(2)[OF ‹x∈S›] 2 ‹t > 0› by (auto simp:
field_simps)
        finally have th1: f y - f x < e .
      }
      moreover
      {
        define w where w = x - t *R (y - x)
        have w ∈ S
         using ‹k > 0› by (auto simp: dist_norm t_def w_def k[THEN subsetD])
        have (1 / (1 + t)) *R x + (t / (1 + t)) *R x = (1 / (1 + t) + t / (1 +
t)) *R x
          by (auto simp: algebra_simps)
        also have ... = x
          using ‹t > 0› by (auto simp:field_simps)
        finally have w: (1 / (1+t)) *R w + (t / (1 + t)) *R y = x
          unfolding w_def using False and ‹t > 0›
          by (auto simp: algebra_simps)
        have 2 * B < e * t
          unfolding t_def
          using ‹0 < e› ‹0 < k› ‹B > 0› and as and False
          by (auto simp:field_simps)
        then have *: (f w - f y) / t < e
          using B(2)[OF ‹w∈S›] and B(2)[OF ‹y∈S›]
          using ‹t > 0›
          by (auto simp:field_simps)
        have f x ≤ 1 / (1 + t) * f w + (t / (1 + t)) * f y
```

```
          using assms(2)[unfolded convex_on_def,rule_format,of w y 1/(1+t) t
/ (1+t),unfolded w]
          using ‹0 < t› ‹2 < t› and ‹y ∈ S› ‹w ∈ S›
          by (auto simp:field_simps)
        also have ... = (f w + t * f y) / (1 + t)
          using ‹t > 0› by (simp add: add_divide_distrib)
        also have ... < e + f y
          using ‹t > 0› * ‹e > 0› by (auto simp: field_simps)
        finally have f x - f y < e by auto
      }
      ultimately show ?thesis by auto
    qed (use ‹0<e› in auto)
  qed (use ‹0<e› ‹0<k› ‹0<B› in ‹auto simp: field_simps›)
 qed
 then show ?thesis
  by (metis continuous_on_iff dist_norm real_norm_def)
qed
```

4.2.14 Upper bound on a ball implies upper and lower bounds

```
lemma convex_bounds_lemma:
 fixes x :: 'a::real_normed_vector
 assumes convex_on (cball x e) f
  and ∀ y ∈ cball x e. f y ≤ b and y: y ∈ cball x e
 shows |f y| ≤ b + 2 * |f x|
proof (cases 0 ≤ e)
 case True
 define z where z = 2 *R x - y
 have *: x - (2 *R x - y) = y - x
  by (simp add: scaleR_2)
 have z: z ∈ cball x e
  using y unfolding z_def mem_cball dist_norm * by (auto simp: norm_minus_commute)
 have (1 / 2) *R y + (1 / 2) *R z = x
  unfolding z_def by (auto simp: algebra_simps)
 then show |f y| ≤ b + 2 * |f x|
  using assms(1)[unfolded convex_on_def,rule_format, OF y z, of 1/2 1/2]
  using assms(2)[rule_format,OF y] assms(2)[rule_format,OF z]
  by (auto simp:field_simps)
next
 case False
 have dist x y < 0
  using False y unfolding mem_cball not_le by (auto simp del: dist_not_less_zero)
 then show |f y| ≤ b + 2 * |f x|
  using zero_le_dist[of x y] by auto
qed
```

Hence a convex function on an open set is continuous

```
lemma real_of_nat_ge_one_iff: 1 ≤ real (n::nat) ⟷ 1 ≤ n
 by auto
```

```
lemma convex_on_continuous:
  assumes open (s::('a::euclidean_space) set) convex_on s f
  shows continuous_on s f
  unfolding continuous_on_eq_continuous_at[OF assms(1)]
proof
  note dimge1 = DIM_positive[where 'a='a]
  fix x
  assume x ∈ s
  then obtain e where e: cball x e ⊆ s e > 0
    using assms(1) unfolding open_contains_cball by auto
  define d where d = e / real DIM('a)
  have 0 < d
    unfolding d_def using ‹e > 0› dimge1 by auto
  let ?d = (∑ i∈Basis. d *R i)::'a
  obtain c
    where c: finite c
    and c1: convex hull c ⊆ cball x e
    and c2: cball x d ⊆ convex hull c
  proof
    define c where c = (∑ i∈Basis. (λa. a *R i) ` {x•i - d, x•i + d})
    show finite c
      unfolding c_def by (simp add: finite_set_sum)
    have ⋀i. i ∈ Basis ⟹ convex hull {x • i - d, x • i + d} = cbox (x • i - d)
(x • i + d)
      using ‹0 < d› convex_hull_eq_real_cbox by auto
    then have 1: convex hull c = {a. ∀ i∈Basis. a • i ∈ cbox (x • i - d) (x • i +
d)}
      unfolding box_eq_set_sum_Basis c_def convex_hull_set_sum
      apply (subst convex_hull_linear_image [symmetric])
      by (force simp add: linear_iff scaleR_add_left)+
    then have 2: convex hull c = {a. ∀ i∈Basis. a • i ∈ cball (x • i) d}
      by (simp add: dist_norm abs_le_iff algebra_simps)
    show cball x d ⊆ convex hull c
      unfolding 2
        by (clarsimp simp: dist_norm) (metis inner_commute inner_diff_right
norm_bound_Basis_le)
    have e': e = (∑ (i::'a)∈Basis. d)
      by (simp add: d_def)
    show convex hull c ⊆ cball x e
      unfolding 2
    proof clarsimp
      show dist x y ≤ e if ∀ i∈Basis. dist (x • i) (y • i) ≤ d for y
      proof -
        have ⋀i. i ∈ Basis ⟹ 0 ≤ dist (x • i) (y • i)
          by simp
        have (∑ i∈Basis. dist (x • i) (y • i)) ≤ e
          using e' sum_mono that by fastforce
        then show ?thesis
```

by (*metis* (*mono_tags*) *euclidean_dist_l2 order_trans* [*OF L2_set_le_sum*] *zero_le_dist*)
qed
qed
qed
define *k* **where** *k* = *Max* (*f* ‘ *c*)
have *convex_on* (*convex hull c*) *f*
using *assms*(*2*) *c1 convex_on_subset e*(*1*) **by** *blast*
then have *k*: ∀ *y*∈*convex hull c*. *f y* ≤ *k*
using *c convex_on_convex_hull_bound k_def* **by** *fastforce*
have *e* ≤ *e* ∗ *real DIM*(*'a*)
using *e*(*2*) *real_of_nat_ge_one_iff* **by** *auto*
then have *d* ≤ *e*
by (*simp add*: *d_def field_split_simps*)
then have *dsube*: *cball x d* ⊆ *cball x e*
by (*rule subset_cball*)
have *conv*: *convex_on* (*cball x d*) *f*
using ‹*convex_on* (*convex hull c*) *f*› *c2 convex_on_subset* **by** *blast*
then have ⋀*y*. *y*∈*cball x d* ⟹ |*f y*| ≤ *k* + *2* ∗ |*f x*|
by (*rule convex_bounds_lemma*) (*use c2 k* **in** *blast*)
then have *continuous_on* (*ball x d*) *f*
by (*meson Elementary_Metric_Spaces.open_ball ball_subset_cball conv convex_on_bounded_continuous*
convex_on_subset mem_ball_imp_mem_cball)
then show *continuous* (*at x*) *f*
unfolding *continuous_on_eq_continuous_at*[*OF open_ball*]
using ‹*d* > *0*› **by** *auto*
qed

end

4.3 Line Segment

theory *Line_Segment*
imports
Convex
Topology_Euclidean_Space
begin

4.3.1 Topological Properties of Convex Sets, Metric Spaces and Functions

lemma *convex_supp_sum*:
assumes *convex S* **and** *1*: *supp_sum u I* = *1*
and ⋀*i*. *i* ∈ *I* ⟹ *0* ≤ *u i* ∧ (*u i* = *0* ∨ *f i* ∈ *S*)
shows *supp_sum* (λ*i*. *u i* ∗$_R$ *f i*) *I* ∈ *S*
proof −
have *fin*: *finite* {*i* ∈ *I*. *u i* ≠ *0*}
using *1 sum.infinite* **by** (*force simp*: *supp_sum_def support_on_def*)

then have *supp_sum* ($\lambda i.\ u\ i *_R f\ i$) I = *sum* ($\lambda i.\ u\ i *_R f\ i$) $\{i \in I.\ u\ i \neq 0\}$
by (*force intro*: *sum.mono_neutral_left simp*: *supp_sum_def support_on_def*)
also have ... $\in S$
using *1 assms* **by** (*force simp*: *supp_sum_def support_on_def intro*: *convex_sum* [*OF fin* ‹*convex S*›])
finally show *?thesis* **.**
qed

lemma *sphere_eq_empty* [*simp*]:
fixes *a* :: *'a*::{*real_normed_vector*, *perfect_space*}
shows *sphere a r* = {} $\longleftrightarrow$ $r < 0$
by (*metis empty_iff linorder_not_less mem_sphere sphere_empty vector_choose_dist*)

lemma *cone_closure*:
fixes *S* :: *'a*::*real_normed_vector set*
assumes *cone S*
shows *cone* (*closure S*)
by (*metis UnCI assms closure_Un_frontier closure_eq_empty closure_scaleR cone_iff*)

corollary *component_complement_connected*:
fixes *S* :: *'a*::*real_normed_vector set*
assumes *connected S C* $\in$ *components* ($-S$)
shows *connected*($-C$)
using *component_diff_connected* [*of S UNIV*] *assms*
by (*auto simp*: *Compl_eq_Diff_UNIV*)

proposition *clopen*:
fixes *S* :: *'a* :: *real_normed_vector set*
shows *closed S* $\wedge$ *open S* $\longleftrightarrow$ *S* = {} $\vee$ *S* = *UNIV*
using *connected_UNIV* **by** (*force simp add*: *connected_clopen*)

corollary *compact_open*:
fixes *S* :: *'a* :: *euclidean_space set*
shows *compact S* $\wedge$ *open S* $\longleftrightarrow$ *S* = {}
by (*auto simp*: *compact_eq_bounded_closed clopen*)

corollary *finite_imp_not_open*:
fixes *S* :: *'a*::{*real_normed_vector*, *perfect_space*} *set*
shows $[\![$*finite S*; *open S*$]\!] \Longrightarrow$ *S*={}
using *clopen* [*of S*] *finite_imp_closed not_bounded_UNIV* **by** *blast*

corollary *empty_interior_finite*:
fixes *S* :: *'a*::{*real_normed_vector*, *perfect_space*} *set*
shows *finite S* $\Longrightarrow$ *interior S* = {}
by (*metis interior_subset finite_subset open_interior* [*of S*] *finite_imp_not_open*)

Balls, being convex, are connected.

lemma *convex_local_global_minimum*:
fixes s :: *'a*::*real_normed_vector set*
assumes $e > 0$
and *convex_on s f*
and *ball x e* $\subseteq$ s
and $\forall\, y{\in}$*ball x e*. $f\, x \leq f\, y$
shows $\forall\, y{\in}s$. $f\, x \leq f\, y$
proof (*rule ccontr*)
have $x \in s$ **using** *assms*(*1*,*3*) **by** *auto*
assume ¬ *?thesis*
then obtain y **where** $y{\in}s$ **and** y: $f\, x > f\, y$ **by** *auto*
then have *xy*: $0 <$ *dist x y* **by** *auto*
then obtain u **where** $0 < u\ u \leq 1$ **and** u: $u < e\ /$ *dist x y*
using *field_lbound_gt_zero*[*of 1 e / dist x y*] *xy* ‹*e*>*0*› **by** *auto*
then have $f\ ((1{-}u) *_R x + u *_R y) \leq (1{-}u) * f\, x + u * f\, y$
using ‹$x{\in}s$› ‹$y{\in}s$› **by** (*smt* (*verit*) *assms*(*2*) *convex_on_def*)
moreover
have $*$: $x - ((1 - u) *_R x + u *_R y) = u *_R (x - y)$
by (*simp add*: *algebra_simps*)
have $(1 - u) *_R x + u *_R y \in$ *ball x e*
by (*smt* (*verit*) $*$ ‹*0 < u*› *dist_norm mem_ball norm_scaleR pos_less_divide_eq u xy*)
then have $f\, x \leq f\ ((1 - u) *_R x + u *_R y)$
using *assms*(*4*) **by** *auto*
ultimately show *False*
using *mult_strict_left_mono*[*OF y* ‹*u*>*0*›]
unfolding *left_diff_distrib*
by *auto*
qed

lemma *convex_ball* [*iff*]:
fixes x :: *'a*::*real_normed_vector*
shows *convex* (*ball x e*)
proof (*auto simp*: *convex_def*)
fix $y\ z$
assume *yz*: *dist x y* $<$ e *dist x z* $<$ e
fix $u\ v$ :: *real*
assume *uv*: $0 \leq u\ 0 \leq v\ u + v = 1$
have *dist x* $(u *_R y + v *_R z) \leq u *$ *dist x y* $+ v *$ *dist x z*
using *uv yz*
using *convex_on_dist* [*of ball x e x*, *unfolded convex_on_def*,
THEN bspec[**where** $x{=}y$], *THEN bspec*[**where** $x{=}z$]]
by *auto*
then show *dist x* $(u *_R y + v *_R z) < e$
using *convex_bound_lt*[*OF yz uv*] **by** *auto*
qed

lemma *convex_cball* [*iff*]:
fixes x :: *'a*::*real_normed_vector*

```
  shows convex (cball x e)
proof −
  {
    fix y z
    assume yz: dist x y ≤ e dist x z ≤ e
    fix u v :: real
    assume uv: 0 ≤ u 0 ≤ v u + v = 1
    have dist x (u *R y + v *R z) ≤ u * dist x y + v * dist x z
      using uv yz
      using convex_on_dist [of cball x e x, unfolded convex_on_def,
        THEN bspec[where x=y], THEN bspec[where x=z]]
      by auto
    then have dist x (u *R y + v *R z) ≤ e
      using convex_bound_le[OF yz uv] by auto
  }
  then show ?thesis by (auto simp: convex_def Ball_def)
qed

lemma connected_ball [iff]:
  fixes x :: 'a::real_normed_vector
  shows connected (ball x e)
  using convex_connected convex_ball by auto

lemma connected_cball [iff]:
  fixes x :: 'a::real_normed_vector
  shows connected (cball x e)
  using convex_connected convex_cball by auto

lemma bounded_convex_hull:
  fixes s :: 'a::real_normed_vector set
  assumes bounded s
  shows bounded (convex hull s)
proof −
  from assms obtain B where B: ∀ x∈s. norm x ≤ B
    unfolding bounded_iff by auto
  show ?thesis
    by (simp add: bounded_subset[OF bounded_cball, of _ 0 B] B subsetI sub-
set_hull)
qed

lemma finite_imp_bounded_convex_hull:
  fixes s :: 'a::real_normed_vector set
  shows finite s ⟹ bounded (convex hull s)
  using bounded_convex_hull finite_imp_bounded
  by auto
```

4.3.2 Midpoint

```
definition midpoint :: 'a::real_vector ⇒ 'a ⇒ 'a
```

```
  where midpoint a b = (inverse (2::real)) *R (a + b)

lemma midpoint_idem [simp]: midpoint x x = x
  unfolding midpoint_def  by simp

lemma midpoint_sym: midpoint a b = midpoint b a
  unfolding midpoint_def by (auto simp add: scaleR_right_distrib)

lemma midpoint_eq_iff: midpoint a b = c ⟷ a + b = c + c
proof −
  have midpoint a b = c ⟷ scaleR 2 (midpoint a b) = scaleR 2 c
    by simp
  then show ?thesis
    unfolding midpoint_def scaleR_2 [symmetric] by simp
qed

lemma
  fixes a::real
  assumes a ≤ b shows ge_midpoint_1: a ≤ midpoint a b
                  and le_midpoint_1: midpoint a b ≤ b
  by (simp_all add: midpoint_def assms)

lemma dist_midpoint:
  fixes a b :: 'a::real_normed_vector shows
  dist a (midpoint a b) = (dist a b) / 2 (is ?t1)
  dist b (midpoint a b) = (dist a b) / 2 (is ?t2)
  dist (midpoint a b) a = (dist a b) / 2 (is ?t3)
  dist (midpoint a b) b = (dist a b) / 2 (is ?t4)
proof −
  have *: ⋀x y::'a. 2 *R x = − y ⟹ norm x = (norm y) / 2
    unfolding equation_minus_iff by auto
  have **: ⋀x y::'a. 2 *R x =   y ⟹ norm x = (norm y) / 2
    by auto
  note scaleR_right_distrib [simp]
  show ?t1
    unfolding midpoint_def dist_norm
    apply (rule **)
    apply (simp add: scaleR_right_diff_distrib)
    apply (simp add: scaleR_2)
    done
  show ?t2
    unfolding midpoint_def dist_norm
    apply (rule *)
    apply (simp add: scaleR_right_diff_distrib)
    apply (simp add: scaleR_2)
    done
  show ?t3
    unfolding midpoint_def dist_norm
    apply (rule *)
```

apply (*simp add*: *scaleR_right_diff_distrib*)
apply (*simp add*: *scaleR_2*)
done
show *?t4*
unfolding *midpoint_def dist_norm*
apply (*rule* ∗∗)
apply (*simp add*: *scaleR_right_diff_distrib*)
apply (*simp add*: *scaleR_2*)
done
qed

lemma *midpoint_eq_endpoint* [*simp*]:
midpoint a b = *a* ⟷ *a* = *b*
midpoint a b = *b* ⟷ *a* = *b*
unfolding *midpoint_eq_iff* **by** *auto*

lemma *midpoint_plus_self* [*simp*]: *midpoint a b* + *midpoint a b* = *a* + *b*
using *midpoint_eq_iff* **by** *metis*

lemma *midpoint_linear_image*:
linear f ⟹ *midpoint*(*f a*)(*f b*) = *f*(*midpoint a b*)
by (*simp add*: *linear_iff midpoint_def*)

4.3.3 Open and closed segments

definition *closed_segment* :: *'a*::*real_vector* ⇒ *'a* ⇒ *'a set*
where *closed_segment a b* = {(1 − *u*) ∗$_R$ *a* + *u* ∗$_R$ *b* | *u*::*real*. 0 ≤ *u* ∧ *u* ≤ 1}

definition *open_segment* :: *'a*::*real_vector* ⇒ *'a* ⇒ *'a set* **where**
open_segment a b ≡ *closed_segment a b* − {*a*,*b*}

lemmas *segment* = *open_segment_def closed_segment_def*

lemma *in_segment*:
x ∈ *closed_segment a b* ⟷ (∃ *u*. 0 ≤ *u* ∧ *u* ≤ 1 ∧ *x* = (1 − *u*) ∗$_R$ *a* + *u* ∗$_R$ *b*)
x ∈ *open_segment a b* ⟷ *a* ≠ *b* ∧ (∃ *u*. 0 < *u* ∧ *u* < 1 ∧ *x* = (1 − *u*) ∗$_R$ *a* + *u* ∗$_R$ *b*)
using *less_eq_real_def* **by** (*auto simp*: *segment algebra_simps*)

lemma *closed_segment_linear_image*:
closed_segment (*f a*) (*f b*) = *f* ‘ (*closed_segment a b*) **if** *linear f*
proof −
interpret *linear f* **by** *fact*
show *?thesis*
by (*force simp add*: *in_segment add scale*)
qed

lemma *open_segment_linear_image*:

⟦*linear f*; *inj f*⟧ ⟹ *open_segment* (*f a*) (*f b*) = *f* ‘ (*open_segment a b*)
by (*force simp*: *open_segment_def closed_segment_linear_image inj_on_def*)

lemma *closed_segment_translation*:
closed_segment (*c* + *a*) (*c* + *b*) = (λ*x*. *c* + *x*) ‘ (*closed_segment a b*) (**is** *?L* = _ ‘ *?R*)
proof −
have ⋀*x*. *x* ∈ *?L* ⟹ *x* − *c* ∈ *?R* ⋀*x*. ⟦*x* ∈ *?R*⟧ ⟹ *c* + *x* ∈ *?L*
by (*auto simp*: *in_segment algebra_simps*)
then show *?thesis* **by** *force*
qed

lemma *open_segment_translation*:
open_segment (*c* + *a*) (*c* + *b*) = *image* (λ*x*. *c* + *x*) (*open_segment a b*)
by (*simp add*: *open_segment_def closed_segment_translation translation_diff*)

lemma *closed_segment_of_real*:
closed_segment (*of_real x*) (*of_real y*) = *of_real* ‘ *closed_segment x y*
by (*simp add*: *closed_segment_linear_image linearI scaleR_conv_of_real*)

lemma *open_segment_of_real*:
open_segment (*of_real x*) (*of_real y*) = *of_real* ‘ *open_segment x y*
by (*simp add*: *closed_segment_of_real image_set_diff inj_of_real open_segment_def*)

lemma *closed_segment_Reals*:
⟦*x* ∈ *Reals*; *y* ∈ *Reals*⟧ ⟹ *closed_segment x y* = *of_real* ‘ *closed_segment* (*Re x*) (*Re y*)
by (*metis closed_segment_of_real of_real_Re*)

lemma *open_segment_Reals*:
⟦*x* ∈ *Reals*; *y* ∈ *Reals*⟧ ⟹ *open_segment x y* = *of_real* ‘ *open_segment* (*Re x*) (*Re y*)
by (*metis open_segment_of_real of_real_Re*)

lemma *open_segment_PairD*:
(*x*, *x′*) ∈ *open_segment* (*a*, *a′*) (*b*, *b′*)
⟹ (*x* ∈ *open_segment a b* ∨ *a* = *b*) ∧ (*x′* ∈ *open_segment a′ b′* ∨ *a′* = *b′*)
by (*auto simp*: *in_segment*)

lemma *closed_segment_PairD*:
(*x*, *x′*) ∈ *closed_segment* (*a*, *a′*) (*b*, *b′*) ⟹ *x* ∈ *closed_segment a b* ∧ *x′* ∈ *closed_segment a′ b′*
by (*auto simp*: *closed_segment_def*)

lemma *closed_segment_translation_eq* [*simp*]:
d + *x* ∈ *closed_segment* (*d* + *a*) (*d* + *b*) ⟷ *x* ∈ *closed_segment a b*
proof −
have ∗: ⋀*d x a b*. *x* ∈ *closed_segment a b* ⟹ *d* + *x* ∈ *closed_segment* (*d* + *a*) (*d* + *b*)

using *closed_segment_translation* **by** *blast*
show *?thesis*
using $*$ [**where** $d = -d$] $*$ **by** *fastforce*
qed

lemma *open_segment_translation_eq* [*simp*]:
$d + x \in$ *open_segment* $(d + a)$ $(d + b) \longleftrightarrow x \in$ *open_segment* a b
by (*simp add*: *open_segment_def*)

lemma *of_real_closed_segment* [*simp*]:
of_real $x \in$ *closed_segment* (*of_real* a) (*of_real* b) $\longleftrightarrow x \in$ *closed_segment* a b
by (*simp add*: *closed_segment_of_real image_iff*)

lemma *of_real_open_segment* [*simp*]:
of_real $x \in$ *open_segment* (*of_real* a) (*of_real* b) $\longleftrightarrow x \in$ *open_segment* a b
by (*simp add*: *image_iff open_segment_of_real*)

lemma *convex_contains_segment*:
convex $S \longleftrightarrow (\forall\, a{\in}S.\ \forall\, b{\in}S.$ *closed_segment* a $b \subseteq S)$
unfolding *convex_alt closed_segment_def* **by** *auto*

lemma *closed_segment_in_Reals*:
$[\![x \in$ *closed_segment* a b; $a \in$ *Reals*; $b \in$ *Reals*$]\!] \Longrightarrow x \in$ *Reals*
by (*meson subsetD convex_Reals convex_contains_segment*)

lemma *open_segment_in_Reals*:
$[\![x \in$ *open_segment* a b; $a \in$ *Reals*; $b \in$ *Reals*$]\!] \Longrightarrow x \in$ *Reals*
by (*metis Diff_iff closed_segment_in_Reals open_segment_def*)

lemma *closed_segment_subset*: $[\![x \in S$; $y \in S$; *convex* $S]\!] \Longrightarrow$ *closed_segment* x y $\subseteq S$
by (*simp add*: *convex_contains_segment*)

lemma *closed_segment_subset_convex_hull*:
$[\![x \in$ *convex hull* S; $y \in$ *convex hull* $S]\!] \Longrightarrow$ *closed_segment* x $y \subseteq$ *convex hull* S
using *convex_contains_segment* **by** *blast*

lemma *segment_convex_hull*:
closed_segment a b = *convex hull* $\{a,b\}$
proof −
have $*$: $\bigwedge x.\ \{x\} \neq \{\}$ **by** *auto*
show *?thesis*
unfolding *segment convex_hull_insert*[*OF* $*$] *convex_hull_singleton*
by (*safe*; *rule_tac x=1 − u* **in** *exI*; *force*)
qed

lemma *open_closed_segment*: $u \in$ *open_segment* w $z \Longrightarrow u \in$ *closed_segment* w z
by (*auto simp add*: *closed_segment_def open_segment_def*)

lemma *segment_open_subset_closed*:
  open_segment a b ⊆ *closed_segment a b*
 by (*simp add*: *open_closed_segment subsetI*)

lemma *bounded_closed_segment*:
 fixes *a* :: *'a*::*real_normed_vector* **shows** *bounded* (*closed_segment a b*)
 by (*simp add*: *bounded_convex_hull segment_convex_hull*)

lemma *bounded_open_segment*:
  fixes *a* :: *'a*::*real_normed_vector* **shows** *bounded* (*open_segment a b*)
 by (*rule bounded_subset* [*OF bounded_closed_segment segment_open_subset_closed*])

lemmas *bounded_segment* = *bounded_closed_segment open_closed_segment*

lemma *ends_in_segment* [*iff*]: *a* ∈ *closed_segment a b b* ∈ *closed_segment a b*
 by (*simp_all add*: *hull_inc segment_convex_hull*)

lemma *eventually_closed_segment*:
 fixes *x0*::*'a*::*real_normed_vector*
 assumes *open X0 x0* ∈ *X0*
 shows ∀ *F x in at x0 within U*. *closed_segment x0 x* ⊆ *X0*
proof −
 from *openE*[*OF assms*]
 obtain *e* **where** *e*: *0* < *e ball x0 e* ⊆ *X0* **.**
 then have ∀ *F x in at x0 within U*. *x* ∈ *ball x0 e*
  by (*auto simp*: *dist_commute eventually_at*)
 then show *?thesis*
 proof *eventually_elim*
  case (*elim x*)
  have *x0* ∈ *ball x0 e* **using** ‹*e* > *0*› **by** *simp*
  then have *closed_segment x0 x* ⊆ *ball x0 e*
   using *closed_segment_subset elim* **by** *blast*
  then show *?case*
   using *e*(*2*) **by** *auto*
 qed
qed

lemma *closed_segment_commute*: *closed_segment a b* = *closed_segment b a*
proof −
 have {*a*, *b*} = {*b*, *a*} **by** *auto*
 thus *?thesis*
  by (*simp add*: *segment_convex_hull*)
qed

lemma *segment_bound1*:
 assumes *x* ∈ *closed_segment a b*
 shows *norm* (*x* − *a*) ≤ *norm* (*b* − *a*)

proof −
obtain *u* **where** *u*: $x = (1 - u) *_R a + u *_R b$ $0 \le u$ $u \le 1$
using *assms* **by** (*auto simp add*: *closed_segment_def*)
then have $norm\ (u *_R b - u *_R a) \le norm\ (b - a)$
by (*simp add*: *mult_left_le_one_le flip*: *scaleR_diff_right*)
with *u* **show** *?thesis*
by (*metis add_diff_cancel_left scaleR_collapse*)
qed

lemma *segment_bound*:
assumes $x \in closed\_segment\ a\ b$
shows $norm\ (x - a) \le norm\ (b - a)$ $norm\ (x - b) \le norm\ (b - a)$
by (*metis assms closed_segment_commute dist_commute dist_norm segment_bound1*)+

lemma *open_segment_bound1*:
assumes $x \in open\_segment\ a\ b$
shows $norm\ (x - a) < norm\ (b - a)$
proof −
obtain *u* **where** *u*: $x = (1 - u) *_R a + u *_R b$ $0 < u$ $u < 1$
by (*meson assms in_segment*)
then have $norm\ (u *_R b - u *_R a) < norm\ (b - a)$
using *assms in_segment*(*2*) *less_eq_real_def* **by** (*fastforce simp flip*: *scaleR_diff_right*)
with *u* **show** *?thesis*
by (*metis add_diff_cancel_left scaleR_collapse*)
qed

lemma *open_segment_commute*: $open\_segment\ a\ b = open\_segment\ b\ a$
by (*simp add*: *closed_segment_commute insert_commute open_segment_def*)

lemma *closed_segment_idem* [*simp*]: $closed\_segment\ a\ a = \{a\}$
unfolding *segment* **by** (*auto simp add*: *algebra_simps*)

lemma *open_segment_idem* [*simp*]: $open\_segment\ a\ a = \{\}$
by (*simp add*: *open_segment_def*)

lemma *closed_segment_eq_open*: $closed\_segment\ a\ b = open\_segment\ a\ b \cup \{a,b\}$
using *open_segment_def* **by** *auto*

lemma *convex_contains_open_segment*:
$convex\ s \longleftrightarrow (\forall\, a{\in}s.\ \forall\, b{\in}s.\ open\_segment\ a\ b \subseteq s)$
by (*simp add*: *convex_contains_segment closed_segment_eq_open*)

lemma *closed_segment_eq_real_ivl1*:
fixes *a b*::*real*
assumes $a \le b$
shows $closed\_segment\ a\ b = \{a\ ..\ b\}$
proof *safe*
fix *x*
assume $x \in closed\_segment\ a\ b$

then obtain u **where** u: $0 \le u$ $u \le 1$ **and** x_def: $x = (1 - u) * a + u * b$
by (*auto simp*: *closed_segment_def*)
have $u * a \le u * b$ $(1 - u) * a \le (1 - u) * b$
by (*auto intro*!: *mult_left_mono u assms*)
then show $x \in \{a .. b\}$
unfolding *x_def* **by** (*auto simp*: *algebra_simps*)
next
show $\bigwedge x.\ x \in \{a..b\} \Longrightarrow x \in$ *closed_segment a b*
by (*force simp*: *closed_segment_def divide_simps algebra_simps*
intro: *exI*[**where** x=$(x - a) / (b - a)$ **for** x])
qed

lemma *closed_segment_eq_real_ivl*:
fixes *a b*::*real*
shows *closed_segment a b* = (*if a* $\le$ *b then* $\{a .. b\}$ *else* $\{b .. a\}$)
by (*metis closed_segment_commute closed_segment_eq_real_ivl1 nle_le*)

lemma *open_segment_eq_real_ivl*:
fixes *a b*::*real*
shows *open_segment a b* = (*if a* $\le$ *b then* $\{a<..<b\}$ *else* $\{b<..<a\}$)
by (*auto simp*: *closed_segment_eq_real_ivl open_segment_def split*: *if_split_asm*)

lemma *closed_segment_real_eq*:
fixes *u*::*real* **shows** *closed_segment u v* = $(\lambda x.\ (v - u) * x + u)$ ‘ $\{0..1\}$
by (*simp add*: *closed_segment_eq_real_ivl image_affinity_atLeastAtMost*)

lemma *closed_segment_same_Re*:
assumes *Re a* = *Re b*
shows *closed_segment a b* = $\{z.$ *Re z* = *Re a* $\wedge$ *Im z* $\in$ *closed_segment* (*Im a*) (*Im b*)$\}$
proof *safe*
fix *z* **assume** *z* $\in$ *closed_segment a b*
then obtain *u* **where** *u*: *u* $\in \{0..1\}$ *z* = *a* + *of_real u* $*$ (*b* $-$ *a*)
by (*auto simp*: *closed_segment_def scaleR_conv_of_real algebra_simps*)
from *assms* **show** *Re z* = *Re a* **by** (*auto simp*: *u*)
from *u*(*1*) **show** *Im z* $\in$ *closed_segment* (*Im a*) (*Im b*)
by (*force simp*: *u closed_segment_def algebra_simps*)
next
fix *z* **assume** [*simp*]: *Re z* = *Re a* **and** *Im z* $\in$ *closed_segment* (*Im a*) (*Im b*)
then obtain *u* **where** *u*: *u* $\in \{0..1\}$ *Im z* = *Im a* + *of_real u* $*$ (*Im b* $-$ *Im a*)
by (*auto simp*: *closed_segment_def scaleR_conv_of_real algebra_simps*)
from *u*(*1*) **show** *z* $\in$ *closed_segment a b* **using** *assms*
by (*force simp*: *u closed_segment_def algebra_simps scaleR_conv_of_real complex_eq_iff*)
qed

lemma *closed_segment_same_Im*:
assumes *Im a* = *Im b*
shows *closed_segment a b* = $\{z.$ *Im z* = *Im a* $\wedge$ *Re z* $\in$ *closed_segment* (*Re*

a) (*Re b*)}
proof *safe*
fix *z* **assume** *z* ∈ *closed_segment a b*
then obtain *u* **where** *u*: *u* ∈ {*0..1*} *z* = *a* + *of_real u* ∗ (*b* − *a*)
by (*auto simp*: *closed_segment_def scaleR_conv_of_real algebra_simps*)
from *assms* **show** *Im z* = *Im a* **by** (*auto simp*: *u*)
from *u*(*1*) **show** *Re z* ∈ *closed_segment* (*Re a*) (*Re b*)
by (*force simp*: *u closed_segment_def algebra_simps*)
next
fix *z* **assume** [*simp*]: *Im z* = *Im a* **and** *Re z* ∈ *closed_segment* (*Re a*) (*Re b*)
then obtain *u* **where** *u*: *u* ∈ {*0..1*} *Re z* = *Re a* + *of_real u* ∗ (*Re b* − *Re a*)
by (*auto simp*: *closed_segment_def scaleR_conv_of_real algebra_simps*)
from *u*(*1*) **show** *z* ∈ *closed_segment a b* **using** *assms*
by (*force simp*: *u closed_segment_def algebra_simps scaleR_conv_of_real complex_eq_iff*)
qed

lemma *dist_in_closed_segment*:
fixes *a* :: *'a* :: *euclidean_space*
assumes *x* ∈ *closed_segment a b*
shows *dist x a* ≤ *dist a b* ∧ *dist x b* ≤ *dist a b*
by (*metis assms dist_commute dist_norm segment_bound*(*2*) *segment_bound1*)

lemma *dist_in_open_segment*:
fixes *a* :: *'a* :: *euclidean_space*
assumes *x* ∈ *open_segment a b*
shows *dist x a* < *dist a b* ∧ *dist x b* < *dist a b*
by (*metis assms dist_commute dist_norm open_segment_bound1 open_segment_commute*)

lemma *dist_decreases_open_segment_0*:
fixes *x* :: *'a* :: *euclidean_space*
assumes *x* ∈ *open_segment 0 b*
shows *dist c x* < *dist c 0* ∨ *dist c x* < *dist c b*
proof (*rule ccontr*, *clarsimp simp*: *not_less*)
obtain *u* **where** *u*: *0* ≠ *b 0* < *u u* < *1* **and** *x*: *x* = *u* ∗$_R$ *b*
using *assms* **by** (*auto simp*: *in_segment*)
have *xb*: *x* · *b* < *b* · *b*
using *u x* **by** *auto*
assume *norm c* ≤ *dist c x*
then have *c* · *c* ≤ (*c* − *x*) · (*c* − *x*)
by (*simp add*: *dist_norm norm_le*)
moreover have *0* < *x* · *b*
using *u x* **by** *auto*
ultimately have *less*: *c* · *b* < *x* · *b*
by (*simp add*: *x algebra_simps inner_commute u*)
assume *dist c b* ≤ *dist c x*
then have (*c* − *b*) · (*c* − *b*) ≤ (*c* − *x*) · (*c* − *x*)
by (*simp add*: *dist_norm norm_le*)
then have (*b* · *b*) ∗ (*1* − *u*∗*u*) ≤ *2* ∗ (*b* · *c*) ∗ (*1*−*u*)

by (*simp add*: *x algebra_simps inner_commute*)
then have (*1*+*u*) ∗ (*b* · *b*) ∗ (*1*−*u*) ≤ *2* ∗ (*b* · *c*) ∗ (*1*−*u*)
by (*simp add*: *algebra_simps*)
then have (*1*+*u*) ∗ (*b* · *b*) ≤ *2* ∗ (*b* · *c*)
using ‹*u* < *1*› **by** *auto*
with *xb* **have** *c* · *b* ≥ *x* · *b*
by (*auto simp*: *x algebra_simps inner_commute*)
with *less* **show** *False* **by** *auto*
qed

proposition *dist_decreases_open_segment*:
fixes *a* :: *'a* :: *euclidean_space*
assumes *x* ∈ *open_segment a b*
shows *dist c x* < *dist c a* ∨ *dist c x* < *dist c b*
proof −
have ∗: *x* − *a* ∈ *open_segment 0* (*b* − *a*) **using** *assms*
by (*metis diff_self open_segment_translation_eq uminus_add_conv_diff*)
show *?thesis*
using *dist_decreases_open_segment_0* [*OF* ∗, *of c*−*a*] *assms*
by (*simp add*: *dist_norm*)
qed

corollary *open_segment_furthest_le*:
fixes *a b x y* :: *'a*::*euclidean_space*
assumes *x* ∈ *open_segment a b*
shows *norm* (*y* − *x*) < *norm* (*y* − *a*) ∨ *norm* (*y* − *x*) < *norm* (*y* − *b*)
by (*metis assms dist_decreases_open_segment dist_norm*)

corollary *dist_decreases_closed_segment*:
fixes *a* :: *'a* :: *euclidean_space*
assumes *x* ∈ *closed_segment a b*
shows *dist c x* ≤ *dist c a* ∨ *dist c x* ≤ *dist c b*
by (*smt* (*verit*, *ccfv_threshold*) *Un_iff assms closed_segment_eq_open dist_norm empty_iff insertE open_segment_furthest_le*)

corollary *segment_furthest_le*:
fixes *a b x y* :: *'a*::*euclidean_space*
assumes *x* ∈ *closed_segment a b*
shows *norm* (*y* − *x*) ≤ *norm* (*y* − *a*) ∨ *norm* (*y* − *x*) ≤ *norm* (*y* − *b*)
by (*metis assms dist_decreases_closed_segment dist_norm*)

lemma *convex_intermediate_ball*:
fixes *a* :: *'a* :: *euclidean_space*
shows ⟦*ball a r* ⊆ *T*; *T* ⊆ *cball a r*⟧ ⟹ *convex T*
by (*smt* (*verit*) *convex_contains_open_segment dist_decreases_open_segment mem_ball mem_cball subset_eq*)

lemma *csegment_midpoint_subset*: *closed_segment* (*midpoint a b*) *b* ⊆ *closed_segment a b*

apply (*clarsimp simp*: *midpoint_def in_segment*)
apply (*rule_tac x*=(*1 + u*) / *2* **in** *exI*)
apply (*simp add*: *algebra_simps add_divide_distrib diff_divide_distrib*)
by (*metis field_sum_of_halves scaleR_left.add*)

lemma *notin_segment_midpoint*:
fixes *a* :: *'a* :: *euclidean_space*
shows *a* ≠ *b* ⟹ *a* ∉ *closed_segment* (*midpoint a b*) *b*
by (*auto simp*: *dist_midpoint dest*!: *dist_in_closed_segment*)

More lemmas, especially for working with the underlying formula

lemma *segment_eq_compose*:
fixes *a* :: *'a* :: *real_vector*
shows (λ*u*. (*1* − *u*) *∗R a* + *u ∗R b*) = (λ*x*. *a* + *x*) *o* (λ*u*. *u ∗R* (*b* − *a*))
by (*simp add*: *o_def algebra_simps*)

lemma *segment_degen_1*:
fixes *a* :: *'a* :: *real_vector*
shows (*1* − *u*) *∗R a* + *u ∗R b* = *b* ⟷ *a*=*b* ∨ *u*=*1*
by (*smt* (*verit*, *best*) *add_right_cancel scaleR_cancel_left scaleR_collapse*)

lemma *segment_degen_0*:
fixes *a* :: *'a* :: *real_vector*
shows (*1* − *u*) *∗R a* + *u ∗R b* = *a* ⟷ *a*=*b* ∨ *u*=*0*
using *segment_degen_1* [*of 1*−*u b a*] **by** (*auto simp*: *algebra_simps*)

lemma *add_scaleR_degen*:
fixes *a b* ::*'a*::*real_vector*
assumes (*u ∗R b* + *v ∗R a*) = (*u ∗R a* + *v ∗R b*) *u* ≠ *v*
shows *a*=*b*
by (*smt* (*verit*) *add_diff_cancel_left' add_diff_eq assms scaleR_cancel_left scaleR_left.diff*)

lemma *closed_segment_image_interval*:
closed_segment a b = (λ*u*. (*1* − *u*) *∗R a* + *u ∗R b*) ‘ {*0*..*1*}
by (*auto simp*: *set_eq_iff image_iff closed_segment_def*)

lemma *open_segment_image_interval*:
open_segment a b = (*if a*=*b then* {} *else* (λ*u*. (*1* − *u*) *∗R a* + *u ∗R b*) ‘ {*0*<..<*1*})
by (*auto simp*: *open_segment_def closed_segment_def segment_degen_0 segment_degen_1*)

lemmas *segment_image_interval* = *closed_segment_image_interval open_segment_image_interval*

lemma *closed_segment_neq_empty* [*simp*]: *closed_segment a b* ≠ {}
by *auto*

lemma *open_segment_eq_empty* [*simp*]: *open_segment a b* = {} ⟷ *a* = *b*
by (*simp add*: *segment_image_interval*(*2*))

lemma *open_segment_eq_empty'* [*simp*]: {} = *open_segment a b* ⟷ *a* = *b*
using *open_segment_eq_empty* **by** *blast*

lemmas *segment_eq_empty* = *closed_segment_neq_empty open_segment_eq_empty*

lemma *inj_segment*:
fixes *a* :: *'a* :: *real_vector*
assumes *a* ≠ *b*
shows *inj_on* (λ*u*. (*1* − *u*) *∗R a* + *u ∗R b*) *I*
proof
fix *x y*
assume (*1* − *x*) *∗R a* + *x ∗R b* = (*1* − *y*) *∗R a* + *y ∗R b*
then have *x ∗R* (*b* − *a*) = *y ∗R* (*b* − *a*)
by (*simp add*: *algebra_simps*)
with *assms* **show** *x* = *y*
by (*simp add*: *real_vector.scale_right_imp_eq*)
qed

lemma *finite_closed_segment* [*simp*]: *finite*(*closed_segment a b*) ⟷ *a* = *b*
using *infinite_Icc* [*OF zero_less_one*] *finite_imageD* [*OF _ inj_segment* [*of a b*]]
unfolding *segment_image_interval*
by (*smt* (*verit*, *del_insts*) *finite.emptyI finite_insert finite_subset image_subset_iff insertCI segment_degen_0*)

lemma *finite_open_segment* [*simp*]: *finite*(*open_segment a b*) ⟷ *a* = *b*
by (*auto simp*: *open_segment_def*)

lemmas *finite_segment* = *finite_closed_segment finite_open_segment*

lemma *closed_segment_eq_sing*: *closed_segment a b* = {*c*} ⟷ *a* = *c* ∧ *b* = *c*
by *auto*

lemma *open_segment_eq_sing*: *open_segment a b* ≠ {*c*}
by (*metis finite_insert finite_open_segment insert_not_empty open_segment_image_interval*)

lemmas *segment_eq_sing* = *closed_segment_eq_sing open_segment_eq_sing*

lemma *compact_segment* [*simp*]:
fixes *a* :: *'a*::*real_normed_vector*
shows *compact* (*closed_segment a b*)
by (*auto simp*: *segment_image_interval intro*!: *compact_continuous_image continuous_intros*)

lemma *closed_segment* [*simp*]:
fixes *a* :: *'a*::*real_normed_vector*

shows *closed* (*closed_segment a b*)
by (*simp add*: *compact_imp_closed*)

lemma *closure_closed_segment* [*simp*]:
fixes *a* :: *'a*::*real_normed_vector*
shows *closure*(*closed_segment a b*) = *closed_segment a b*
by *simp*

lemma *open_segment_bound*:
assumes *x* ∈ *open_segment a b*
shows *norm* (*x* − *a*) < *norm* (*b* − *a*) *norm* (*x* − *b*) < *norm* (*b* − *a*)
by (*metis assms norm_minus_commute open_segment_bound1 open_segment_commute*)+

lemma *closure_open_segment* [*simp*]:
closure (*open_segment a b*) = (*if a* = *b then* {} *else closed_segment a b*)
for *a* :: *'a*::*euclidean_space*
proof (*cases a* = *b*)
case *True*
then show *?thesis*
by *simp*
next
case *False*
have *closure* ((λ*u*. *u* ∗*R* (*b* − *a*)) ‘ {*0*<..<*1*}) = (λ*u*. *u* ∗*R* (*b* − *a*)) ‘ *closure* {*0*<..<*1*}
proof (*rule closure_injective_linear_image* [*symmetric*])
qed (*use False* **in** ‹*auto intro*!: *injI*›)
then have *closure*
((λ*u*. (*1* − *u*) ∗*R* *a* + *u* ∗*R* *b*) ‘ {*0*<..<*1*}) = (λ*x*. (*1* − *x*) ∗*R* *a* + *x* ∗*R* *b*) ‘ *closure* {*0*<..<*1*}
using *closure_translation* [*of a* ((λ*x*. *x* ∗*R* *b* − *x* ∗*R* *a*) ‘ {*0*<..<*1*})]
by (*simp add*: *segment_eq_compose field_simps scaleR_diff_left scaleR_diff_right image_image*)
then show *?thesis*
by (*simp add*: *segment_image_interval closure_greaterThanLessThan* [*symmetric*] *del*: *closure_greaterThanLessThan*)
qed

lemma *closed_open_segment_iff* [*simp*]:
fixes *a* :: *'a*::*euclidean_space* **shows** *closed*(*open_segment a b*) ⟷ *a* = *b*
by (*metis open_segment_def DiffE closure_eq closure_open_segment ends_in_segment*(*1*) *insert_iff segment_image_interval*(*2*))

lemma *compact_open_segment_iff* [*simp*]:
fixes *a* :: *'a*::*euclidean_space* **shows** *compact*(*open_segment a b*) ⟷ *a* = *b*
by (*simp add*: *bounded_open_segment compact_eq_bounded_closed*)

lemma *convex_closed_segment* [*iff*]: *convex* (*closed_segment a b*)
unfolding *segment_convex_hull* **by**(*rule convex_convex_hull*)

lemma *convex_open_segment* [*iff*]: *convex* (*open_segment a b*)
proof −
have *convex* ((λ*u*. *u* ∗*R* (*b* − *a*)) ‘ {*0*<..<*1*})
by (*rule convex_linear_image*) *auto*
then have *convex* ((+) *a* ‘ (λ*u*. *u* ∗*R* (*b* − *a*)) ‘ {*0*<..<*1*})
by (*rule convex_translation*)
then show *?thesis*
by (*simp add*: *image_image open_segment_image_interval segment_eq_compose field_simps scaleR_diff_left scaleR_diff_right*)
qed

lemmas *convex_segment* = *convex_closed_segment convex_open_segment*

lemma *subset_closed_segment*:
closed_segment a b ⊆ *closed_segment c d* ⟷
a ∈ *closed_segment c d* ∧ *b* ∈ *closed_segment c d*
using *closed_segment_subset convex_closed_segment ends_in_segment in_mono*
by *blast*

lemma *subset_co_segment*:
closed_segment a b ⊆ *open_segment c d* ⟷
a ∈ *open_segment c d* ∧ *b* ∈ *open_segment c d*
using *closed_segment_subset* **by** *blast*

lemma *subset_open_segment*:
fixes *a* :: *'a*::*euclidean_space*
shows *open_segment a b* ⊆ *open_segment c d* ⟷
a = *b* ∨ *a* ∈ *closed_segment c d* ∧ *b* ∈ *closed_segment c d*
(**is** *?lhs* = *?rhs*)
proof (*cases a* = *b*)
case *True* **then show** *?thesis* **by** *simp*
next
case *False* **show** *?thesis*
proof
assume *rhs*: *?rhs*
with ‹*a* ≠ *b*› **have** *c* ≠ *d*
using *closed_segment_idem singleton_iff* **by** *auto*
have ∃*uc*. (*1* − *u*) ∗*R* ((*1* − *ua*) ∗*R* *c* + *ua* ∗*R* *d*) + *u* ∗*R* ((*1* − *ub*) ∗*R* *c* + *ub* ∗*R* *d*) =
(*1* − *uc*) ∗*R* *c* + *uc* ∗*R* *d* ∧ *0* < *uc* ∧ *uc* < *1*
if *neq*: (*1* − *ua*) ∗*R* *c* + *ua* ∗*R* *d* ≠ (*1* − *ub*) ∗*R* *c* + *ub* ∗*R* *d* *c* ≠ *d*
and *a* = (*1* − *ua*) ∗*R* *c* + *ua* ∗*R* *d* *b* = (*1* − *ub*) ∗*R* *c* + *ub* ∗*R* *d*
and *u*: *0* < *u* *u* < *1* **and** *uab*: *0* ≤ *ua* *ua* ≤ *1* *0* ≤ *ub* *ub* ≤ *1*
for *u ua ub*
proof −
have *ua* ≠ *ub*
using *neq* **by** *auto*
moreover have (*u* − *1*) ∗ *ua* ≤ *0* **using** *u uab*
by (*simp add*: *mult_nonpos_nonneg*)

```
    ultimately have lt: (u − 1) ∗ ua < u ∗ ub using u uab
      by (metis antisym_conv diff_ge_0_iff_ge le_less_trans mult_eq_0_iff
mult_le_0_iff not_less)
    have p ∗ ua + q ∗ ub < p+q if p: 0 < p and  q: 0 < q for p q
    proof −
      have ¬ p ≤ 0 ¬ q ≤ 0
        using p q not_less by blast+
      then show ?thesis
        by (smt (verit) ‹ua ≠ ub› mult_cancel_left1 mult_left_le uab(2) uab(4))
    qed
    then have (1 − u) ∗ ua + u ∗ ub < 1 using u ‹ua ≠ ub›
      by (metis diff_add_cancel diff_gt_0_iff_gt)
    with lt show ?thesis
      by (rule_tac x=ua + u∗(ub−ua) in exI) (simp add: algebra_simps)
  qed
  with rhs ‹a ≠ b› ‹c ≠ d› show ?lhs
    unfolding open_segment_image_interval closed_segment_def
    by (fastforce simp add:)
 next
  assume lhs: ?lhs
  with ‹a ≠ b› have c ≠ d
    by (meson finite_open_segment rev_finite_subset)
  have closure (open_segment a b) ⊆ closure (open_segment c d)
    using lhs closure_mono by blast
  then have closed_segment a b ⊆ closed_segment c d
    by (simp add: ‹a ≠ b› ‹c ≠ d›)
  then show ?rhs
    by (force simp: ‹a ≠ b›)
 qed
qed

lemma closed_segment_same_fst:
 fst a = fst b ⟹ closed_segment a b = {fst a} × closed_segment (snd a) (snd b)
 by (auto simp: closed_segment_def scaleR_prod_def)

lemma closed_segment_same_snd:
 snd a = snd b ⟹ closed_segment a b = closed_segment (fst a) (fst b) × {snd
a}
 by (auto simp: closed_segment_def scaleR_prod_def)

lemma subset_oc_segment:
 fixes a :: 'a::euclidean_space
 shows open_segment a b ⊆ closed_segment c d ⟷
       a = b ∨ a ∈ closed_segment c d ∧ b ∈ closed_segment c d
   (is ?lhs = ?rhs)
proof
 show ?lhs ⟹ ?rhs
   by (metis closure_closed_segment closure_mono closure_open_segment sub-
set_closed_segment)
```

show *?rhs* $\Longrightarrow$ *?lhs*
by (*meson dual_order.trans segment_open_subset_closed subset_open_segment*)
qed

lemmas *subset_segment* = *subset_closed_segment subset_co_segment subset_oc_segment subset_open_segment*

lemma *dist_half_times2*:
fixes *a* :: *'a* :: *real_normed_vector*
shows *dist* ((*1* / *2*) $*_R$ (*a* + *b*)) *x* * *2* = *dist* (*a*+*b*) (*2* $*_R$ *x*)
proof −
have *norm* ((*1* / *2*) $*_R$ (*a* + *b*) − *x*) * *2* = *norm* (*2* $*_R$ ((*1* / *2*) $*_R$ (*a* + *b*) − *x*))
by *simp*
also have ... = *norm* ((*a* + *b*) − *2* $*_R$ *x*)
by (*simp add*: *real_vector.scale_right_diff_distrib*)
finally show *?thesis*
by (*simp only*: *dist_norm*)
qed

lemma *closed_segment_as_ball*:
closed_segment a b = *affine hull* {*a*,*b*} ∩ *cball*(*inverse 2* $*_R$ (*a* + *b*))(*norm*(*b* − *a*) / *2*)
proof (*cases b* = *a*)
case *True* **then show** *?thesis* **by** (*auto simp*: *hull_inc*)
next
case *False*
then have *: ((∃ *u v*. *x* = *u* $*_R$ *a* + *v* $*_R$ *b* ∧ *u* + *v* = *1*) ∧
dist ((*1* / *2*) $*_R$ (*a* + *b*)) *x* * *2* ≤ *norm* (*b* − *a*)) =
(∃ *u*. *x* = (*1* − *u*) $*_R$ *a* + *u* $*_R$ *b* ∧ *0* ≤ *u* ∧ *u* ≤ *1*) **for** *x*
proof −
have ((∃ *u v*. *x* = *u* $*_R$ *a* + *v* $*_R$ *b* ∧ *u* + *v* = *1*) ∧
dist ((*1* / *2*) $*_R$ (*a* + *b*)) *x* * *2* ≤ *norm* (*b* − *a*)) =
((∃ *u*. *x* = (*1* − *u*) $*_R$ *a* + *u* $*_R$ *b*) ∧
dist ((*1* / *2*) $*_R$ (*a* + *b*)) *x* * *2* ≤ *norm* (*b* − *a*))
unfolding *eq_diff_eq* [*symmetric*] **by** *simp*
also have ... = (∃ *u*. *x* = (*1* − *u*) $*_R$ *a* + *u* $*_R$ *b* ∧
norm ((*a*+*b*) − (*2* $*_R$ *x*)) ≤ *norm* (*b* − *a*))
by (*simp add*: *dist_half_times2*) (*simp add*: *dist_norm*)
also have ... = (∃ *u*. *x* = (*1* − *u*) $*_R$ *a* + *u* $*_R$ *b* ∧
norm ((*a*+*b*) − (*2* $*_R$ ((*1* − *u*) $*_R$ *a* + *u* $*_R$ *b*))) ≤ *norm* (*b* − *a*))
by *auto*
also have ... = (∃ *u*. *x* = (*1* − *u*) $*_R$ *a* + *u* $*_R$ *b* ∧
norm ((*1* − *u* * *2*) $*_R$ (*b* − *a*)) ≤ *norm* (*b* − *a*))
by (*simp add*: *algebra_simps scaleR_2*)
also have ... = (∃ *u*. *x* = (*1* − *u*) $*_R$ *a* + *u* $*_R$ *b* ∧
|*1* − *u* * *2*| * *norm* (*b* − *a*) ≤ *norm* (*b* − *a*))
by *simp*
also have ... = (∃ *u*. *x* = (*1* − *u*) $*_R$ *a* + *u* $*_R$ *b* ∧ |*1* − *u* * *2*| ≤ *1*)

```
    by (simp add: mult_le_cancel_right2 False)
  also have ... = (∃u. x = (1 − u) *R a + u *R b ∧ 0 ≤ u ∧ u ≤ 1)
    by auto
  finally show ?thesis .
 qed
 show ?thesis
  by (simp add: affine_hull_2 Set.set_eq_iff closed_segment_def *)
qed

lemma open_segment_as_ball:
   open_segment a b =
    affine hull {a,b} ∩ ball(inverse 2 *R (a + b))(norm(b − a) / 2)
proof (cases b = a)
 case True then show ?thesis by (auto simp: hull_inc)
next
 case False
 then have *: ((∃u v. x = u *R a + v *R b ∧ u + v = 1) ∧
               dist ((1 / 2) *R (a + b)) x * 2 < norm (b − a)) =
              (∃u. x = (1 − u) *R a + u *R b ∧ 0 < u ∧ u < 1) for x
 proof −
  have ((∃u v. x = u *R a + v *R b ∧ u + v = 1) ∧
               dist ((1 / 2) *R (a + b)) x * 2 < norm (b − a)) =
        ((∃u. x = (1 − u) *R a + u *R b) ∧
               dist ((1 / 2) *R (a + b)) x * 2 < norm (b − a))
    unfolding eq_diff_eq [symmetric] by simp
  also have ... = (∃u. x = (1 − u) *R a + u *R b ∧
                       norm ((a+b) − (2 *R x)) < norm (b − a))
    by (simp add: dist_half_times2) (simp add: dist_norm)
  also have ... = (∃u. x = (1 − u) *R a + u *R b ∧
         norm ((a+b) − (2 *R ((1 − u) *R a + u *R b))) < norm (b − a))
    by auto
  also have ... = (∃u. x = (1 − u) *R a + u *R b ∧
            norm ((1 − u * 2) *R (b − a)) < norm (b − a))
    by (simp add: algebra_simps scaleR_2)
  also have ... = (∃u. x = (1 − u) *R a + u *R b ∧
                       |1 − u * 2| * norm (b − a) < norm (b − a))
    by simp
  also have ... = (∃u. x = (1 − u) *R a + u *R b ∧ |1 − u * 2| < 1)
    by (simp add: mult_le_cancel_right2 False)
  also have ... = (∃u. x = (1 − u) *R a + u *R b ∧ 0 < u ∧ u < 1)
    by auto
  finally show ?thesis .
 qed
 show ?thesis
  using False by (force simp: affine_hull_2 Set.set_eq_iff open_segment_image_interval
*)
qed

lemmas segment_as_ball = closed_segment_as_ball open_segment_as_ball
```

lemma *connected_segment* [*iff*]:
fixes $x :: 'a :: real\_normed\_vector$
shows *connected* (*closed_segment x y*)
by (*simp add*: *convex_connected*)

lemma *is_interval_closed_segment_1*[*intro*, *simp*]: *is_interval* (*closed_segment a b*) **for** *a b*::*real*
unfolding *closed_segment_eq_real_ivl*
by (*auto simp*: *is_interval_def*)

lemma *IVT'_closed_segment_real*:
fixes $f :: real \Rightarrow real$
assumes $y \in$ *closed_segment* (*f a*) (*f b*)
assumes *continuous_on* (*closed_segment a b*) *f*
shows $\exists x \in$ *closed_segment a b*. $f\ x = y$
using *IVT'*[*of f a y b*]
IVT'[*of* $-f$ *a* $-y$ *b*]
IVT'[*of f b y a*]
IVT'[*of* $-f$ *b* $-y$ *a*] *assms*
by (*cases* $a \leq b$; *cases f b* $\geq$ *f a*) (*auto simp*: *closed_segment_eq_real_ivl continuous_on_minus*)

4.3.4 Betweenness

definition *between* = $(\lambda(a,b)\ x.\ x \in$ *closed_segment a b*)

lemma *betweenI*:
assumes $0 \leq u\ u \leq 1\ x = (1 - u) *_R a + u *_R b$
shows *between* (*a*, *b*) *x*
using *assms* **unfolding** *between_def closed_segment_def* **by** *auto*

lemma *betweenE*:
assumes *between* (*a*, *b*) *x*
obtains *u* **where** $0 \leq u\ u \leq 1\ x = (1 - u) *_R a + u *_R b$
using *assms* **unfolding** *between_def closed_segment_def* **by** *auto*

lemma *between_implies_scaled_diff*:
assumes *between* (*S*, *T*) *X between* (*S*, *T*) *Y S* $\neq$ *Y*
obtains *c* **where** $(X - Y) = c *_R (S - Y)$
proof −
from ‹*between* (*S*, *T*) *X*› **obtain** u_X **where** *X*: $X = u_X *_R S + (1 - u_X) *_R T$
by (*metis add.commute betweenE eq_diff_eq*)
from ‹*between* (*S*, *T*) *Y*› **obtain** u_Y **where** *Y*: $Y = u_Y *_R S + (1 - u_Y) *_R T$
by (*metis add.commute betweenE eq_diff_eq*)
have $X - Y = (u_X - u_Y) *_R (S - T)$
by (*simp add*: *X Y scaleR_left.diff scaleR_right_diff_distrib*)

```
  moreover from Y have S − Y = (1 − u_Y) *_R (S − T)
   by (simp add: real_vector.scale_left_diff_distrib real_vector.scale_right_diff_distrib)
  moreover note ‹S ≠ Y›
  ultimately have (X − Y) = ((u_X − u_Y) / (1 − u_Y)) *_R (S − Y) by auto
  from this that show thesis by blast
qed

lemma between_mem_segment: between (a,b) x ⟷ x ∈ closed_segment a b
  unfolding between_def by auto

lemma between: between (a, b) (x::'a::euclidean_space) ⟷ dist a b = (dist a x)
+ (dist x b)
proof (cases a = b)
  case True
  then show ?thesis
    by (auto simp add: between_def dist_commute)
next
  case False
  then have Fal: norm (a − b) ≠ 0 and Fal2: norm (a − b) > 0
    by auto
  have *: ⋀u. a − ((1 − u) *_R a + u *_R b) = u *_R (a − b)
    by (auto simp add: algebra_simps)
  have norm (a − x) *_R (x − b) = norm (x − b) *_R (a − x) if x = (1 − u) *_R
a + u *_R b 0 ≤ u u ≤ 1 for u
  proof −
    have *: a − x = u *_R (a − b) x − b = (1 − u) *_R (a − b)
      unfolding that(1) by (auto simp add:algebra_simps)
    show norm (a − x) *_R (x − b) = norm (x − b) *_R (a − x)
      unfolding norm_minus_commute[of x a] * using ‹0 ≤ u› ‹u ≤ 1›
      by simp
  qed
  moreover have ∃u. x = (1 − u) *_R a + u *_R b ∧ 0 ≤ u ∧ u ≤ 1 if dist a b
= dist a x + dist x b
  proof −
    let ?β = norm (a − x) / norm (a − b)
    show ∃u. x = (1 − u) *_R a + u *_R b ∧ 0 ≤ u ∧ u ≤ 1
    proof (intro exI conjI)
      show ?β ≤ 1
        using Fal2 unfolding that[unfolded dist_norm] norm_ge_zero by auto
      show x = (1 − ?β) *_R a + (?β) *_R b
      proof (subst euclidean_eq_iff; intro ballI)
        fix i :: 'a
        assume i: i ∈ Basis
        have ((1 − ?β) *_R a + (?β) *_R b) • i
              = ((norm (a − b) − norm (a − x)) * (a • i) + norm (a − x) * (b •
i)) / norm (a − b)
          using Fal by (auto simp add: field_simps inner_simps)
        also have ... = x•i
          apply (rule divide_eq_imp[OF Fal])
```

```
        unfolding that[unfolded dist_norm]
        using that[unfolded dist_triangle_eq] i
        apply (subst (asm) euclidean_eq_iff)
         apply (auto simp add: field_simps inner_simps)
        done
      finally show x · i = ((1 − ?β) *R a + (?β) *R b) · i
        by auto
    qed
  qed (use Fal2 in auto)
 qed
 ultimately show ?thesis
  by (force simp add: between_def closed_segment_def dist_triangle_eq)
qed

lemma between_midpoint:
 fixes a :: 'a::euclidean_space
 shows between (a,b) (midpoint a b) (is ?t1)
  and between (b,a) (midpoint a b) (is ?t2)
proof −
 have ∗: ⋀x y z. x = (1/2::real) *R z ⟹ y = (1/2) *R z ⟹ norm z = norm
x + norm y
  by auto
 show ?t1 ?t2
  unfolding between midpoint_def dist_norm
   by (auto simp add: field_simps inner_simps euclidean_eq_iff[where 'a='a]
intro!: ∗)
qed

lemma between_mem_convex_hull:
 between (a,b) x ⟷ x ∈ convex hull {a,b}
 unfolding between_mem_segment segment_convex_hull ..

lemma between_triv_iff [simp]: between (a,a) b ⟷ a=b
 by (auto simp: between_def)

lemma between_triv1 [simp]: between (a,b) a
 by (auto simp: between_def)

lemma between_triv2 [simp]: between (a,b) b
 by (auto simp: between_def)

lemma between_commute:
 between (a,b) = between (b,a)
 by (auto simp: between_def closed_segment_commute)

lemma between_antisym:
 fixes a :: 'a :: euclidean_space
 shows ⟦between (b,c) a; between (a,c) b⟧ ⟹ a = b
 by (auto simp: between dist_commute)
```

lemma *between_trans*:
fixes *a* :: *'a* :: *euclidean_space*
shows ⟦*between* (*b*,*c*) *a*; *between* (*a*,*c*) *d*⟧ ⟹ *between* (*b*,*c*) *d*
using *dist_triangle2* [*of b c d*] *dist_triangle3* [*of b d a*]
by (*auto simp*: *between dist_commute*)

lemma *between_norm*:
fixes *a* :: *'a* :: *euclidean_space*
shows *between* (*a*,*b*) *x* ⟷ *norm*(*x* − *a*) *∗R* (*b* − *x*) = *norm*(*b* − *x*) *∗R* (*x* − *a*)
by (*auto simp*: *between dist_triangle_eq norm_minus_commute algebra_simps*)

lemma *between_swap*:
fixes *A B X Y* :: *'a*::*euclidean_space*
assumes *between* (*A*, *B*) *X*
assumes *between* (*A*, *B*) *Y*
shows *between* (*X*, *B*) *Y* ⟷ *between* (*A*, *Y*) *X*
using *assms* **by** (*auto simp add*: *between*)

lemma *between_translation* [*simp*]: *between* (*a* + *y*,*a* + *z*) (*a* + *x*) ⟷ *between* (*y*,*z*) *x*
by (*auto simp*: *between_def*)

lemma *between_trans_2*:
fixes *a* :: *'a* :: *euclidean_space*
shows ⟦*between* (*b*,*c*) *a*; *between* (*a*,*b*) *d*⟧ ⟹ *between* (*c*,*d*) *a*
by (*metis between_commute between_swap between_trans*)

lemma *between_scaleR_lift* [*simp*]:
fixes *v* :: *'a*::*euclidean_space*
shows *between* (*a ∗R v*, *b ∗R v*) (*c ∗R v*) ⟷ *v* = *0* ∨ *between* (*a*, *b*) *c*
by (*simp add*: *between dist_norm flip*: *scaleR_left_diff_distrib distrib_right*)

lemma *between_1*:
fixes *x*::*real*
shows *between* (*a*,*b*) *x* ⟷ (*a* ≤ *x* ∧ *x* ≤ *b*) ∨ (*b* ≤ *x* ∧ *x* ≤ *a*)
by (*auto simp*: *between_mem_segment closed_segment_eq_real_ivl*)

end

Chapter 5

Unsorted

theory *Starlike*
imports
Convex_Euclidean_Space
Line_Segment
begin

lemma *affine_hull_closed_segment* [*simp*]:
affine hull (*closed_segment a b*) = *affine hull* {*a*,*b*}
by (*simp add*: *segment_convex_hull*)

lemma *affine_hull_open_segment* [*simp*]:
fixes *a* :: *'a*::*euclidean_space*
shows *affine hull* (*open_segment a b*) = (*if a* = *b then* {} *else affine hull* {*a*,*b*})
by (*metis affine_hull_convex_hull affine_hull_empty closure_open_segment closure_same_affine_hull segment_convex_hull*)

lemma *rel_interior_closure_convex_segment*:
fixes *S* :: *_*::*euclidean_space set*
assumes *convex S a* ∈ *rel_interior S b* ∈ *closure S*
shows *open_segment a b* ⊆ *rel_interior S*
proof
fix *x*
have [*simp*]: (*1* − *u*) *∗R a* + *u ∗R b* = *b* − (*1* − *u*) *∗R* (*b* − *a*) **for** *u*
by (*simp add*: *algebra_simps*)
assume *x* ∈ *open_segment a b*
then show *x* ∈ *rel_interior S*
unfolding *closed_segment_def open_segment_def* **using** *assms*
by (*auto intro*: *rel_interior_closure_convex_shrink*)
qed

lemma *convex_hull_insert_segments*:
convex hull (*insert a S*) =
(*if S* = {} *then* {*a*} *else* ⋃*x* ∈ *convex hull S*. *closed_segment a x*)
by (*force simp add*: *convex_hull_insert_alt in_segment*)

lemma *Int_convex_hull_insert_rel_exterior*:
fixes *z* :: *'a*::*euclidean_space*
assumes *convex C T* ⊆ *C* **and** *z*: *z* ∈ *rel_interior C* **and** *dis*: *disjnt S* (*rel_interior C*)
shows *S* ∩ (*convex hull* (*insert z T*)) = *S* ∩ (*convex hull T*) (**is** *?lhs* = *?rhs*)
proof
have *T* = {} ⟹ *z* ∉ *S*
using *dis z* **by** (*auto simp add*: *disjnt_def*)
then show *?lhs* ⊆ *?rhs*
proof (*clarsimp simp add*: *convex_hull_insert_segments*)
fix *x y*
assume *x* ∈ *S* **and** *y*: *y* ∈ *convex hull T* **and** *x* ∈ *closed_segment z y*
have *y* ∈ *closure C*
by (*metis y* ‹*convex C*› ‹*T* ⊆ *C*› *closure_subset contra_subsetD convex_hull_eq hull_mono*)
moreover have *x* ∉ *rel_interior C*
by (*meson* ‹*x* ∈ *S*› *dis disjnt_iff*)
moreover have *x* ∈ *open_segment z y* ∪ {*z*, *y*}
using ‹*x* ∈ *closed_segment z y*› *closed_segment_eq_open* **by** *blast*
ultimately show *x* ∈ *convex hull T*
using *rel_interior_closure_convex_segment* [*OF* ‹*convex C*› *z*]
using *y z* **by** *blast*
qed
show *?rhs* ⊆ *?lhs*
by (*meson hull_mono inf_mono subset_insertI subset_refl*)
qed

5.0.1 Shrinking towards the interior of a convex set

lemma *mem_interior_convex_shrink*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *convex S*
and *c* ∈ *interior S*
and *x* ∈ *S*
and *0* < *e*
and *e* ≤ *1*
shows *x* − *e* ∗*R* (*x* − *c*) ∈ *interior S*
proof −
obtain *d* **where** *d* > *0* **and** *d*: *ball c d* ⊆ *S*
using *assms*(*2*) **unfolding** *mem_interior* **by** *auto*
show *?thesis*
unfolding *mem_interior*
proof (*intro exI subsetI conjI*)
fix *y*
assume *y* ∈ *ball* (*x* − *e* ∗*R* (*x* − *c*)) (*e*∗*d*)
then have *as*: *dist* (*x* − *e* ∗*R* (*x* − *c*)) *y* < *e* ∗ *d*
by *simp*
have ∗: *y* = (*1* − (*1* − *e*)) ∗*R* ((*1* / *e*) ∗*R* *y* − ((*1* − *e*) / *e*) ∗*R* *x*) + (*1* − *e*)

```
*R x
    using ‹e > 0› by (auto simp add: scaleR_left_diff_distrib scaleR_right_diff_distrib)
    have c − ((1 / e) *R y − ((1 − e) / e) *R x) = (1 / e) *R (e *R c − y + (1
− e) *R x)
      using ‹e > 0›
      by (auto simp add: euclidean_eq_iff[where 'a='a] field_simps inner_simps)
    then have dist c ((1 / e) *R y − ((1 − e) / e) *R x) = |1/e| * norm (e *R c
− y + (1 − e) *R x)
      by (simp add: dist_norm)
    also have ... = |1/e| * norm (x − e *R (x − c) − y)
      by (auto intro!:arg_cong[where f=norm] simp add: algebra_simps)
    also have ... < d
      using as[unfolded dist_norm] and ‹e > 0›
      by (auto simp add:pos_divide_less_eq[OF ‹e > 0›] mult.commute)
    finally have (1 − (1 − e)) *R ((1 / e) *R y − ((1 − e) / e) *R x) + (1 −
e) *R x ∈ S
      using assms(3−5) d
      by (intro convexD_alt [OF ‹convex S›]) (auto intro: convexD_alt [OF ‹convex
S›])
    with ‹e > 0› show y ∈ S
      by (auto simp add: scaleR_left_diff_distrib scaleR_right_diff_distrib)
  qed (use ‹e>0› ‹d>0› in auto)
qed

lemma mem_interior_closure_convex_shrink:
  fixes S :: 'a::euclidean_space set
  assumes convex S
    and c ∈ interior S
    and x ∈ closure S
    and 0 < e
    and e ≤ 1
  shows x − e *R (x − c) ∈ interior S
proof −
  obtain d where d > 0 and d: ball c d ⊆ S
    using assms(2) unfolding mem_interior by auto
  have ∃ y∈S. norm (y − x) * (1 − e) < e * d
  proof (cases x ∈ S)
    case True
    then show ?thesis
      using ‹e > 0› ‹d > 0› by force
  next
    case False
    then have x: x islimpt S
      using assms(3)[unfolded closure_def] by auto
    show ?thesis
    proof (cases e = 1)
      case True
      obtain y where y ∈ S y ≠ x dist y x < 1
        using x[unfolded islimpt_approachable,THEN spec[where x=1]] by auto
```

```
      then show ?thesis
        using True ‹0 < d› by auto
    next
      case False
      then have 0 < e * d / (1 - e) and *: 1 - e > 0
        using ‹e ≤ 1› ‹e > 0› ‹d > 0› by auto
      then obtain y where y ∈ S y ≠ x dist y x < e * d / (1 - e)
        using islimpt_approachable x by blast
      then have norm (y - x) * (1 - e) < e * d
        by (metis * dist_norm mult_imp_div_pos_le not_less)
      then show ?thesis
        using ‹y ∈ S› by blast
    qed
  qed
  then obtain y where y ∈ S and y: norm (y - x) * (1 - e) < e * d
    by auto
  define z where z = c + ((1 - e) / e) *R (x - y)
  have *: x - e *R (x - c) = y - e *R (y - z)
    unfolding z_def using ‹e > 0›
   by (auto simp add: scaleR_right_diff_distrib scaleR_right_distrib scaleR_left_diff_distrib)
  have (1 - e) * norm (x - y) / e < d
    using y ‹0 < e› by (simp add: field_simps norm_minus_commute)
  then have z ∈ interior (ball c d)
     using ‹0 < e› ‹e ≤ 1› by (simp add: interior_open[OF open_ball] z_def
dist_norm)
  then have z ∈ interior S
    using d interiorI interior_ball by blast
  then show ?thesis
    unfolding * using mem_interior_convex_shrink ‹y ∈ S› assms by blast
qed

lemma in_interior_closure_convex_segment:
  fixes S :: 'a::euclidean_space set
  assumes convex S and a: a ∈ interior S and b: b ∈ closure S
    shows open_segment a b ⊆ interior S
proof (clarsimp simp: in_segment)
  fix u::real
  assume u: 0 < u u < 1
  have (1 - u) *R a + u *R b = b - (1 - u) *R (b - a)
    by (simp add: algebra_simps)
   also have ... ∈ interior S using mem_interior_closure_convex_shrink [OF
assms] u
    by simp
  finally show (1 - u) *R a + u *R b ∈ interior S .
qed

lemma convex_closure_interior:
  fixes S :: 'a::euclidean_space set
  assumes convex S and int: interior S ≠ {}
```

```
  shows closure(interior S) = closure S
proof −
  obtain a where a: a ∈ interior S
    using int by auto
  have closure S ⊆ closure(interior S)
  proof
    fix x
    assume x: x ∈ closure S
    show x ∈ closure (interior S)
    proof (cases x=a)
      case True
      then show ?thesis
        using ‹a ∈ interior S› closure_subset by blast
    next
      case False
      show ?thesis
      proof (clarsimp simp add: closure_def islimpt_approachable)
        fix e::real
        assume xnotS: x ∉ interior S and 0 < e
        show ∃x'∈interior S. x' ≠ x ∧ dist x' x < e
        proof (intro bexI conjI)
          show x − min (e/2 / norm (x − a)) 1 *R (x − a) ≠ x
            using False ‹0 < e› by (auto simp: algebra_simps min_def)
          show dist (x − min (e/2 / norm (x − a)) 1 *R (x − a)) x < e
            using ‹0 < e› by (auto simp: dist_norm min_def)
          show x − min (e/2 / norm (x − a)) 1 *R (x − a) ∈ interior S
            using ‹0 < e› False
         by (auto simp add: min_def a intro: mem_interior_closure_convex_shrink
[OF ‹convex S› a x])
        qed
      qed
    qed
  qed
  then show ?thesis
    by (simp add: closure_mono interior_subset subset_antisym)
qed

lemma closure_convex_Int_superset:
  fixes S :: 'a::euclidean_space set
  assumes convex S interior S ≠ {} interior S ⊆ closure T
  shows closure(S ∩ T) = closure S
proof −
  have closure S ⊆ closure(interior S)
    by (simp add: convex_closure_interior assms)
  also have ... ⊆ closure (S ∩ T)
    using interior_subset [of S] assms
   by (metis (no_types, lifting) Int_assoc Int_lower2 closure_mono closure_open_Int_superset
inf.orderE open_interior)
  finally show ?thesis
```

by (*simp add*: *closure_mono dual_order.antisym*)
qed

5.0.2 Some obvious but surprisingly hard simplex lemmas

lemma *simplex*:
assumes *finite S*
and *0 ∉ S*
shows *convex hull* (*insert 0 S*) = {*y*. ∃ *u*. (∀ *x*∈*S*. *0* ≤ *u x*) ∧ *sum u S* ≤ *1* ∧ *sum* (λ*x*. *u x* ∗*R x*) *S* = *y*}
proof (*simp add*: *convex_hull_finite set_eq_iff assms*, *safe*)
fix *x* **and** *u* :: *'a* ⇒ *real*
assume *0* ≤ *u 0* ∀ *x*∈*S*. *0* ≤ *u x u 0* + *sum u S* = *1*
then show ∃ *v*. (∀ *x*∈*S*. *0* ≤ *v x*) ∧ *sum v S* ≤ *1* ∧ (∑ *x*∈*S*. *v x* ∗*R x*) = (∑ *x*∈*S*. *u x* ∗*R x*)
by *force*
next
fix *x* **and** *u* :: *'a* ⇒ *real*
assume ∀ *x*∈*S*. *0* ≤ *u x sum u S* ≤ *1*
then show ∃ *v*. *0* ≤ *v 0* ∧ (∀ *x*∈*S*. *0* ≤ *v x*) ∧ *v 0* + *sum v S* = *1* ∧ (∑ *x*∈*S*. *v x* ∗*R x*) = (∑ *x*∈*S*. *u x* ∗*R x*)
by (*rule_tac x*=λ*x*. *if x* = *0 then 1* − *sum u S else u x* **in** *exI*) (*auto simp*: *sum_delta_notmem assms if_smult*)
qed

lemma *substd_simplex*:
assumes *d*: *d* ⊆ *Basis*
shows *convex hull* (*insert 0 d*) =
{*x*. (∀ *i*∈*Basis*. *0* ≤ *x*•*i*) ∧ (∑ *i*∈*d*. *x*•*i*) ≤ *1* ∧ (∀ *i*∈*Basis*. *i* ∉ *d* ⟶ *x*•*i* = *0*)}
(**is** *convex hull* (*insert 0 ?p*) = *?s*)
proof −
let *?D* = *d*
have *0* ∉ *?p*
using *assms* **by** (*auto simp*: *image_def*)
from *d* **have** *finite d*
by (*blast intro*: *finite_subset finite_Basis*)
show *?thesis*
unfolding *simplex*[*OF* ‹*finite d*› ‹*0* ∉ *?p*›]
proof (*intro set_eqI*; *safe*)
fix *u* :: *'a* ⇒ *real*
assume *as*: ∀ *x*∈*?D*. *0* ≤ *u x sum u ?D* ≤ *1*
let *?x* = (∑ *x*∈*?D*. *u x* ∗*R x*)
have *ind*: ∀ *i*∈*Basis*. *i* ∈ *d* ⟶ *u i* = *?x* • *i*
and *notind*: (∀ *i*∈*Basis*. *i* ∉ *d* ⟶ *?x* • *i* = *0*)
using *substdbasis_expansion_unique*[*OF assms*] **by** *blast*+
then have ∗∗: *sum u ?D* = *sum* ((•) *?x*) *?D*
using *assms* **by** (*auto intro*!: *sum.cong*)
show *0* ≤ *?x* • *i* **if** *i* ∈ *Basis* **for** *i*
using *as*(*1*) *ind notind that* **by** *fastforce*

```
    show sum ((·) ?x) ?D ≤ 1
      using ** as(2) by linarith
    show ?x · i = 0 if i ∈ Basis i ∉ d for i
      using notind that by blast
  next
    fix x
    assume ∀ i∈Basis. 0 ≤ x · i sum ((·) x) ?D ≤ 1 (∀ i∈Basis. i ∉ d ⟶ x · i
= 0)
    with d show ∃ u. (∀ x∈?D. 0 ≤ u x) ∧ sum u ?D ≤ 1 ∧ (∑ x∈?D. u x *R x)
= x
      unfolding substdbasis_expansion_unique[OF assms]
      by (rule_tac x=inner x in exI) auto
  qed
qed

lemma std_simplex:
  convex hull (insert 0 Basis) =
    {x::'a::euclidean_space. (∀ i∈Basis. 0 ≤ x·i) ∧ sum (λi. x·i) Basis ≤ 1}
  using substd_simplex[of Basis] by auto

lemma interior_std_simplex:
  interior (convex hull (insert 0 Basis)) =
    {x::'a::euclidean_space. (∀ i∈Basis. 0 < x·i) ∧ sum (λi. x·i) Basis < 1}
  unfolding set_eq_iff mem_interior std_simplex
proof (intro allI iffI CollectI; clarify)
  fix x :: 'a
  fix e
  assume e > 0 and as: ball x e ⊆ {x. (∀ i∈Basis. 0 ≤ x · i) ∧ sum ((·) x) Basis
≤ 1}
  show (∀ i∈Basis. 0 < x · i) ∧ sum ((·) x) Basis < 1
  proof safe
    fix i :: 'a
    assume i: i ∈ Basis
    then show 0 < x · i
      using as[THEN subsetD[where c=x − (e/2) *R i]] and ‹e > 0›
      by (force simp add: inner_simps)
  next
    have **: dist x (x + (e/2) *R (SOME i. i∈Basis)) < e using ‹e > 0›
      unfolding dist_norm
      by (auto intro!: mult_strict_left_mono simp: SOME_Basis)
    have ⋀i. i ∈ Basis ⟹ (x + (e/2) *R (SOME i. i∈Basis)) · i =
      x·i + (if i = (SOME i. i∈Basis) then e/2 else 0)
      by (auto simp: SOME_Basis inner_Basis inner_simps)
    then have *: sum ((·) (x + (e/2) *R (SOME i. i∈Basis))) Basis =
      sum (λi. x·i + (if (SOME i. i∈Basis) = i then e/2 else 0)) Basis
      by (auto simp: intro!: sum.cong)
    have sum ((·) x) Basis < sum ((·) (x + (e/2) *R (SOME i. i∈Basis))) Basis
      using ‹e > 0› DIM_positive by (auto simp: SOME_Basis sum.distrib *)
    also have ... ≤ 1
```

```
    using ** as by force
   finally show sum ((·) x) Basis < 1 by auto
  qed
next
  fix x :: 'a
  assume as: ∀ i∈Basis. 0 < x · i sum ((·) x) Basis < 1
  obtain a :: 'b where a ∈ UNIV using UNIV_witness ..
  let ?d = (1 - sum ((·) x) Basis) / real (DIM('a))
  show ∃ e>0. ball x e ⊆ {x. (∀ i∈Basis. 0 ≤ x · i) ∧ sum ((·) x) Basis ≤ 1}
   proof (rule_tac x=min (Min (((·) x) ' Basis)) D for D in exI, intro conjI
subsetI CollectI)
    fix y
    assume y: y ∈ ball x (min (Min ((·) x ' Basis)) ?d)
    have sum ((·) y) Basis ≤ sum (λi. x·i + ?d) Basis
    proof (rule sum_mono)
     fix i :: 'a
     assume i: i ∈ Basis
     have |y·i - x·i| ≤ norm (y - x)
      by (metis Basis_le_norm i inner_commute inner_diff_right)
     also have ... < ?d
      using y by (simp add: dist_norm norm_minus_commute)
     finally have |y·i - x·i| < ?d .
     then show y · i ≤ x · i + ?d by auto
    qed
    also have ... ≤ 1
     unfolding sum.distrib sum_constant
     by (auto simp add: Suc_le_eq)
    finally show sum ((·) y) Basis ≤ 1 .
    show (∀ i∈Basis. 0 ≤ y · i)
    proof safe
     fix i :: 'a
     assume i: i ∈ Basis
     have norm (x - y) < Min (((·) x) ' Basis)
      using y by (auto simp: dist_norm less_eq_real_def)
     also have ... ≤ x·i
      using i by auto
     finally have norm (x - y) < x·i .
     then show 0 ≤ y·i
      using Basis_le_norm[OF i, of x - y] and as(1)[rule_format, OF i]
      by (auto simp: inner_simps)
    qed
  next
    have Min (((·) x) ' Basis) > 0
     using as by simp
    moreover have ?d > 0
     using as by (auto simp: Suc_le_eq)
    ultimately show 0 < min (Min ((·) x ' Basis)) ((1 - sum ((·) x) Basis) /
real DIM('a))
     by linarith
```

```
  qed
qed

lemma interior_std_simplex_nonempty:
  obtains a :: 'a::euclidean_space where
    a ∈ interior(convex hull (insert 0 Basis))
proof −
  let ?D = Basis :: 'a set
  let ?a = sum (λb::'a. inverse (2 ∗ real DIM('a)) ∗R b) Basis
  {
    fix i :: 'a
    assume i: i ∈ Basis
    have ?a • i = inverse (2 ∗ real DIM('a))
      by (rule trans[of _ sum (λj. if i = j then inverse (2 ∗ real DIM('a)) else 0)
?D])
         (simp_all add: sum.If_cases i) }
  note ∗∗ = this
  show ?thesis
  proof
    show ?a ∈ interior(convex hull (insert 0 Basis))
      unfolding interior_std_simplex mem_Collect_eq
    proof safe
      fix i :: 'a
      assume i: i ∈ Basis
      show 0 < ?a • i
        unfolding ∗∗[OF i] by (auto simp add: Suc_le_eq)
    next
      have sum ((•) ?a) ?D = sum (λi. inverse (2 ∗ real DIM('a))) ?D
        by (auto intro: sum.cong)
      also have ... < 1
        unfolding sum_constant divide_inverse[symmetric]
        by (auto simp add: field_simps)
      finally show sum ((•) ?a) ?D < 1 by auto
    qed
  qed
qed

lemma rel_interior_substd_simplex:
  assumes D: D ⊆ Basis
  shows rel_interior (convex hull (insert 0 D)) =
        {x::'a::euclidean_space. (∀ i∈D. 0 < x•i) ∧ (∑ i∈D. x•i) < 1 ∧ (∀ i∈Basis.
i ∉ D ⟶ x•i = 0)}
    (is _ = ?s)
proof −
  have finite D
    using D finite_Basis finite_subset by blast
  show ?thesis
  proof (cases D = {})
    case True
```

```
  then show ?thesis
    using rel_interior_sing using euclidean_eq_iff[of _ 0] by auto
 next
  case False
  have h0: affine hull (convex hull (insert 0 D)) =
          {x::'a::euclidean_space. (∀ i∈Basis. i ∉ D ⟶ x•i = 0)}
    using affine_hull_convex_hull affine_hull_substd_basis assms by auto
  have aux: ⋀x::'a. ∀ i∈Basis. (∀ i∈D. 0 ≤ x•i) ∧ (∀ i∈Basis. i ∉ D ⟶ x•i =
0) ⟶ 0 ≤ x•i
    by auto
  {
    fix x :: 'a::euclidean_space
    assume x: x ∈ rel_interior (convex hull (insert 0 D))
    then obtain e where e > 0 and
      ball x e ∩ {xa. (∀ i∈Basis. i ∉ D ⟶ xa•i = 0)} ⊆ convex hull (insert 0 D)
      using mem_rel_interior_ball[of x convex hull (insert 0 D)] h0 by auto
    then have as: ⋀y. ⟦dist x y < e ∧ (∀ i∈Basis. i ∉ D ⟶ y•i = 0)⟧ ⟹
                        (∀ i∈D. 0 ≤ y • i) ∧ sum ((•) y) D ≤ 1
      using assms by (force simp: substd_simplex)
    have x0: (∀ i∈Basis. i ∉ D ⟶ x•i = 0)
      using x rel_interior_subset  substd_simplex[OF assms] by auto
    have (∀ i∈D. 0 < x • i) ∧ sum ((•) x) D < 1 ∧ (∀ i∈Basis. i ∉ D ⟶ x•i =
0)
    proof (intro conjI ballI)
      fix i :: 'a
      assume i ∈ D
      then have ∀ j∈D. 0 ≤ (x − (e/2) *R i) • j
        using D ‹e > 0› x0
          by (intro as[THEN conjunct1]) (force simp: dist_norm inner_simps
inner_Basis)
      then show 0 < x • i
        using ‹e > 0› ‹i ∈ D› D  by (force simp: inner_simps inner_Basis)
    next
      obtain a where a: a ∈ D
        using ‹D ≠ {}› by auto
      then have **: dist x (x + (e/2) *R a) < e
        using ‹e > 0› norm_Basis[of a] D by (auto simp: dist_norm)
       have ⋀i. i ∈ Basis ⟹ (x + (e/2) *R a) • i = x•i + (if i = a then e/2
else 0)
        using a D by (auto simp: inner_simps inner_Basis)
      then have *: sum ((•) (x + (e/2) *R a)) D = sum (λi. x•i + (if a = i then
e/2 else 0)) D
        using D by (intro sum.cong) auto
      have a ∈ Basis
        using ‹a ∈ D› D by auto
      then have h1: (∀ i∈Basis. i ∉ D ⟶ (x + (e/2) *R a) • i = 0)
        using x0 D ‹a∈D› by (auto simp add: inner_add_left inner_Basis)
      have sum ((•) x) D < sum ((•) (x + (e/2) *R a)) D
        using ‹e > 0› ‹a ∈ D› ‹finite D› by (auto simp add: * sum.distrib)
```

```
      also have ... ≤ 1
        using ** h1 as[rule_format, of x + (e/2) *R a]
        by auto
      finally show sum ((•) x) D < 1 ⋀i. i∈Basis ⟹ i ∉ D ⟶ x•i = 0
        using x0 by auto
    qed
  }
  moreover
  {
    fix x :: 'a::euclidean_space
    assume as: x ∈ ?s
    have ∀ i. 0 < x•i ∨ 0 = x•i ⟶ 0 ≤ x•i
      by auto
    moreover have ∀ i. i ∈ D ∨ i ∉ D by auto
    ultimately
    have ∀ i. (∀ i∈D. 0 < x•i) ∧ (∀ i. i ∉ D ⟶ x•i = 0) ⟶ 0 ≤ x•i
      by metis
    then have h2: x ∈ convex hull (insert 0 D)
      using as assms by (force simp add: substd_simplex)
    obtain a where a: a ∈ D
      using ‹D ≠ {}› by auto
    define d where d ≡ (1 - sum ((•) x) D) / real (card D)
    have ∃ e>0. ball x e ∩ {x. ∀ i∈Basis. i ∉ D ⟶ x • i = 0} ⊆ convex hull
insert 0 D
      unfolding substd_simplex[OF assms]
    proof (intro exI; safe)
      have 0 < card D using ‹D ≠ {}› ‹finite D›
        by (simp add: card_gt_0_iff)
      have Min (((•) x) ‘ D) > 0
        using as ‹D ≠ {}› ‹finite D› by (simp)
      moreover have d > 0
        using as ‹0 < card D› by (auto simp: d_def)
      ultimately show min (Min (((•) x) ‘ D)) d > 0
        by auto
      fix y :: 'a
      assume y2: ∀ i∈Basis. i ∉ D ⟶ y•i = 0
      assume y ∈ ball x (min (Min ((•) x ‘ D)) d)
      then have y: dist x y < min (Min ((•) x ‘ D)) d
        by auto
      have sum ((•) y) D ≤ sum (λi. x•i + d) D
      proof (rule sum_mono)
        fix i
        assume i ∈ D
        with D have i: i ∈ Basis
          by auto
        have |y•i - x•i| ≤ norm (y - x)
            by (metis i inner_commute inner_diff_right norm_bound_Basis_le
order_refl)
        also have ... < d
```

```
        by (metis dist_norm min_less_iff_conj norm_minus_commute y)
       finally have |y•i − x•i| < d .
       then show y · i ≤ x · i + d by auto
      qed
      also have ... ≤ 1
       unfolding sum.distrib sum_constant d_def using ‹0 < card D›
       by auto
      finally show sum ((•) y) D ≤ 1 .

      fix i :: 'a
      assume i: i ∈ Basis
      then show 0 ≤ y•i
      proof (cases i∈D)
       case True
       have norm (x − y) < x•i
        using y Min_gr_iff[of (•) x ' D norm (x − y)] ‹0 < card D› ‹i ∈ D›
        by (simp add: dist_norm card_gt_0_iff)
       then show 0 ≤ y•i
        using Basis_le_norm[OF i, of x − y] and as(1)[rule_format]
        by (auto simp: inner_simps)
      qed (use y2 in auto)
     qed
     then have x ∈ rel_interior (convex hull (insert 0 D))
      using h0 h2 rel_interior_ball by force
    }
    ultimately have
     ⋀x. x ∈ rel_interior (convex hull insert 0 D) ⟷
      x ∈ {x. (∀ i∈D. 0 < x · i) ∧ sum ((•) x) D < 1 ∧ (∀ i∈Basis. i ∉ D ⟶ x
· i = 0)}
     by blast
    then show ?thesis by (rule set_eqI)
  qed
qed

lemma rel_interior_substd_simplex_nonempty:
  assumes D ≠ {}
   and D ⊆ Basis
  obtains a :: 'a::euclidean_space
   where a ∈ rel_interior (convex hull (insert 0 D))
proof −
  let ?a = sum (λb::'a::euclidean_space. inverse (2 ∗ real (card D)) ∗R b) D
  have finite D
   using assms finite_Basis infinite_super by blast
  then have d1: 0 < real (card D)
   using ‹D ≠ {}› by auto
  {
   fix i
   assume i ∈ D
   have ?a · i = sum (λj. if i = j then inverse (2 ∗ real (card D)) else 0) D
```

```
    unfolding inner_sum_left
      using ‹i ∈ D› by (auto simp: inner_Basis subsetD[OF assms(2)] intro:
sum.cong)
   also have ... = inverse (2 * real (card D))
    using ‹i ∈ D› ‹finite D› by auto
   finally have ?a · i = inverse (2 * real (card D)) .
  }
  note ** = this
  show ?thesis
  proof
   show ?a ∈ rel_interior (convex hull (insert 0 D))
    unfolding rel_interior_substd_simplex[OF assms(2)]
   proof safe
    fix i
    assume i ∈ D
    have 0 < inverse (2 * real (card D))
     using d1 by auto
    also have ... = ?a · i using **[of i] ‹i ∈ D›
     by auto
    finally show 0 < ?a · i by auto
   next
    have sum ((·) ?a) D = sum (λi. inverse (2 * real (card D))) D
     by (rule sum.cong) (rule refl, rule **)
    also have ... < 1
     unfolding sum_constant divide_real_def[symmetric]
     by (auto simp add: field_simps)
    finally show sum ((·) ?a) D < 1 by auto
   next
    fix i
    assume i ∈ Basis and i ∉ D
    have ?a ∈ span D
    proof (rule span_sum[of D (λb. b /R (2 * real (card D))) D])
     {
      fix x :: 'a::euclidean_space
      assume x ∈ D
      then have x ∈ span D
       using span_base[of _ D] by auto
      then have x /R (2 * real (card D)) ∈ span D
       using span_mul[of x D (inverse (real (card D)) / 2)] by auto
     }
     then show ⋀x. x∈D ⟹ x /R (2 * real (card D)) ∈ span D
      by auto
    qed
    then show ?a · i = 0
      using ‹i ∉ D› unfolding span_substd_basis[OF assms(2)] using ‹i ∈
Basis› by auto
   qed
  qed
qed
```

5.0.3 Relative interior of convex set

lemma *rel_interior_convex_nonempty_aux*:
fixes $S :: 'n::euclidean\_space\ set$
assumes *convex S*
and $0 \in S$
shows $rel\_interior\ S \neq \{\}$
proof (*cases* $S = \{0\}$)
case *True*
then show *?thesis* **using** *rel_interior_sing* **by** *auto*
next
case *False*
obtain *B* **where** *B*: *independent* $B \wedge B \leq S \wedge S \leq span\ B \wedge card\ B = dim\ S$
using *basis_exists*[*of S*] **by** *metis*
then have $B \neq \{\}$
using *B assms* ‹$S \neq \{0\}$› *span_empty* **by** *auto*
have $insert\ 0\ B \leq span\ B$
using *subspace_span*[*of B*] *subspace_0*[*of span B*]
span_superset **by** *auto*
then have $span\ (insert\ 0\ B) \leq span\ B$
using *span_span*[*of B*] *span_mono*[*of insert 0 B span B*] **by** *blast*
then have $convex\ hull\ insert\ 0\ B \leq span\ B$
using *convex_hull_subset_span*[*of insert 0 B*] **by** *auto*
then have $span\ (convex\ hull\ insert\ 0\ B) \leq span\ B$
using *span_span*[*of B*]
span_mono[*of convex hull insert 0 B span B*] **by** *blast*
then have ∗: $span\ (convex\ hull\ insert\ 0\ B) = span\ B$
using *span_mono*[*of B convex hull insert 0 B*] *hull_subset*[*of insert 0 B*] **by** *auto*
then have $span\ (convex\ hull\ insert\ 0\ B) = span\ S$
using *B span_mono*[*of B S*] *span_mono*[*of S span B*]
span_span[*of B*] **by** *auto*
moreover have $0 \in affine\ hull\ (convex\ hull\ insert\ 0\ B)$
using *hull_subset*[*of convex hull insert 0 B*] *hull_subset*[*of insert 0 B*] **by** *auto*
ultimately have ∗∗: $affine\ hull\ (convex\ hull\ insert\ 0\ B) = affine\ hull\ S$
using *affine_hull_span_0*[*of convex hull insert 0 B*] *affine_hull_span_0*[*of S*]
assms hull_subset[*of S*]
by *auto*
obtain *d* **and** *f* :: $'n \Rightarrow 'n$ **where**
fd: $card\ d = card\ B \wedge linear\ f \wedge f\ `\ B = d$
$\wedge\ f\ `\ span\ B = \{x.\ \forall i \in Basis.\ i \notin d \longrightarrow x \cdot i = (0::real)\} \wedge inj\_on\ f\ (span\ B)$
and *d*: $d \subseteq Basis$
using *basis_to_substdbasis_subspace_isomorphism*[*of B*,*OF _] B* **by** *auto*
then have *bounded_linear f*
using *linear_conv_bounded_linear* **by** *auto*
have $d \neq \{\}$
using *fd B* ‹$B \neq \{\}$› **by** *auto*
have $insert\ 0\ d = f\ `\ (insert\ 0\ B)$
using *fd linear_0* **by** *auto*
then have $(convex\ hull\ (insert\ 0\ d)) = f\ `\ (convex\ hull\ (insert\ 0\ B))$

```
    using convex_hull_linear_image[of f (insert 0 d)]
      convex_hull_linear_image[of f (insert 0 B)] ‹linear f›
    by auto
  moreover have rel_interior (f ‘ (convex hull insert 0 B)) = f ‘ rel_interior
(convex hull insert 0 B)
  proof (rule rel_interior_injective_on_span_linear_image[OF ‹bounded_linear
f›])
    show inj_on f (span (convex hull insert 0 B))
      using fd ∗ by auto
  qed
  ultimately have rel_interior (convex hull insert 0 B) ≠ {}
    using rel_interior_substd_simplex_nonempty[OF ‹d ≠ {}› d] by fastforce
  moreover have convex hull (insert 0 B) ⊆ S
    using B assms hull_mono[of insert 0 B S convex] convex_hull_eq by auto
  ultimately show ?thesis
    using subset_rel_interior[of convex hull insert 0 B S] ∗∗ by auto
qed

lemma rel_interior_eq_empty:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows rel_interior S = {} ⟷ S = {}
proof −
  {
    assume S ≠ {}
    then obtain a where a ∈ S by auto
    then have 0 ∈ (+) (−a) ‘ S
      using assms exI[of (λx. x ∈ S ∧ − a + x = 0) a] by auto
    then have rel_interior ((+) (−a) ‘ S) ≠ {}
      using rel_interior_convex_nonempty_aux[of (+) (−a) ‘ S]
        convex_translation[of S −a] assms
      by auto
    then have rel_interior S ≠ {}
      using rel_interior_translation [of − a] by simp
  }
  then show ?thesis by auto
qed

lemma interior_simplex_nonempty:
  fixes S :: 'N :: euclidean_space set
  assumes independent S finite S card S = DIM('N)
  obtains a where a ∈ interior (convex hull (insert 0 S))
proof −
  have affine hull (insert 0 S) = UNIV
    by (simp add: hull_inc affine_hull_span_0 dim_eq_full[symmetric]
        assms(1) assms(3) dim_eq_card_independent)
  moreover have rel_interior (convex hull insert 0 S) ≠ {}
    using rel_interior_eq_empty [of convex hull (insert 0 S)] by auto
  ultimately have interior (convex hull insert 0 S) ≠ {}
```

```
    by (simp add: rel_interior_interior)
  with that show ?thesis
    by auto
qed

lemma convex_rel_interior:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows convex (rel_interior S)
proof -
  {
    fix x y and u :: real
    assume assm: x ∈ rel_interior S y ∈ rel_interior S 0 ≤ u u ≤ 1
    then have x ∈ S
      using rel_interior_subset by auto
    have x - u *R (x-y) ∈ rel_interior S
    proof (cases 0 = u)
      case False
      then have 0 < u using assm by auto
      then show ?thesis
        using assm rel_interior_convex_shrink[of S y x u] assms ‹x ∈ S› by auto
    next
      case True
      then show ?thesis using assm by auto
    qed
    then have (1 - u) *R x + u *R y ∈ rel_interior S
      by (simp add: algebra_simps)
  }
  then show ?thesis
    unfolding convex_alt by auto
qed

lemma convex_closure_rel_interior:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows closure (rel_interior S) = closure S
proof -
  have h1: closure (rel_interior S) ≤ closure S
    using closure_mono[of rel_interior S S] rel_interior_subset[of S] by auto
  show ?thesis
  proof (cases S = {})
    case False
    then obtain a where a: a ∈ rel_interior S
      using rel_interior_eq_empty assms by auto
    { fix x
      assume x: x ∈ closure S
      {
        assume x = a
        then have x ∈ closure (rel_interior S)
```

```
      using a unfolding closure_def by auto
    }
    moreover
    {
     assume x ≠ a
      {
       fix e :: real
       assume e > 0
       define e1 where e1 = min 1 (e/norm (x − a))
       then have e1: e1 > 0 e1 ≤ 1 e1 ∗ norm (x − a) ≤ e
         using ‹x ≠ a› ‹e > 0› le_divide_eq[of e1 e norm (x − a)]
         by simp_all
       then have ∗: x − e1 ∗R (x − a) ∈ rel_interior S
       using rel_interior_closure_convex_shrink[of S a x e1] assms x a e1_def
         by auto
       have ∃ y. y ∈ rel_interior S ∧ y ≠ x ∧ dist y x ≤ e
         using ∗ ‹x ≠ a› e1 by force
      }
      then have x islimpt rel_interior S
        unfolding islimpt_approachable_le by auto
      then have x ∈ closure(rel_interior S)
        unfolding closure_def by auto
    }
    ultimately have x ∈ closure(rel_interior S) by auto
   }
   then show ?thesis using h1 by auto
  qed auto
qed

lemma rel_interior_same_affine_hull:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows affine hull (rel_interior S) = affine hull S
  by (metis assms closure_same_affine_hull convex_closure_rel_interior)

lemma rel_interior_aff_dim:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows aff_dim (rel_interior S) = aff_dim S
  by (metis aff_dim_affine_hull2 assms rel_interior_same_affine_hull)

lemma rel_interior_rel_interior:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows rel_interior (rel_interior S) = rel_interior S
proof −
  have openin (top_of_set (affine hull (rel_interior S))) (rel_interior S)
   using openin_rel_interior[of S] rel_interior_same_affine_hull[of S] assms by
auto
```

```
  then show ?thesis
    using rel_interior_def by auto
qed

lemma rel_interior_rel_open:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows rel_open (rel_interior S)
  unfolding rel_open_def using rel_interior_rel_interior assms by auto

lemma convex_rel_interior_closure_aux:
  fixes x y z :: 'n::euclidean_space
  assumes 0 < a 0 < b (a + b) *R z = a *R x + b *R y
  obtains e where 0 < e e < 1 z = y - e *R (y - x)
proof -
  define e where e = a / (a + b)
  have z = (1 / (a + b)) *R ((a + b) *R z)
    using assms  by (simp add: eq_vector_fraction_iff)
  also have ... = (1 / (a + b)) *R (a *R x + b *R y)
    using assms scaleR_cancel_left[of 1/(a+b) (a + b) *R z a *R x + b *R y]
    by auto
  also have ... = y - e *R (y-x)
    using e_def assms
   by (simp add: divide_simps vector_fraction_eq_iff) (simp add: algebra_simps)
  finally have z = y - e *R (y-x)
    by auto
  moreover have e > 0 e < 1 using e_def assms by auto
  ultimately show ?thesis using that[of e] by auto
qed

lemma convex_rel_interior_closure:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows rel_interior (closure S) = rel_interior S
proof (cases S = {})
  case True
  then show ?thesis
    using assms rel_interior_eq_empty by auto
next
  case False
  have rel_interior (closure S) ⊇ rel_interior S
   using subset_rel_interior[of S closure S] closure_same_affine_hull closure_subset
    by auto
  moreover
  {
    fix z
    assume z: z ∈ rel_interior (closure S)
    obtain x where x: x ∈ rel_interior S
      using ‹S ≠ {}› assms rel_interior_eq_empty by auto
```

```
    have z ∈ rel_interior S
    proof (cases x = z)
      case True
      then show ?thesis using x by auto
    next
      case False
      obtain e where e: e > 0 cball z e ∩ affine hull closure S ≤ closure S
        using z rel_interior_cball[of closure S] by auto
      hence ∗: 0 < e/norm(z−x) using e False by auto
      define y where y = z + (e/norm(z−x)) ∗R (z−x)
      have yball: y ∈ cball z e
        using y_def dist_norm[of z y] e by auto
      have x ∈ affine hull closure S
        using x rel_interior_subset_closure hull_inc[of x closure S] by blast
      moreover have z ∈ affine hull closure S
        using z rel_interior_subset hull_subset[of closure S] by blast
      ultimately have y ∈ affine hull closure S
        using y_def affine_affine_hull[of closure S]
         mem_affine_3_minus [of affine hull closure S z z x e/norm(z−x)] by auto
      then have y ∈ closure S using e yball by auto
      have (1 + (e/norm(z−x))) ∗R z = (e/norm(z−x)) ∗R x + y
        using y_def by (simp add: algebra_simps)
      then obtain e1 where 0 < e1 e1 < 1 z = y − e1 ∗R (y − x)
        using ∗ convex_rel_interior_closure_aux[of e / norm (z − x) 1 z x y]
        by (auto simp add: algebra_simps)
      then show ?thesis
        using rel_interior_closure_convex_shrink assms x ‹y ∈ closure S›
        by fastforce
    qed
  }
  ultimately show ?thesis by auto
qed

lemma convex_interior_closure:
  fixes S :: 'n::euclidean_space set
  assumes convex S
  shows interior (closure S) = interior S
  using closure_aff_dim[of S] interior_rel_interior_gen[of S]
    interior_rel_interior_gen[of closure S]
    convex_rel_interior_closure[of S] assms
  by auto

lemma open_subset_closure_of_interval:
  assumes open U is_interval S
  shows U ⊆ closure S ⟷ U ⊆ interior S
 by (metis assms convex_interior_closure is_interval_convex open_subset_interior)

lemma closure_eq_rel_interior_eq:
  fixes S1 S2 :: 'n::euclidean_space set
```

assumes *convex S1*
and *convex S2*
shows *closure S1 = closure S2* $\longleftrightarrow$ *rel_interior S1 = rel_interior S2*
by (*metis convex_rel_interior_closure convex_closure_rel_interior assms*)

lemma *closure_eq_between*:
fixes *S1 S2* :: *'n*::*euclidean_space set*
assumes *convex S1*
and *convex S2*
shows *closure S1 = closure S2* $\longleftrightarrow$ *rel_interior S1* $\leq$ *S2* $\wedge$ *S2* $\subseteq$ *closure S1*
(**is** *?A* $\longleftrightarrow$ *?B*)
proof
assume *?A*
then show *?B*
by (*metis assms closure_subset convex_rel_interior_closure rel_interior_subset*)
next
assume *?B*
then have *closure S1* $\subseteq$ *closure S2*
by (*metis assms*(*1*) *convex_closure_rel_interior closure_mono*)
moreover from ‹*?B*› **have** *closure S1* $\supseteq$ *closure S2*
by (*metis closed_closure closure_minimal*)
ultimately show *?A* **..**
qed

lemma *open_inter_closure_rel_interior*:
fixes *S A* :: *'n*::*euclidean_space set*
assumes *convex S*
and *open A*
shows *A* $\cap$ *closure S* = {} $\longleftrightarrow$ *A* $\cap$ *rel_interior S* = {}
by (*metis assms convex_closure_rel_interior open_Int_closure_eq_empty*)

lemma *rel_interior_open_segment*:
fixes *a* :: *'a* :: *euclidean_space*
shows *rel_interior*(*open_segment a b*) = *open_segment a b*
proof (*cases a* = *b*)
case *True* **then show** *?thesis* **by** *auto*
next
case *False* **then**
have *open_segment a b = affine hull* {*a*, *b*} $\cap$ *ball* ((*a* + *b*) /$_R$ *2*) (*norm* (*b* − *a*) / *2*)
by (*simp add*: *open_segment_as_ball*)
then show *?thesis*
unfolding *rel_interior_eq openin_open*
by (*metis Elementary_Metric_Spaces.open_ball False affine_hull_open_segment*)
qed

lemma *rel_interior_closed_segment*:
fixes *a* :: *'a* :: *euclidean_space*
shows *rel_interior*(*closed_segment a b*) =

(*if* $a = b$ *then* $\{a\}$ *else open_segment* a b)
proof (*cases* $a = b$)
case *True* **then show** *?thesis* **by** *auto*
next
case *False* **then show** *?thesis*
by *simp*
(*metis closure_open_segment convex_open_segment convex_rel_interior_closure*
rel_interior_open_segment)
qed

lemmas *rel_interior_segment* = *rel_interior_closed_segment rel_interior_open_segment*

5.0.4 The relative frontier of a set

definition *rel_frontier* S = *closure* S − *rel_interior* S

lemma *rel_frontier_empty* [*simp*]: *rel_frontier* {} = {}
by (*simp add*: *rel_frontier_def*)

lemma *rel_frontier_eq_empty*:
fixes S :: *'n::euclidean_space set*
shows *rel_frontier* S = {} ⟷ *affine* S
unfolding *rel_frontier_def*
using *rel_interior_subset_closure* **by** (*auto simp add*: *rel_interior_eq_closure* [*symmetric*])

lemma *rel_frontier_sing* [*simp*]:
fixes a :: *'n::euclidean_space*
shows *rel_frontier* $\{a\}$ = {}
by (*simp add*: *rel_frontier_def*)

lemma *rel_frontier_affine_hull*:
fixes S :: *'a::euclidean_space set*
shows *rel_frontier* $S \subseteq$ *affine hull* S
using *closure_affine_hull rel_frontier_def* **by** *fastforce*

lemma *rel_frontier_cball* [*simp*]:
fixes a :: *'n::euclidean_space*
shows *rel_frontier*(*cball* a r) = (*if* $r = 0$ *then* {} *else sphere* a r)
proof (*cases rule*: *linorder_cases* [*of* r *0*])
case *less* **then show** *?thesis*
by (*force simp*: *sphere_def*)
next
case *equal* **then show** *?thesis* **by** *simp*
next
case *greater* **then show** *?thesis*
by *simp* (*metis centre_in_ball empty_iff frontier_cball frontier_def interior_cball interior_rel_interior_gen rel_frontier_def*)
qed

lemma *rel_frontier_translation*:
fixes *a* :: *'a*::*euclidean_space*
shows *rel_frontier*((λ*x*. *a* + *x*) ' *S*) = (λ*x*. *a* + *x*) ' (*rel_frontier S*)
by (*simp add*: *rel_frontier_def translation_diff rel_interior_translation closure_translation*)

lemma *rel_frontier_nonempty_interior*:
fixes *S* :: *'n*::*euclidean_space set*
shows *interior S* ≠ {} ⟹ *rel_frontier S* = *frontier S*
by (*metis frontier_def interior_rel_interior_gen rel_frontier_def*)

lemma *rel_frontier_frontier*:
fixes *S* :: *'n*::*euclidean_space set*
shows *affine hull S* = *UNIV* ⟹ *rel_frontier S* = *frontier S*
by (*simp add*: *frontier_def rel_frontier_def rel_interior_interior*)

lemma *closest_point_in_rel_frontier*:
⟦*closed S*; *S* ≠ {}; *x* ∈ *affine hull S* − *rel_interior S*⟧
⟹ *closest_point S x* ∈ *rel_frontier S*
by (*simp add*: *closest_point_in_rel_interior closest_point_in_set rel_frontier_def*)

lemma *closed_rel_frontier* [*iff*]:
fixes *S* :: *'n*::*euclidean_space set*
shows *closed* (*rel_frontier S*)
proof −
have ∗: *closedin* (*top_of_set* (*affine hull S*)) (*closure S* − *rel_interior S*)
by (*simp add*: *closed_subset closedin_diff closure_affine_hull openin_rel_interior*)
show *?thesis*
proof (*rule closedin_closed_trans*[*of affine hull S rel_frontier S*])
show *closedin* (*top_of_set* (*affine hull S*)) (*rel_frontier S*)
by (*simp add*: ∗ *rel_frontier_def*)
qed *simp*
qed

lemma *closed_rel_boundary*:
fixes *S* :: *'n*::*euclidean_space set*
shows *closed S* ⟹ *closed*(*S* − *rel_interior S*)
by (*metis closed_rel_frontier closure_closed rel_frontier_def*)

lemma *compact_rel_boundary*:
fixes *S* :: *'n*::*euclidean_space set*
shows *compact S* ⟹ *compact*(*S* − *rel_interior S*)
by (*metis bounded_diff closed_rel_boundary closure_eq compact_closure compact_imp_closed*)

lemma *bounded_rel_frontier*:
fixes *S* :: *'n*::*euclidean_space set*
shows *bounded S* ⟹ *bounded*(*rel_frontier S*)

by (*simp add*: *bounded_closure bounded_diff rel_frontier_def*)

lemma *compact_rel_frontier_bounded*:
fixes *S* :: *'n*::*euclidean_space set*
shows *bounded S* ⟹ *compact*(*rel_frontier S*)
using *bounded_rel_frontier closed_rel_frontier compact_eq_bounded_closed* **by** *blast*

lemma *compact_rel_frontier*:
fixes *S* :: *'n*::*euclidean_space set*
shows *compact S* ⟹ *compact*(*rel_frontier S*)
by (*meson compact_eq_bounded_closed compact_rel_frontier_bounded*)

lemma *convex_same_rel_interior_closure*:
fixes *S* :: *'n*::*euclidean_space set*
shows ⟦*convex S*; *convex T*⟧
⟹ *rel_interior S* = *rel_interior T* ⟷ *closure S* = *closure T*
by (*simp add*: *closure_eq_rel_interior_eq*)

lemma *convex_same_rel_interior_closure_straddle*:
fixes *S* :: *'n*::*euclidean_space set*
shows ⟦*convex S*; *convex T*⟧
⟹ *rel_interior S* = *rel_interior T* ⟷
rel_interior S ⊆ *T* ∧ *T* ⊆ *closure S*
by (*simp add*: *closure_eq_between convex_same_rel_interior_closure*)

lemma *convex_rel_frontier_aff_dim*:
fixes *S1 S2* :: *'n*::*euclidean_space set*
assumes *convex S1*
and *convex S2*
and *S2* ≠ {}
and *S1* ≤ *rel_frontier S2*
shows *aff_dim S1* < *aff_dim S2*
proof −
have *S1* ⊆ *closure S2*
using *assms* **unfolding** *rel_frontier_def* **by** *auto*
then have ∗: *affine hull S1* ⊆ *affine hull S2*
using *hull_mono*[*of S1 closure S2*] *closure_same_affine_hull*[*of S2*] **by** *blast*
then have *aff_dim S1* ≤ *aff_dim S2*
using ∗ *aff_dim_affine_hull*[*of S1*] *aff_dim_affine_hull*[*of S2*]
aff_dim_subset[*of affine hull S1 affine hull S2*]
by *auto*
moreover
{
assume *eq*: *aff_dim S1* = *aff_dim S2*
then have *S1* ≠ {}
using *aff_dim_empty*[*of S1*] *aff_dim_empty*[*of S2*] ‹*S2* ≠ {}› **by** *auto*
have ∗∗: *affine hull S1* = *affine hull S2*
by (*simp_all add*: ∗ *eq* ‹*S1* ≠ {}› *affine_dim_equal*)

```
    obtain a where a: a ∈ rel_interior S1
      using ‹S1 ≠ {}› rel_interior_eq_empty assms by auto
    obtain T where T: open T a ∈ T ∩ S1 T ∩ affine hull S1 ⊆ S1
       using mem_rel_interior[of a S1] a by auto
    then have a ∈ T ∩ closure S2
      using a assms unfolding rel_frontier_def by auto
    then obtain b where b: b ∈ T ∩ rel_interior S2
      using open_inter_closure_rel_interior[of S2 T] assms T by auto
    then have b ∈ affine hull S1
      using rel_interior_subset hull_subset[of S2] ** by auto
    then have b ∈ S1
      using T b by auto
    then have False
      using b assms unfolding rel_frontier_def by auto
  }
  ultimately show ?thesis
    using less_le by auto
qed

lemma convex_rel_interior_if:
  fixes S :: 'n::euclidean_space set
  assumes convex S
    and z ∈ rel_interior S
  shows ∀ x∈affine hull S. ∃ m. m > 1 ∧ (∀ e. e > 1 ∧ e ≤ m ⟶ (1 − e) *R x
+ e *R z ∈ S)
proof −
  obtain e1 where e1: e1 > 0 ∧ cball z e1 ∩ affine hull S ⊆ S
    using mem_rel_interior_cball[of z S] assms by auto
  {
    fix x
    assume x: x ∈ affine hull S
    {
      assume x ≠ z
      define m where m = 1 + e1/norm(x−z)
      hence m > 1 using e1 ‹x ≠ z› by auto
      {
        fix e
        assume e: e > 1 ∧ e ≤ m
        have z ∈ affine hull S
          using assms rel_interior_subset hull_subset[of S] by auto
        then have ∗: (1 − e)∗R x + e *R z ∈ affine hull S
          using mem_affine[of affine hull S x z (1−e) e] affine_affine_hull[of S] x
          by auto
        have norm (z + e *R x − (x + e *R z)) = norm ((e − 1) *R (x − z))
          by (simp add: algebra_simps)
        also have ... = (e − 1) ∗ norm (x−z)
          using norm_scaleR e by auto
        also have ... ≤ (m − 1) ∗ norm (x − z)
          using e mult_right_mono[of _ _ norm(x−z)] by auto
```

```
      also have ... = (e1 / norm (x - z)) * norm (x - z)
        using m_def by auto
      also have ... = e1
        using ‹x ≠ z› e1 by simp
      finally have **: norm (z + e *R x - (x + e *R z)) ≤ e1
        by auto
      have (1 - e)*R x+ e *R z ∈ cball z e1
        using m_def **
        unfolding cball_def dist_norm
        by (auto simp add: algebra_simps)
      then have (1 - e) *R x+ e *R z ∈ S
        using e * e1 by auto
    }
    then have ∃m. m > 1 ∧ (∀e. e > 1 ∧ e ≤ m ⟶ (1 - e) *R x + e *R z
∈ S )
      using ‹m> 1 › by auto
  }
  moreover
  {
    assume x = z
    define m where m = 1 + e1
    then have m > 1
      using e1 by auto
    {
      fix e
      assume e: e > 1 ∧ e ≤ m
      then have (1 - e) *R x + e *R z ∈ S
        using e1 x ‹x = z› by (auto simp add: algebra_simps)
      then have (1 - e) *R x + e *R z ∈ S
        using e by auto
    }
    then have ∃m. m > 1 ∧ (∀e. e > 1 ∧ e ≤ m ⟶ (1 - e) *R x + e *R z
∈ S)
      using ‹m > 1› by auto
  }
  ultimately have ∃m. m > 1 ∧ (∀e. e > 1 ∧ e ≤ m ⟶ (1 - e) *R x + e
*R z ∈ S )
    by blast
 }
 then show ?thesis by auto
qed

lemma convex_rel_interior_if2:
 fixes S :: 'n::euclidean_space set
 assumes convex S
 assumes z ∈ rel_interior S
 shows ∀x∈affine hull S. ∃e. e > 1 ∧ (1 - e)*R x + e *R z ∈ S
 using convex_rel_interior_if[of S z] assms by auto
```

lemma *convex_rel_interior_only_if*:
fixes S :: *'n::euclidean_space set*
assumes *convex S*
and $S \neq \{\}$
assumes $\forall x{\in}S.\ \exists e.\ e > 1 \wedge (1 - e) *_R x + e *_R z \in S$
shows $z \in$ *rel_interior S*
proof −
obtain x **where** x: $x \in$ *rel_interior S*
using *rel_interior_eq_empty assms* **by** *auto*
then have $x \in S$
using *rel_interior_subset* **by** *auto*
then obtain e **where** e: $e > 1 \wedge (1 - e) *_R x + e *_R z \in S$
using *assms* **by** *auto*
define y **where** [*abs_def*]: $y = (1 - e) *_R x + e *_R z$
then have $y \in S$ **using** e **by** *auto*
define *e1* **where** *e1* = *1*/*e*
then have *0* < *e1* ∧ *e1* < *1* **using** e **by** *auto*
then have $z = y - (1 - e1) *_R (y - x)$
using *e1_def y_def* **by** (*auto simp add*: *algebra_simps*)
then show *?thesis*
using *rel_interior_convex_shrink*[*of S x y 1−e1*] ‹*0* < *e1* ∧ *e1* < *1*› ‹*y* ∈ *S*› *x assms*
by *auto*
qed

lemma *convex_rel_interior_iff*:
fixes S :: *'n::euclidean_space set*
assumes *convex S*
and $S \neq \{\}$
shows $z \in$ *rel_interior S* $\longleftrightarrow$ $(\forall x{\in}S.\ \exists e.\ e > 1 \wedge (1 - e) *_R x + e *_R z \in S)$
using *assms hull_subset*[*of S affine*]
convex_rel_interior_if[*of S z*] *convex_rel_interior_only_if*[*of S z*]
by *auto*

lemma *convex_rel_interior_iff2*:
fixes S :: *'n::euclidean_space set*
assumes *convex S*
and $S \neq \{\}$
shows $z \in$ *rel_interior S* $\longleftrightarrow$ $(\forall x{\in}$*affine hull S*$.\ \exists e.\ e > 1 \wedge (1 - e) *_R x + e *_R z \in S)$
using *assms hull_subset*[*of S*] *convex_rel_interior_if2*[*of S z*] *convex_rel_interior_only_if*[*of S z*]
by *auto*

lemma *convex_interior_iff*:
fixes S :: *'n::euclidean_space set*
assumes *convex S*
shows $z \in$ *interior S* $\longleftrightarrow$ $(\forall x.\ \exists e.\ e > 0 \wedge z + e *_R x \in S)$
proof (*cases aff_dim S = int DIM*(*'n*))

```
  case False
  { assume z ∈ interior S
    then have False
      using False interior_rel_interior_gen[of S] by auto }
  moreover
  { assume r: ∀x. ∃e. e > 0 ∧ z + e *R x ∈ S
    { fix x
      obtain e1 where e1: e1 > 0 ∧ z + e1 *R (x − z) ∈ S
        using r by auto
      obtain e2 where e2: e2 > 0 ∧ z + e2 *R (z − x) ∈ S
        using r by auto
      define x1 where [abs_def]: x1 = z + e1 *R (x − z)
      then have x1: x1 ∈ affine hull S
        using e1 hull_subset[of S] by auto
      define x2 where [abs_def]: x2 = z + e2 *R (z − x)
      then have x2: x2 ∈ affine hull S
        using e2 hull_subset[of S] by auto
      have ∗: e1/(e1+e2) + e2/(e1+e2) = 1
        using add_divide_distrib[of e1 e2 e1+e2] e1 e2 by simp
      then have z = (e2/(e1+e2)) *R x1 + (e1/(e1+e2)) *R x2
        by (simp add: x1_def x2_def algebra_simps) (simp add: ∗ pth_8)
      then have z: z ∈ affine hull S
        using mem_affine[of affine hull S x1 x2 e2/(e1+e2) e1/(e1+e2)]
          x1 x2 affine_affine_hull[of S] ∗
        by auto
      have x1 − x2 = (e1 + e2) *R (x − z)
        using x1_def x2_def by (auto simp add: algebra_simps)
      then have x = z+(1/(e1+e2)) *R (x1−x2)
        using e1 e2 by simp
      then have x ∈ affine hull S
        using mem_affine_3_minus[of affine hull S z x1 x2 1/(e1+e2)]
          x1 x2 z affine_affine_hull[of S]
        by auto
    }
    then have affine hull S = UNIV
      by auto
    then have aff_dim S = int DIM('n)
      using aff_dim_affine_hull[of S] by (simp)
    then have False
      using False by auto
  }
  ultimately show ?thesis by auto
next
  case True
  then have S ≠ {}
    using aff_dim_empty[of S] by auto
  have ∗: affine hull S = UNIV
    using True affine_hull_UNIV by auto
  {
```

```
    assume z ∈ interior S
    then have z ∈ rel_interior S
      using True interior_rel_interior_gen[of S] by auto
    then have ∗∗: ∀ x. ∃ e. e > 1 ∧ (1 − e) ∗R x + e ∗R z ∈ S
      using convex_rel_interior_iff2[of S z] assms ‹S ≠ {}› ∗ by auto
    fix x
    obtain e1 where e1: e1 > 1 (1 − e1) ∗R (z − x) + e1 ∗R z ∈ S
      using ∗∗[rule_format, of z−x] by auto
    define e where [abs_def]: e = e1 − 1
    then have (1 − e1) ∗R (z − x) + e1 ∗R z = z + e ∗R x
      by (simp add: algebra_simps)
    then have e > 0 z + e ∗R x ∈ S
      using e1 e_def by auto
    then have ∃ e. e > 0 ∧ z + e ∗R x ∈ S
      by auto
  }
  moreover
  {
    assume r: ∀ x. ∃ e. e > 0 ∧ z + e ∗R x ∈ S
    {
      fix x
      obtain e1 where e1: e1 > 0 z + e1 ∗R (z − x) ∈ S
        using r[rule_format, of z−x] by auto
      define e where e = e1 + 1
      then have z + e1 ∗R (z − x) = (1 − e) ∗R x + e ∗R z
        by (simp add: algebra_simps)
      then have e > 1 (1 − e)∗R x + e ∗R z ∈ S
        using e1 e_def by auto
      then have ∃ e. e > 1 ∧ (1 − e) ∗R x + e ∗R z ∈ S by auto
    }
    then have z ∈ rel_interior S
      using convex_rel_interior_iff2[of S z] assms ‹S ≠ {}› by auto
    then have z ∈ interior S
      using True interior_rel_interior_gen[of S] by auto
  }
  ultimately show ?thesis by auto
qed
```

Relative interior and closure under common operations

```
lemma rel_interior_inter_aux: ⋂{rel_interior S |S. S ∈ I} ⊆ ⋂I
proof −
  {
    fix y
    assume y ∈ ⋂{rel_interior S |S. S ∈ I}
    then have y: ∀ S ∈ I. y ∈ rel_interior S
      by auto
    {
      fix S
```

```
    assume S ∈ I
    then have y ∈ S
      using rel_interior_subset y by auto
   }
   then have y ∈ ⋂I by auto
  }
  then show ?thesis by auto
qed

lemma convex_closure_rel_interior_inter:
  assumes ∀S∈I. convex (S :: 'n::euclidean_space set)
    and ⋂{rel_interior S |S. S ∈ I} ≠ {}
  shows ⋂{closure S |S. S ∈ I} ≤ closure (⋂{rel_interior S |S. S ∈ I})
proof −
  obtain x where x: ∀S∈I. x ∈ rel_interior S
    using assms by auto
  {
   fix y
   assume y ∈ ⋂{closure S |S. S ∈ I}
   then have y: ∀S ∈ I. y ∈ closure S
     by auto
   {
    assume y = x
    then have y ∈ closure (⋂{rel_interior S |S. S ∈ I})
      using x closure_subset[of ⋂{rel_interior S |S. S ∈ I}] by auto
   }
   moreover
   {
    assume y ≠ x
    { fix e :: real
     assume e: e > 0
     define e1 where e1 = min 1 (e/norm (y − x))
     then have e1: e1 > 0 e1 ≤ 1 e1 ∗ norm (y − x) ≤ e
       using ‹y ≠ x› ‹e > 0› le_divide_eq[of e1 e norm (y − x)]
       by simp_all
     define z where z = y − e1 ∗R (y − x)
     {
      fix S
      assume S ∈ I
      then have z ∈ rel_interior S
          using rel_interior_closure_convex_shrink[of S x y e1] assms x y e1
z_def
        by auto
     }
     then have ∗: z ∈ ⋂{rel_interior S |S. S ∈ I}
       by auto
     have ∃z. z ∈ ⋂{rel_interior S |S. S ∈ I} ∧ z ≠ y ∧ dist z y ≤ e
       using ‹y ≠ x› z_def ∗ e1 e dist_norm[of z y]
       by (rule_tac x=z in exI) auto
```

```
    }
    then have y islimpt ⋂{rel_interior S |S. S ∈ I}
      unfolding islimpt_approachable_le by blast
    then have y ∈ closure (⋂{rel_interior S |S. S ∈ I})
      unfolding closure_def by auto
  }
  ultimately have y ∈ closure (⋂{rel_interior S |S. S ∈ I})
    by auto
 }
 then show ?thesis by auto
qed

lemma convex_closure_inter:
 assumes ∀S∈I. convex (S :: 'n::euclidean_space set)
  and ⋂{rel_interior S |S. S ∈ I} ≠ {}
 shows closure (⋂I) = ⋂{closure S |S. S ∈ I}
proof −
 have ⋂{closure S |S. S ∈ I} ≤ closure (⋂{rel_interior S |S. S ∈ I})
  using convex_closure_rel_interior_inter assms by auto
 moreover
 have closure (⋂{rel_interior S |S. S ∈ I}) ≤ closure (⋂I)
  using rel_interior_inter_aux closure_mono[of ⋂{rel_interior S |S. S ∈ I}
⋂I]
  by auto
 ultimately show ?thesis
  using closure_Int[of I] by auto
qed

lemma convex_inter_rel_interior_same_closure:
 assumes ∀S∈I. convex (S :: 'n::euclidean_space set)
  and ⋂{rel_interior S |S. S ∈ I} ≠ {}
 shows closure (⋂{rel_interior S |S. S ∈ I}) = closure (⋂I)
proof −
 have ⋂{closure S |S. S ∈ I} ≤ closure (⋂{rel_interior S |S. S ∈ I})
  using convex_closure_rel_interior_inter assms by auto
 moreover
 have closure (⋂{rel_interior S |S. S ∈ I}) ≤ closure (⋂I)
  using rel_interior_inter_aux closure_mono[of ⋂{rel_interior S |S. S ∈ I}
⋂I]
  by auto
 ultimately show ?thesis
  using closure_Int[of I] by auto
qed

lemma convex_rel_interior_inter:
 assumes ∀S∈I. convex (S :: 'n::euclidean_space set)
  and ⋂{rel_interior S |S. S ∈ I} ≠ {}
 shows rel_interior (⋂I) ⊆ ⋂{rel_interior S |S. S ∈ I}
proof −
```

```
  have convex (⋂ I)
    using assms convex_Inter by auto
  moreover
  have convex (⋂{rel_interior S |S. S ∈ I})
    using assms convex_rel_interior by (force intro: convex_Inter)
  ultimately
  have rel_interior (⋂{rel_interior S |S. S ∈ I}) = rel_interior (⋂ I)
    using convex_inter_rel_interior_same_closure assms
      closure_eq_rel_interior_eq[of ⋂{rel_interior S |S. S ∈ I} ⋂ I]
    by blast
  then show ?thesis
    using rel_interior_subset[of ⋂{rel_interior S |S. S ∈ I}] by auto
qed

lemma convex_rel_interior_finite_inter:
  assumes ∀ S∈I. convex (S :: 'n::euclidean_space set)
    and ⋂{rel_interior S |S. S ∈ I} ≠ {}
    and finite I
  shows rel_interior (⋂ I) = ⋂{rel_interior S |S. S ∈ I}
proof −
  have ⋂ I ≠ {}
    using assms rel_interior_inter_aux[of I] by auto
  have convex (⋂ I)
    using convex_Inter assms by auto
  show ?thesis
  proof (cases I = {})
    case True
    then show ?thesis
      using Inter_empty rel_interior_UNIV by auto
  next
    case False
    {
      fix z
      assume z: z ∈ ⋂{rel_interior S |S. S ∈ I}
      {
        fix x
        assume x: x ∈ ⋂ I
        {
          fix S
          assume S: S ∈ I
          then have z ∈ rel_interior S x ∈ S
            using z x by auto
          then have ∃ m. m > 1 ∧ (∀ e. e > 1 ∧ e ≤ m ⟶ (1 − e)*R x + e *R
z ∈ S)
            using convex_rel_interior_if[of S z] S assms hull_subset[of S] by auto
        }
        then obtain mS where
          mS: ∀ S∈I. mS S > 1 ∧ (∀ e. e > 1 ∧ e ≤ mS S ⟶ (1 − e) *R x + e
*R z ∈ S) by metis
```

```
      define e where e = Min (mS ‘ I)
      then have e ∈ mS ‘ I using assms ‹I ≠ {}› by simp
      then have e > 1 using mS by auto
      moreover have ∀ S∈I. e ≤ mS S
        using e_def assms by auto
      ultimately have ∃ e > 1. (1 − e) *R x + e *R z ∈ ⋂ I
        using mS by auto
    }
    then have z ∈ rel_interior (⋂ I)
     using convex_rel_interior_iff[of ⋂ I z] ‹⋂ I ≠ {}› ‹convex (⋂ I)› by auto
  }
  then show ?thesis
    using convex_rel_interior_inter[of I] assms by auto
 qed
qed

lemma convex_closure_inter_two:
 fixes S T :: 'n::euclidean_space set
 assumes convex S
   and convex T
 assumes rel_interior S ∩ rel_interior T ≠ {}
 shows closure (S ∩ T) = closure S ∩ closure T
 using convex_closure_inter[of {S,T}] assms by auto

lemma convex_rel_interior_inter_two:
 fixes S T :: 'n::euclidean_space set
 assumes convex S
   and convex T
   and rel_interior S ∩ rel_interior T ≠ {}
 shows rel_interior (S ∩ T) = rel_interior S ∩ rel_interior T
 using convex_rel_interior_finite_inter[of {S,T}] assms by auto

lemma convex_affine_closure_Int:
 fixes S T :: 'n::euclidean_space set
 assumes convex S
   and affine T
   and rel_interior S ∩ T ≠ {}
 shows closure (S ∩ T) = closure S ∩ T
 by (metis affine_imp_convex assms convex_closure_inter_two rel_interior_affine
rel_interior_eq_closure)

lemma connected_component_1_gen:
 fixes S :: 'a :: euclidean_space set
 assumes DIM('a) = 1
 shows connected_component S a b ⟷ closed_segment a b ⊆ S
unfolding connected_component_def
by (metis (no_types, lifting) assms subsetD subsetI convex_contains_segment con-
vex_segment(1)
          ends_in_segment connected_convex_1_gen)
```

lemma *connected_component_1*:
fixes S :: *real set*
shows *connected_component S a b* $\longleftrightarrow$ *closed_segment a b* $\subseteq S$
by (*simp add*: *connected_component_1_gen*)

lemma *convex_affine_rel_interior_Int*:
fixes S T :: $'n$::*euclidean_space set*
assumes *convex S*
and *affine T*
and *rel_interior S* $\cap$ $T \neq \{\}$
shows *rel_interior* $(S \cap T)$ = *rel_interior S* $\cap$ T
by (*simp add*: *affine_imp_convex assms convex_rel_interior_inter_two rel_interior_affine*)

lemma *convex_affine_rel_frontier_Int*:
fixes S T :: $'n$::*euclidean_space set*
assumes *convex S*
and *affine T*
and *interior S* $\cap$ $T \neq \{\}$
shows *rel_frontier*$(S \cap T)$ = *frontier S* $\cap$ T
using *assms*
unfolding *rel_frontier_def frontier_def*
using *convex_affine_closure_Int convex_affine_rel_interior_Int rel_interior_nonempty_interior*
by *fastforce*

lemma *rel_interior_convex_Int_affine*:
fixes S :: $'a$::*euclidean_space set*
assumes *convex S affine T interior S* $\cap$ $T \neq \{\}$
shows *rel_interior*$(S \cap T)$ = *interior S* $\cap$ T
by (*metis Int_emptyI assms convex_affine_rel_interior_Int empty_iff interior_rel_interior_gen*)

lemma *subset_rel_interior_convex*:
fixes S T :: $'n$::*euclidean_space set*
assumes *convex S*
and *convex T*
and $S \leq$ *closure T*
and $\neg$ $S \subseteq$ *rel_frontier T*
shows *rel_interior S* $\subseteq$ *rel_interior T*
proof −
have $*$: *S* $\cap$ *closure T* = *S*
using *assms* **by** *auto*
have $\neg$ *rel_interior S* $\subseteq$ *rel_frontier T*
using *closure_mono*[*of rel_interior S rel_frontier T*] *closed_rel_frontier*[*of T*]
closure_closed[*of S*] *convex_closure_rel_interior*[*of S*] *closure_subset*[*of S*]
assms
by *auto*
then have *rel_interior S* $\cap$ *rel_interior* (*closure T*) $\neq \{\}$
using *assms rel_frontier_def*[*of T*] *rel_interior_subset convex_rel_interior_closure*[*of*

```
T]
    by auto
  then have rel_interior S ∩ rel_interior T = rel_interior (S ∩ closure T)
    using assms convex_closure convex_rel_interior_inter_two[of S closure T]
      convex_rel_interior_closure[of T]
    by auto
  also have ... = rel_interior S
    using * by auto
  finally show ?thesis
    by auto
qed

lemma rel_interior_convex_linear_image:
  fixes f :: 'm::euclidean_space ⇒ 'n::euclidean_space
  assumes linear f
    and convex S
  shows f ` (rel_interior S) = rel_interior (f ` S)
proof (cases S = {})
  case True
  then show ?thesis
    using assms by auto
next
  case False
  interpret linear f by fact
  have *: f ` (rel_interior S) ⊆ f ` S
    unfolding image_mono using rel_interior_subset by auto
  have f ` S ⊆ f ` (closure S)
    unfolding image_mono using closure_subset by auto
  also have ... = f ` (closure (rel_interior S))
    using convex_closure_rel_interior assms by auto
  also have ... ⊆ closure (f ` (rel_interior S))
    using closure_linear_image_subset assms by auto
  finally have closure (f ` S) = closure (f ` rel_interior S)
    using closure_mono[of f ` S closure (f ` rel_interior S)] closure_closure
      closure_mono[of f ` rel_interior S f ` S] *
    by auto
  then have rel_interior (f ` S) = rel_interior (f ` rel_interior S)
    using assms convex_rel_interior
      linear_conv_bounded_linear[of f] convex_linear_image[of _ S]
      convex_linear_image[of _ rel_interior S]
      closure_eq_rel_interior_eq[of f ` S f ` rel_interior S]
    by auto
  then have rel_interior (f ` S) ⊆ f ` rel_interior S
    using rel_interior_subset by auto
  moreover
  {
    fix z
    assume z ∈ f ` rel_interior S
    then obtain z1 where z1: z1 ∈ rel_interior S f z1 = z by auto
```

```
    {
      fix x
      assume x ∈ f ‘ S
      then obtain x1 where x1: x1 ∈ S f x1 = x by auto
      then obtain e where e: e > 1 (1 − e) *R x1 + e *R z1 ∈ S
        using convex_rel_interior_iff[of S z1] ‹convex S› x1 z1 by auto
      moreover have f ((1 − e) *R x1 + e *R z1) = (1 − e) *R x + e *R z
        using x1 z1 by (simp add: linear_add linear_scale ‹linear f›)
      ultimately have (1 − e) *R x + e *R z ∈ f ‘ S
        using imageI[of (1 − e) *R x1 + e *R z1 S f] by auto
      then have ∃ e. e > 1 ∧ (1 − e) *R x + e *R z ∈ f ‘ S
        using e by auto
    }
    then have z ∈ rel_interior (f ‘ S)
      using convex_rel_interior_iff[of f ‘ S z] ‹convex S› ‹linear f›
        ‹S ≠ {}› convex_linear_image[of f S]  linear_conv_bounded_linear[of f]
      by auto
  }
  ultimately show ?thesis by auto
qed

lemma rel_interior_convex_linear_preimage:
  fixes f :: 'm::euclidean_space ⇒ 'n::euclidean_space
  assumes linear f
    and convex S
    and f −‘ (rel_interior S) ≠ {}
  shows rel_interior (f −‘ S) = f −‘ (rel_interior S)
proof −
  interpret linear f by fact
  have S ≠ {}
    using assms by auto
  have nonemp: f −‘ S ≠ {}
    by (metis assms(3) rel_interior_subset subset_empty vimage_mono)
  then have S ∩ (range f) ≠ {}
    by auto
  have conv: convex (f −‘ S)
    using convex_linear_vimage assms by auto
  then have convex (S ∩ range f)
    by (simp add: assms(2) convex_Int convex_linear_image linear_axioms)
  {
    fix z
    assume z ∈ f −‘ (rel_interior S)
    then have z: f z ∈ rel_interior S
      by auto
    {
      fix x
      assume x ∈ f −‘ S
      then have f x ∈ S by auto
      then obtain e where e: e > 1 (1 − e) *R f x + e *R f z ∈ S
```

```
      using convex_rel_interior_iff[of S f z] z assms ‹S ≠ {}› by auto
    moreover have (1 − e) *R f x + e *R f z = f ((1 − e) *R x + e *R z)
      using ‹linear f› by (simp add: linear_iff)
    ultimately have ∃ e. e > 1 ∧ (1 − e) *R x + e *R z ∈ f −‘ S
      using e by auto
  }
  then have z ∈ rel_interior (f −‘ S)
    using convex_rel_interior_iff[of f −‘ S z] conv nonemp by auto
}
moreover
{
  fix z
  assume z: z ∈ rel_interior (f −‘ S)
  {
    fix x
    assume x ∈ S ∩ range f
    then obtain y where y: f y = x y ∈ f −‘ S by auto
    then obtain e where e: e > 1 (1 − e) *R y + e *R z ∈ f −‘ S
      using convex_rel_interior_iff[of f −‘ S z] z conv by auto
    moreover have (1 − e) *R x + e *R f z = f ((1 − e) *R y + e *R z)
      using ‹linear f› y by (simp add: linear_iff)
    ultimately have ∃ e. e > 1 ∧ (1 − e) *R x + e *R f z ∈ S ∩ range f
      using e by auto
  }
  then have f z ∈ rel_interior (S ∩ range f)
    using ‹convex (S ∩ (range f))› ‹S ∩ range f ≠ {}›
      convex_rel_interior_iff[of S ∩ (range f) f z]
    by auto
  moreover have affine (range f)
    by (simp add: linear_axioms linear_subspace_image subspace_imp_affine)
  ultimately have f z ∈ rel_interior S
    using convex_affine_rel_interior_Int[of S range f] assms by auto
  then have z ∈ f −‘ (rel_interior S)
    by auto
}
ultimately show ?thesis by auto
qed

lemma rel_interior_Times:
  fixes S :: 'n::euclidean_space set
    and T :: 'm::euclidean_space set
  assumes convex S
    and convex T
  shows rel_interior (S × T) = rel_interior S × rel_interior T
proof (cases S = {} ∨ T = {})
  case True
  then show ?thesis
    by auto
next
```

case *False*
then have $S \neq \{\}$ $T \neq \{\}$
by *auto*
then have *ri*: *rel_interior* $S \neq \{\}$ *rel_interior* $T \neq \{\}$
using *rel_interior_eq_empty assms* **by** *auto*
then have *fst* $-`$ *rel_interior* $S \neq \{\}$
using *fst_vimage_eq_Times*[*of rel_interior* S] **by** *auto*
then have *rel_interior* $((fst :: 'n * 'm \Rightarrow 'n) -` S) = fst -` rel\_interior\ S$
using *linear_fst* ‹*convex* S› *rel_interior_convex_linear_preimage*[*of fst* S] **by** *auto*
then have *s*: *rel_interior* $(S \times (UNIV :: 'm\ set)) = rel\_interior\ S \times UNIV$
by (*simp add*: *fst_vimage_eq_Times*)
from *ri* **have** *snd* $-`$ *rel_interior* $T \neq \{\}$
using *snd_vimage_eq_Times*[*of rel_interior* T] **by** *auto*
then have *rel_interior* $((snd :: 'n * 'm \Rightarrow 'm) -` T) = snd -` rel\_interior\ T$
using *linear_snd* ‹*convex* T› *rel_interior_convex_linear_preimage*[*of snd* T] **by** *auto*
then have *t*: *rel_interior* $((UNIV :: 'n\ set) \times T) = UNIV \times rel\_interior\ T$
by (*simp add*: *snd_vimage_eq_Times*)
from *s t* **have** $*$: *rel_interior* $(S \times (UNIV :: 'm\ set)) \cap rel\_interior\ ((UNIV :: 'n\ set) \times T) =$
rel_interior $S \times$ *rel_interior* T **by** *auto*
have $S \times T = S \times (UNIV :: 'm\ set) \cap (UNIV :: 'n\ set) \times T$
by *auto*
then have *rel_interior* $(S \times T) = rel\_interior\ ((S \times (UNIV :: 'm\ set)) \cap ((UNIV :: 'n\ set) \times T))$
by *auto*
also have $\ldots$ = *rel_interior* $(S \times (UNIV :: 'm\ set)) \cap rel\_interior\ ((UNIV :: 'n\ set) \times T)$
using $*$ *ri assms convex_Times*
by (*subst convex_rel_interior_inter_two*) *auto*
finally show *?thesis* **using** $*$ **by** *auto*
qed

lemma *rel_interior_scaleR*:
fixes $S :: 'n::euclidean\_space\ set$
assumes $c \neq 0$
shows $((*_R)\ c)\ `\ (rel\_interior\ S) = rel\_interior\ (((*_R)\ c)\ `\ S)$
using *rel_interior_injective_linear_image*[*of* $((*_R)\ c)$ S] *linear_conv_bounded_linear*[*of* $(*_R)$ c] *linear_scaleR injective_scaleR*[*of* c] *assms*
by *auto*

lemma *rel_interior_convex_scaleR*:
fixes $S :: 'n::euclidean\_space\ set$
assumes *convex* S
shows $((*_R)\ c)\ `\ (rel\_interior\ S) = rel\_interior\ (((*_R)\ c)\ `\ S)$
by (*metis assms linear_scaleR rel_interior_convex_linear_image*)

lemma *convex_rel_open_scaleR*:
fixes $S :: 'n::euclidean\_space\ set$
assumes *convex S*
and *rel_open S*
shows *convex* $(((*_R)\ c)\ `\ S) \wedge$ *rel_open* $(((*_R)\ c)\ `\ S)$
by (*metis assms convex_scaling rel_interior_convex_scaleR rel_open_def*)

lemma *convex_rel_open_finite_inter*:
assumes $\forall S{\in}I.$ *convex* $(S :: 'n::euclidean\_space\ set) \wedge$ *rel_open S*
and *finite I*
shows *convex* $(\bigcap I) \wedge$ *rel_open* $(\bigcap I)$
proof (*cases* $\bigcap \{rel\_interior\ S\ |S.\ S \in I\} = \{\}$)
case *True*
then have $\bigcap I = \{\}$
using *assms* **unfolding** *rel_open_def* **by** *auto*
then show *?thesis*
unfolding *rel_open_def* **by** *auto*
next
case *False*
then have *rel_open* $(\bigcap I)$
using *assms* **unfolding** *rel_open_def*
using *convex_rel_interior_finite_inter*[*of I*]
by *auto*
then show *?thesis*
using *convex_Inter assms* **by** *auto*
qed

lemma *convex_rel_open_linear_image*:
fixes $f :: 'm::euclidean\_space \Rightarrow 'n::euclidean\_space$
assumes *linear f*
and *convex S*
and *rel_open S*
shows *convex* $(f\ `\ S) \wedge$ *rel_open* $(f\ `\ S)$
by (*metis assms convex_linear_image rel_interior_convex_linear_image rel_open_def*)

lemma *convex_rel_open_linear_preimage*:
fixes $f :: 'm::euclidean\_space \Rightarrow 'n::euclidean\_space$
assumes *linear f*
and *convex S*
and *rel_open S*
shows *convex* $(f -` S) \wedge$ *rel_open* $(f -` S)$
proof (*cases* $f -` (rel\_interior\ S) = \{\}$)
case *True*
then have $f -` S = \{\}$
using *assms* **unfolding** *rel_open_def* **by** *auto*
then show *?thesis*
unfolding *rel_open_def* **by** *auto*
next
case *False*

```
  then have rel_open (f −‘ S)
    using assms unfolding rel_open_def
    using rel_interior_convex_linear_preimage[of f S]
    by auto
  then show ?thesis
    using convex_linear_vimage assms
    by auto
qed

lemma rel_interior_projection:
  fixes S :: ('m::euclidean_space × 'n::euclidean_space) set
    and f :: 'm::euclidean_space ⇒ 'n::euclidean_space set
  assumes convex S
    and f = (λy. {z. (y, z) ∈ S})
  shows (y, z) ∈ rel_interior S ⟷ (y ∈ rel_interior {y. (f y ≠ {})} ∧ z ∈
rel_interior (f y))
proof −
  {
    fix y
    assume y ∈ {y. f y ≠ {}}
    then obtain z where (y, z) ∈ S
      using assms by auto
    then have ∃ x. x ∈ S ∧ y = fst x
      by auto
    then obtain x where x ∈ S y = fst x
      by blast
    then have y ∈ fst ‘ S
      unfolding image_def by auto
  }
  then have fst ‘ S = {y. f y ≠ {}}
    unfolding fst_def using assms by auto
  then have h1: fst ‘ rel_interior S = rel_interior {y. f y ≠ {}}
    using rel_interior_convex_linear_image[of fst S] assms linear_fst by auto
  {
    fix y
    assume y ∈ rel_interior {y. f y ≠ {}}
    then have y ∈ fst ‘ rel_interior S
      using h1 by auto
    then have ∗: rel_interior S ∩ fst −‘ {y} ≠ {}
      by auto
    moreover have aff: affine (fst −‘ {y})
      unfolding affine_alt by (simp add: algebra_simps)
    ultimately have ∗∗: rel_interior (S ∩ fst −‘ {y}) = rel_interior S ∩ fst −‘
{y}
      using convex_affine_rel_interior_Int[of S fst −‘ {y}] assms by auto
    have conv: convex (S ∩ fst −‘ {y})
      using convex_Int assms aff affine_imp_convex by auto
    {
      fix x
```

```
    assume x ∈ f y
    then have (y, x) ∈ S ∩ (fst -` {y})
      using assms by auto
    moreover have x = snd (y, x) by auto
    ultimately have x ∈ snd ` (S ∩ fst -` {y})
      by blast
  }
  then have snd ` (S ∩ fst -` {y}) = f y
    using assms by auto
  then have ***: rel_interior (f y) = snd ` rel_interior (S ∩ fst -` {y})
    using rel_interior_convex_linear_image[of snd S ∩ fst -` {y}] linear_snd
conv
    by auto
  {
    fix z
    assume z ∈ rel_interior (f y)
    then have z ∈ snd ` rel_interior (S ∩ fst -` {y})
      using *** by auto
    moreover have {y} = fst ` rel_interior (S ∩ fst -` {y})
      using * ** rel_interior_subset by auto
    ultimately have (y, z) ∈ rel_interior (S ∩ fst -` {y})
      by force
    then have (y,z) ∈ rel_interior S
      using ** by auto
  }
  moreover
  {
    fix z
    assume (y, z) ∈ rel_interior S
    then have (y, z) ∈ rel_interior (S ∩ fst -` {y})
      using ** by auto
    then have z ∈ snd ` rel_interior (S ∩ fst -` {y})
      by (metis Range_iff snd_eq_Range)
    then have z ∈ rel_interior (f y)
      using *** by auto
  }
  ultimately have ⋀z. (y, z) ∈ rel_interior S ⟷ z ∈ rel_interior (f y)
    by auto
}
then have h2: ⋀y z. y ∈ rel_interior {t. f t ≠ {}} ⟹
  (y, z) ∈ rel_interior S ⟷ z ∈ rel_interior (f y)
  by auto
{
  fix y z
  assume asm: (y, z) ∈ rel_interior S
  then have y ∈ fst ` rel_interior S
    by (metis Domain_iff fst_eq_Domain)
  then have y ∈ rel_interior {t. f t ≠ {}}
    using h1 by auto
```

```
    then have y ∈ rel_interior {t. f t ≠ {}} and (z ∈ rel_interior (f y))
      using h2 asm by auto
  }
  then show ?thesis using h2 by blast
qed

lemma rel_frontier_Times:
  fixes S :: 'n::euclidean_space set
    and T :: 'm::euclidean_space set
  assumes convex S
    and convex T
  shows rel_frontier S × rel_frontier T ⊆ rel_frontier (S × T)
    by (force simp: rel_frontier_def rel_interior_Times assms closure_Times)
```

Relative interior of convex cone

```
lemma cone_rel_interior:
  fixes S :: 'm::euclidean_space set
  assumes cone S
  shows cone ({0} ∪ rel_interior S)
proof (cases S = {})
  case True
  then show ?thesis
    by (simp add: cone_0)
next
  case False
  then have ∗: 0 ∈ S ∧ (∀ c. c > 0 ⟶ (∗R) c ‘ S = S)
    using cone_iff[of S] assms by auto
  then have ∗: 0 ∈ ({0} ∪ rel_interior S)
    and ∀ c. c > 0 ⟶ (∗R) c ‘ ({0} ∪ rel_interior S) = ({0} ∪ rel_interior S)
    by (auto simp add: rel_interior_scaleR)
  then show ?thesis
    using cone_iff[of {0} ∪ rel_interior S] by auto
qed

lemma rel_interior_convex_cone_aux:
  fixes S :: 'm::euclidean_space set
  assumes convex S
  shows (c, x) ∈ rel_interior (cone hull ({(1 :: real)} × S)) ⟷
    c > 0 ∧ x ∈ (((∗R) c) ‘ (rel_interior S))
proof (cases S = {})
  case True
  then show ?thesis
    by (simp add: cone_hull_empty)
next
  case False
  then obtain s where s ∈ S by auto
  have conv: convex ({(1 :: real)} × S)
    using convex_Times[of {(1 :: real)} S] assms convex_singleton[of 1 :: real]
```

```
    by auto
  define f where f y = {z. (y, z) ∈ cone hull ({1 :: real} × S)} for y
  then have *: (c, x) ∈ rel_interior (cone hull ({(1 :: real)} × S)) =
    (c ∈ rel_interior {y. f y ≠ {}} ∧ x ∈ rel_interior (f c))
    using convex_cone_hull[of {(1 :: real)} × S] conv
    by (subst rel_interior_projection) auto
  {
    fix y :: real
    assume y ≥ 0
    then have y *R (1,s) ∈ cone hull ({1 :: real} × S)
      using cone_hull_expl[of {(1 :: real)} × S] ‹s ∈ S› by auto
    then have f y ≠ {}
      using f_def by auto
  }
  then have {y. f y ≠ {}} = {0..}
    using f_def cone_hull_expl[of {1 :: real} × S] by auto
  then have **: rel_interior {y. f y ≠ {}} = {0<..}
    using rel_interior_real_semiline by auto
  {
    fix c :: real
    assume c > 0
    then have f c = ((*R) c ‘ S)
      using f_def cone_hull_expl[of {1 :: real} × S] by auto
    then have rel_interior (f c) = (*R) c ‘ rel_interior S
      using rel_interior_convex_scaleR[of S c] assms by auto
  }
  then show ?thesis using * ** by auto
qed

lemma rel_interior_convex_cone:
  fixes S :: 'm::euclidean_space set
  assumes convex S
  shows rel_interior (cone hull ({1 :: real} × S)) =
    {(c, c *R x) | c x. c > 0 ∧ x ∈ rel_interior S}
  (is ?lhs = ?rhs)
proof −
  {
    fix z
    assume z ∈ ?lhs
    have *: z = (fst z, snd z)
      by auto
    then have z ∈ ?rhs
      using rel_interior_convex_cone_aux[of S fst z snd z] assms ‹z ∈ ?lhs› by
fastforce
  }
  moreover
  {
    fix z
    assume z ∈ ?rhs
```

```
    then have z ∈ ?lhs
      using rel_interior_convex_cone_aux[of S fst z snd z] assms
      by auto
  }
  ultimately show ?thesis by blast
qed

lemma convex_hull_finite_union:
  assumes finite I
  assumes ∀ i∈I. convex (S i) ∧ (S i) ≠ {}
  shows convex hull (⋃(S ‘ I)) =
    {sum (λi. c i *R s i) I | c s. (∀ i∈I. c i ≥ 0) ∧ sum c I = 1 ∧ (∀ i∈I. s i ∈ S
i)}
  (is ?lhs = ?rhs)
proof −
  have ?lhs ⊇ ?rhs
  proof
    fix x
    assume x ∈ ?rhs
    then obtain c s where ∗: sum (λi. c i *R s i) I = x sum c I = 1
      (∀ i∈I. c i ≥ 0) ∧ (∀ i∈I. s i ∈ S i) by auto
    then have ∀ i∈I. s i ∈ convex hull (⋃(S ‘ I))
      using hull_subset[of ⋃(S ‘ I) convex] by auto
    then show x ∈ ?lhs
      unfolding ∗(1)[symmetric]
      using ∗ assms convex_convex_hull
      by (subst convex_sum) auto
  qed
  {
    fix i
    assume i ∈ I
    with assms have ∃ p. p ∈ S i by auto
  }
  then obtain p where p: ∀ i∈I. p i ∈ S i by metis
  {
    fix i
    assume i ∈ I
    {
      fix x
      assume x ∈ S i
      define c where c j = (if j = i then 1::real else 0) for j
      then have ∗: sum c I = 1
        using ‹finite I› ‹i ∈ I› sum.delta[of I i λj::'a. 1::real]
        by auto
      define s where s j = (if j = i then x else p j) for j
      then have ∀ j. c j *R s j = (if j = i then x else 0)
        using c_def by (auto simp add: algebra_simps)
      then have x = sum (λi. c i *R s i) I
        using s_def c_def ‹finite I› ‹i ∈ I› sum.delta[of I i λj::'a. x]
```

```
      by auto
    moreover have (∀ i∈I. 0 ≤ c i) ∧ sum c I = 1 ∧ (∀ i∈I. s i ∈ S i)
      using ∗ c_def s_def p ‹x ∈ S i› by auto
    ultimately have x ∈ ?rhs
      by force
   }
   then have ?rhs ⊇ S i by auto
  }
  then have ∗: ?rhs ⊇ ⋃(S ‘ I) by auto

  {
   fix u v :: real
   assume uv: u ≥ 0 ∧ v ≥ 0 ∧ u + v = 1
   fix x y
   assume xy: x ∈ ?rhs ∧ y ∈ ?rhs
   from xy obtain c s where
     xc: x = sum (λi. c i ∗R s i) I ∧ (∀ i∈I. c i ≥ 0) ∧ sum c I = 1 ∧ (∀ i∈I. s
i ∈ S i)
     by auto
   from xy obtain d t where
     yc: y = sum (λi. d i ∗R t i) I ∧ (∀ i∈I. d i ≥ 0) ∧ sum d I = 1 ∧ (∀ i∈I. t
i ∈ S i)
     by auto
   define e where e i = u ∗ c i + v ∗ d i for i
   have ge0: ∀ i∈I. e i ≥ 0
     using e_def xc yc uv by simp
   have sum (λi. u ∗ c i) I = u ∗ sum c I
     by (simp add: sum_distrib_left)
   moreover have sum (λi. v ∗ d i) I = v ∗ sum d I
     by (simp add: sum_distrib_left)
   ultimately have sum1: sum e I = 1
     using e_def xc yc uv by (simp add: sum.distrib)
   define q where q i = (if e i = 0 then p i else (u ∗ c i / e i) ∗R s i + (v ∗ d i
/ e i) ∗R t i)
     for i
   {
    fix i
    assume i: i ∈ I
    have q i ∈ S i
    proof (cases e i = 0)
     case True
     then show ?thesis using i p q_def by auto
    next
     case False
     then show ?thesis
       using mem_convex_alt[of S i s i t i u ∗ (c i) v ∗ (d i)]
         mult_nonneg_nonneg[of u c i] mult_nonneg_nonneg[of v d i]
         assms q_def e_def i False xc yc uv
       by (auto simp del: mult_nonneg_nonneg)
```

```
    qed
   }
   then have qs: ∀i∈I. q i ∈ S i by auto
   {
    fix i
    assume i: i ∈ I
    have (u * c i) *R s i + (v * d i) *R t i = e i *R q i
    proof (cases e i = 0)
     case True
     have ge: u * (c i) ≥ 0 ∧ v * d i ≥ 0
      using xc yc uv i by simp
     moreover from ge have u * c i ≤ 0 ∧ v * d i ≤ 0
      using True e_def i by simp
     ultimately have u * c i = 0 ∧ v * d i = 0 by auto
     with True show ?thesis by auto
    next
     case False
     then have (u * (c i)/(e i))*R (s i)+(v * (d i)/(e i))*R (t i) = q i
      using q_def by auto
     then have e i *R ((u * (c i)/(e i))*R (s i)+(v * (d i)/(e i))*R (t i))
           = (e i) *R (q i) by auto
     with False show ?thesis by (simp add: algebra_simps)
    qed
   }
   then have *: ∀i∈I. (u * c i) *R s i + (v * d i) *R t i = e i *R q i
    by auto
   have u *R x + v *R y = sum (λi. (u * c i) *R s i + (v * d i) *R t i) I
    using xc yc by (simp add: algebra_simps scaleR_right.sum sum.distrib)
   also have ... = sum (λi. e i *R q i) I
    using * by auto
   finally have u *R x + v *R y = sum (λi. (e i) *R (q i)) I
    by auto
   then have u *R x + v *R y ∈ ?rhs
    using ge0 sum1 qs by auto
  }
  then have convex ?rhs unfolding convex_def by auto
  then show ?thesis
   using ‹?lhs ⊇ ?rhs› * hull_minimal[of ⋃(S ‘ I) ?rhs convex]
   by blast
qed

lemma convex_hull_union_two:
 fixes S T :: 'm::euclidean_space set
 assumes convex S
   and S ≠ {}
   and convex T
   and T ≠ {}
 shows convex hull (S ∪ T) =
   {u *R s + v *R t | u v s t. u ≥ 0 ∧ v ≥ 0 ∧ u + v = 1 ∧ s ∈ S ∧ t ∈ T}
```

```
  (is ?lhs = ?rhs)
proof
  define I :: nat set where I = {1, 2}
  define s where s i = (if i = (1::nat) then S else T) for i
  have ⋃(s ` I) = S ∪ T
    using s_def I_def by auto
  then have convex hull (⋃(s ` I)) = convex hull (S ∪ T)
    by auto
  moreover have convex hull ⋃(s ` I) =
    {∑ i∈I. c i *R sa i | c sa. (∀ i∈I. 0 ≤ c i) ∧ sum c I = 1 ∧ (∀ i∈I. sa i ∈ s
i)}
      using assms s_def I_def
      by (subst convex_hull_finite_union) auto
  moreover have
    {∑ i∈I. c i *R sa i | c sa. (∀ i∈I. 0 ≤ c i) ∧ sum c I = 1 ∧ (∀ i∈I. sa i ∈ s
i)} ≤ ?rhs
    using s_def I_def by auto
  ultimately show ?lhs ⊆ ?rhs by auto
  {
    fix x
    assume x ∈ ?rhs
    then obtain u v s t where *: x = u *R s + v *R t ∧ u ≥ 0 ∧ v ≥ 0 ∧ u +
v = 1 ∧ s ∈ S ∧ t ∈ T
      by auto
    then have x ∈ convex hull {s, t}
      using convex_hull_2[of s t] by auto
    then have x ∈ convex hull (S ∪ T)
      using * hull_mono[of {s, t} S ∪ T] by auto
  }
  then show ?lhs ⊇ ?rhs by blast
qed

proposition ray_to_rel_frontier:
  fixes a :: 'a::real_inner
  assumes bounded S
      and a: a ∈ rel_interior S
      and aff: (a + l) ∈ affine hull S
      and l ≠ 0
  obtains d where 0 < d (a + d *R l) ∈ rel_frontier S
          ⋀e. ⟦0 ≤ e; e < d⟧ ⟹ (a + e *R l) ∈ rel_interior S
proof −
  have aaff: a ∈ affine hull S
    by (meson a hull_subset rel_interior_subset rev_subsetD)
  let ?D = {d. 0 < d ∧ a + d *R l ∉ rel_interior S}
  obtain B where B > 0 and B: S ⊆ ball a B
    using bounded_subset_ballD [OF ‹bounded S›] by blast
  have a + (B / norm l) *R l ∉ ball a B
    by (simp add: dist_norm ‹l ≠ 0›)
  with B have a + (B / norm l) *R l ∉ rel_interior S
```

```
    using rel_interior_subset subsetCE by blast
  with ‹B > 0› ‹l ≠ 0› have nonMT: ?D ≠ {}
    using divide_pos_pos zero_less_norm_iff by fastforce
  have bdd: bdd_below ?D
    by (metis (no_types, lifting) bdd_belowI le_less mem_Collect_eq)
  have relin_Ex: ⋀x. x ∈ rel_interior S ⟹
                 ∃e>0. ∀x'∈affine hull S. dist x' x < e ⟶ x' ∈ rel_interior S
   using openin_rel_interior [of S] by (simp add: openin_euclidean_subtopology_iff)
  define d where d = Inf ?D
 obtain ε where 0 < ε and ε: ⋀η. ⟦0 ≤ η; η < ε⟧ ⟹ (a + η *R l) ∈ rel_interior
S
  proof −
    obtain e where e>0
          and e: ⋀x'. x' ∈ affine hull S ⟹ dist x' a < e ⟹ x' ∈ rel_interior S
      using relin_Ex a by blast
    show thesis
    proof (rule_tac ε = e / norm l in that)
      show 0 < e / norm l by (simp add: ‹0 < e› ‹l ≠ 0›)
    next
      show a + η *R l ∈ rel_interior S if 0 ≤ η η < e / norm l for η
      proof (rule e)
        show a + η *R l ∈ affine hull S
       by (metis (no_types) add_diff_cancel_left' aff affine_affine_hull mem_affine_3_minus
aaff)
        show dist (a + η *R l) a < e
          using that by (simp add: ‹l ≠ 0› dist_norm pos_less_divide_eq)
      qed
    qed
  qed
  have inint: ⋀e. ⟦0 ≤ e; e < d⟧ ⟹ a + e *R l ∈ rel_interior S
    unfolding d_def using cInf_lower [OF _ bdd]
     by (metis (no_types, lifting) a add.right_neutral le_less mem_Collect_eq
not_less real_vector.scale_zero_left)
  have ε ≤ d
    unfolding d_def
    using ε dual_order.strict_implies_order le_less_linear
    by (blast intro: cInf_greatest [OF nonMT])
  with ‹0 < ε› have 0 < d by simp
  have a + d *R l ∉ rel_interior S
  proof
    assume adl: a + d *R l ∈ rel_interior S
    obtain e where e > 0
          and e: ⋀x'. x' ∈ affine hull S ⟹ dist x' (a + d *R l) < e ⟹ x' ∈
rel_interior S
      using relin_Ex adl by blast
    have d + e / norm l ≤ Inf {d. 0 < d ∧ a + d *R l ∉ rel_interior S}
    proof (rule cInf_greatest [OF nonMT], clarsimp)
      fix x::real
      assume 0 < x and nonrel: a + x *R l ∉ rel_interior S
```

```
      show d + e / norm l ≤ x
      proof (cases x < d)
        case True with inint nonrel ‹0 < x›
          show ?thesis by auto
      next
        case False
          then have dle: x < d + e / norm l ⟹ dist (a + x *R l) (a + d *R l)
< e
            by (simp add: field_simps ‹l ≠ 0›)
          have ain: a + x *R l ∈ affine hull S
         by (metis add_diff_cancel_left' aff affine_affine_hull mem_affine_3_minus
aaff)
          show ?thesis
            using e [OF ain] nonrel dle by force
      qed
    qed
    then show False
      using ‹0 < e› ‹l ≠ 0› by (simp add: d_def [symmetric] field_simps)
  qed
  moreover have a + d *R l ∈ closure S
  proof (clarsimp simp: closure_approachable)
    fix η::real assume 0 < η
    have 1: a + (d − min d (η / 2 / norm l)) *R l ∈ S
    proof (rule subsetD [OF rel_interior_subset inint])
      show d − min d (η / 2 / norm l) < d
        using ‹l ≠ 0› ‹0 < d› ‹0 < η› by auto
    qed auto
    have norm l * min d (η / (norm l * 2)) ≤ norm l * (η / (norm l * 2))
      by (metis min_def mult_left_mono norm_ge_zero order_refl)
    also have ... < η
      using ‹l ≠ 0› ‹0 < η› by (simp add: field_simps)
    finally have 2: norm l * min d (η / (norm l * 2)) < η .
    show ∃ y∈S. dist y (a + d *R l) < η
      using 1 2 ‹0 < d› ‹0 < η›
      by (rule_tac x=a + (d − min d (η / 2 / norm l)) *R l in bexI) (auto simp:
algebra_simps)
  qed
  ultimately have infront: a + d *R l ∈ rel_frontier S
    by (simp add: rel_frontier_def)
  show ?thesis
    by (rule that [OF ‹0 < d› infront inint])
qed

corollary ray_to_frontier:
  fixes a :: 'a::euclidean_space
  assumes bounded S
      and a: a ∈ interior S
      and l ≠ 0
  obtains d where 0 < d (a + d *R l) ∈ frontier S
```

$\bigwedge e.\ [\![0 \le e;\ e < d]\!] \Longrightarrow (a + e *_R l) \in \mathit{interior}\ S$
proof −
have §: *interior S = rel_interior S*
using *a rel_interior_nonempty_interior* **by** *auto*
then have *a ∈ rel_interior S*
using *a* **by** *simp*
moreover have *a + l ∈ affine hull S*
using *a affine_hull_nonempty_interior* **by** *blast*
ultimately show *thesis*
by (*metis* § ‹*bounded S*› ‹*l ≠ 0*› *frontier_def ray_to_rel_frontier rel_frontier_def that*)
qed

lemma *segment_to_rel_frontier_aux*:
fixes *x* :: *'a*::*euclidean_space*
assumes *convex S bounded S* **and** *x*: *x ∈ rel_interior S* **and** *y*: *y ∈ S* **and** *xy*: *x ≠ y*
obtains *z* **where** *z ∈ rel_frontier S y ∈ closed_segment x z*
open_segment x z ⊆ rel_interior S
proof −
have *x + (y − x) ∈ affine hull S*
using *hull_inc* [*OF y*] **by** *auto*
then obtain *d* **where** *0 < d* **and** *df*: *(x + d *$_R$ (y−x)) ∈ rel_frontier S*
and *di*: $\bigwedge e.\ [\![0 \le e;\ e < d]\!] \Longrightarrow (x + e *_R (y-x)) \in \mathit{rel\_interior}\ S$
by (*rule ray_to_rel_frontier* [*OF* ‹*bounded S*› *x*]) (*use xy* **in** *auto*)
show *?thesis*
proof
show *x + d *$_R$ (y − x) ∈ rel_frontier S*
by (*simp add*: *df*)
next
have *open_segment x y ⊆ rel_interior S*
using *rel_interior_closure_convex_segment* [*OF* ‹*convex S*› *x*] *closure_subset y* **by** *blast*
moreover have *x + d *$_R$ (y − x) ∈ open_segment x y* **if** *d < 1*
using *xy* ‹*0 < d*› *that* **by** (*force simp*: *in_segment algebra_simps*)
ultimately have *1 ≤ d*
using *df rel_frontier_def* **by** *fastforce*
moreover have *x = (1 / d) *$_R$ x + ((d − 1) / d) *$_R$ x*
by (*metis* ‹*0 < d*› *add.commute add_divide_distrib diff_add_cancel divide_self_if less_irrefl scaleR_add_left scaleR_one*)
ultimately show *y ∈ closed_segment x (x + d *$_R$ (y − x))*
unfolding *in_segment*
by (*rule_tac x=1/d* **in** *exI*) (*auto simp*: *algebra_simps*)
next
show *open_segment x (x + d *$_R$ (y − x)) ⊆ rel_interior S*
proof (*rule rel_interior_closure_convex_segment* [*OF* ‹*convex S*› *x*])
show *x + d *$_R$ (y − x) ∈ closure S*
using *df rel_frontier_def* **by** *auto*

```
    qed
  qed
qed

lemma segment_to_rel_frontier:
  fixes x :: 'a::euclidean_space
  assumes S: convex S bounded S and x: x ∈ rel_interior S
      and y: y ∈ S and xy: ¬(x = y ∧ S = {x})
  obtains z where z ∈ rel_frontier S y ∈ closed_segment x z
                  open_segment x z ⊆ rel_interior S
proof (cases x=y)
  case True
  with xy have S ≠ {x}
    by blast
  with True show ?thesis
    by (metis Set.set_insert all_not_in_conv ends_in_segment(1) insert_iff seg-
ment_to_rel_frontier_aux[OF S x] that y)
next
  case False
  then show ?thesis
    using segment_to_rel_frontier_aux [OF S x y] that by blast
qed

proposition rel_frontier_not_sing:
  fixes a :: 'a::euclidean_space
  assumes bounded S
    shows rel_frontier S ≠ {a}
proof (cases S = {})
  case True  then show ?thesis  by simp
next
  case False
  then obtain z where z ∈ S
    by blast
  then show ?thesis
  proof (cases S = {z})
    case True then show ?thesis  by simp
  next
    case False
    then obtain w where w ∈ S w ≠ z
      using ‹z ∈ S› by blast
    show ?thesis
    proof
      assume rel_frontier S = {a}
      then consider w ∉ rel_frontier S | z ∉ rel_frontier S
        using ‹w ≠ z› by auto
      then show False
      proof cases
        case 1
        then have w: w ∈ rel_interior S
```

using ‹$w \in S$› *closure_subset rel_frontier_def* **by** *fastforce*
have $w + (w - z) \in$ *affine hull* S
by (*metis* ‹$w \in S$› ‹$z \in S$› *affine_affine_hull hull_inc mem_affine_3_minus scaleR_one*)
then obtain e **where** $0 < e$ $(w + e *_R (w - z)) \in$ *rel_frontier* S
using ‹$w \neq z$› ‹$z \in S$› **by** (*metis assms ray_to_rel_frontier right_minus_eq w*)
moreover obtain d **where** $0 < d$ $(w + d *_R (z - w)) \in$ *rel_frontier* S
using *ray_to_rel_frontier* [*OF* ‹*bounded* S› w, *of* $1 *_R (z - w)$] ‹$w \neq z$› ‹$z \in S$›
by (*metis add.commute add.right_neutral diff_add_cancel hull_inc scaleR_one*)
ultimately have $d *_R (z - w) = e *_R (w - z)$
using ‹*rel_frontier* $S = \{a\}$› **by** *force*
moreover have $e \neq -d$
using ‹$0 < e$› ‹$0 < d$› **by** *force*
ultimately show *False*
by (*metis* (*no_types*, *lifting*) ‹$w \neq z$› *eq_iff_diff_eq_0 minus_diff_eq real_vector.scale_cancel_right real_vector.scale_minus_right scaleR_left.minus*)
next
case *2*
then have z: $z \in$ *rel_interior* S
using ‹$z \in S$› *closure_subset rel_frontier_def* **by** *fastforce*
have $z + (z - w) \in$ *affine hull* S
by (*metis* ‹$z \in S$› ‹$w \in S$› *affine_affine_hull hull_inc mem_affine_3_minus scaleR_one*)
then obtain e **where** $0 < e$ $(z + e *_R (z - w)) \in$ *rel_frontier* S
using ‹$w \neq z$› ‹$w \in S$› **by** (*metis assms ray_to_rel_frontier right_minus_eq z*)
moreover obtain d **where** $0 < d$ $(z + d *_R (w - z)) \in$ *rel_frontier* S
using *ray_to_rel_frontier* [*OF* ‹*bounded* S› z, *of* $1 *_R (w - z)$] ‹$w \neq z$› ‹$w \in S$›
by (*metis add.commute add.right_neutral diff_add_cancel hull_inc scaleR_one*)
ultimately have $d *_R (w - z) = e *_R (z - w)$
using ‹*rel_frontier* $S = \{a\}$› **by** *force*
moreover have $e \neq -d$
using ‹$0 < e$› ‹$0 < d$› **by** *force*
ultimately show *False*
by (*metis* (*no_types*, *lifting*) ‹$w \neq z$› *eq_iff_diff_eq_0 minus_diff_eq real_vector.scale_cancel_right real_vector.scale_minus_right scaleR_left.minus*)
qed
qed
qed
qed

5.0.5 Convexity on direct sums

lemma *closure_sum*:

fixes S T :: $'a$::*real_normed_vector set*
shows *closure* S + *closure* T ⊆ *closure* (S + T)
unfolding *set_plus_image closure_Times* [*symmetric*] *split_def*
by (*intro closure_bounded_linear_image_subset bounded_linear_add*
bounded_linear_fst bounded_linear_snd)

lemma *fst_snd_linear*: *linear* (λ(x,y). x + y)
unfolding *linear_iff* **by** (*simp add*: *algebra_simps*)

lemma *rel_interior_sum*:
fixes S T :: $'n$::*euclidean_space set*
assumes *convex* S
and *convex* T
shows *rel_interior* (S + T) = *rel_interior* S + *rel_interior* T
proof −
have *rel_interior* S + *rel_interior* T = (λ(x,y). x + y) ‘ (*rel_interior* S × *rel_interior* T)
by (*simp add*: *set_plus_image*)
also have ... = (λ(x,y). x + y) ‘ *rel_interior* (S × T)
using *rel_interior_Times assms* **by** *auto*
also have ... = *rel_interior* (S + T)
using *fst_snd_linear convex_Times assms*
rel_interior_convex_linear_image[*of* (λ(x,y). x + y) S × T]
by (*auto simp add*: *set_plus_image*)
finally show *?thesis* **..**
qed

lemma *rel_interior_sum_gen*:
fixes S :: $'a$ ⇒ $'n$::*euclidean_space set*
assumes ⋀i. i∈I ⟹ *convex* (S i)
shows *rel_interior* (*sum* S I) = *sum* (λi. *rel_interior* (S i)) I
using *rel_interior_sum rel_interior_sing*[*of* 0] *assms*
by (*subst sum_set_cond_linear*[*of convex*], *auto simp add*: *convex_set_plus*)

lemma *convex_rel_open_direct_sum*:
fixes S T :: $'n$::*euclidean_space set*
assumes *convex* S
and *rel_open* S
and *convex* T
and *rel_open* T
shows *convex* (S × T) ∧ *rel_open* (S × T)
by (*metis assms convex_Times rel_interior_Times rel_open_def*)

lemma *convex_rel_open_sum*:
fixes S T :: $'n$::*euclidean_space set*
assumes *convex* S
and *rel_open* S
and *convex* T
and *rel_open* T

```
  shows convex (S + T) ∧ rel_open (S + T)
  by (metis assms convex_set_plus rel_interior_sum rel_open_def)

lemma convex_hull_finite_union_cones:
  assumes finite I
    and I ≠ {}
  assumes ⋀i. i∈I ⟹ convex (S i) ∧ cone (S i) ∧ S i ≠ {}
  shows convex hull (⋃(S ` I)) = sum S I
  (is ?lhs = ?rhs)
proof −
  {
    fix x
    assume x ∈ ?lhs
    then obtain c xs where
      x: x = sum (λi. c i *R xs i) I ∧ (∀ i∈I. c i ≥ 0) ∧ sum c I = 1 ∧ (∀ i∈I. xs
i ∈ S i)
      using convex_hull_finite_union[of I S] assms by auto
    define s where s i = c i *R xs i for i
    have ∀ i∈I. s i ∈ S i
      using s_def x assms by (simp add: mem_cone)
    moreover have x = sum s I using x s_def by auto
    ultimately have x ∈ ?rhs
      using set_sum_alt[of I S] assms by auto
  }
  moreover
  {
    fix x
    assume x ∈ ?rhs
    then obtain s where x: x = sum s I ∧ (∀ i∈I. s i ∈ S i)
      using set_sum_alt[of I S] assms by auto
    define xs where xs i = of_nat(card I) *R s i for i
    then have x = sum (λi. ((1 :: real) / of_nat(card I)) *R xs i) I
      using x assms by auto
    moreover have ∀ i∈I. xs i ∈ S i
      using x xs_def assms by (simp add: cone_def)
    moreover have ∀ i∈I. (1 :: real) / of_nat (card I) ≥ 0
      by auto
    moreover have sum (λi. (1 :: real) / of_nat (card I)) I = 1
      using assms by auto
    ultimately have x ∈ ?lhs
      using assms
      apply (simp add: convex_hull_finite_union[of I S])
      by (rule_tac x = (λi. 1 / (card I)) in exI) auto
  }
  ultimately show ?thesis by auto
qed

lemma convex_hull_union_cones_two:
  fixes S T :: 'm::euclidean_space set
```

```
  assumes convex S
    and cone S
    and S ≠ {}
  assumes convex T
    and cone T
    and T ≠ {}
  shows convex hull (S ∪ T) = S + T
proof −
  define I :: nat set where I = {1, 2}
  define A where A i = (if i = (1::nat) then S else T) for i
  have ⋃(A ‘ I) = S ∪ T
    using A_def I_def by auto
  then have convex hull (⋃(A ‘ I)) = convex hull (S ∪ T)
    by auto
  moreover have convex hull ⋃(A ‘ I) = sum A I
    using A_def I_def
     by (metis assms convex_hull_finite_union_cones empty_iff finite.emptyI fi-
nite.insertI insertI1)
  moreover have sum A I = S + T
    using A_def I_def by (force simp add: set_plus_def)
  ultimately show ?thesis by auto
qed

lemma rel_interior_convex_hull_union:
  fixes S :: 'a ⇒ 'n::euclidean_space set
  assumes finite I
    and ∀ i∈I. convex (S i) ∧ S i ≠ {}
  shows rel_interior (convex hull (⋃(S ‘ I))) =
    {sum (λi. c i ∗R s i) I | c s. (∀ i∈I. c i > 0) ∧ sum c I = 1 ∧
      (∀ i∈I. s i ∈ rel_interior(S i))}
  (is ?lhs = ?rhs)
proof (cases I = {})
  case True
  then show ?thesis
    using convex_hull_empty by auto
next
  case False
  define C0 where C0 = convex hull (⋃(S ‘ I))
  have ∀ i∈I. C0 ≥ S i
    unfolding C0_def using hull_subset[of ⋃(S ‘ I)] by auto
  define K0 where K0 = cone hull ({1 :: real} × C0)
  define K where K i = cone hull ({1 :: real} × S i) for i
  have ∀ i∈I. K i ≠ {}
    unfolding K_def using assms
    by (simp add: cone_hull_empty_iff[symmetric])
  have convK: ∀ i∈I. convex (K i)
    unfolding K_def
    by (simp add: assms(2) convex_Times convex_cone_hull)
  have K0 ⊇ K i if  i ∈ I for i
```

```
    unfolding K0_def K_def
    by (simp add: Sigma_mono ‹∀ i∈I. S i ⊆ C0› hull_mono that)
  then have K0 ⊇ ⋃(K ‘ I) by auto
  moreover have convex K0
    unfolding K0_def by (simp add: C0_def convex_Times convex_cone_hull)
  ultimately have geq: K0 ⊇ convex hull (⋃(K ‘ I))
    using hull_minimal[of _ K0 convex] by blast
  have ∀ i∈I. K i ⊇ {1 :: real} × S i
    using K_def by (simp add: hull_subset)
  then have ⋃(K ‘ I) ⊇ {1 :: real} × ⋃(S ‘ I)
    by auto
  then have convex hull ⋃(K ‘ I) ⊇ convex hull ({1 :: real} × ⋃(S ‘ I))
    by (simp add: hull_mono)
  then have convex hull ⋃(K ‘ I) ⊇ {1 :: real} × C0
    unfolding C0_def
    using convex_hull_Times[of {(1 :: real)} ⋃(S ‘ I)] convex_hull_singleton
    by auto
  moreover have cone (convex hull (⋃(K ‘ I)))
    by (simp add: K_def cone_Union cone_cone_hull cone_convex_hull)
  ultimately have convex hull (⋃(K ‘ I)) ⊇ K0
    unfolding K0_def
    using hull_minimal[of _ convex hull (⋃(K ‘ I)) cone]
    by blast
  then have K0 = convex hull (⋃(K ‘ I))
    using geq by auto
  also have ... = sum K I
    using assms False ‹∀ i∈I. K i ≠ {}› cone_hull_eq convK
    by (intro convex_hull_finite_union_cones; fastforce simp: K_def)
  finally have K0 = sum K I by auto
  then have ∗: rel_interior K0 = sum (λi. (rel_interior (K i))) I
    using rel_interior_sum_gen[of I K] convK by auto
  {
    fix x
    assume x ∈ ?lhs
    then have (1::real, x) ∈ rel_interior K0
      using K0_def C0_def rel_interior_convex_cone_aux[of C0 1::real x] con-
vex_convex_hull
      by auto
    then obtain k where k: (1::real, x) = sum k I ∧ (∀ i∈I. k i ∈ rel_interior
(K i))
      using ‹finite I› ∗ set_sum_alt[of I λi. rel_interior (K i)] by auto
    {
      fix i
      assume i ∈ I
      then have convex (S i) ∧ k i ∈ rel_interior (cone hull {1} × S i)
        using k K_def assms by auto
      then have ∃ ci si. k i = (ci, ci ∗R si) ∧ 0 < ci ∧ si ∈ rel_interior (S i)
        using rel_interior_convex_cone[of S i] by auto
    }
```

```
    then obtain c s where cs: ∀ i∈I. k i = (c i, c i *R s i) ∧ 0 < c i ∧ s i ∈
rel_interior (S i)
      by metis
    then have x = (∑ i∈I. c i *R s i) ∧ sum c I = 1
      using k by (simp add: sum_prod)
    then have x ∈ ?rhs
      using k cs by auto
  }
  moreover
  {
    fix x
    assume x ∈ ?rhs
    then obtain c s where cs: x = sum (λi. c i *R s i) I ∧
        (∀ i∈I. c i > 0) ∧ sum c I = 1 ∧ (∀ i∈I. s i ∈ rel_interior (S i))
      by auto
    define k where k i = (c i, c i *R s i) for i
    {
      fix i assume i ∈ I
      then have k i ∈ rel_interior (K i)
        using k_def K_def assms cs rel_interior_convex_cone[of S i]
        by auto
    }
    then have (1, x) ∈ rel_interior K0
      using * set_sum_alt[of I (λi. rel_interior (K i))] assms cs
      by (simp add: k_def) (metis (mono_tags, lifting) sum_prod)
    then have x ∈ ?lhs
      using K0_def C0_def rel_interior_convex_cone_aux[of C0 1 x]
      by auto
  }
  ultimately show ?thesis by blast
qed


lemma convex_le_Inf_differential:
  fixes f :: real ⇒ real
  assumes convex_on I f
    and x ∈ interior I
    and y ∈ I
  shows f y ≥ f x + Inf ((λt. (f x − f t) / (x − t)) ‘ ({x<..} ∩ I)) * (y − x)
  (is _ ≥ _ + Inf (?F x) * (y − x))
proof (cases rule: linorder_cases)
  assume x < y
  moreover
  have open (interior I) by auto
  from openE[OF this ‹x ∈ interior I›]
  obtain e where e: 0 < e ball x e ⊆ interior I .
  moreover define t where t = min (x + e / 2) ((x + y) / 2)
  ultimately have x < t t < y t ∈ ball x e
    by (auto simp: dist_real_def field_simps split: split_min)
```

```
  with ‹x ∈ interior I› e interior_subset[of I] have t ∈ I x ∈ I by auto

  define K where K = x − e / 2
  with ‹0 < e› have K ∈ ball x e K < x
    by (auto simp: dist_real_def)
  then have K ∈ I
    using ‹interior I ⊆ I› e(2) by blast

  have Inf (?F x) ≤ (f x − f y) / (x − y)
  proof (intro bdd_belowI cInf_lower2)
    show (f x − f t) / (x − t) ∈ ?F x
      using ‹t ∈ I› ‹x < t› by auto
    show (f x − f t) / (x − t) ≤ (f x − f y) / (x − y)
      using ‹convex_on I f› ‹x ∈ I› ‹y ∈ I› ‹x < t› ‹t < y›
      by (rule convex_on_diff)
  next
    fix y
    assume y ∈ ?F x
    with order_trans[OF convex_on_diff[OF ‹convex_on I f› ‹K ∈ I› _ ‹K <
x› _]]
    show (f K − f x) / (K − x) ≤ y by auto
  qed
  then show ?thesis
    using ‹x < y› by (simp add: field_simps)
next
  assume y < x
  moreover
  have open (interior I) by auto
  from openE[OF this ‹x ∈ interior I›]
  obtain e where e: 0 < e ball x e ⊆ interior I .
  moreover define t where t = x + e / 2
  ultimately have x < t t ∈ ball x e
    by (auto simp: dist_real_def field_simps)
  with ‹x ∈ interior I› e interior_subset[of I] have t ∈ I x ∈ I by auto

  have (f x − f y) / (x − y) ≤ Inf (?F x)
  proof (rule cInf_greatest)
    have (f x − f y) / (x − y) = (f y − f x) / (y − x)
      using ‹y < x› by (auto simp: field_simps)
    also
    fix z
    assume z ∈ ?F x
    with order_trans[OF convex_on_diff[OF ‹convex_on I f› ‹y ∈ I› _ ‹y < x›]]
    have (f y − f x) / (y − x) ≤ z
      by auto
    finally show (f x − f y) / (x − y) ≤ z .
  next
    have x + e / 2 ∈ ball x e
      using e by (auto simp: dist_real_def)
```

with *e interior_subset*[*of I*] **have** $x + e / 2 \in \{x<..\} \cap I$
by *auto*
then show *?F x* $\neq \{\}$
by *blast*
qed
then show *?thesis*
using ‹$y < x$› **by** (*simp add*: *field_simps*)
qed *simp*

5.0.6 Explicit formulas for interior and relative interior of convex hull

lemma *at_within_cbox_finite*:
assumes $x \in$ *box a b x* $\notin S$ *finite S*
shows (*at x within cbox a b* $- S$) = *at x*
proof −
have *interior* (*cbox a b* $- S$) = *box a b* $- S$
using ‹*finite S*› **by** (*simp add*: *interior_diff finite_imp_closed*)
then show *?thesis*
using *at_within_interior assms* **by** *fastforce*
qed

lemma *affine_independent_convex_affine_hull*:
fixes *S* :: *'a*::*euclidean_space set*
assumes ¬ *affine_dependent S T* $\subseteq S$
shows *convex hull T* = *affine hull T* $\cap$ *convex hull S*
proof −
have *fin*: *finite S finite T* **using** *assms aff_independent_finite finite_subset* **by** *auto*
have *convex hull T* $\subseteq$ *affine hull T*
using *convex_hull_subset_affine_hull* **by** *blast*
moreover have *convex hull T* $\subseteq$ *convex hull S*
using *assms hull_mono* **by** *blast*
moreover have *affine hull T* $\cap$ *convex hull S* $\subseteq$ *convex hull T*
proof −
have *0*: $\bigwedge u$. *sum u S* = *0* $\Longrightarrow$ $(\forall v{\in}S.\ u\ v = 0) \vee (\sum v{\in}S.\ u\ v *_R v) \neq 0$
using *affine_dependent_explicit_finite assms*(*1*) *fin*(*1*) **by** *auto*
show *?thesis*
proof (*clarsimp simp add*: *affine_hull_finite fin*)
fix *u*
assume *S*: $(\sum v{\in}T.\ u\ v *_R v) \in$ *convex hull S*
and *T1*: *sum u T* = *1*
then obtain *v* **where** *v*: $\forall x{\in}S.\ 0 \le v\ x$ *sum v S* = *1* $(\sum x{\in}S.\ v\ x *_R x) = (\sum v{\in}T.\ u\ v *_R v)$
by (*auto simp add*: *convex_hull_finite fin*)
{ **fix** *x*
assume*x* $\in T$
then have *S*: $S = (S - T) \cup T$ — split into separate cases
using *assms* **by** *auto*

```
      have [simp]: (∑x∈T. v x *R x) + (∑x∈S - T. v x *R x) = (∑x∈T. u
x *R x)
        sum v T + sum v (S - T) = 1
        using v fin S
        by (auto simp: sum.union_disjoint [symmetric] Un_commute)
      have (∑x∈S. if x ∈ T then v x - u x else v x) = 0
          (∑x∈S. (if x ∈ T then v x - u x else v x) *R x) = 0
        using v fin T1
      by (subst S, subst sum.union_disjoint, auto simp: algebra_simps sum_subtractf)+
    } note [simp] = this
    have (∀x∈T. 0 ≤ u x)
     using 0 [of λx. if x ∈ T then v x - u x else v x] ‹T ⊆ S› v(1) by fastforce
    then show (∑v∈T. u v *R v) ∈ convex hull T
      using 0 [of λx. if x ∈ T then v x - u x else v x] ‹T ⊆ S› T1
      by (fastforce simp add: convex_hull_finite fin)
  qed
 qed
 ultimately show ?thesis
  by blast
qed

lemma affine_independent_span_eq:
 fixes S :: 'a::euclidean_space set
 assumes ¬ affine_dependent S card S = Suc (DIM ('a))
  shows affine hull S = UNIV
proof (cases S = {})
 case True then show ?thesis
  using assms by simp
next
 case False
  then obtain a T where T: a ∉ T S = insert a T
   by blast
  then have fin: finite T using assms
   by (metis finite_insert aff_independent_finite)
  have UNIV ⊆ (+) a ‘ span ((λx. x - a) ‘ T)
  proof (intro card_ge_dim_independent Fun.vimage_subsetD)
   show independent ((λx. x - a) ‘ T)
    using T affine_dependent_iff_dependent assms(1) by auto
   show dim ((+) a -‘ UNIV) ≤ card ((λx. x - a) ‘ T)
    using assms T fin by (auto simp: card_image inj_on_def)
  qed (use surj_plus in auto)
  then show ?thesis
   using T(2) affine_hull_insert_span_gen equalityI by fastforce
qed

lemma affine_independent_span_gt:
 fixes S :: 'a::euclidean_space set
 assumes ind: ¬ affine_dependent S and dim: DIM ('a) < card S
  shows affine hull S = UNIV
```

proof (*intro affine_independent_span_eq* [*OF ind*] *antisym*)
show *card S* ≤ *Suc DIM*('*a*)
using *aff_independent_finite affine_dependent_biggerset ind* **by** *fastforce*
show *Suc DIM*('*a*) ≤ *card S*
using *Suc_leI dim* **by** *blast*
qed

lemma *empty_interior_affine_hull*:
fixes *S* :: '*a*::*euclidean_space set*
assumes *finite S* **and** *dim*: *card S* ≤ *DIM* ('*a*)
shows *interior*(*affine hull S*) = {}
using *assms*
proof (*induct S rule*: *finite_induct*)
case (*insert x S*)
then have *dim* (*span* ((λ*y*. *y* − *x*) ' *S*)) < *DIM*('*a*)
by (*auto simp*: *Suc_le_lessD card_image_le dual_order.trans intro*!: *dim_le_card*'[*THEN le_less_trans*])
then show *?case*
by (*simp add*: *empty_interior_lowdim affine_hull_insert_span_gen interior_translation*)
qed *auto*

lemma *empty_interior_convex_hull*:
fixes *S* :: '*a*::*euclidean_space set*
assumes *finite S* **and** *dim*: *card S* ≤ *DIM* ('*a*)
shows *interior*(*convex hull S*) = {}
by (*metis Diff_empty Diff_eq_empty_iff convex_hull_subset_affine_hull interior_mono empty_interior_affine_hull* [*OF assms*])

lemma *explicit_subset_rel_interior_convex_hull*:
fixes *S* :: '*a*::*euclidean_space set*
shows *finite S*
⟹ {*y*. ∃ *u*. (∀ *x* ∈ *S*. 0 < *u x* ∧ *u x* < 1) ∧ *sum u S* = 1 ∧ *sum* (λ*x*. *u x* *$_R$ *x*) *S* = *y*}
⊆ *rel_interior* (*convex hull S*)
by (*force simp add*: *rel_interior_convex_hull_union* [**where** *S*=λ*x*. {*x*} **and** *I*=*S*, *simplified*])

lemma *explicit_subset_rel_interior_convex_hull_minimal*:
fixes *S* :: '*a*::*euclidean_space set*
shows *finite S*
⟹ {*y*. ∃ *u*. (∀ *x* ∈ *S*. 0 < *u x*) ∧ *sum u S* = 1 ∧ *sum* (λ*x*. *u x* *$_R$ *x*) *S* = *y*}
⊆ *rel_interior* (*convex hull S*)
by (*force simp add*: *rel_interior_convex_hull_union* [**where** *S*=λ*x*. {*x*} **and** *I*=*S*, *simplified*])

lemma *rel_interior_convex_hull_explicit*:
fixes *S* :: '*a*::*euclidean_space set*
assumes ¬ *affine_dependent S*

```
  shows rel_interior(convex hull S) =
        {y. ∃u. (∀x ∈ S. 0 < u x) ∧ sum u S = 1 ∧ sum (λx. u x *R x) S = y}
        (is ?lhs = ?rhs)
proof
  show ?rhs ≤ ?lhs
   by (simp add: aff_independent_finite explicit_subset_rel_interior_convex_hull_minimal
assms)
next
  show ?lhs ≤ ?rhs
  proof (cases ∃a. S = {a})
    case True then show ?lhs ≤ ?rhs
      by force
  next
    case False
    have fs: finite S
      using assms by (simp add: aff_independent_finite)
    { fix a b and d::real
      assume ab: a ∈ S b ∈ S a ≠ b
      then have S: S = (S - {a,b}) ∪ {a,b} — split into separate cases
        by auto
      have (∑x∈S. if x = a then - d else if x = b then d else 0) = 0
            (∑x∈S. (if x = a then - d else if x = b then d else 0) *R x) = d *R b
- d *R a
        using ab fs
        by (subst S, subst sum.union_disjoint, auto)+
    } note [simp] = this
    { fix y
      assume y: y ∈ convex hull S y ∉ ?rhs
      have *: False if
        ua: ∀x∈S. 0 ≤ u x sum u S = 1 ¬ 0 < u a a ∈ S
        and yT: y = (∑x∈S. u x *R x) y ∈ T open T
         and sb: T ∩ affine hull S ⊆ {w. ∃u. (∀x∈S. 0 ≤ u x) ∧ sum u S = 1 ∧
(∑x∈S. u x *R x) = w}
      for u T a
      proof -
        have ua0: u a = 0
          using ua by auto
        obtain b where b: b∈S a ≠ b
          using ua False by auto
        obtain e where e: 0 < e ball (∑x∈S. u x *R x) e ⊆ T
          using yT by (auto elim: openE)
        with b obtain d where d: 0 < d norm(d *R (a-b)) < e
          by (auto intro: that [of e / 2 / norm(a-b)])
        have (∑x∈S. u x *R x) ∈ affine hull S
          using yT y by (metis affine_hull_convex_hull hull_redundant_eq)
        then have (∑x∈S. u x *R x) - d *R (a - b) ∈ affine hull S
          using ua b by (auto simp: hull_inc intro: mem_affine_3_minus2)
        then have y - d *R (a - b) ∈ T ∩ affine hull S
          using d e yT by auto
```

```
      then obtain v where v: ∀x∈S. 0 ≤ v x
        sum v S = 1
        (∑x∈S. v x *R x) = (∑x∈S. u x *R x) - d *R (a - b)
        using subsetD [OF sb] yT
        by auto
      have aff: ⋀u. sum u S = 0 ⟹ (∀v∈S. u v = 0) ∨ (∑v∈S. u v *R v) ≠ 0
        using assms by (simp add: affine_dependent_explicit_finite fs)
      show False
        using ua b d v aff [of λx. (v x - u x) - (if x = a then -d else if x = b
then d else 0)]
        by (auto simp: algebra_simps sum_subtractf sum.distrib)
    qed
    have y ∉ rel_interior (convex hull S)
      using y
      apply (simp add: mem_rel_interior)
      apply (auto simp: convex_hull_finite [OF fs])
      apply (drule_tac x=u in spec)
      apply (auto intro: *)
      done
  } with rel_interior_subset show ?lhs ≤ ?rhs
    by blast
 qed
qed

lemma interior_convex_hull_explicit_minimal:
 fixes S :: 'a::euclidean_space set
 assumes ¬ affine_dependent S
 shows
  interior(convex hull S) =
          (if card(S) ≤ DIM('a) then {}
           else {y. ∃u. (∀x ∈ S. 0 < u x) ∧ sum u S = 1 ∧ (∑x∈S. u x *R x)
= y})
  (is _ = (if _ then _ else ?rhs))
proof (clarsimp simp: aff_independent_finite empty_interior_convex_hull assms)
 assume S: ¬ card S ≤ DIM('a)
 have interior (convex hull S) = rel_interior(convex hull S)
  using assms S by (simp add: affine_independent_span_gt rel_interior_interior)
 then show interior(convex hull S) = ?rhs
   by (simp add: assms S rel_interior_convex_hull_explicit)
qed

lemma interior_convex_hull_explicit:
 fixes S :: 'a::euclidean_space set
 assumes ¬ affine_dependent S
 shows
  interior(convex hull S) =
          (if card(S) ≤ DIM('a) then {}
           else {y. ∃u. (∀x ∈ S. 0 < u x ∧ u x < 1) ∧ sum u S = 1 ∧ (∑x∈S.
u x *R x) = y})
```

```
proof −
  { fix u :: 'a ⇒ real and a
    assume card Basis < card S and u: ⋀x. x∈S ⟹ 0 < u x sum u S = 1 and
a: a ∈ S
    then have cs: Suc 0 < card S
      by (metis DIM_positive less_trans_Suc)
    obtain b where b: b ∈ S a ≠ b
    proof (cases S ≤ {a})
      case True
      then show thesis
        using cs subset_singletonD by fastforce
    qed blast
    have u a + u b ≤ sum u {a,b}
      using a b by simp
    also have ... ≤ sum u S
      using a b u
     by (intro Groups_Big.sum_mono2) (auto simp: less_imp_le aff_independent_finite
assms)
    finally have u a < 1
      using ‹b ∈ S› u by fastforce
  } note [simp] = this
  show ?thesis
   using assms by (force simp add: not_le interior_convex_hull_explicit_minimal)
qed

lemma interior_closed_segment_ge2:
  fixes a :: 'a::euclidean_space
  assumes 2 ≤ DIM('a)
    shows  interior(closed_segment a b) = {}
using assms unfolding segment_convex_hull
proof −
  have card {a, b} ≤ DIM('a)
    using assms
    by (simp add: card_insert_if linear not_less_eq_eq numeral_2_eq_2)
  then show interior (convex hull {a, b}) = {}
    by (metis empty_interior_convex_hull finite.insertI finite.emptyI)
qed

lemma interior_open_segment:
  fixes a :: 'a::euclidean_space
  shows  interior(open_segment a b) =
                (if 2 ≤ DIM('a) then {} else open_segment a b)
proof (simp add: not_le, intro conjI impI)
  assume 2 ≤ DIM('a)
  then show interior (open_segment a b) = {}
   using interior_closed_segment_ge2 interior_mono segment_open_subset_closed
by blast
next
  assume le2: DIM('a) < 2
```

```
  show interior (open_segment a b) = open_segment a b
  proof (cases a = b)
    case True then show ?thesis by auto
  next
    case False
    with le2 have affine hull (open_segment a b) = UNIV
      by (simp add: False affine_independent_span_gt)
    then show interior (open_segment a b) = open_segment a b
      using rel_interior_interior rel_interior_open_segment by blast
  qed
qed

lemma interior_closed_segment:
  fixes a :: 'a::euclidean_space
  shows interior(closed_segment a b) =
                (if 2 ≤ DIM('a) then {} else open_segment a b)
proof (cases a = b)
  case True then show ?thesis by simp
next
  case False
  then have closure (open_segment a b) = closed_segment a b
    by simp
  then show ?thesis
     by (metis (no_types) convex_interior_closure convex_open_segment inte-
rior_open_segment)
qed

lemmas interior_segment = interior_closed_segment interior_open_segment

lemma closed_segment_eq [simp]:
  fixes a :: 'a::euclidean_space
  shows closed_segment a b = closed_segment c d ⟷ {a,b} = {c,d}
proof
  assume abcd: closed_segment a b = closed_segment c d
  show {a,b} = {c,d}
  proof (cases a=b ∨ c=d)
    case True with abcd show ?thesis by force
  next
    case False
    then have neq: a ≠ b ∧ c ≠ d by force
    have ∗: closed_segment c d − {a, b} = rel_interior (closed_segment c d)
     using neq abcd by (metis (no_types) open_segment_def rel_interior_closed_segment)
    have b ∈ {c, d}
    proof −
      have insert b (closed_segment c d) = closed_segment c d
        using abcd by blast
      then show ?thesis
      by (metis DiffD2 Diff_insert2 False ∗ insertI1 insert_Diff_if open_segment_def
rel_interior_closed_segment)
```

qed
moreover have $a \in \{c,\ d\}$
by (*metis Diff_iff False* ∗ *abcd ends_in_segment*(*1*) *insertI1 open_segment_def rel_interior_closed_segment*)
ultimately show $\{a,\ b\} = \{c,\ d\}$
using *neq* **by** *fastforce*
qed
next
assume $\{a,b\} = \{c,d\}$
then show *closed_segment a b* = *closed_segment c d*
by (*simp add*: *segment_convex_hull*)
qed

lemma *closed_open_segment_eq* [*simp*]:
fixes *a* :: *'a*::*euclidean_space*
shows *closed_segment a b* ≠ *open_segment c d*
by (*metis DiffE closed_segment_neq_empty closure_closed_segment closure_open_segment ends_in_segment*(*1*) *insertI1 open_segment_def*)

lemma *open_closed_segment_eq* [*simp*]:
fixes *a* :: *'a*::*euclidean_space*
shows *open_segment a b* ≠ *closed_segment c d*
using *closed_open_segment_eq* **by** *blast*

lemma *open_segment_eq* [*simp*]:
fixes *a* :: *'a*::*euclidean_space*
shows *open_segment a b* = *open_segment c d* ⟷ *a* = *b* ∧ *c* = *d* ∨ $\{a,b\} = \{c,d\}$
(**is** *?lhs* = *?rhs*)
proof
assume *abcd*: *?lhs*
show *?rhs*
proof (*cases a*=*b* ∨ *c*=*d*)
case *True* **with** *abcd* **show** *?thesis*
using *finite_open_segment* **by** *fastforce*
next
case *False*
then have *a2*: *a* ≠ *b* ∧ *c* ≠ *d* **by** *force*
with *abcd* **show** *?rhs*
unfolding *open_segment_def*
by (*metis* (*no_types*) *abcd closed_segment_eq closure_open_segment*)
qed
next
assume *?rhs*
then show *?lhs*
by (*metis Diff_cancel convex_hull_singleton insert_absorb2 open_segment_def segment_convex_hull*)
qed

5.0.7 Similar results for closure and (relative or absolute) frontier

lemma *closure_convex_hull* [*simp*]:
fixes S :: *'a*::*euclidean_space set*
shows *compact S* ==> *closure*(*convex hull S*) = *convex hull S*
by (*simp add*: *compact_imp_closed compact_convex_hull*)

lemma *rel_frontier_convex_hull_explicit*:
fixes S :: *'a*::*euclidean_space set*
assumes ¬ *affine_dependent S*
shows *rel_frontier*(*convex hull S*) =
{*y*. ∃ *u*. (∀ *x* ∈ *S*. 0 ≤ *u x*) ∧ (∃ *x* ∈ *S*. *u x* = 0) ∧ *sum u S* = 1 ∧ *sum* (λ*x*. *u x* ∗*R x*) *S* = *y*}
proof −
have *fs*: *finite S*
using *assms* **by** (*simp add*: *aff_independent_finite*)
have ⋀*u y v*.
⟦*y* ∈ *S*; *u y* = 0; *sum u S* = 1; ∀ *x*∈*S*. 0 < *v x*;
sum v S = 1; (∑ *x*∈*S*. *v x* ∗*R x*) = (∑ *x*∈*S*. *u x* ∗*R x*)⟧
⟹ ∃ *u*. *sum u S* = 0 ∧ (∃ *v*∈*S*. *u v* ≠ 0) ∧ (∑ *v*∈*S*. *u v* ∗*R v*) = 0
apply (*rule_tac x* = λ*x*. *u x* − *v x* **in** *exI*)
apply (*force simp*: *sum_subtractf scaleR_diff_left*)
done
then show *?thesis*
using *fs assms*
apply (*simp add*: *rel_frontier_def finite_imp_compact rel_interior_convex_hull_explicit*)
apply (*auto simp*: *convex_hull_finite*)
apply (*metis less_eq_real_def*)
by (*simp add*: *affine_dependent_explicit_finite*)
qed

lemma *frontier_convex_hull_explicit*:
fixes S :: *'a*::*euclidean_space set*
assumes ¬ *affine_dependent S*
shows *frontier*(*convex hull S*) =
{*y*. ∃ *u*. (∀ *x* ∈ *S*. 0 ≤ *u x*) ∧ (*DIM* (*'a*) < *card S* ⟶ (∃ *x* ∈ *S*. *u x* = 0))
∧
sum u S = 1 ∧ *sum* (λ*x*. *u x* ∗*R x*) *S* = *y*}
proof −
have *fs*: *finite S*
using *assms* **by** (*simp add*: *aff_independent_finite*)
show *?thesis*
proof (*cases DIM* (*'a*) < *card S*)
case *True*
with *assms fs* **show** *?thesis*
by (*simp add*: *rel_frontier_def frontier_def rel_frontier_convex_hull_explicit*
[*symmetric*]
interior_convex_hull_explicit_minimal rel_interior_convex_hull_explicit)
next

```
    case False
    then have card S ≤ DIM ('a)
      by linarith
    then show ?thesis
      using assms fs
    apply (simp add: frontier_def interior_convex_hull_explicit finite_imp_compact)
      apply (simp add: convex_hull_finite)
      done
  qed
qed

lemma rel_frontier_convex_hull_cases:
  fixes S :: 'a::euclidean_space set
  assumes ¬ affine_dependent S
  shows rel_frontier(convex hull S) = ⋃{convex hull (S - {x}) |x. x ∈ S}
proof -
  have fs: finite S
    using assms by (simp add: aff_independent_finite)
  { fix u a
  have ∀x∈S. 0 ≤ u x ⟹ a ∈ S ⟹ u a = 0 ⟹ sum u S = 1 ⟹
          ∃x v. x ∈ S ∧
               (∀x∈S - {x}. 0 ≤ v x) ∧
                  sum v (S - {x}) = 1 ∧ (∑x∈S - {x}. v x *R x) = (∑x∈S. u
x *R x)
    apply (rule_tac x=a in exI)
    apply (rule_tac x=u in exI)
    apply (simp add: Groups_Big.sum_diff1 fs)
    done }
  moreover
  { fix a u
    have a ∈ S ⟹ ∀x∈S - {a}. 0 ≤ u x ⟹ sum u (S - {a}) = 1 ⟹
          ∃v. (∀x∈S. 0 ≤ v x) ∧
              (∃x∈S. v x = 0) ∧ sum v S = 1 ∧ (∑x∈S. v x *R x) = (∑x∈S -
{a}. u x *R x)
    apply (rule_tac x=λx. if x = a then 0 else u x in exI)
    apply (auto simp: sum.If_cases Diff_eq if_smult fs)
    done }
  ultimately show ?thesis
    using assms
    apply (simp add: rel_frontier_convex_hull_explicit)
    apply (auto simp add: convex_hull_finite fs Union_SetCompr_eq)
    done
qed

lemma frontier_convex_hull_eq_rel_frontier:
  fixes S :: 'a::euclidean_space set
  assumes ¬ affine_dependent S
  shows frontier(convex hull S) =
        (if card S ≤ DIM ('a) then convex hull S else rel_frontier(convex hull S))
```

using *assms*
unfolding *rel_frontier_def frontier_def*
by (*simp add*: *affine_independent_span_gt rel_interior_interior*
finite_imp_compact empty_interior_convex_hull aff_independent_finite)

lemma *frontier_convex_hull_cases*:
fixes $S :: 'a::euclidean\_space\ set$
assumes $\neg$ *affine_dependent S*
shows *frontier*(*convex hull S*) =
(*if card S* $\leq$ *DIM* ($'a$) *then convex hull S else* $\bigcup$ {*convex hull* ($S - \{x\}$)
$|x.\ x \in S\}$)
by (*simp add*: *assms frontier_convex_hull_eq_rel_frontier rel_frontier_convex_hull_cases*)

lemma *in_frontier_convex_hull*:
fixes $S :: 'a::euclidean\_space\ set$
assumes *finite S card S* $\leq$ *Suc* (*DIM* ($'a$)) $x \in S$
shows $x \in$ *frontier*(*convex hull S*)
proof (*cases affine_dependent S*)
case *True*
with *assms* **obtain** y **where** $y \in S$ **and** y: $y \in$ *affine hull* ($S - \{y\}$)
by (*auto simp*: *affine_dependent_def*)
moreover have $x \in$ *closure* (*convex hull S*)
by (*meson closure_subset hull_inc subset_eq* ‹$x \in S$›)
moreover have $x \notin$ *interior* (*convex hull S*)
using *assms*
by (*metis Suc_mono affine_hull_convex_hull affine_hull_nonempty_interior*
‹$y \in S$› *y card.remove empty_iff empty_interior_affine_hull finite_Diff hull_redundant*
insert_Diff interior_UNIV not_less)
ultimately show *?thesis*
unfolding *frontier_def* **by** *blast*
next
case *False*
{ assume *card S* = *Suc* (*card Basis*)
then have *cs*: *Suc 0* < *card S*
by (*simp*)
with *subset_singletonD* **have** $\exists\, y \in S.\ y \neq x$
by (*cases* $S \leq \{x\}$) *fastforce+*
} note [*dest*!] = *this*
show *?thesis* **using** *assms*
unfolding *frontier_convex_hull_cases* [*OF False*] *Union_SetCompr_eq*
by (*auto simp*: *le_Suc_eq hull_inc*)
qed

lemma *not_in_interior_convex_hull*:
fixes $S :: 'a::euclidean\_space\ set$
assumes *finite S card S* $\leq$ *Suc* (*DIM* ($'a$)) $x \in S$
shows $x \notin$ *interior*(*convex hull S*)
using *in_frontier_convex_hull* [*OF assms*]
by (*metis Diff_iff frontier_def*)

lemma *interior_convex_hull_eq_empty*:
fixes $S :: 'a::euclidean\_space\ set$
assumes $card\ S = Suc\ (DIM\ ('a))$
shows $interior(convex\ hull\ S) = \{\} \longleftrightarrow affine\_dependent\ S$
proof
show $affine\_dependent\ S \Longrightarrow interior\ (convex\ hull\ S) = \{\}$
proof (*clarsimp simp*: *affine_dependent_def*)
fix *a b*
assume $b \in S\ b \in affine\ hull\ (S - \{b\})$
then have $interior(affine\ hull\ S) = \{\}$ **using** *assms*
by (*metis DIM_positive One_nat_def Suc_mono card.remove card.infinite empty_interior_affine_hull eq_iff hull_redundant insert_Diff not_less zero_le_one*)
then show $interior\ (convex\ hull\ S) = \{\}$
using *affine_hull_nonempty_interior* **by** *fastforce*
qed
next
show $interior\ (convex\ hull\ S) = \{\} \Longrightarrow affine\_dependent\ S$
by (*metis affine_hull_convex_hull affine_hull_empty affine_independent_span_eq assms convex_convex_hull empty_not_UNIV rel_interior_eq_empty rel_interior_interior*)
qed

5.0.8 Coplanarity, and collinearity in terms of affine hull

definition *coplanar* **where**
$coplanar\ S \equiv \exists\, u\ v\ w.\ S \subseteq affine\ hull\ \{u,v,w\}$

lemma *collinear_affine_hull*:
$collinear\ S \longleftrightarrow (\exists\, u\ v.\ S \subseteq affine\ hull\ \{u,v\})$
proof (*cases* $S=\{\}$)
case *True* **then show** *?thesis*
by *simp*
next
case *False*
then obtain x **where** x: $x \in S$ **by** *auto*
{ fix *u*
assume $*$: $\bigwedge x\ y.\ [\![x\in S;\ y\in S]\!] \Longrightarrow \exists\, c.\ x - y = c *_R u$
have $\bigwedge y\ c.\ x - y = c *_R u \Longrightarrow \exists\, a\ b.\ y = a *_R x + b *_R (x + u) \wedge a + b = 1$
by (*rule_tac x=1+c* **in** *exI*, *rule_tac x=−c* **in** *exI*, *simp add*: *algebra_simps*)
then have $\exists\, u\ v.\ S \subseteq \{a *_R u + b *_R v\ |a\ b.\ a + b = 1\}$
using $*$ [*OF x*] **by** (*rule_tac x=x* **in** *exI*, *rule_tac x=x+u* **in** *exI*, *force*)
} moreover
{ fix *u v x y*
assume $*$: $S \subseteq \{a *_R u + b *_R v\ |a\ b.\ a + b = 1\}$
have $\exists\, c.\ x - y = c *_R (v-u)$ **if** $x\in S\ y\in S$
proof −
obtain $a\ r$ **where** $a + r = 1\ x = a *_R u + r *_R v$
using $*$ ‹$x \in S$› **by** *blast*
moreover

```
    obtain b s where b + s = 1 y = b *R u + s *R v
      using * ‹y ∈ S› by blast
    ultimately have x − y = (r−s) *R (v−u)
      by (simp add: algebra_simps) (metis scaleR_left.add)
    then show ?thesis
      by blast
  qed
 } ultimately
 show ?thesis
 unfolding collinear_def affine_hull_2
  by blast
qed

lemma collinear_closed_segment [simp]: collinear (closed_segment a b)
 by (metis affine_hull_convex_hull collinear_affine_hull hull_subset segment_convex_hull)

lemma collinear_open_segment [simp]: collinear (open_segment a b)
 unfolding open_segment_def
 by (metis convex_hull_subset_affine_hull segment_convex_hull dual_order.trans
     convex_hull_subset_affine_hull Diff_subset collinear_affine_hull)

lemma collinear_between_cases:
 fixes c :: 'a::euclidean_space
 shows collinear {a,b,c} ⟷ between (b,c) a ∨ between (c,a) b ∨ between (a,b) c
       (is ?lhs = ?rhs)
proof
 assume ?lhs
 then obtain u v where uv: ⋀x. x ∈ {a, b, c} ⟹ ∃ c. x = u + c *R v
  by (auto simp: collinear_alt)
 show ?rhs
  using uv [of a] uv [of b] uv [of c] by (auto simp: between_1)
next
 assume ?rhs
 then show ?lhs
  unfolding between_mem_convex_hull
 by (metis (no_types, opaque_lifting) collinear_closed_segment collinear_subset
hull_redundant hull_subset insert_commute segment_convex_hull)
qed


lemma subset_continuous_image_segment_1:
 fixes f :: 'a::euclidean_space ⇒ real
 assumes continuous_on (closed_segment a b) f
 shows closed_segment (f a) (f b) ⊆ image f (closed_segment a b)
by (metis connected_segment convex_contains_segment ends_in_segment imageI
      is_interval_connected_1 is_interval_convex connected_continuous_image
[OF assms])

lemma continuous_injective_image_segment_1:
```

```
  fixes f :: 'a::euclidean_space ⇒ real
  assumes contf: continuous_on (closed_segment a b) f
      and injf: inj_on f (closed_segment a b)
  shows f ‘ (closed_segment a b) = closed_segment (f a) (f b)
proof
  show closed_segment (f a) (f b) ⊆ f ‘ closed_segment a b
    by (metis subset_continuous_image_segment_1 contf)
  show f ‘ closed_segment a b ⊆ closed_segment (f a) (f b)
  proof (cases a = b)
    case True
    then show ?thesis by auto
  next
    case False
    then have fnot: f a ≠ f b
      using inj_onD injf by fastforce
    moreover
    have f a ∉ open_segment (f c) (f b) if c: c ∈ closed_segment a b for c
    proof (clarsimp simp add: open_segment_def)
      assume fa: f a ∈ closed_segment (f c) (f b)
      moreover have closed_segment (f c) (f b) ⊆ f ‘ closed_segment c b
      by (meson closed_segment_subset contf continuous_on_subset convex_closed_segment
ends_in_segment(2) subset_continuous_image_segment_1 that)
      ultimately have f a ∈ f ‘ closed_segment c b
        by blast
      then have a: a ∈ closed_segment c b
      by (meson ends_in_segment inj_on_image_mem_iff injf subset_closed_segment
that)
      have cb: closed_segment c b ⊆ closed_segment a b
        by (simp add: closed_segment_subset that)
      show f a = f c
      proof (rule between_antisym)
        show between (f c, f b) (f a)
          by (simp add: between_mem_segment fa)
        show between (f a, f b) (f c)
            by (metis a cb between_antisym between_mem_segment between_triv1
subset_iff)
      qed
    qed
    moreover
    have f b ∉ open_segment (f a) (f c) if c: c ∈ closed_segment a b for c
    proof (clarsimp simp add: open_segment_def fnot eq_commute)
      assume fb: f b ∈ closed_segment (f a) (f c)
      moreover have closed_segment (f a) (f c) ⊆ f ‘ closed_segment a c
      by (meson contf continuous_on_subset ends_in_segment(1) subset_closed_segment
subset_continuous_image_segment_1 that)
      ultimately have f b ∈ f ‘ closed_segment a c
        by blast
      then have b: b ∈ closed_segment a c
      by (meson ends_in_segment inj_on_image_mem_iff injf subset_closed_segment
```

that)
    have *ca*: *closed_segment a c* ⊆ *closed_segment a b*
     by (*simp add*: *closed_segment_subset that*)
    show *f b* = *f c*
    proof (*rule between_antisym*)
     show *between* (*f c*, *f a*) (*f b*)
      by (*simp add*: *between_commute between_mem_segment fb*)
     show *between* (*f b*, *f a*) (*f c*)
      by (*metis b between_antisym between_commute between_mem_segment between_triv2 that*)
    qed
   qed
   ultimately show *?thesis*
    by (*force simp*: *closed_segment_eq_real_ivl open_segment_eq_real_ivl split*: *if_split_asm*)
  qed
qed

lemma *continuous_injective_image_open_segment_1*:
  fixes *f* :: *'a*::*euclidean_space* ⇒ *real*
  assumes *contf*: *continuous_on* (*closed_segment a b*) *f*
     and *injf*: *inj_on f* (*closed_segment a b*)
    shows *f* ‘ (*open_segment a b*) = *open_segment* (*f a*) (*f b*)
proof −
  have *f* ‘ (*open_segment a b*) = *f* ‘ (*closed_segment a b*) − {*f a*, *f b*}
     by (*metis* (*no_types*, *opaque_lifting*) *empty_subsetI ends_in_segment image_insert image_is_empty inj_on_image_set_diff injf insert_subset open_segment_def segment_open_subset_closed*)
  also have ... = *open_segment* (*f a*) (*f b*)
    using *continuous_injective_image_segment_1* [*OF assms*]
    by (*simp add*: *open_segment_def inj_on_image_set_diff* [*OF injf*])
  finally show *?thesis* **.**
qed

lemma *collinear_imp_coplanar*:
  collinear s ==> *coplanar s*
by (*metis collinear_affine_hull coplanar_def insert_absorb2*)

lemma *collinear_small*:
  assumes *finite s card s* ≤ *2*
    shows *collinear s*
proof −
  have *card s* = *0* ∨ *card s* = *1* ∨ *card s* = *2*
    using *assms* **by** *linarith*
  then show *?thesis* **using** *assms*
    using *card_eq_SucD numeral_2_eq_2* **by** (*force simp*: *card_1_singleton_iff*)
qed

lemma *coplanar_small*:

```
  assumes finite s card s ≤ 3
    shows coplanar s
proof −
  consider card s ≤ 2 | card s = Suc (Suc (Suc 0))
    using assms by linarith
  then show ?thesis
  proof cases
    case 1
    then show ?thesis
      by (simp add: ‹finite s› collinear_imp_coplanar collinear_small)
  next
    case 2
    then show ?thesis
      using hull_subset [of {_,_,_}]
      by (fastforce simp: coplanar_def dest!: card_eq_SucD)
  qed
qed

lemma coplanar_empty: coplanar {}
  by (simp add: coplanar_small)

lemma coplanar_sing: coplanar {a}
  by (simp add: coplanar_small)

lemma coplanar_2: coplanar {a,b}
  by (auto simp: card_insert_if coplanar_small)

lemma coplanar_3: coplanar {a,b,c}
  by (auto simp: card_insert_if coplanar_small)

lemma collinear_affine_hull_collinear: collinear(affine hull s) ⟷ collinear s
  unfolding collinear_affine_hull
  by (metis affine_affine_hull subset_hull hull_hull hull_mono)

lemma coplanar_affine_hull_coplanar: coplanar(affine hull s) ⟷ coplanar s
  unfolding coplanar_def
  by (metis affine_affine_hull subset_hull hull_hull hull_mono)

lemma coplanar_linear_image:
  fixes f :: 'a::euclidean_space ⇒ 'b::real_normed_vector
  assumes coplanar S linear f shows coplanar(f ‘ S)
proof −
  { fix u v w
    assume S ⊆ affine hull {u, v, w}
    then have f ‘ S ⊆ f ‘ (affine hull {u, v, w})
      by (simp add: image_mono)
    then have f ‘ S ⊆ affine hull (f ‘ {u, v, w})
      by (metis assms(2) linear_conv_bounded_linear affine_hull_linear_image)
  } then
```

show *?thesis*
by *auto* (*meson assms*(*1*) *coplanar_def*)
qed

lemma *coplanar_translation_imp*:
assumes *coplanar S* **shows** *coplanar* ((λ*x*. *a* + *x*) ‘ *S*)
proof −
obtain *u v w* **where** *S* ⊆ *affine hull* {*u*,*v*,*w*}
by (*meson assms coplanar_def*)
then have (+) *a* ‘ *S* ⊆ *affine hull* {*u* + *a*, *v* + *a*, *w* + *a*}
using *affine_hull_translation* [*of a* {*u*,*v*,*w*} **for** *u v w*]
by (*force simp*: *add.commute*)
then show *?thesis*
unfolding *coplanar_def* **by** *blast*
qed

lemma *coplanar_translation_eq*: *coplanar*((λ*x*. *a* + *x*) ‘ *S*) ⟷ *coplanar S*
by (*metis* (*no_types*) *coplanar_translation_imp translation_galois*)

lemma *coplanar_linear_image_eq*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *linear f inj f* **shows** *coplanar*(*f* ‘ *S*) = *coplanar S*
proof
assume *coplanar S*
then show *coplanar* (*f* ‘ *S*)
using *assms*(*1*) *coplanar_linear_image* **by** *blast*
next
obtain *g* **where** *g*: *linear g g* ∘ *f* = *id*
using *linear_injective_left_inverse* [*OF assms*]
by *blast*
assume *coplanar* (*f* ‘ *S*)
then show *coplanar S*
by (*metis coplanar_linear_image g*(*1*) *g*(*2*) *id_apply image_comp image_id*)
qed

lemma *coplanar_subset*: ⟦*coplanar t*; *S* ⊆ *t*⟧ ⟹ *coplanar S*
by (*meson coplanar_def order_trans*)

lemma *affine_hull_3_imp_collinear*: *c* ∈ *affine hull* {*a*,*b*} ⟹ *collinear* {*a*,*b*,*c*}
by (*metis collinear_2 collinear_affine_hull_collinear hull_redundant insert_commute*)

lemma *collinear_3_imp_in_affine_hull*:
assumes *collinear* {*a*,*b*,*c*} *a* ≠ *b* **shows** *c* ∈ *affine hull* {*a*,*b*}
proof −
obtain *u x y* **where** *b* − *a* = *y* ∗*R u c* − *a* = *x* ∗*R u*
using *assms* **unfolding** *collinear_def* **by** *auto*
with ‹*a* ≠ *b*› **have** ∃*v*. *c* = (*1* − *x* / *y*) ∗*R a* + *v* ∗*R b* ∧ *1* − *x* / *y* + *v* = *1*
by (*simp add*: *algebra_simps*)
then show *?thesis*

by (*simp add*: *hull_inc mem_affine*)
qed

lemma *collinear_3_affine_hull*:
assumes $a \neq b$
shows *collinear* $\{a,b,c\} \longleftrightarrow c \in$ *affine hull* $\{a,b\}$
using *affine_hull_3_imp_collinear assms collinear_3_imp_in_affine_hull* **by** *blast*

lemma *collinear_3_eq_affine_dependent*:
collinear$\{a,b,c\} \longleftrightarrow a = b \vee a = c \vee b = c \vee$ *affine_dependent* $\{a,b,c\}$
proof (*cases* $a = b \vee a = c \vee b = c$)
case *True*
then show *?thesis*
by (*auto simp*: *insert_commute*)
next
case *False*
then have *collinear*$\{a,b,c\}$ **if** *affine_dependent* $\{a,b,c\}$
using *that* **unfolding** *affine_dependent_def*
by (*auto simp*: *insert_Diff_if*; *metis affine_hull_3_imp_collinear insert_commute*)
moreover
have *affine_dependent* $\{a,b,c\}$ **if** *collinear*$\{a,b,c\}$
using *False that* **by** (*auto simp*: *affine_dependent_def collinear_3_affine_hull insert_Diff_if*)
ultimately
show *?thesis*
using *False* **by** *blast*
qed

lemma *affine_dependent_imp_collinear_3*:
affine_dependent $\{a,b,c\} \Longrightarrow$ *collinear*$\{a,b,c\}$
by (*simp add*: *collinear_3_eq_affine_dependent*)

lemma *collinear_3*: *NO_MATCH 0 x* $\Longrightarrow$ *collinear* $\{x,y,z\} \longleftrightarrow$ *collinear* $\{0, x-y, z-y\}$
by (*auto simp add*: *collinear_def*)

lemma *collinear_3_expand*:
collinear$\{a,b,c\} \longleftrightarrow a = c \vee (\exists u.\ b = u *_R a + (1 - u) *_R c)$
proof −
have *collinear*$\{a,b,c\}$ = *collinear*$\{a,c,b\}$
by (*simp add*: *insert_commute*)
also have ... = *collinear* $\{0, a - c, b - c\}$
by (*simp add*: *collinear_3*)
also have ... $\longleftrightarrow (a = c \vee b = c \vee (\exists ca.\ b - c = ca *_R (a - c)))$
by (*simp add*: *collinear_lemma*)
also have ... $\longleftrightarrow a = c \vee (\exists u.\ b = u *_R a + (1 - u) *_R c)$
by (*cases* $a = c \vee b = c$) (*auto simp*: *algebra_simps*)
finally show *?thesis* **.**

qed

lemma *collinear_aff_dim*: *collinear S* $\longleftrightarrow$ *aff_dim S* $\leq$ *1*
proof
assume *collinear S*
then obtain *u* **and** *v* :: *'a* **where** *aff_dim S* $\leq$ *aff_dim* {*u*,*v*}
by (*metis* ‹*collinear S*› *aff_dim_affine_hull aff_dim_subset collinear_affine_hull*)
then show *aff_dim S* $\leq$ *1*
using *order_trans* **by** *fastforce*
next
assume *aff_dim S* $\leq$ *1*
then have *le1*: *aff_dim* (*affine hull S*) $\leq$ *1*
by *simp*
obtain *B* **where** *B* $\subseteq$ *S* **and** *B*: $\neg$ *affine_dependent B affine hull S* = *affine hull B*
using *affine_basis_exists* [*of S*] **by** *auto*
then have *finite B card B* $\leq$ *2*
using *B le1* **by** (*auto simp*: *affine_independent_iff_card*)
then have *collinear B*
by (*rule collinear_small*)
then show *collinear S*
by (*metis* ‹*affine hull S* = *affine hull B*› *collinear_affine_hull_collinear*)
qed

lemma *collinear_midpoint*: *collinear*{*a*, *midpoint a b*, *b*}
proof −
have §: ⟦*a* $\neq$ *midpoint a b*; *b* − *midpoint a b* $\neq$ − *1* $*_R$ (*a* − *midpoint a b*)⟧ $\Longrightarrow$ *b* = *midpoint a b*
by (*simp add*: *algebra_simps*)
show *?thesis*
by (*auto simp*: *collinear_3 collinear_lemma intro*: §)
qed

lemma *midpoint_collinear*:
fixes *a b c* :: *'a*::*real_normed_vector*
assumes *a* $\neq$ *c*
shows *b* = *midpoint a c* $\longleftrightarrow$ *collinear*{*a*,*b*,*c*} $\wedge$ *dist a b* = *dist b c*
proof −
have $*$: *a* − (*u* $*_R$ *a* + (*1* − *u*) $*_R$ *c*) = (*1* − *u*) $*_R$ (*a* − *c*)
u $*_R$ *a* + (*1* − *u*) $*_R$ *c* − *c* = *u* $*_R$ (*a* − *c*)
|*1* − *u*| = |*u*| $\longleftrightarrow$ *u* = *1*/*2* **for** *u*::*real*
by (*auto simp*: *algebra_simps*)
have *b* = *midpoint a c* $\Longrightarrow$ *collinear*{*a*,*b*,*c*}
using *collinear_midpoint* **by** *blast*
moreover have *b* = *midpoint a c* $\longleftrightarrow$ *dist a b* = *dist b c* **if** *collinear*{*a*,*b*,*c*}
proof −
consider *a* = *c* | *u* **where** *b* = *u* $*_R$ *a* + (*1* − *u*) $*_R$ *c*
using ‹*collinear* {*a*,*b*,*c*}› **unfolding** *collinear_3_expand* **by** *blast*
then show *?thesis*

```
    proof cases
      case 2
      with assms have dist a b = dist b c ⟹ b = midpoint a c
      by (simp add: dist_norm * midpoint_def scaleR_add_right del: divide_const_simps)
      then show ?thesis
        by (auto simp: dist_midpoint)
    qed (use assms in auto)
  qed
  ultimately show ?thesis by blast
qed

lemma between_imp_collinear:
  fixes x :: 'a :: euclidean_space
  assumes between (a,b) x
    shows collinear {a,x,b}
proof (cases x = a ∨ x = b ∨ a = b)
  case True with assms show ?thesis
    by (auto simp: dist_commute)
next
  case False
  then have False if ⋀c. b - x ≠ c *R (a - x)
    using that [of -(norm(b - x) / norm(x - a))] assms
   by (simp add: between_norm vector_add_divide_simps flip: real_vector.scale_minus_right)
  then show ?thesis
    by (auto simp: collinear_3 collinear_lemma)
qed

lemma midpoint_between:
  fixes a b :: 'a::euclidean_space
  shows b = midpoint a c ⟷ between (a,c) b ∧ dist a b = dist b c
proof (cases a = c)
  case False
  show ?thesis
     using False between_imp_collinear between_midpoint(1) midpoint_collinear
by blast
qed (auto simp: dist_commute)

lemma collinear_triples:
  assumes a ≠ b
    shows collinear(insert a (insert b S)) ⟷ (∀x ∈ S. collinear{a,b,x})
          (is ?lhs = ?rhs)
proof safe
  fix x
  assume ?lhs and x ∈ S
  then show collinear {a, b, x}
    using collinear_subset by force
next
  assume ?rhs
  then have ∀x ∈ S. collinear{a,x,b}
```

```
    by (simp add: insert_commute)
  then have *: ∃u. x = u *R a + (1 - u) *R b if x ∈ insert a (insert b S) for x
    using that assms collinear_3_expand by fastforce+
  have ∃c. x - y = c *R (b - a)
    if x: x ∈ insert a (insert b S) and y: y ∈ insert a (insert b S) for x y
  proof -
    obtain u v where x = u *R a + (1 - u) *R b y = v *R a + (1 - v) *R b
      using * x y by presburger
    then have x - y = (v - u) *R (b - a)
      by (simp add: scale_left_diff_distrib scale_right_diff_distrib)
    then show ?thesis ..
  qed
  then show ?lhs
    unfolding collinear_def by metis
qed

lemma collinear_4_3:
  assumes a ≠ b
    shows collinear {a,b,c,d} ⟷ collinear{a,b,c} ∧ collinear{a,b,d}
  using collinear_triples [OF assms, of {c,d}] by (force simp:)

lemma collinear_3_trans:
  assumes collinear{a,b,c} collinear{b,c,d} b ≠ c
    shows collinear{a,b,d}
proof -
  have collinear{b,c,a,d}
    by (metis (full_types) assms collinear_4_3 insert_commute)
  then show ?thesis
    by (simp add: collinear_subset)
qed

lemma affine_hull_2_alt:
  fixes a b :: 'a::real_vector
  shows affine hull {a,b} = range (λu. a + u *R (b - a))
proof -
  have 1: u *R a + v *R b = a + v *R (b - a) if u + v = 1 for u v
    using that
    by (simp add: algebra_simps flip: scaleR_add_left)
  have 2: a + u *R (b - a) = (1 - u) *R a + u *R b for u
    by (auto simp: algebra_simps)
  show ?thesis
    by (force simp add: affine_hull_2 dest: 1 intro!: 2)
qed

lemma interior_convex_hull_3_minimal:
  fixes a :: 'a::euclidean_space
  assumes ¬ collinear{a,b,c} and 2: DIM('a) = 2
  shows interior(convex hull {a,b,c}) =
        {v. ∃x y z. 0 < x ∧ 0 < y ∧ 0 < z ∧ x + y + z = 1 ∧ x *R a + y *R b
```

$+ z *_R c = v\}$
(**is** *?lhs = ?rhs*)
proof
have *abc*: $a \neq b$ $a \neq c$ $b \neq c$ $\neg$ *affine_dependent* $\{a, b, c\}$
using *assms* **by** (*auto simp*: *collinear_3_eq_affine_dependent*)
with *2* **show** *?lhs* $\subseteq$ *?rhs*
by (*fastforce simp add*: *interior_convex_hull_explicit_minimal*)
show *?rhs* $\subseteq$ *?lhs*
using *abc 2*
apply (*clarsimp simp add*: *interior_convex_hull_explicit_minimal*)
subgoal for *x y z*
by (*rule_tac x*=λr. (*if r*=*a then x else if r*=*b then y else if r*=*c then z else 0*) **in** *exI*) *auto*
done
qed

5.0.9 Basic lemmas about hyperplanes and halfspaces

lemma *halfspace_Int_eq*:
$\{x.\ a \cdot x \leq b\} \cap \{x.\ b \leq a \cdot x\} = \{x.\ a \cdot x = b\}$
$\{x.\ b \leq a \cdot x\} \cap \{x.\ a \cdot x \leq b\} = \{x.\ a \cdot x = b\}$
by *auto*

lemma *hyperplane_eq_Ex*:
assumes $a \neq 0$ **obtains** x **where** $a \cdot x = b$
by (*rule_tac x* = $(b\ /\ (a \cdot a)) *_R a$ **in** *that*) (*simp add*: *assms*)

lemma *hyperplane_eq_empty*:
$\{x.\ a \cdot x = b\} = \{\} \longleftrightarrow a = 0 \wedge b \neq 0$
using *hyperplane_eq_Ex*
by (*metis* (*mono_tags*, *lifting*) *empty_Collect_eq inner_zero_left*)

lemma *hyperplane_eq_UNIV*:
$\{x.\ a \cdot x = b\} = UNIV \longleftrightarrow a = 0 \wedge b = 0$
proof −
have $a = 0 \wedge b = 0$ **if** $UNIV \subseteq \{x.\ a \cdot x = b\}$
using *subsetD* [*OF that*, **where** $c = ((b+1)\ /\ (a \cdot a)) *_R a$]
by (*simp add*: *field_split_simps split*: *if_split_asm*)
then show *?thesis* **by** *force*
qed

lemma *halfspace_eq_empty_lt*:
$\{x.\ a \cdot x < b\} = \{\} \longleftrightarrow a = 0 \wedge b \leq 0$
proof −
have $a = 0 \wedge b \leq 0$ **if** $\{x.\ a \cdot x < b\} \subseteq \{\}$
using *subsetD* [*OF that*, **where** $c = ((b-1)\ /\ (a \cdot a)) *_R a$]
by (*force simp add*: *field_split_simps split*: *if_split_asm*)
then show *?thesis* **by** *force*
qed

lemma *halfspace_eq_empty_gt*:
 $\{x.\ a \cdot x > b\} = \{\} \longleftrightarrow a = 0 \wedge b \geq 0$
 using *halfspace_eq_empty_lt* [*of* $-a$ $-b$]
 by *simp*

lemma *halfspace_eq_empty_le*:
 $\{x.\ a \cdot x \leq b\} = \{\} \longleftrightarrow a = 0 \wedge b < 0$
proof −
 have $a = 0 \wedge b < 0$ **if** $\{x.\ a \cdot x \leq b\} \subseteq \{\}$
  using *subsetD* [*OF that*, **where** $c = ((b-1)\ /\ (a \cdot a)) *_R a$]
  by (*force simp add*: *field_split_simps split*: *if_split_asm*)
 then show *?thesis* **by** *force*
qed

lemma *halfspace_eq_empty_ge*:
 $\{x.\ a \cdot x \geq b\} = \{\} \longleftrightarrow a = 0 \wedge b > 0$
 using *halfspace_eq_empty_le* [*of* $-a$ $-b$] **by** *simp*

5.0.10 Use set distance for an easy proof of separation properties

proposition *separation_closures*:
 fixes S :: *'a*::*euclidean_space set*
 assumes $S \cap$ *closure* $T = \{\}$ $T \cap$ *closure* $S = \{\}$
 obtains U V **where** $U \cap V = \{\}$ *open* U *open* V $S \subseteq U$ $T \subseteq V$
proof (*cases* $S = \{\} \vee T = \{\}$)
 case *True* **with** *that* **show** *?thesis* **by** *auto*
next
 case *False*
 define f **where** $f \equiv \lambda x.$ *setdist* $\{x\}$ T − *setdist* $\{x\}$ S
 have *contf*: *continuous_on UNIV f*
  unfolding *f_def* **by** (*intro continuous_intros continuous_on_setdist*)
 show *?thesis*
 proof (*rule_tac* $U = \{x.\ f\ x > 0\}$ **and** $V = \{x.\ f\ x < 0\}$ **in** *that*)
  show $\{x.\ 0 < f\ x\} \cap \{x.\ f\ x < 0\} = \{\}$
   by *auto*
  show *open* $\{x.\ 0 < f\ x\}$
   by (*simp add*: *open_Collect_less contf*)
  show *open* $\{x.\ f\ x < 0\}$
   by (*simp add*: *open_Collect_less contf*)
  have $\bigwedge x.\ x \in S \Longrightarrow$ *setdist* $\{x\}$ $T \neq 0$ $\bigwedge x.\ x \in T \Longrightarrow$ *setdist* $\{x\}$ $S \neq 0$
   by (*meson False assms disjoint_iff setdist_eq_0_sing_1*)+
  then show $S \subseteq \{x.\ 0 < f\ x\}$ $T \subseteq \{x.\ f\ x < 0\}$
   using *less_eq_real_def* **by** (*fastforce simp add*: *f_def setdist_sing_in_set*)+
 qed
qed

lemma *separation_normal*:

fixes $S :: 'a::euclidean\_space\ set$
assumes *closed S closed T* $S \cap T = \{\}$
obtains *U V* **where** *open U open V* $S \subseteq U$ $T \subseteq V$ $U \cap V = \{\}$
using *separation_closures* [*of S T*]
by (*metis assms closure_closed disjnt_def inf_commute*)

lemma *separation_normal_local*:
fixes $S :: 'a::euclidean\_space\ set$
assumes *US*: *closedin* (*top_of_set U*) *S*
and *UT*: *closedin* (*top_of_set U*) *T*
and $S \cap T = \{\}$
obtains S' T' **where** *openin* (*top_of_set U*) S'
openin (*top_of_set U*) T'
$S \subseteq S'$ $T \subseteq T'$ $S' \cap T' = \{\}$
proof (*cases* $S = \{\} \vee T = \{\}$)
case *True* **with** *that* **show** *?thesis*
using *UT US* **by** (*blast dest*: *closedin_subset*)
next
case *False*
define *f* **where** $f \equiv \lambda x.$ *setdist* $\{x\}$ T − *setdist* $\{x\}$ *S*
have *contf*: *continuous_on U f*
unfolding *f_def* **by** (*intro continuous_intros*)
show *?thesis*
proof (*rule_tac* $S' = (U \cap f -` \{0<..\})$ **and** $T' = (U \cap f -` \{..<0\})$ **in** *that*)
show $(U \cap f -` \{0<..\}) \cap (U \cap f -` \{..<0\}) = \{\}$
by *auto*
show *openin* (*top_of_set U*) $(U \cap f -` \{0<..\})$
by (*rule continuous_openin_preimage* [**where** *T*=*UNIV*]) (*simp_all add*: *contf*)
next
show *openin* (*top_of_set U*) $(U \cap f -` \{..<0\})$
by (*rule continuous_openin_preimage* [**where** *T*=*UNIV*]) (*simp_all add*: *contf*)
next
have $S \subseteq U$ $T \subseteq U$
using *closedin_imp_subset assms* **by** *blast*+
then show $S \subseteq U \cap f -` \{0<..\}$ $T \subseteq U \cap f -` \{..<0\}$
using *assms False* **by** (*force simp add*: *f_def setdist_sing_in_set intro*!: *setdist_gt_0_closedin*)+
qed
qed

lemma *separation_normal_compact*:
fixes $S :: 'a::euclidean\_space\ set$
assumes *compact S closed T* $S \cap T = \{\}$
obtains *U V* **where** *open U compact*(*closure U*) *open V* $S \subseteq U$ $T \subseteq V$ $U \cap V = \{\}$
proof −
have *closed S bounded S*

using *assms* **by** (*auto simp*: *compact_eq_bounded_closed*)
then obtain *r* **where** *r>0* **and** *r*: $S \subseteq$ *ball 0 r*
by (*auto dest*!: *bounded_subset_ballD*)
have $**$: *closed* ($T \cup -$ *ball 0 r*) $S \cap$ ($T \cup -$ *ball 0 r*) = {}
using *assms r* **by** *blast+*
then obtain *U V* **where** *UV*: *open U open V S* $\subseteq$ *U T* $\cup -$ *ball 0 r* $\subseteq$ *V U* $\cap$ *V* = {}
by (*meson* ‹*closed S*› *separation_normal*)
then have *compact*(*closure U*)
by (*meson bounded_ball bounded_subset compact_closure compl_le_swap2 disjoint_eq_subset_Compl le_sup_iff*)
with *UV* **show** *thesis*
using *that* **by** *auto*
qed

5.0.11 Connectedness of the intersection of a chain

proposition *connected_chain*:
fixes $\mathcal{F}$ :: *'a* :: *euclidean_space set set*
assumes *cc*: $\bigwedge S.\ S \in \mathcal{F} \Longrightarrow$ *compact S* $\wedge$ *connected S*
and *linear*: $\bigwedge S\ T.\ S \in \mathcal{F} \wedge T \in \mathcal{F} \Longrightarrow S \subseteq T \vee T \subseteq S$
shows *connected*($\bigcap \mathcal{F}$)
proof (*cases* $\mathcal{F}$ = {})
case *True* **then show** *?thesis*
by *auto*
next
case *False*
then have *cf*: *compact*($\bigcap \mathcal{F}$)
by (*simp add*: *cc compact_Inter*)
have *False* **if** *AB*: *closed A closed B A* $\cap$ *B* = {}
and *ABeq*: *A* $\cup$ *B* = $\bigcap \mathcal{F}$ **and** *A* $\neq$ {} *B* $\neq$ {} **for** *A B*
proof −
obtain *U V* **where** *open U open V A* $\subseteq$ *U B* $\subseteq$ *V U* $\cap$ *V* = {}
using *separation_normal* [*OF AB*] **by** *metis*
obtain *K* **where** *K* $\in \mathcal{F}$ *compact K*
using *cc False* **by** *blast*
then obtain *N* **where** *open N* **and** *K* $\subseteq$ *N*
by *blast*
let *?C* = *insert* (*U* $\cup$ *V*) ((λ*S*. *N* − *S*) ‘ $\mathcal{F}$)
obtain $\mathcal{D}$ **where** $\mathcal{D} \subseteq$ *?C finite* $\mathcal{D}$ *K* $\subseteq \bigcup \mathcal{D}$
proof (*rule compactE* [*OF* ‹*compact K*›])
show *K* $\subseteq \bigcup$(*insert* (*U* $\cup$ *V*) ((−) *N* ‘ $\mathcal{F}$))
using ‹*K* $\subseteq$ *N*› *ABeq* ‹*A* $\subseteq$ *U*› ‹*B* $\subseteq$ *V*› **by** *auto*
show $\bigwedge B.\ B \in$ *insert* (*U* $\cup$ *V*) ((−) *N* ‘ $\mathcal{F}$) $\Longrightarrow$ *open B*
by (*auto simp*: ‹*open U*› ‹*open V*› *open_Un* ‹*open N*› *cc compact_imp_closed open_Diff*)
qed
then have *finite*($\mathcal{D}$ − {*U* $\cup$ *V*})
by *blast*

moreover have $\mathcal{D} - \{U \cup V\} \subseteq (\lambda S.\ N - S)\ `\ \mathcal{F}$
using ‹$\mathcal{D} \subseteq ?\mathcal{C}$› **by** *blast*
ultimately obtain $\mathcal{G}$ **where** $\mathcal{G} \subseteq \mathcal{F}$ *finite* $\mathcal{G}$ **and** *Deq*: $\mathcal{D} - \{U \cup V\} = (\lambda S.\ N - S)\ `\ \mathcal{G}$
using *finite_subset_image* **by** *metis*
obtain J **where** $J \in \mathcal{F}$ **and** J: $(\bigcup S \in \mathcal{G}.\ N - S) \subseteq N - J$
proof (*cases* $\mathcal{G} = \{\}$)
case *True*
with ‹$\mathcal{F} \neq \{\}$› *that* **show** *?thesis*
by *auto*
next
case *False*
have $\bigwedge S\ T.\ [\![S \in \mathcal{G};\ T \in \mathcal{G}]\!] \Longrightarrow S \subseteq T \vee T \subseteq S$
by (*meson* ‹$\mathcal{G} \subseteq \mathcal{F}$› *in_mono local.linear*)
with ‹*finite* $\mathcal{G}$› ‹$\mathcal{G} \neq \{\}$›
have $\exists J \in \mathcal{G}.\ (\bigcup S \in \mathcal{G}.\ N - S) \subseteq N - J$
proof *induction*
case (*insert* X $\mathcal{H}$)
show *?case*
proof (*cases* $\mathcal{H} = \{\}$)
case *True* **then show** *?thesis* **by** *auto*
next
case *False*
then have $\bigwedge S\ T.\ [\![S \in \mathcal{H};\ T \in \mathcal{H}]\!] \Longrightarrow S \subseteq T \vee T \subseteq S$
by (*simp add*: *insert.prems*)
with *insert.IH False* **obtain** J **where** $J \in \mathcal{H}$ **and** J: $(\bigcup Y \in \mathcal{H}.\ N - Y) \subseteq N - J$
by *metis*
have $N - J \subseteq N - X \vee N - X \subseteq N - J$
by (*meson Diff_mono* ‹$J \in \mathcal{H}$› *insert.prems*(*2*) *insert_iff order_refl*)
then show *?thesis*
proof
assume $N - J \subseteq N - X$ **with** J **show** *?thesis*
by *auto*
next
assume $N - X \subseteq N - J$
with J **have** $N - X \cup \bigcup\ ((-)\ N\ `\ \mathcal{H}) \subseteq N - J$
by *auto*
with ‹$J \in \mathcal{H}$› **show** *?thesis*
by *blast*
qed
qed
qed *simp*
with ‹$\mathcal{G} \subseteq \mathcal{F}$› **show** *?thesis* **by** (*blast intro*: *that*)
qed
have $K \subseteq \bigcup(\mathit{insert}\ (U \cup V)\ (\mathcal{D} - \{U \cup V\}))$
using ‹$K \subseteq \bigcup \mathcal{D}$› **by** *auto*
also have $\ldots \subseteq (U \cup V) \cup (N - J)$
by (*metis* (*no_types*, *opaque_lifting*) *Deq Un_subset_iff Un_upper2 J Union_insert*

order_trans sup_ge1)
finally have $J \cap K \subseteq U \cup V$
by *blast*
moreover have *connected*$(J \cap K)$
by (*metis Int_absorb1* ‹$J \in \mathcal{F}$› ‹$K \in \mathcal{F}$› *cc inf.orderE local.linear*)
moreover have $U \cap (J \cap K) \neq \{\}$
using *ABeq* ‹$J \in \mathcal{F}$› ‹$K \in \mathcal{F}$› ‹$A \neq \{\}$› ‹$A \subseteq U$› **by** *blast*
moreover have $V \cap (J \cap K) \neq \{\}$
using *ABeq* ‹$J \in \mathcal{F}$› ‹$K \in \mathcal{F}$› ‹$B \neq \{\}$› ‹$B \subseteq V$› **by** *blast*
ultimately show *False*
using *connectedD* [*of* $J \cap K$ U V] ‹*open* U› ‹*open* V› ‹$U \cap V = \{\}$› **by** *auto*
qed
with *cf* **show** *?thesis*
by (*auto simp*: *connected_closed_set compact_imp_closed*)
qed

lemma *connected_chain_gen*:
fixes $\mathcal{F}$ :: *'a* :: *euclidean_space set set*
assumes X: $X \in \mathcal{F}$ *compact* X
and *cc*: $\bigwedge T.\ T \in \mathcal{F} \Longrightarrow$ *closed* $T \wedge$ *connected* T
and *linear*: $\bigwedge S\ T.\ S \in \mathcal{F} \wedge T \in \mathcal{F} \Longrightarrow S \subseteq T \vee T \subseteq S$
shows *connected*$(\bigcap \mathcal{F})$
proof −
have $\bigcap \mathcal{F} = (\bigcap T \in \mathcal{F}.\ X \cap T)$
using X **by** *blast*
moreover have *connected* $(\bigcap T \in \mathcal{F}.\ X \cap T)$
proof (*rule connected_chain*)
show $\bigwedge T.\ T \in (\cap)\ X$ ‘ $\mathcal{F} \Longrightarrow$ *compact* $T \wedge$ *connected* T
using *cc* X **by** *auto* (*metis inf.absorb2 inf.orderE local.linear*)
show $\bigwedge S\ T.\ S \in (\cap)\ X$ ‘ $\mathcal{F} \wedge T \in (\cap)\ X$ ‘ $\mathcal{F} \Longrightarrow S \subseteq T \vee T \subseteq S$
using *local.linear* **by** *blast*
qed
ultimately show *?thesis*
by *metis*
qed

lemma *connected_nest*:
fixes S :: *'a*::*linorder* ⇒ *'b*::*euclidean_space set*
assumes S: $\bigwedge n.$ *compact*$(S\ n)$ $\bigwedge n.$ *connected*$(S\ n)$
and *nest*: $\bigwedge m\ n.\ m \leq n \Longrightarrow S\ n \subseteq S\ m$
shows *connected*$(\bigcap$ (*range* S))
proof (*rule connected_chain*)
show $\bigwedge A\ T.\ A \in$ *range* $S \wedge T \in$ *range* $S \Longrightarrow A \subseteq T \vee T \subseteq A$
by (*metis image_iff le_cases nest*)
qed (*use* S **in** *blast*)

lemma *connected_nest_gen*:
fixes S :: *'a*::*linorder* ⇒ *'b*::*euclidean_space set*

assumes S: $\bigwedge n.\ closed(S\ n)$ $\bigwedge n.\ connected(S\ n)$ $compact(S\ k)$
and *nest*: $\bigwedge m\ n.\ m \leq n \Longrightarrow S\ n \subseteq S\ m$
shows $connected(\bigcap\ (range\ S))$
proof (*rule connected_chain_gen* [*of S k*])
show $\bigwedge A\ T.\ A \in range\ S \wedge T \in range\ S \Longrightarrow A \subseteq T \vee T \subseteq A$
by (*metis imageE le_cases nest*)
qed (*use S* **in** *auto*)

5.0.12 Proper maps, including projections out of compact sets

lemma *finite_indexed_bound*:
assumes A: *finite* A $\bigwedge x.\ x \in A \Longrightarrow \exists n{::}'a{::}linorder.\ P\ x\ n$
shows $\exists m.\ \forall x \in A.\ \exists k{\leq}m.\ P\ x\ k$
using A
proof (*induction A*)
case *empty* **then show** *?case* **by** *force*
next
case (*insert a A*)
then obtain $m\ n$ **where** $\forall x \in A.\ \exists k{\leq}m.\ P\ x\ k\ P\ a\ n$
by *force*
then show *?case*
by (*metis dual_order.trans insert_iff le_cases*)
qed

proposition *proper_map*:
fixes $f :: 'a{::}heine\_borel \Rightarrow 'b{::}heine\_borel$
assumes *closedin* (*top_of_set S*) K
and *com*: $\bigwedge U.\ [\![U \subseteq T;\ compact\ U]\!] \Longrightarrow compact\ (S \cap f\ -`\ U)$
and $f\ `\ S \subseteq T$
shows *closedin* (*top_of_set T*) $(f\ `\ K)$
proof −
have $K \subseteq S$
using *assms closedin_imp_subset* **by** *metis*
obtain C **where** *closed* C **and** *Keq*: $K = S \cap C$
using *assms* **by** (*auto simp*: *closedin_closed*)
have $*$: $y \in f\ `\ K$ **if** $y \in T$ **and** $y$: $y$ *islimpt* $f\ `\ K$ **for** y
proof −
obtain h **where** $\forall n.\ (\exists x{\in}K.\ h\ n = f\ x) \wedge h\ n \neq y$ *inj* h **and** *hlim*: $(h \longrightarrow y)$ *sequentially*
using ‹$y \in T$› y **by** (*force simp*: *limpt_sequential_inj*)
then obtain X **where** X: $\bigwedge n.\ X\ n \in K \wedge h\ n = f\ (X\ n) \wedge h\ n \neq y$
by *metis*
then have *fX*: $\bigwedge n.\ f\ (X\ n) = h\ n$
by *metis*
define Ψ **where** $\Psi \equiv \lambda n.\ \{a \in K.\ f\ a \in insert\ y\ (range\ (\lambda i.\ f\ (X\ (n + i))))\}$
have *compact* $(C \cap (S \cap f\ -`\ insert\ y\ (range\ (\lambda i.\ f(X(n + i))))))$ **for** n
proof (*intro closed_Int_compact* [*OF* ‹*closed C*› *com*] *compact_sequence_with_limit*)
show *insert* y $(range\ (\lambda i.\ f\ (X\ (n + i)))) \subseteq T$

```
      using X ‹K ⊆ S› ‹f ‘ S ⊆ T› ‹y ∈ T› by blast
    show (λi. f (X (n + i))) ⟶ y
     by (simp add: fX add.commute [of n] LIMSEQ_ignore_initial_segment [OF
hlim])
  qed
  then have comf: compact (Ψ n) for n
    by (simp add: Keq Int_def Ψ_def conj_commute)
  have ne: ⋂ℱ ≠ {}
        if finite ℱ
          and ℱ: ⋀t. t ∈ ℱ ⟹ (∃ n. t = Ψ n)
        for ℱ
  proof −
    obtain m where m: ⋀t. t ∈ ℱ ⟹ ∃ k≤m. t = Ψ k
      by (rule exE [OF finite_indexed_bound [OF ‹finite ℱ› ℱ]], force+)
    have X m ∈ ⋂ℱ
      using X le_Suc_ex by (fastforce simp: Ψ_def dest: m)
    then show ?thesis by blast
  qed
  have (⋂ n. Ψ n) ≠ {}
  proof (rule compact_fip_Heine_Borel)
    show ⋀ℱ'. ⟦finite ℱ'; ℱ' ⊆ range Ψ⟧ ⟹ ⋂ ℱ' ≠ {}
      by (meson ne rangeE subset_eq)
  qed (use comf in blast)
  then obtain x where x ∈ K ⋀n. (f x = y ∨ (∃ u. f x = h (n + u)))
    by (force simp add: Ψ_def fX)
  then show ?thesis
      unfolding image_iff by (metis ‹inj h› le_add1 not_less_eq_eq rangeI
range_ex1_eq)
 qed
 with assms closedin_subset show ?thesis
   by (force simp: closedin_limpt)
qed


lemma compact_continuous_image_eq:
 fixes f :: 'a::heine_borel ⇒ 'b::heine_borel
 assumes f: inj_on f S
 shows continuous_on S f ⟷ (∀ T. compact T ∧ T ⊆ S ⟶ compact(f ‘ T))
        (is ?lhs = ?rhs)
proof
 assume ?lhs then show ?rhs
   by (metis continuous_on_subset compact_continuous_image)
next
 assume RHS: ?rhs
 obtain g where gf: ⋀x. x ∈ S ⟹ g (f x) = x
   by (metis inv_into_f_f f)
 then have ∗: (S ∩ f −‘ U) = g ‘ U if U ⊆ f ‘ S for U
   using that by fastforce
 have gfim: g ‘ f ‘ S ⊆ S using gf by auto
```

```
  have **: compact (f ‘ S ∩ g −‘ C) if C: C ⊆ S compact C for C
  proof −
    obtain h where h C ∈ C ∧ h C ∉ S ∨ compact (f ‘ C)
      by (force simp: C RHS)
    moreover have f ‘ C = (f ‘ S ∩ g −‘ C)
      using C gf by auto
    ultimately show ?thesis
      using C by auto
  qed
  show ?lhs
    using proper_map [OF _ _ gfim] **
    by (simp add: continuous_on_closed * closedin_imp_subset)
qed
```

5.0.13 Trivial fact: convexity equals connectedness for collinear sets

```
lemma convex_connected_collinear:
  fixes S :: 'a::euclidean_space set
  assumes collinear S
    shows convex S ⟷ connected S
proof
  assume convex S
  then show connected S
    using convex_connected by blast
next
  assume S: connected S
  show convex S
  proof (cases S = {})
    case True
    then show ?thesis by simp
  next
    case False
    then obtain a where a ∈ S by auto
    have collinear (affine hull S)
      by (simp add: assms collinear_affine_hull_collinear)
    then obtain z where z ≠ 0 ⋀x. x ∈ affine hull S ⟹ ∃ c. x − a = c *R z
      by (meson ‹a ∈ S› collinear hull_inc)
    then obtain f where f: ⋀x. x ∈ affine hull S ⟹ x − a = f x *R z
      by metis
    then have inj_f: inj_on f (affine hull S)
      by (metis diff_add_cancel inj_onI)
    have diff: x − y = (f x − f y) *R z if x: x ∈ affine hull S and y: y ∈ affine
hull S for x y
    proof −
      have f x *R z = x − a
        by (simp add: f hull_inc x)
      moreover have f y *R z = y − a
        by (simp add: f hull_inc y)
```

```
    ultimately show ?thesis
      by (simp add: scaleR_left.diff)
  qed
  have cont_f: continuous_on (affine hull S) f
  proof (clarsimp simp: dist_norm continuous_on_iff diff)
    show ⋀x e. 0 < e ⟹ ∃d>0. ∀y∈affine hull S. |f y − f x| ∗ norm z < d
⟶ |f y − f x| < e
      by (metis ‹z ≠ 0› mult_pos_pos mult_less_iff1 zero_less_norm_iff)
  qed
  then have conn_fS: connected (f ‘ S)
   by (meson S connected_continuous_image continuous_on_subset hull_subset)
  show ?thesis
  proof (clarsimp simp: convex_contains_segment)
    fix x y z
    assume x ∈ S y ∈ S z ∈ closed_segment x y
    have False if z ∉ S
    proof −
      have f ‘ (closed_segment x y) = closed_segment (f x) (f y)
      proof (rule continuous_injective_image_segment_1)
        show continuous_on (closed_segment x y) f
         by (meson ‹x ∈ S› ‹y ∈ S› convex_affine_hull convex_contains_segment
hull_inc continuous_on_subset [OF cont_f])
        show inj_on f (closed_segment x y)
         by (meson ‹x ∈ S› ‹y ∈ S› convex_affine_hull convex_contains_segment
hull_inc inj_on_subset [OF inj_f])
      qed
      then have fz: f z ∈ closed_segment (f x) (f y)
        using ‹z ∈ closed_segment x y› by blast
      have z ∈ affine hull S
        by (meson ‹x ∈ S› ‹y ∈ S› ‹z ∈ closed_segment x y› convex_affine_hull
convex_contains_segment hull_inc subset_eq)
      then have fz_notin: f z ∉ f ‘ S
        using hull_subset inj_f inj_onD that by fastforce
      moreover have {..<f z} ∩ f ‘ S ≠ {} {f z<..} ∩ f ‘ S ≠ {}
      proof −
        consider f x ≤ f z ∧ f z ≤ f y | f y ≤ f z ∧ f z ≤ f x
          using fz
          by (auto simp add: closed_segment_eq_real_ivl split: if_split_asm)
        then have {..<f z} ∩ f ‘ {x,y} ≠ {} ∧ {f z<..} ∩ f ‘ {x,y} ≠ {}
          by cases (use fz_notin ‹x ∈ S› ‹y ∈ S› in ‹auto simp: image_iff›)
        then show {..<f z} ∩ f ‘ S ≠ {} {f z<..} ∩ f ‘ S ≠ {}
          using ‹x ∈ S› ‹y ∈ S› by blast+
      qed
      ultimately show False
        using connectedD [OF conn_fS, of {..<f z} {f z<..}] by force
    qed
    then show z ∈ S by meson
  qed
qed
```

qed

lemma *compact_convex_collinear_segment_alt*:
fixes S :: *'a*::*euclidean_space set*
assumes $S \neq \{\}$ *compact S connected S collinear S*
obtains *a b* **where** S = *closed_segment a b*
proof −
obtain ξ **where** $\xi \in S$ **using** ‹$S \neq \{\}$› **by** *auto*
have *collinear* (*affine hull S*)
by (*simp add*: *assms collinear_affine_hull_collinear*)
then obtain *z* **where** $z \neq 0$ $\bigwedge x.\ x \in$ *affine hull S* $\Longrightarrow \exists c.\ x - \xi = c *_R z$
by (*meson* ‹$\xi \in S$› *collinear hull_inc*)
then obtain *f* **where** *f*: $\bigwedge x.\ x \in$ *affine hull S* $\Longrightarrow x - \xi = f\ x *_R z$
by *metis*
let *?g* = $\lambda r.\ r *_R z + \xi$
have *gf*: *?g* (*f x*) = *x* **if** *x* ∈ *affine hull S* **for** *x*
by (*metis diff_add_cancel f that*)
then have *inj_f*: *inj_on f* (*affine hull S*)
by (*metis inj_onI*)
have *diff*: $x - y = (f\ x - f\ y) *_R z$ **if** *x*: *x* ∈ *affine hull S* **and** *y*: *y* ∈ *affine hull S* **for** *x y*
proof −
have $f\ x *_R z = x - \xi$
by (*simp add*: *f hull_inc x*)
moreover have $f\ y *_R z = y - \xi$
by (*simp add*: *f hull_inc y*)
ultimately show *?thesis*
by (*simp add*: *scaleR_left.diff*)
qed
have *cont_f*: *continuous_on* (*affine hull S*) *f*
proof (*clarsimp simp*: *dist_norm continuous_on_iff diff*)
show $\bigwedge x\ e.\ 0 < e \Longrightarrow \exists d{>}0.\ \forall y \in$ *affine hull S*. $|f\ y - f\ x| * norm\ z < d \longrightarrow |f\ y - f\ x| < e$
by (*metis* ‹$z \neq 0$› *mult_pos_pos mult_less_iff1 zero_less_norm_iff*)
qed
then have *connected* (*f* ‘ *S*)
by (*meson* ‹*connected S*› *connected_continuous_image continuous_on_subset hull_subset*)
moreover have *compact* (*f* ‘ *S*)
by (*meson* ‹*compact S*› *compact_continuous_image_eq cont_f hull_subset inj_f*)
ultimately obtain *x y* **where** *f* ‘ *S* = {*x*..*y*}
by (*meson connected_compact_interval_1*)
then have *fS_eq*: *f* ‘ *S* = *closed_segment x y*
using ‹$S \neq \{\}$› *closed_segment_eq_real_ivl* **by** *auto*
obtain *a b* **where** *a* ∈ *S f a* = *x b* ∈ *S f b* = *y*
by (*metis* (*full_types*) *ends_in_segment fS_eq imageE*)
have *f* ‘ (*closed_segment a b*) = *closed_segment* (*f a*) (*f b*)
proof (*rule continuous_injective_image_segment_1*)

```
    show continuous_on (closed_segment a b) f
      by (meson ‹a ∈ S› ‹b ∈ S› convex_affine_hull convex_contains_segment
hull_inc continuous_on_subset [OF cont_f])
    show inj_on f (closed_segment a b)
      by (meson ‹a ∈ S› ‹b ∈ S› convex_affine_hull convex_contains_segment
hull_inc inj_on_subset [OF inj_f])
  qed
  then have f ‘ (closed_segment a b) = f ‘ S
    by (simp add: ‹f a = x› ‹f b = y› fS_eq)
  then have ?g ‘ f ‘ (closed_segment a b) = ?g ‘ f ‘ S
    by simp
  moreover have (λx. f x *R z + ξ) ‘ closed_segment a b = closed_segment a b
    unfolding image_def using ‹a ∈ S› ‹b ∈ S›
   by (safe; metis (mono_tags, lifting) convex_affine_hull convex_contains_segment
gf hull_subset subsetCE)
  ultimately have closed_segment a b = S
    using gf by (simp add: image_comp o_def hull_inc cong: image_cong)
  then show ?thesis
    using that by blast
qed

lemma compact_convex_collinear_segment:
  fixes S :: 'a::euclidean_space set
  assumes S ≠ {} compact S convex S collinear S
  obtains a b where S = closed_segment a b
 using assms convex_connected_collinear compact_convex_collinear_segment_alt
by blast


lemma proper_map_from_compact:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes contf: continuous_on S f and imf: f ∈ S → T and compact S
        closedin (top_of_set T) K
  shows compact (S ∩ f −‘ K)
by (rule closedin_compact [OF ‹compact S›] continuous_closedin_preimage_gen
assms)+

lemma proper_map_fst:
  assumes compact T K ⊆ S compact K
    shows compact (S × T ∩ fst −‘ K)
proof −
  have (S × T ∩ fst −‘ K) = K × T
    using assms by auto
  then show ?thesis by (simp add: assms compact_Times)
qed

lemma closed_map_fst:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes compact T closedin (top_of_set (S × T)) c
```

```
  shows closedin (top_of_set S) (fst ‘ c)
proof −
  have ∗: fst ‘ (S × T) ⊆ S
    by auto
  show ?thesis
    using proper_map [OF _ _ ∗] by (simp add: proper_map_fst assms)
qed

lemma proper_map_snd:
  assumes compact S K ⊆ T compact K
    shows compact (S × T ∩ snd −‘ K)
proof −
  have (S × T ∩ snd −‘ K) = S × K
    using assms by auto
  then show ?thesis by (simp add: assms compact_Times)
qed

lemma closed_map_snd:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes compact S closedin (top_of_set (S × T)) c
   shows closedin (top_of_set T) (snd ‘ c)
proof −
  have ∗: snd ‘ (S × T) ⊆ T
    by auto
  show ?thesis
    using proper_map [OF _ _ ∗] by (simp add: proper_map_snd assms)
qed

lemma closedin_compact_projection:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes compact S and clo: closedin (top_of_set (S × T)) U
    shows closedin (top_of_set T) {y. ∃x. x ∈ S ∧ (x, y) ∈ U}
proof −
  have U ⊆ S × T
    by (metis clo closedin_imp_subset)
  then have {y. ∃x. x ∈ S ∧ (x, y) ∈ U} = snd ‘ U
    by force
  moreover have closedin (top_of_set T) (snd ‘ U)
    by (rule closed_map_snd [OF assms])
  ultimately show ?thesis
    by simp
qed


lemma closed_compact_projection:
  fixes S :: 'a::euclidean_space set
    and T :: ('a ∗ 'b::euclidean_space) set
  assumes compact S and clo: closed T
    shows closed {y. ∃x. x ∈ S ∧ (x, y) ∈ T}
```

proof −
have ∗: {*y*. ∃ *x*. *x* ∈ *S* ∧ *Pair x y* ∈ *T*} = {*y*. ∃ *x*. *x* ∈ *S* ∧ *Pair x y* ∈ ((*S* × *UNIV*) ∩ *T*)}
by *auto*
show *?thesis*
unfolding ∗
by (*intro clo closedin_closed_Int closedin_closed_trans* [*OF* _ *closed_UNIV*] *closedin_compact_projection* [*OF* ‹*compact S*›])
qed

Representing affine hull as a finite intersection of hyperplanes

proposition *affine_hull_convex_Int_nonempty_interior*:
fixes *S* :: *'a*::*real_normed_vector set*
assumes *convex S S* ∩ *interior T* ≠ {}
shows *affine hull* (*S* ∩ *T*) = *affine hull S*
proof
show *affine hull* (*S* ∩ *T*) ⊆ *affine hull S*
by (*simp add*: *hull_mono*)
next
obtain *a* **where** *a* ∈ *S a* ∈ *T* **and** *at*: *a* ∈ *interior T*
using *assms interior_subset* **by** *blast*
then obtain *e* **where** *e* > *0* **and** *e*: *cball a e* ⊆ *T*
using *mem_interior_cball* **by** *blast*
have ∗: *x* ∈ (+) *a* ‘ *span* ((λ*x*. *x* − *a*) ‘ (*S* ∩ *T*)) **if** *x* ∈ *S* **for** *x*
proof (*cases x* = *a*)
case *True* **with** *that span_0 eq_add_iff image_def mem_Collect_eq* **show** *?thesis*
by *blast*
next
case *False*
define *k* **where** *k* = *min* (*1*/*2*) (*e* / *norm* (*x*−*a*))
have *k*: *0* < *k k* < *1*
using ‹*e* > *0*› *False* **by** (*auto simp*: *k_def*)
then have *xa*: (*x*−*a*) = *inverse k* ∗*R k* ∗*R* (*x*−*a*)
by *simp*
have *e* / *norm* (*x* − *a*) ≥ *k*
using *k_def* **by** *linarith*
then have *a* + *k* ∗*R* (*x* − *a*) ∈ *cball a e*
using ‹*0* < *k*› *False*
by (*simp add*: *dist_norm*) (*simp add*: *field_simps*)
then have *T*: *a* + *k* ∗*R* (*x* − *a*) ∈ *T*
using *e* **by** *blast*
have *S*: *a* + *k* ∗*R* (*x* − *a*) ∈ *S*
using *k* ‹*a* ∈ *S*› *convexD* [*OF* ‹*convex S*› ‹*a* ∈ *S*› ‹*x* ∈ *S*›, *of* *1*−*k k*]
by (*simp add*: *algebra_simps*)
have *inverse k* ∗*R k* ∗*R* (*x*−*a*) ∈ *span* ((λ*x*. *x* − *a*) ‘ (*S* ∩ *T*))
by (*intro span_mul* [*OF span_base*] *image_eqI* [**where** *x* = *a* + *k* ∗*R* (*x* − *a*)]) (*auto simp*: *S T*)

```
    with xa image_iff show ?thesis  by fastforce
  qed
  have S ⊆ affine hull (S ∩ T)
    by (force simp: ∗ ‹a ∈ S› ‹a ∈ T› hull_inc affine_hull_span_gen [of a])
  then show affine hull S ⊆ affine hull (S ∩ T)
    by (simp add: subset_hull)
qed

corollary affine_hull_convex_Int_open:
  fixes S :: 'a::real_normed_vector set
  assumes convex S open T S ∩ T ≠ {}
  shows affine hull (S ∩ T) = affine hull S
  using affine_hull_convex_Int_nonempty_interior assms interior_eq by blast

corollary affine_hull_affine_Int_nonempty_interior:
  fixes S :: 'a::real_normed_vector set
  assumes affine S S ∩ interior T ≠ {}
  shows affine hull (S ∩ T) = affine hull S
  by (simp add: affine_hull_convex_Int_nonempty_interior affine_imp_convex
assms)

corollary affine_hull_affine_Int_open:
  fixes S :: 'a::real_normed_vector set
  assumes affine S open T S ∩ T ≠ {}
  shows affine hull (S ∩ T) = affine hull S
  by (simp add: affine_hull_convex_Int_open affine_imp_convex assms)

corollary affine_hull_convex_Int_openin:
  fixes S :: 'a::real_normed_vector set
  assumes convex S openin (top_of_set (affine hull S)) T S ∩ T ≠ {}
  shows affine hull (S ∩ T) = affine hull S
  using assms unfolding openin_open
  by (metis affine_hull_convex_Int_open hull_subset inf.orderE inf_assoc)

corollary affine_hull_openin:
  fixes S :: 'a::real_normed_vector set
  assumes openin (top_of_set (affine hull T)) S S ≠ {}
  shows affine hull S = affine hull T
  using assms unfolding openin_open
  by (metis affine_affine_hull affine_hull_affine_Int_open hull_hull)

corollary affine_hull_open:
  fixes S :: 'a::real_normed_vector set
  assumes open S S ≠ {}
  shows affine hull S = UNIV
  by (metis affine_hull_convex_Int_nonempty_interior assms convex_UNIV hull_UNIV
inf_top.left_neutral interior_open)

lemma aff_dim_convex_Int_nonempty_interior:
```

fixes S :: *'a*::*euclidean_space set*
shows $[\![convex\ S;\ S \cap interior\ T \neq \{\}]\!] \Longrightarrow aff\_dim(S \cap T) = aff\_dim\ S$
using *aff_dim_affine_hull2 affine_hull_convex_Int_nonempty_interior* **by** *blast*

lemma *aff_dim_convex_Int_open*:
fixes S :: *'a*::*euclidean_space set*
shows $[\![convex\ S;\ open\ T;\ S \cap T \neq \{\}]\!] \Longrightarrow aff\_dim(S \cap T) = aff\_dim\ S$
using *aff_dim_convex_Int_nonempty_interior interior_eq* **by** *blast*

lemma *affine_hull_Diff*:
fixes S:: *'a*::*real_normed_vector set*
assumes *ope*: *openin* (*top_of_set* (*affine hull S*)) S **and** *finite F* $F \subset S$
shows *affine hull* $(S - F) =$ *affine hull S*
proof −
have *clo*: *closedin* (*top_of_set S*) *F*
using *assms finite_imp_closedin* **by** *auto*
moreover have $S - F \neq \{\}$
using *assms* **by** *auto*
ultimately show *?thesis*
by (*metis ope closedin_def topspace_euclidean_subtopology affine_hull_openin openin_trans*)
qed

lemma *affine_hull_halfspace_lt*:
fixes a :: *'a*::*euclidean_space*
shows *affine hull* $\{x.\ a \cdot x < r\} =$ (*if* $a = 0 \wedge r \leq 0$ *then* $\{\}$ *else UNIV*)
using *halfspace_eq_empty_lt* [*of a r*]
by (*simp add*: *open_halfspace_lt affine_hull_open*)

lemma *affine_hull_halfspace_le*:
fixes a :: *'a*::*euclidean_space*
shows *affine hull* $\{x.\ a \cdot x \leq r\} =$ (*if* $a = 0 \wedge r < 0$ *then* $\{\}$ *else UNIV*)
proof (*cases* $a = 0$)
case *True* **then show** *?thesis* **by** *simp*
next
case *False*
then have *affine hull closure* $\{x.\ a \cdot x < r\} = UNIV$
using *affine_hull_halfspace_lt closure_same_affine_hull* **by** *fastforce*
moreover have $\{x.\ a \cdot x < r\} \subseteq \{x.\ a \cdot x \leq r\}$
by (*simp add*: *Collect_mono*)
ultimately show *?thesis* **using** *False antisym_conv hull_mono top_greatest*
by (*metis affine_hull_halfspace_lt*)
qed

lemma *affine_hull_halfspace_gt*:
fixes a :: *'a*::*euclidean_space*
shows *affine hull* $\{x.\ a \cdot x > r\} =$ (*if* $a = 0 \wedge r \geq 0$ *then* $\{\}$ *else UNIV*)
using *halfspace_eq_empty_gt* [*of r a*]
by (*simp add*: *open_halfspace_gt affine_hull_open*)

lemma *affine_hull_halfspace_ge*:
fixes *a* :: *'a*::*euclidean_space*
shows *affine hull* {*x*. *a* · *x* ≥ *r*} = (*if a* = *0* ∧ *r* > *0 then* {} *else UNIV*)
using *affine_hull_halfspace_le* [*of* −*a* −*r*] **by** *simp*

lemma *aff_dim_halfspace_lt*:
fixes *a* :: *'a*::*euclidean_space*
shows *aff_dim* {*x*. *a* · *x* < *r*} =
(*if a* = *0* ∧ *r* ≤ *0 then* −*1 else DIM*(*'a*))
by *simp* (*metis aff_dim_open halfspace_eq_empty_lt open_halfspace_lt*)

lemma *aff_dim_halfspace_le*:
fixes *a* :: *'a*::*euclidean_space*
shows *aff_dim* {*x*. *a* · *x* ≤ *r*} =
(*if a* = *0* ∧ *r* < *0 then* −*1 else DIM*(*'a*))
proof −
have *int* (*DIM*(*'a*)) = *aff_dim* (*UNIV*::*'a set*)
by (*simp*)
then have *aff_dim* (*affine hull* {*x*. *a* · *x* ≤ *r*}) = *DIM*(*'a*) **if** (*a* = *0* ⟶ *r* ≥ *0*)
using *that* **by** (*simp add*: *affine_hull_halfspace_le not_less*)
then show *?thesis*
by (*force*)
qed

lemma *aff_dim_halfspace_gt*:
fixes *a* :: *'a*::*euclidean_space*
shows *aff_dim* {*x*. *a* · *x* > *r*} =
(*if a* = *0* ∧ *r* ≥ *0 then* −*1 else DIM*(*'a*))
by *simp* (*metis aff_dim_open halfspace_eq_empty_gt open_halfspace_gt*)

lemma *aff_dim_halfspace_ge*:
fixes *a* :: *'a*::*euclidean_space*
shows *aff_dim* {*x*. *a* · *x* ≥ *r*} =
(*if a* = *0* ∧ *r* > *0 then* −*1 else DIM*(*'a*))
using *aff_dim_halfspace_le* [*of* −*a* −*r*] **by** *simp*

proposition *aff_dim_eq_hyperplane*:
fixes *S* :: *'a*::*euclidean_space set*
shows *aff_dim S* = *DIM*(*'a*) − *1* ⟷ (∃ *a b*. *a* ≠ *0* ∧ *affine hull S* = {*x*. *a* · *x* = *b*})
(**is** *?lhs* = *?rhs*)
proof (*cases S* = {})
case *True* **then show** *?thesis*
by (*auto simp*: *dest*: *hyperplane_eq_Ex*)
next
case *False*
then obtain *c* **where** *c* ∈ *S* **by** *blast*

show *?thesis*
proof (*cases* $c = 0$)
case *True*
have *?lhs* $\longleftrightarrow$ $(\exists\, a.\ a \neq 0 \wedge$ *span* $((\lambda x.\ x - c)\ `\ S) = \{x.\ a \cdot x = 0\})$
by (*simp add*: *aff_dim_eq_dim* [*of c*] ‹$c \in S$› *hull_inc dim_eq_hyperplane del*: *One_nat_def*)
also have ... $\longleftrightarrow$ *?rhs*
using *span_zero* [*of S*] *True* ‹$c \in S$› *affine_hull_span_0 hull_inc*
by (*fastforce simp add*: *affine_hull_span_gen* [*of c*] ‹$c = 0$›)
finally show *?thesis* **.**
next
case *False*
have *xc_im*: $x \in (+)\ c\ `\ \{y.\ a \cdot y = 0\}$ **if** $a \cdot x = a \cdot c$ **for** $a\ x$
proof −
have $\exists\, y.\ a \cdot y = 0 \wedge c + y = x$
by (*metis that add.commute diff_add_cancel inner_commute inner_diff_left right_minus_eq*)
then show $x \in (+)\ c\ `\ \{y.\ a \cdot y = 0\}$
by *blast*
qed
have *2*: *span* $((\lambda x.\ x - c)\ `\ S) = \{x.\ a \cdot x = 0\}$
if $(+)\ c\ `$ *span* $((\lambda x.\ x - c)\ `\ S) = \{x.\ a \cdot x = b\}$ **for** $a\ b$
proof −
have $b = a \cdot c$
using *span_0 that* **by** *fastforce*
with *that* **have** $(+)\ c\ `$ *span* $((\lambda x.\ x - c)\ `\ S) = \{x.\ a \cdot x = a \cdot c\}$
by *simp*
then have *span* $((\lambda x.\ x - c)\ `\ S) = (\lambda x.\ x - c)\ `\ \{x.\ a \cdot x = a \cdot c\}$
by (*metis* (*no_types*) *image_cong translation_galois uminus_add_conv_diff*)
also have ... $= \{x.\ a \cdot x = 0\}$
by (*force simp*: *inner_distrib inner_diff_right*
intro: *image_eqI* [**where** $x{=}x{+}c$ **for** x])
finally show *?thesis* **.**
qed
have *?lhs* $= (\exists\, a.\ a \neq 0 \wedge$ *span* $((\lambda x.\ x - c)\ `\ S) = \{x.\ a \cdot x = 0\})$
by (*simp add*: *aff_dim_eq_dim* [*of c*] ‹$c \in S$› *hull_inc dim_eq_hyperplane del*: *One_nat_def*)
also have ... $=$ *?rhs*
by (*fastforce simp add*: *affine_hull_span_gen* [*of c*] ‹$c \in S$› *hull_inc inner_distrib intro*: *xc_im intro*!: *2*)
finally show *?thesis* **.**
qed
qed

corollary *aff_dim_hyperplane* [*simp*]:
fixes $a :: 'a{::}euclidean\_space$
shows $a \neq 0 \Longrightarrow$ *aff_dim* $\{x.\ a \cdot x = r\} = DIM('a) - 1$
by (*metis aff_dim_eq_hyperplane affine_hull_eq affine_hyperplane*)

5.0.14 Some stepping theorems

lemma *aff_dim_insert*:
fixes *a* :: *'a*::*euclidean_space*
shows *aff_dim* (*insert a S*) = (*if a* ∈ *affine hull S then aff_dim S else aff_dim S* + *1*)
proof (*cases S* = {})
case *True* **then show** *?thesis*
by *simp*
next
case *False*
then obtain *x s'* **where** *S*: *S* = *insert x s' x* ∉ *s'*
by (*meson Set.set_insert all_not_in_conv*)
show *?thesis* **using** *S*
by (*force simp add*: *affine_hull_insert_span_gen span_zero insert_commute* [*of a*] *aff_dim_eq_dim* [*of x*] *dim_insert*)
qed

lemma *affine_dependent_choose*:
fixes *a* :: *'a* :: *euclidean_space*
assumes ¬(*affine_dependent S*)
shows *affine_dependent*(*insert a S*) ⟷ *a* ∉ *S* ∧ *a* ∈ *affine hull S*
(**is** *?lhs* = *?rhs*)
proof *safe*
assume *affine_dependent* (*insert a S*) **and** *a* ∈ *S*
then show *False*
using ‹*a* ∈ *S*› *assms insert_absorb* **by** *fastforce*
next
assume *lhs*: *affine_dependent* (*insert a S*)
then have *a* ∉ *S*
by (*metis* (*no_types*) *assms insert_absorb*)
moreover have *finite S*
using *affine_independent_iff_card assms* **by** *blast*
moreover have *aff_dim* (*insert a S*) ≠ *int* (*card S*)
using ‹*finite S*› *affine_independent_iff_card* ‹*a* ∉ *S*› *lhs* **by** *fastforce*
ultimately show *a* ∈ *affine hull S*
by (*metis aff_dim_affine_independent aff_dim_insert assms*)
next
assume *a* ∉ *S* **and** *a* ∈ *affine hull S*
show *affine_dependent* (*insert a S*)
by (*simp add*: ‹*a* ∈ *affine hull S*› ‹*a* ∉ *S*› *affine_dependent_def*)
qed

lemma *affine_independent_insert*:
fixes *a* :: *'a* :: *euclidean_space*
shows ⟦¬ *affine_dependent S*; *a* ∉ *affine hull S*⟧ ⟹ ¬ *affine_dependent*(*insert a S*)
by (*simp add*: *affine_dependent_choose*)

lemma *subspace_bounded_eq_trivial*:

fixes S :: *'a*::*real_normed_vector set*
assumes *subspace S*
shows *bounded S* $\longleftrightarrow$ $S = \{0\}$
proof −
have *False* **if** *bounded S* $x \in S$ $x \neq 0$ **for** x
proof −
obtain B **where** B: $\bigwedge y.\ y \in S \Longrightarrow$ *norm* $y < B$ $B > 0$
using ‹*bounded S*› **by** (*force simp*: *bounded_pos_less*)
have $(B\ /$ *norm* $x) *_R x \in S$
using *assms subspace_mul* ‹$x \in S$› **by** *auto*
moreover have *norm* $((B\ /$ *norm* $x) *_R x) = B$
using *that* B **by** (*simp add*: *algebra_simps*)
ultimately show *False* **using** B **by** *force*
qed
then have *bounded S* $\Longrightarrow$ $S = \{0\}$
using *assms subspace_0* **by** *fastforce*
then show *?thesis*
by *blast*
qed

lemma *affine_bounded_eq_trivial*:
fixes S :: *'a*::*real_normed_vector set*
assumes *affine S*
shows *bounded S* $\longleftrightarrow$ $S = \{\} \vee (\exists a.\ S = \{a\})$
proof (*cases* $S = \{\}$)
case *True* **then show** *?thesis*
by *simp*
next
case *False*
then obtain b **where** $b \in S$ **by** *blast*
with *False assms*
have *bounded S* $\Longrightarrow$ $S = \{b\}$
using *affine_diffs_subspace* [*OF assms* ‹$b \in S$›]
by (*metis* (*no_types*, *lifting*) *ab_group_add_class.ab_left_minus bounded_translation image_empty image_insert subspace_bounded_eq_trivial translation_invert*)
then show *?thesis* **by** *auto*
qed

lemma *affine_bounded_eq_lowdim*:
fixes S :: *'a*::*euclidean_space set*
assumes *affine S*
shows *bounded S* $\longleftrightarrow$ *aff_dim* $S \leq 0$
proof
show *aff_dim* $S \leq 0 \Longrightarrow$ *bounded S*
by (*metis aff_dim_sing aff_dim_subset affine_dim_equal affine_sing all_not_in_conv assms bounded_empty bounded_insert dual_order.antisym empty_subsetI insert_subset*)
qed (*use affine_bounded_eq_trivial assms* **in** *fastforce*)

lemma *bounded_hyperplane_eq_trivial_0*:
fixes $a :: 'a::euclidean\_space$
assumes $a \neq 0$
shows $bounded\ \{x.\ a \cdot x = 0\} \longleftrightarrow DIM('a) = 1$
proof
assume $bounded\ \{x.\ a \cdot x = 0\}$
then have $aff\_dim\ \{x.\ a \cdot x = 0\} \leq 0$
by (*simp add*: *affine_bounded_eq_lowdim affine_hyperplane*)
with *assms* **show** $DIM('a) = 1$
by (*simp add*: *le_Suc_eq*)
next
assume $DIM('a) = 1$
then show $bounded\ \{x.\ a \cdot x = 0\}$
by (*simp add*: *affine_bounded_eq_lowdim affine_hyperplane assms*)
qed

lemma *bounded_hyperplane_eq_trivial*:
fixes $a :: 'a::euclidean\_space$
shows $bounded\ \{x.\ a \cdot x = r\} \longleftrightarrow (if\ a = 0\ then\ r \neq 0\ else\ DIM('a) = 1)$
proof (*simp add*: *bounded_hyperplane_eq_trivial_0*, *clarify*)
assume $r \neq 0\ a \neq 0$
have $aff\_dim\ \{x.\ y \cdot x = 0\} = aff\_dim\ \{x.\ a \cdot x = r\}$ **if** $y \neq 0$ **for** $y::'a$
by (*metis that* ‹$a \neq 0$› *aff_dim_hyperplane*)
then show $bounded\ \{x.\ a \cdot x = r\} = (DIM('a) = Suc\ 0)$
by (*metis One_nat_def* ‹$a \neq 0$› *affine_bounded_eq_lowdim affine_hyperplane bounded_hyperplane_eq_trivial_0*)
qed

5.0.15 General case without assuming closure and getting non-strict separation

proposition *separating_hyperplane_closed_point_inset*:
fixes $S :: 'a::euclidean\_space\ set$
assumes *convex S closed S* $S \neq \{\}$ $z \notin S$
obtains *a b* **where** $a \in S$ $(a - z) \cdot z < b$ $\bigwedge x.\ x \in S \Longrightarrow b < (a - z) \cdot x$
proof −
obtain *y* **where** $y \in S$ **and** y: $\bigwedge u.\ u \in S \Longrightarrow dist\ z\ y \leq dist\ z\ u$
using *distance_attains_inf* [*of S z*] *assms* **by** *auto*
then have $*$: $(y - z) \cdot z < (y - z) \cdot z + (norm\ (y - z))^2\ /\ 2$
using ‹$y \in S$› ‹$z \notin S$› **by** *auto*
show *?thesis*
proof (*rule that* [*OF* ‹$y \in S$› $*$])
fix *x*
assume $x \in S$
have *yz*: $0 < (y - z) \cdot (y - z)$
using ‹$y \in S$› ‹$z \notin S$› **by** *auto*
{ assume *0*: $0 < ((z - y) \cdot (x - y))$
with *any_closest_point_dot* [*OF* ‹*convex S*› ‹*closed S*›]
have *False*

```
      using y ‹x ∈ S› ‹y ∈ S› not_less by blast
    }
    then have 0 ≤ ((y − z) · (x − y))
      by (force simp: not_less inner_diff_left)
    with yz have 0 < 2 ∗ ((y − z) · (x − y)) + (y − z) · (y − z)
      by (simp add: algebra_simps)
    then show (y − z) · z + (norm (y − z))^2 / 2 < (y − z) · x
     by (simp add: field_simps inner_diff_left inner_diff_right dot_square_norm
[symmetric])
  qed
qed

lemma separating_hyperplane_closed_0_inset:
  fixes S :: 'a::euclidean_space set
  assumes convex S closed S S ≠ {} 0 ∉ S
  obtains a b where a ∈ S a ≠ 0 0 < b ⋀x. x ∈ S ⟹ a · x > b
  using separating_hyperplane_closed_point_inset [OF assms] by simp (metis ‹0
∉ S›)


proposition separating_hyperplane_set_0_inspan:
  fixes S :: 'a::euclidean_space set
  assumes convex S S ≠ {} 0 ∉ S
  obtains a where a ∈ span S a ≠ 0 ⋀x. x ∈ S ⟹ 0 ≤ a · x
proof −
  define k where [abs_def]: k c = {x. 0 ≤ c · x} for c :: 'a
  have span S ∩ frontier (cball 0 1) ∩ ⋂f' ≠ {}
        if f': finite f' f' ⊆ k ‘ S for f'
  proof −
    obtain C where C ⊆ S finite C and C: f' = k ‘ C
      using finite_subset_image [OF f'] by blast
    obtain a where a ∈ S a ≠ 0
      using ‹S ≠ {}› ‹0 ∉ S› ex_in_conv by blast
    then have norm (a /R (norm a)) = 1
      by simp
    moreover have a /R (norm a) ∈ span S
      by (simp add: ‹a ∈ S› span_scale span_base)
    ultimately have ass: a /R (norm a) ∈ span S ∩ sphere 0 1
      by simp
    show ?thesis
    proof (cases C = {})
      case True with C ass show ?thesis
        by auto
    next
      case False
      have closed (convex hull C)
      using ‹finite C› compact_eq_bounded_closed finite_imp_compact_convex_hull
by auto
      moreover have convex hull C ≠ {}
```

```
      by (simp add: False)
    moreover have 0 ∉ convex hull C
      by (metis ‹C ⊆ S› ‹convex S› ‹0 ∉ S› convex_hull_subset hull_same
insert_absorb insert_subset)
    ultimately obtain a b
      where a ∈ convex hull C a ≠ 0 0 < b
        and ab: ⋀x. x ∈ convex hull C ⟹ a · x > b
      using separating_hyperplane_closed_0_inset by blast
    then have a ∈ S
      by (metis ‹C ⊆ S› assms(1) subsetCE subset_hull)
    moreover have norm (a /R (norm a)) = 1
      using ‹a ≠ 0› by simp
    moreover have a /R (norm a) ∈ span S
      by (simp add: ‹a ∈ S› span_scale span_base)
    ultimately have ass: a /R (norm a) ∈ span S ∩ sphere 0 1
      by simp
    have ⋀x. ⟦a ≠ 0; x ∈ C⟧ ⟹ 0 ≤ x · a
      using ab ‹0 < b› by (metis hull_inc inner_commute less_eq_real_def
less_trans)
    then have aa: a /R (norm a) ∈ (⋂c∈C. {x. 0 ≤ c · x})
      by (auto simp add: field_split_simps)
    show ?thesis
      unfolding C k_def
      using ass aa Int_iff empty_iff by force
  qed
 qed
 moreover have ⋀T. T ∈ k ‘ S ⟹ closed T
  using closed_halfspace_ge k_def by blast
 ultimately have (span S ∩ frontier(cball 0 1)) ∩ (⋂ (k ‘ S)) ≠ {}
  by (metis compact_imp_fip closed_Int_compact closed_span compact_cball
compact_frontier)
 then show ?thesis
  unfolding set_eq_iff k_def
  by simp (metis inner_commute norm_eq_zero that zero_neq_one)
qed


lemma separating_hyperplane_set_point_inaff:
 fixes S :: 'a::euclidean_space set
 assumes convex S S ≠ {} and zno: z ∉ S
 obtains a b where (z + a) ∈ affine hull (insert z S)
            and a ≠ 0 and a · z ≤ b
            and ⋀x. x ∈ S ⟹ a · x ≥ b
proof −
 from separating_hyperplane_set_0_inspan [of image (λx. −z + x) S]
 have convex ((+) (− z) ‘ S)
  using ‹convex S› by simp
 moreover have (+) (− z) ‘ S ≠ {}
  by (simp add: ‹S ≠ {}›)
```

```
  moreover have 0 ∉ (+) (− z) ‘ S
    using zno by auto
  ultimately obtain a where a ∈ span ((+) (− z) ‘ S) a ≠ 0
                and a: ⋀x. x ∈ ((+) (− z) ‘ S) ⟹ 0 ≤ a · x
    using separating_hyperplane_set_0_inspan [of image (λx. −z + x) S]
    by blast
  then have szx: ⋀x. x ∈ S ⟹ a · z ≤ a · x
    by (metis (no_types, lifting) imageI inner_minus_right inner_right_distrib
minus_add neg_le_0_iff_le neg_le_iff_le real_add_le_0_iff)
  moreover
  have z + a ∈ affine hull insert z S
    using ‹a ∈ span ((+) (− z) ‘ S)› affine_hull_insert_span_gen by blast
  ultimately show ?thesis
    using ‹a ≠ 0› szx that by auto
qed

proposition supporting_hyperplane_rel_boundary:
  fixes S :: 'a::euclidean_space set
  assumes convex S x ∈ S and xno: x ∉ rel_interior S
  obtains a where a ≠ 0
            and ⋀y. y ∈ S ⟹ a · x ≤ a · y
            and ⋀y. y ∈ rel_interior S ⟹ a · x < a · y
proof −
  obtain a b where aff: (x + a) ∈ affine hull (insert x (rel_interior S))
                and a ≠ 0 and a · x ≤ b
                and ageb: ⋀u. u ∈ (rel_interior S) ⟹ a · u ≥ b
    using separating_hyperplane_set_point_inaff [of rel_interior S x] assms
    by (auto simp: rel_interior_eq_empty convex_rel_interior)
  have le_ay: a · x ≤ a · y if y ∈ S for y
  proof −
    have con: continuous_on (closure (rel_interior S)) ((·) a)
      by (rule continuous_intros continuous_on_subset | blast)+
    have y: y ∈ closure (rel_interior S)
      using ‹convex S› closure_def convex_closure_rel_interior ‹y ∈ S›
      by fastforce
    show ?thesis
      using continuous_ge_on_closure [OF con y] ageb ‹a · x ≤ b›
      by fastforce
  qed
  have 3: a · x < a · y if y ∈ rel_interior S for y
  proof −
    obtain e where 0 < e y ∈ S and e: cball y e ∩ affine hull S ⊆ S
      using ‹y ∈ rel_interior S› by (force simp: rel_interior_cball)
    define y' where y' = y − (e / norm a) *R ((x + a) − x)
    have y' ∈ cball y e
      unfolding y'_def using ‹0 < e› by force
    moreover have y' ∈ affine hull S
      unfolding y'_def
      by (metis ‹x ∈ S› ‹y ∈ S› ‹convex S› aff affine_affine_hull hull_redundant
```

rel_interior_same_affine_hull hull_inc mem_affine_3_minus2)
ultimately have $y' \in S$
using *e* **by** *auto*
have $a \cdot x \leq a \cdot y$
using *le_ay* ‹$a \neq 0$› ‹$y \in S$› **by** *blast*
moreover have $a \cdot x \neq a \cdot y$
using *le_ay* [*OF* ‹$y' \in S$›] ‹$a \neq 0$› ‹$0 < e$› *not_le*
by (*fastforce simp add*: *y'_def inner_diff dot_square_norm power2_eq_square*)
ultimately show *?thesis* **by** *force*
qed
show *?thesis*
by (*rule that* [*OF* ‹$a \neq 0$› *le_ay 3*])
qed

lemma *supporting_hyperplane_relative_frontier*:
fixes S :: *'a*::*euclidean_space set*
assumes *convex S x* ∈ *closure S x* ∉ *rel_interior S*
obtains *a* **where** $a \neq 0$
and $\bigwedge y.\ y \in$ *closure S* $\Longrightarrow a \cdot x \leq a \cdot y$
and $\bigwedge y.\ y \in$ *rel_interior S* $\Longrightarrow a \cdot x < a \cdot y$
using *supporting_hyperplane_rel_boundary* [*of closure S x*]
by (*metis assms convex_closure convex_rel_interior_closure*)

5.0.16 Some results on decomposing convex hulls: intersections, simplicial subdivision

lemma
fixes S :: *'a*::*euclidean_space set*
assumes ¬ *affine_dependent*($S \cup T$)
shows *convex_hull_Int_subset*: *convex hull S* ∩ *convex hull T* ⊆ *convex hull* ($S \cap T$) (**is** *?C*)
and *affine_hull_Int_subset*: *affine hull S* ∩ *affine hull T* ⊆ *affine hull* ($S \cap T$) (**is** *?A*)
proof −
have [*simp*]: *finite S finite T*
using *aff_independent_finite assms* **by** *blast*+
have *sum u* ($S \cap T$) = 1 ∧
($\sum v \in S \cap T.\ u\ v *_R v$) = ($\sum v \in S.\ u\ v *_R v$)
if [*simp*]: *sum u S* = 1
sum v T = 1
and *eq*: ($\sum x \in T.\ v\ x *_R x$) = ($\sum x \in S.\ u\ x *_R x$) **for** *u v*
proof −
define *f* **where** *f x* = (*if x* ∈ *S then u x else* 0) − (*if x* ∈ *T then v x else* 0)
for *x*
have *sum f* ($S \cup T$) = 0
by (*simp add*: *f_def sum_Un sum_subtractf flip*: *sum.inter_restrict*)
moreover have ($\sum x \in (S \cup T).\ f\ x *_R x$) = 0
by (*simp add*: *eq f_def sum_Un scaleR_left_diff_distrib sum_subtractf if_smult flip*: *sum.inter_restrict cong*: *if_cong*)

```
    ultimately have ⋀v. v ∈ S ∪ T ⟹ f v = 0
      using aff_independent_finite assms unfolding affine_dependent_explicit
      by blast
    then have u [simp]: ⋀x. x ∈ S ⟹ u x = (if x ∈ T then v x else 0)
      by (simp add: f_def) presburger
    have sum u (S ∩ T) = sum u S
      by (simp add: sum.inter_restrict)
    then have sum u (S ∩ T) = 1
      using that by linarith
    moreover have (∑ v∈S ∩ T. u v *R v) = (∑ v∈S. u v *R v)
    by (auto simp: if_smult sum.inter_restrict intro: sum.cong)
   ultimately show ?thesis
    by force
   qed
   then show ?A ?C
    by (auto simp: convex_hull_finite affine_hull_finite)
qed


proposition affine_hull_Int:
  fixes S :: 'a::euclidean_space set
  assumes ¬ affine_dependent(S ∪ T)
    shows affine hull (S ∩ T) = affine hull S ∩ affine hull T
  by (simp add: affine_hull_Int_subset assms hull_mono subset_antisym)

proposition convex_hull_Int:
  fixes S :: 'a::euclidean_space set
  assumes ¬ affine_dependent(S ∪ T)
    shows convex hull (S ∩ T) = convex hull S ∩ convex hull T
  by (simp add: convex_hull_Int_subset assms hull_mono subset_antisym)

proposition
  fixes S :: 'a::euclidean_space set set
  assumes ¬ affine_dependent (⋃S)
    shows affine_hull_Inter: affine hull (⋂S) = (⋂T∈S. affine hull T) (is ?A)
     and convex_hull_Inter: convex hull (⋂S) = (⋂T∈S. convex hull T) (is ?C)
proof −
  have finite S
    using aff_independent_finite assms finite_UnionD by blast
  then have ?A ∧ ?C using assms
  proof (induction S rule: finite_induct)
    case empty then show ?case by auto
  next
    case (insert T F)
    then show ?case
    proof (cases F={})
      case True then show ?thesis by simp
    next
      case False
```

```
    with insert.prems have [simp]: ¬ affine_dependent (T ∪ ⋂F)
      by (auto intro: affine_dependent_subset)
    have [simp]: ¬ affine_dependent (⋃F)
      using affine_independent_subset insert.prems by fastforce
    show ?thesis
      by (simp add: affine_hull_Int convex_hull_Int insert.IH)
  qed
 qed
 then show ?A ?C
  by auto
qed

proposition in_convex_hull_exchange_unique:
 fixes S :: 'a::euclidean_space set
 assumes naff: ¬ affine_dependent S and a: a ∈ convex hull S
    and S: T ⊆ S T' ⊆ S
    and x:  x ∈ convex hull (insert a T)
    and x': x ∈ convex hull (insert a T')
  shows x ∈ convex hull (insert a (T ∩ T'))
proof (cases a ∈ S)
 case True
 then have ¬ affine_dependent (insert a T ∪ insert a T')
  using affine_dependent_subset assms by auto
 then have x ∈ convex hull (insert a T ∩ insert a T')
  by (metis IntI convex_hull_Int x x')
 then show ?thesis
  by simp
next
 case False
 then have anot: a ∉ T a ∉ T'
  using assms by auto
 have [simp]: finite S
  by (simp add: aff_independent_finite assms)
 then obtain b where b0: ⋀s. s ∈ S ⟹ 0 ≤ b s
                 and b1: sum b S = 1 and aeq: a = (∑s∈S. b s *R s)
  using a by (auto simp: convex_hull_finite)
 have fin [simp]: finite T finite T'
  using assms infinite_super ‹finite S› by blast+
 then obtain c c' where c0: ⋀t. t ∈ insert a T ⟹ 0 ≤ c t
                    and c1: sum c (insert a T) = 1
                    and xeq: x = (∑t ∈ insert a T. c t *R t)
                    and c'0: ⋀t. t ∈ insert a T' ⟹ 0 ≤ c' t
                    and c'1: sum c' (insert a T') = 1
                    and x'eq: x = (∑t ∈ insert a T'. c' t *R t)
  using x x' by (auto simp: convex_hull_finite)
 with fin anot
 have sumTT': sum c T = 1 − c a sum c' T' = 1 − c' a
  and wsumT: (∑t ∈ T. c t *R t) = x − c a *R a
  by simp_all
```

have *wsumT'*: $(\sum t \in T'.\ c'\ t *_R t) = x - c'\ a *_R a$
using *x'eq fin anot* **by** *simp*
define *cc* **where** $cc \equiv \lambda x.\ if\ x \in T\ then\ c\ x\ else\ 0$
define *cc'* **where** $cc' \equiv \lambda x.\ if\ x \in T'\ then\ c'\ x\ else\ 0$
define *dd* **where** $dd \equiv \lambda x.\ cc\ x - cc'\ x + (c\ a - c'\ a) * b\ x$
have *sumSS'*: *sum cc S* $= 1 - c\ a$ *sum cc' S* $= 1 - c'\ a$
unfolding *cc_def cc'_def* **using** *S*
by (*simp_all add*: *Int_absorb1 Int_absorb2 sum_subtractf sum.inter_restrict* [*symmetric*] *sumTT'*)
have *wsumSS*: $(\sum t \in S.\ cc\ t *_R t) = x - c\ a *_R a$ $(\sum t \in S.\ cc'\ t *_R t) = x - c'\ a *_R a$
unfolding *cc_def cc'_def* **using** *S*
by (*simp_all add*: *Int_absorb1 Int_absorb2 if_smult sum.inter_restrict* [*symmetric*] *wsumT wsumT' cong*: *if_cong*)
have *sum_dd0*: *sum dd S* $= 0$
unfolding *dd_def* **using** *S*
by (*simp add*: *sumSS' comm_monoid_add_class.sum.distrib sum_subtractf algebra_simps sum_distrib_right* [*symmetric*] *b1*)
have $(\sum v{\in}S.\ (b\ v * x) *_R v) = x *_R (\sum v{\in}S.\ b\ v *_R v)$ **for** *x*
by (*simp add*: *pth_5 real_vector.scale_sum_right mult.commute*)
then have $*$: $(\sum v{\in}S.\ (b\ v * x) *_R v) = x *_R a$ **for** *x*
using *aeq* **by** *blast*
have $(\sum v \in S.\ dd\ v *_R v) = 0$
unfolding *dd_def* **using** *S*
by (*simp add*: $*$ *wsumSS sum.distrib sum_subtractf algebra_simps*)
then have *dd0*: *dd v* $= 0$ **if** $v \in S$ **for** *v*
using *naff* [*unfolded affine_dependent_explicit not_ex*, *rule_format*, *of S dd*]
using *that sum_dd0* **by** *force*
consider $c'\ a \le c\ a \mid c\ a \le c'\ a$ **by** *linarith*
then show *?thesis*
proof *cases*
case *1*
then have *sum cc S* $\le$ *sum cc' S*
by (*simp add*: *sumSS'*)
then have *le*: *cc x* $\le$ *cc' x* **if** $x \in S$ **for** *x*
using *dd0* [*OF that*] *1 b0 mult_left_mono that*
by (*fastforce simp add*: *dd_def algebra_simps*)
have *cc0*: *cc x* $= 0$ **if** $x \in S\ x \notin T \cap T'$ **for** *x*
using *le* [*OF* ‹$x \in S$›] *that c0*
by (*force simp*: *cc_def cc'_def split*: *if_split_asm*)
show *?thesis*
proof (*simp add*: *convex_hull_finite*, *intro exI conjI*)
show $\forall x{\in}T \cap T'.\ 0 \le (cc(a := c\ a))\ x$
by (*simp add*: *c0 cc_def*)
show $0 \le (cc(a := c\ a))\ a$
by (*simp add*: *c0*)
have *sum* $(cc(a := c\ a))$ $(insert\ a\ (T \cap T'))$ $= c\ a +$ *sum* $(cc(a := c\ a))$ $(T \cap T')$
by (*simp add*: *anot*)

```
    also have ... = c a + sum (cc(a := c a)) S
    using ‹T ⊆ S› False cc0 cc_def ‹a ∉ S› by (fastforce intro!: sum.mono_neutral_left
split: if_split_asm)
    also have ... = c a + (1 - c a)
      by (metis ‹a ∉ S› fun_upd_other sum.cong sumSS'(1))
    finally show sum (cc(a := c a)) (insert a (T ∩ T')) = 1
      by simp
    have (∑x∈insert a (T ∩ T'). (cc(a := c a)) x *R x) = c a *R a + (∑x ∈
T ∩ T'. (cc(a := c a)) x *R x)
      by (simp add: anot)
    also have ... = c a *R a + (∑x ∈ S. (cc(a := c a)) x *R x)
      using ‹T ⊆ S› False cc0 by (fastforce intro!: sum.mono_neutral_left split:
if_split_asm)
    also have ... = c a *R a + x - c a *R a
      by (simp add: wsumSS ‹a ∉ S› if_smult sum_delta_notmem)
    finally show (∑x∈insert a (T ∩ T'). (cc(a := c a)) x *R x) = x
      by simp
  qed
 next
  case 2
  then have sum cc' S ≤ sum cc S
    by (simp add: sumSS')
  then have le: cc' x ≤ cc x if x ∈ S for x
    using dd0 [OF that] 2 b0 mult_left_mono that
    by (fastforce simp add: dd_def algebra_simps)
  have cc0: cc' x = 0 if x ∈ S x ∉ T ∩ T' for x
    using le [OF ‹x ∈ S›] that c'0
    by (force simp: cc_def cc'_def split: if_split_asm)
  show ?thesis
  proof (simp add: convex_hull_finite, intro exI conjI)
    show ∀x∈T ∩ T'. 0 ≤ (cc'(a := c' a)) x
      by (simp add: c'0 cc'_def)
    show 0 ≤ (cc'(a := c' a)) a
      by (simp add: c'0)
    have sum (cc'(a := c' a)) (insert a (T ∩ T')) = c' a + sum (cc'(a := c' a))
(T ∩ T')
      by (simp add: anot)
    also have ... = c' a + sum (cc'(a := c' a)) S
      using ‹T ⊆ S› False cc0 by (fastforce intro!: sum.mono_neutral_left split:
if_split_asm)
    also have ... = c' a + (1 - c' a)
      by (metis ‹a ∉ S› fun_upd_other sum.cong sumSS')
    finally show sum (cc'(a := c' a)) (insert a (T ∩ T')) = 1
      by simp
    have (∑x∈insert a (T ∩ T'). (cc'(a := c' a)) x *R x) = c' a *R a + (∑x
∈ T ∩ T'. (cc'(a := c' a)) x *R x)
      by (simp add: anot)
    also have ... = c' a *R a + (∑x ∈ S. (cc'(a := c' a)) x *R x)
      using ‹T ⊆ S› False cc0 by (fastforce intro!: sum.mono_neutral_left split:
```

```
if_split_asm)
      also have ... = c a *R a + x - c a *R a
        by (simp add: wsumSS ‹a ∉ S› if_smult sum_delta_notmem)
      finally show (∑x∈insert a (T ∩ T'). (cc'(a := c' a)) x *R x) = x
        by simp
    qed
  qed
qed

corollary convex_hull_exchange_Int:
  fixes a :: 'a::euclidean_space
  assumes ¬ affine_dependent S a ∈ convex hull S T ⊆ S T' ⊆ S
  shows (convex hull (insert a T)) ∩ (convex hull (insert a T')) =
        convex hull (insert a (T ∩ T')) (is ?lhs = ?rhs)
proof (rule subset_antisym)
  show ?lhs ⊆ ?rhs
    using in_convex_hull_exchange_unique assms by blast
  show ?rhs ⊆ ?lhs
    by (metis hull_mono inf_le1 inf_le2 insert_inter_insert le_inf_iff)
qed

lemma Int_closed_segment:
  fixes b :: 'a::euclidean_space
  assumes b ∈ closed_segment a c ∨ ¬ collinear{a,b,c}
    shows closed_segment a b ∩ closed_segment b c = {b}
proof (cases c = a)
  case True
  then show ?thesis
    using assms collinear_3_eq_affine_dependent by fastforce
next
  case False
  from assms show ?thesis
  proof
    assume b ∈ closed_segment a c
    moreover have ¬ affine_dependent {a, c}
      by (simp)
    ultimately show ?thesis
      using False convex_hull_exchange_Int [of {a,c} b {a} {c}]
      by (simp add: segment_convex_hull insert_commute)
  next
    assume ncoll: ¬ collinear {a, b, c}
    have False if closed_segment a b ∩ closed_segment b c ≠ {b}
    proof −
      have b ∈ closed_segment a b and b ∈ closed_segment b c
        by auto
     with that obtain d where b ≠ d d ∈ closed_segment a b d ∈ closed_segment
b c
        by force
      then have d: collinear {a, d, b}  collinear {b, d, c}
```

```
      by (auto simp: between_mem_segment between_imp_collinear)
    have collinear {a, b, c}
      by (metis (full_types) ‹b ≠ d› collinear_3_trans d insert_commute)
    with ncoll show False ..
  qed
  then show ?thesis
    by blast
 qed
qed

lemma affine_hull_finite_intersection_hyperplanes:
 fixes S :: 'a::euclidean_space set
 obtains ℱ where
   finite ℱ
   of_nat (card ℱ) + aff_dim S = DIM('a)
   affine hull S = ⋂ℱ
   ⋀h. h ∈ ℱ ⟹ ∃a b. a ≠ 0 ∧ h = {x. a • x = b}
proof −
 obtain b where b ⊆ S
           and indb: ¬ affine_dependent b
           and eq: affine hull S = affine hull b
   using affine_basis_exists by blast
 obtain c where indc: ¬ affine_dependent c and b ⊆ c
           and affc: affine hull c = UNIV
  by (metis extend_to_affine_basis affine_UNIV hull_same indb subset_UNIV)
 then have finite c
   by (simp add: aff_independent_finite)
 then have fbc: finite b card b ≤ card c
   using ‹b ⊆ c› infinite_super by (auto simp: card_mono)
 have imeq: (λx. affine hull x) ‘ ((λa. c − {a}) ‘ (c − b)) = ((λa. affine hull (c
− {a})) ‘ (c − b))
   by blast
 have card_cb: (card (c − b)) + aff_dim S = DIM('a)
 proof −
   have aff: aff_dim (UNIV::'a set) = aff_dim c
     by (metis aff_dim_affine_hull affc)
   have aff_dim b = aff_dim S
     by (metis (no_types) aff_dim_affine_hull eq)
   then have int (card b) = 1 + aff_dim S
     by (simp add: aff_dim_affine_independent indb)
   then show ?thesis
     using fbc aff
     by (simp add: ‹¬ affine_dependent c› ‹b ⊆ c› aff_dim_affine_independent
card_Diff_subset of_nat_diff)
 qed
 show ?thesis
 proof (cases c = b)
   case True show ?thesis
   proof
```

```
      show int (card {}) + aff_dim S = int DIM('a)
        using True card_cb by auto
      show affine hull S = ⋂ {}
        using True affc eq by blast
    qed auto
  next
    case False
    have ind: ¬ affine_dependent (⋃a∈c - b. c - {a})
      by (rule affine_independent_subset [OF indc]) auto
    have *: 1 + aff_dim (c - {t}) = int (DIM('a)) if t: t ∈ c for t
    proof -
      have insert t c = c
        using t by blast
      then show ?thesis
        by (metis (full_types) add.commute aff_dim_affine_hull aff_dim_insert
aff_dim_UNIV affc affine_dependent_def indc insert_Diff_single t)
    qed
    let ?F = (λx. affine hull x) ` ((λa. c - {a}) ` (c - b))
    show ?thesis
    proof
      have card ((λa. affine hull (c - {a})) ` (c - b)) = card (c - b)
      proof (rule card_image)
        show inj_on (λa. affine hull (c - {a})) (c - b)
          unfolding inj_on_def
       by (metis Diff_eq_empty_iff Diff_iff indc affine_dependent_def hull_subset
insert_iff)
      qed
      then show int (card ?F) + aff_dim S = int DIM('a)
        by (simp add: imeq card_cb)
      show affine hull S = ⋂ ?F
       by (metis Diff_eq_empty_iff INT_simps(4) UN_singleton order_refl ‹b ⊆
c› False eq double_diff affine_hull_Inter [OF ind])
      have ⋀a. ⟦a ∈ c; a ∉ b⟧ ⟹ aff_dim (c - {a}) = int (DIM('a) - Suc 0)
       by (metis * DIM_ge_Suc0 One_nat_def add_diff_cancel_left' int_ops(2)
of_nat_diff)
      then show ⋀h. h ∈ ?F ⟹ ∃a b. a ≠ 0 ∧ h = {x. a · x = b}
        by (auto simp only: One_nat_def aff_dim_eq_hyperplane [symmetric])
      qed (use ‹finite c› in auto)
  qed
qed

lemma affine_hyperplane_sums_eq_UNIV_0:
  fixes S :: 'a :: euclidean_space set
  assumes affine S
    and 0 ∈ S and w ∈ S
    and a · w ≠ 0
   shows {x + y| x y. x ∈ S ∧ a · y = 0} = UNIV
proof -
  have subspace S
```

```
    by (simp add: assms subspace_affine)
  have span1: span {y. a · y = 0} ⊆ span {x + y |x y. x ∈ S ∧ a · y = 0}
    using ‹0 ∈ S› add.left_neutral by (intro span_mono) force
  have w ∉ span {y. a · y = 0}
    using ‹a · w ≠ 0› span_induct subspace_hyperplane by auto
  moreover have w ∈ span {x + y |x y. x ∈ S ∧ a · y = 0}
    using ‹w ∈ S›
   by (metis (mono_tags, lifting) inner_zero_right mem_Collect_eq pth_d span_base)
  ultimately have span2: span {y. a · y = 0} ≠ span {x + y |x y. x ∈ S ∧ a ·
y = 0}
    by blast
  have a ≠ 0 using assms inner_zero_left by blast
  then have DIM('a) − 1 = dim {y. a · y = 0}
    by (simp add: dim_hyperplane)
  also have ... < dim {x + y |x y. x ∈ S ∧ a · y = 0}
    using span1 span2 by (blast intro: dim_psubset)
  finally have DIM('a) − 1 < dim {x + y |x y. x ∈ S ∧ a · y = 0} .
  then have DD: dim {x + y |x y. x ∈ S ∧ a · y = 0} = DIM('a)
    using antisym dim_subset_UNIV lowdim_subset_hyperplane not_le by fast-
force
  have subs: subspace {x + y| x y. x ∈ S ∧ a · y = 0}
    using subspace_sums [OF ‹subspace S› subspace_hyperplane] by simp
  moreover have span {x + y| x y. x ∈ S ∧ a · y = 0} = UNIV
    using DD dim_eq_full by blast
  ultimately show ?thesis
    by (simp add: subs) (metis (lifting) span_eq_iff subs)
qed

proposition affine_hyperplane_sums_eq_UNIV:
  fixes S :: 'a :: euclidean_space set
  assumes affine S
      and S ∩ {v. a · v = b} ≠ {}
      and S − {v. a · v = b} ≠ {}
    shows {x + y| x y. x ∈ S ∧ a · y = b} = UNIV
proof (cases a = 0)
  case True with assms show ?thesis
    by (auto simp: if_splits)
next
  case False
  obtain c where c ∈ S and c: a · c = b
    using assms by force
  with affine_diffs_subspace [OF ‹affine S›]
  have subspace ((+) (− c) ‘ S) by blast
  then have aff: affine ((+) (− c) ‘ S)
    by (simp add: subspace_imp_affine)
  have 0: 0 ∈ (+) (− c) ‘ S
    by (simp add: ‹c ∈ S›)
  obtain d where d ∈ S and a · d ≠ b and dc: d−c ∈ (+) (− c) ‘ S
    using assms by auto
```

```
  then have adc: a · (d − c) ≠ 0
    by (simp add: c inner_diff_right)
  define U where U ≡ {x + y |x y. x ∈ (+) (− c) ‘ S ∧ a · y = 0}
  have u + v ∈ (+) (c+c) ‘ U
    if u ∈ S b = a · v for u v
  proof
    show u + v = c + c + (u + v − c − c)
      by (simp add: algebra_simps)
    have ∃ x y. u + v − c − c = x + y ∧ (∃ xa∈S. x = xa − c) ∧ a · y = 0
    proof (intro exI conjI)
      show u + v − c − c = (u−c) + (v−c) a · (v − c) = 0
        by (simp_all add: algebra_simps c that)
    qed (use that in auto)
    then show u + v − c − c ∈ U
      by (auto simp: U_def image_def)
  qed
  moreover have ⟦a · v = 0; u ∈ S⟧
      ⟹ ∃ x ya. v + (u + c) = x + ya ∧ x ∈ S ∧ a · ya = b for v u
    by (metis add.left_commute c inner_right_distrib pth_d)
  ultimately have {x + y |x y. x ∈ S ∧ a · y = b} = (+) (c+c) ‘ U
    by (fastforce simp: algebra_simps U_def)
  also have ... = range ((+) (c + c))
   by (simp only: U_def affine_hyperplane_sums_eq_UNIV_0 [OF aff 0 dc adc])
  also have ... = UNIV
    by simp
  finally show ?thesis .
qed

lemma aff_dim_sums_Int_0:
  assumes affine S
      and affine T
      and 0 ∈ S 0 ∈ T
    shows aff_dim {x + y| x y. x ∈ S ∧ y ∈ T} = (aff_dim S + aff_dim T) −
aff_dim(S ∩ T)
proof −
  have 0 ∈ {x + y |x y. x ∈ S ∧ y ∈ T}
    using assms by force
  then have 0: 0 ∈ affine hull {x + y |x y. x ∈ S ∧ y ∈ T}
    by (metis (lifting) hull_inc)
  have sub: subspace S  subspace T
    using assms by (auto simp: subspace_affine)
  show ?thesis
   using dim_sums_Int [OF sub] by (simp add: aff_dim_zero assms 0 hull_inc)
qed

proposition aff_dim_sums_Int:
  assumes affine S
      and affine T
      and S ∩ T ≠ {}
```

shows *aff_dim* $\{x + y |\ x\ y.\ x \in S \wedge y \in T\} = (aff\_dim\ S + aff\_dim\ T) - aff\_dim(S \cap T)$
proof −
obtain *a* **where** *a*: $a \in S\ a \in T$ **using** *assms* **by** *force*
have *aff*: *affine* $((+)\ (-a)\ `\ S)$ *affine* $((+)\ (-a)\ `\ T)$
using *affine_translation* [*symmetric*, *of* − *a*] *assms* **by** (*simp_all cong*: *image_cong_simp*)
have *zero*: $0 \in ((+)\ (-a)\ `\ S)$ $0 \in ((+)\ (-a)\ `\ T)$
using *a assms* **by** *auto*
have $\{x + y\ |x\ y.\ x \in (+)\ (-\ a)\ `\ S \wedge y \in (+)\ (-\ a)\ `\ T\} =$
$(+)\ (-\ 2 *_R\ a)\ `\ \{x + y |\ x\ y.\ x \in S \wedge y \in T\}$
by (*force simp*: *algebra_simps scaleR_2*)
moreover have $(+)\ (-\ a)\ `\ S \cap (+)\ (-\ a)\ `\ T = (+)\ (-\ a)\ `\ (S \cap T)$
by *auto*
ultimately show *?thesis*
using *aff_dim_sums_Int_0* [*OF aff zero*] *aff_dim_translation_eq*
by (*metis* (*lifting*))
qed

lemma *aff_dim_affine_Int_hyperplane*:
fixes *a* :: *'a*::*euclidean_space*
assumes *affine S*
shows $aff\_dim(S \cap \{x.\ a \cdot x = b\}) =$
(*if* $S \cap \{v.\ a \cdot v = b\} = \{\}$ *then* − *1*
else if $S \subseteq \{v.\ a \cdot v = b\}$ *then aff_dim S*
else aff_dim S − *1*)
proof (*cases* $a = 0$)
case *True* **with** *assms* **show** *?thesis*
by *auto*
next
case *False*
then have $aff\_dim\ (S \cap \{x.\ a \cdot x = b\}) = aff\_dim\ S - 1$
if $x \in S\ a \cdot x \neq b$ **and** *non*: $S \cap \{v.\ a \cdot v = b\} \neq \{\}$ **for** *x*
proof −
have [*simp*]: $\{x + y |\ x\ y.\ x \in S \wedge a \cdot y = b\} = UNIV$
using *affine_hyperplane_sums_eq_UNIV* [*OF assms non*] *that* **by** *blast*
show *?thesis*
using *aff_dim_sums_Int* [*OF assms affine_hyperplane non*]
by (*simp add*: *of_nat_diff False*)
qed
then show *?thesis*
by (*metis* (*mono_tags*, *lifting*) *inf.orderE aff_dim_empty_eq mem_Collect_eq subsetI*)
qed

lemma *aff_dim_lt_full*:
fixes *S* :: *'a*::*euclidean_space set*
shows $aff\_dim\ S < DIM('a) \longleftrightarrow (affine\ hull\ S \neq UNIV)$
by (*metis* (*no_types*) *aff_dim_affine_hull aff_dim_le_DIM aff_dim_UNIV affine_hull_UNIV*

less_le)

lemma *aff_dim_openin*:
fixes S :: *'a*::*euclidean_space set*
assumes *ope*: *openin* (*top_of_set* T) S **and** *affine* T $S \neq \{\}$
shows *aff_dim* S = *aff_dim* T
proof −
show *?thesis*
proof (*rule order_antisym*)
show *aff_dim* $S \leq$ *aff_dim* T
by (*blast intro*: *aff_dim_subset* [*OF openin_imp_subset*] *ope*)
next
obtain a **where** $a \in S$
using ‹$S \neq \{\}$› **by** *blast*
have $S \subseteq T$
using *ope openin_imp_subset* **by** *auto*
then have $a \in T$
using ‹$a \in S$› **by** *auto*
then have *subT'*: *subspace* ((λ*x*. − *a* + *x*) ‘ T)
using *affine_diffs_subspace* ‹*affine* T› **by** *auto*
then obtain B **where** *Bsub*: $B \subseteq$ ((λ*x*. − *a* + *x*) ‘ T) **and** *po*: *pairwise orthogonal B*
and *eq1*: $\bigwedge x.\ x \in B \Longrightarrow$ *norm* x = *1* **and** *independent B*
and *cardB*: *card* B = *dim* ((λ*x*. − *a* + *x*) ‘ T)
and *spanB*: *span* B = ((λ*x*. − *a* + *x*) ‘ T)
by (*rule orthonormal_basis_subspace*) *auto*
obtain *e* **where** *0* < *e* **and** *e*: *cball a e* ∩ $T \subseteq S$
by (*meson* ‹$a \in S$› *openin_contains_cball ope*)
have *aff_dim* T = *aff_dim* ((λ*x*. − *a* + *x*) ‘ T)
by (*metis aff_dim_translation_eq*)
also have ... = *dim* ((λ*x*. − *a* + *x*) ‘ T)
using *aff_dim_subspace subT'* **by** *blast*
also have ... = *card B*
by (*simp add*: *cardB*)
also have ... = *card* ((λ*x*. *e* ∗$_R$ *x*) ‘ B)
using ‹*0* < *e*› **by** (*force simp*: *inj_on_def card_image*)
also have ... ≤ *dim* ((λ*x*. − *a* + *x*) ‘ S)
proof (*simp*, *rule independent_card_le_dim*)
have *e'*: *cball 0 e* ∩ (λ*x*. *x* − *a*) ‘ $T \subseteq$ (λ*x*. *x* − *a*) ‘ S
using *e* **by** (*auto simp*: *dist_norm norm_minus_commute subset_eq*)
have (λ*x*. *e* ∗$_R$ *x*) ‘ $B \subseteq$ *cball 0 e* ∩ (λ*x*. *x* − *a*) ‘ T
using *Bsub* ‹*0* < *e*› *eq1 subT'* ‹$a \in T$› **by** (*auto simp*: *subspace_def*)
then show (λ*x*. *e* ∗$_R$ *x*) ‘ $B \subseteq$ (λ*x*. *x* − *a*) ‘ S
using *e'* **by** *blast*
have *inj_on* ((∗$_R$) *e*) (*span B*)
using ‹*0* < *e*› *inj_on_def* **by** *fastforce*
then show *independent* ((λ*x*. *e* ∗$_R$ *x*) ‘ B)
using *linear_scale_self* ‹*independent B*› *linear_dependent_inj_imageD* **by** *blast*

```
    qed
    also have ... = aff_dim S
      using ‹a ∈ S› aff_dim_eq_dim hull_inc by (force cong: image_cong_simp)
    finally show aff_dim T ≤ aff_dim S .
  qed
qed

lemma dim_openin:
  fixes S :: 'a::euclidean_space set
  assumes ope: openin (top_of_set T) S and subspace T S ≠ {}
  shows dim S = dim T
proof (rule order_antisym)
  show dim S ≤ dim T
    by (metis ope dim_subset openin_subset topspace_euclidean_subtopology)
next
  have dim T = aff_dim S
    using aff_dim_openin
    by (metis aff_dim_subspace ‹subspace T› ‹S ≠ {}› ope subspace_affine)
  also have ... ≤ dim S
    by (metis aff_dim_subset aff_dim_subspace dim_span span_superset
        subspace_span)
  finally show dim T ≤ dim S by simp
qed
```

5.0.17 Lower-dimensional affine subsets are nowhere dense

```
proposition dense_complement_subspace:
  fixes S :: 'a :: euclidean_space set
  assumes dim_less: dim T < dim S and subspace S shows closure(S - T) = S
proof -
  have closure(S - U) = S if dim U < dim S U ⊆ S for U
  proof -
    have span U ⊂ span S
      by (metis neq_iff psubsetI span_eq_dim span_mono that)
   then obtain a where a ≠ 0 a ∈ span S and a: ⋀y. y ∈ span U ⟹ orthogonal
a y
      using orthogonal_to_subspace_exists_gen by metis
    show ?thesis
    proof
      have closed S
        by (simp add: ‹subspace S› closed_subspace)
      then show closure (S - U) ⊆ S
        by (simp add: closure_minimal)
      show S ⊆ closure (S - U)
      proof (clarsimp simp: closure_approachable)
        fix x and e::real
        assume x ∈ S 0 < e
        show ∃y∈S - U. dist y x < e
        proof (cases x ∈ U)
```

```
        case True
        let ?y = x + (e/2 / norm a) *R a
        show ?thesis
        proof
          show dist ?y x < e
            using ‹0 < e› by (simp add: dist_norm)
        next
          have ?y ∈ S
            by (metis ‹a ∈ span S› ‹x ∈ S› assms(2) span_eq_iff subspace_add
subspace_scale)
          moreover have ?y ∉ U
          proof −
            have e/2 / norm a ≠ 0
              using ‹0 < e› ‹a ≠ 0› by auto
            then show ?thesis
                  by (metis True ‹a ≠ 0› a orthogonal_scaleR orthogonal_self
real_vector.scale_eq_0_iff span_add_eq span_base)
          qed
          ultimately show ?y ∈ S − U by blast
        qed
      next
        case False
        with ‹0 < e› ‹x ∈ S› show ?thesis by force
      qed
    qed
  qed
  moreover have S − S ∩ T = S−T
    by blast
  moreover have dim (S ∩ T) < dim S
    by (metis dim_less dim_subset inf.cobounded2 inf.orderE inf.strict_boundedE
not_le)
  ultimately show ?thesis
    by force
qed

corollary dense_complement_affine:
  fixes S :: 'a :: euclidean_space set
  assumes less: aff_dim T < aff_dim S and affine S shows closure(S − T) = S
proof (cases S ∩ T = {})
  case True
  then show ?thesis
    by (metis Diff_triv affine_hull_eq ‹affine S› closure_same_affine_hull clo-
sure_subset hull_subset subset_antisym)
next
  case False
  then obtain z where z: z ∈ S ∩ T by blast
  then have subspace ((+) (− z) ‘ S)
    by (meson IntD1 affine_diffs_subspace ‹affine S›)
```

moreover have *int* (*dim* ((+) (− *z*) ‘ *T*)) < *int* (*dim* ((+) (− *z*) ‘ *S*))
thm *aff_dim_eq_dim*
using *z less* **by** (*simp add*: *aff_dim_eq_dim_subtract* [*of z*] *hull_inc cong*: *image_cong_simp*)
ultimately have *closure*(((+) (− *z*) ‘ *S*) − ((+) (− *z*) ‘ *T*)) = ((+) (− *z*) ‘ *S*)
by (*simp add*: *dense_complement_subspace*)
then show *?thesis*
by (*metis closure_translation translation_diff translation_invert*)
qed

corollary *dense_complement_openin_affine_hull*:
fixes *S* :: *'a* :: *euclidean_space set*
assumes *less*: *aff_dim T* < *aff_dim S*
and *ope*: *openin* (*top_of_set* (*affine hull S*)) *S*
shows *closure*(*S* − *T*) = *closure S*
proof −
have *affine hull S* − *T* ⊆ *affine hull S*
by *blast*
then have *closure* (*S* ∩ *closure* (*affine hull S* − *T*)) = *closure* (*S* ∩ (*affine hull S* − *T*))
by (*rule closure_openin_Int_closure* [*OF ope*])
then show *?thesis*
by (*metis Int_Diff aff_dim_affine_hull affine_affine_hull dense_complement_affine hull_subset inf.orderE less*)
qed

corollary *dense_complement_convex*:
fixes *S* :: *'a* :: *euclidean_space set*
assumes *aff_dim T* < *aff_dim S convex S*
shows *closure*(*S* − *T*) = *closure S*
proof
show *closure* (*S* − *T*) ⊆ *closure S*
by (*simp add*: *closure_mono*)
have *closure* (*rel_interior S* − *T*) = *closure* (*rel_interior S*)
by (*simp add*: *assms dense_complement_openin_affine_hull openin_rel_interior rel_interior_aff_dim rel_interior_same_affine_hull*)
then show *closure S* ⊆ *closure* (*S* − *T*)
by (*metis Diff_mono* ‹*convex S*› *closure_mono convex_closure_rel_interior order_refl rel_interior_subset*)
qed

corollary *dense_complement_convex_closed*:
fixes *S* :: *'a* :: *euclidean_space set*
assumes *aff_dim T* < *aff_dim S convex S closed S*
shows *closure*(*S* − *T*) = *S*
by (*simp add*: *assms dense_complement_convex*)

5.0.18 Parallel slices, etc

If we take a slice out of a set, we can do it perpendicularly, with the normal vector to the slice parallel to the affine hull.

proposition *affine_parallel_slice*:
fixes $S :: 'a :: euclidean\_space\ set$
assumes *affine* S
and $S \cap \{x.\ a \cdot x \leq b\} \neq \{\}$
and $\neg\ (S \subseteq \{x.\ a \cdot x \leq b\})$
obtains $a'\ b'$ **where** $a' \neq 0$
$S \cap \{x.\ a' \cdot x \leq b'\} = S \cap \{x.\ a \cdot x \leq b\}$
$S \cap \{x.\ a' \cdot x = b'\} = S \cap \{x.\ a \cdot x = b\}$
$\bigwedge w.\ w \in S \Longrightarrow (w + a') \in S$
proof (*cases* $S \cap \{x.\ a \cdot x = b\} = \{\}$)
case *True*
then obtain $u\ v$ **where** $u \in S\ v \in S\ a \cdot u \leq b\ a \cdot v > b$
using *assms* **by** (*auto simp*: *not_le*)
define η **where** $\eta = u + ((b - a \cdot u)\ /\ (a \cdot v - a \cdot u)) *_R (v - u)$
have $\eta \in S$
by (*simp add*: *η_def* ‹$u \in S$› ‹$v \in S$› ‹*affine* S› *mem_affine_3_minus*)
moreover have $a \cdot \eta = b$
using ‹$a \cdot u \leq b$› ‹$b < a \cdot v$›
by (*simp add*: *η_def algebra_simps*) (*simp add*: *field_simps*)
ultimately have *False*
using *True* **by** *force*
then show *?thesis* **..**
next
case *False*
then obtain z **where** $z \in S$ **and** z: $a \cdot z = b$
using *assms* **by** *auto*
with *affine_diffs_subspace* [*OF* ‹*affine* S›]
have *sub*: *subspace* ((+) (− z) ‘ S) **by** *blast*
then have *aff*: *affine* ((+) (− z) ‘ S) **and** *span*: *span* ((+) (− z) ‘ S) = ((+) (− z) ‘ S)
by (*auto simp*: *subspace_imp_affine*)
obtain $a'\ a''$ **where** a': $a' \in span\ ((+)\ (-\ z)\ `\ S)$ **and** a: $a = a' + a''$
and $\bigwedge w.\ w \in span\ ((+)\ (-\ z)\ `\ S) \Longrightarrow orthogonal\ a''\ w$
using *orthogonal_subspace_decomp_exists* [*of* (+) (− z) ‘ S a] **by** *metis*
then have $\bigwedge w.\ w \in S \Longrightarrow a'' \cdot (w - z) = 0$
by (*simp add*: *span_base orthogonal_def*)
then have a'': $\bigwedge w.\ w \in S \Longrightarrow a'' \cdot w = (a - a') \cdot z$
by (*simp add*: *a inner_diff_right*)
then have *ba''*: $\bigwedge w.\ w \in S \Longrightarrow a'' \cdot w = b - a' \cdot z$
by (*simp add*: *inner_diff_left z*)
show *?thesis*
proof (*cases* $a' = 0$)
case *True*
with *a assms True a'' diff_zero less_irrefl* **show** *?thesis*
by *auto*

```
  next
    case False
    show ?thesis
    proof
      show S ∩ {x. a' • x ≤ a' • z} = S ∩ {x. a • x ≤ b}
        S ∩ {x. a' • x = a' • z} = S ∩ {x. a • x = b}
        by (auto simp: a ba'' inner_left_distrib)
      have ⋀w. w ∈ (+) (− z) ` S ⟹ (w + a') ∈ (+) (− z) ` S
        by (metis subspace_add a' span_eq_iff sub)
      then show ⋀w. w ∈ S ⟹ (w + a') ∈ S
        by fastforce
    qed (use False in auto)
  qed
qed

lemma diffs_affine_hull_span:
  assumes a ∈ S
    shows (λx. x − a) ` (affine hull S) = span ((λx. x − a) ` S)
proof −
  have ∗: ((λx. x − a) ` (S − {a})) = ((λx. x − a) ` S) − {0}
    by (auto simp: algebra_simps)
  show ?thesis
    by (auto simp add: algebra_simps affine_hull_span2 [OF assms] ∗)
qed

lemma aff_dim_dim_affine_diffs:
  fixes S :: 'a :: euclidean_space set
  assumes affine S a ∈ S
    shows aff_dim S = dim ((λx. x − a) ` S)
proof −
  obtain B where aff: affine hull B = affine hull S
             and ind: ¬ affine_dependent B
             and card: of_nat (card B) = aff_dim S + 1
    using aff_dim_basis_exists by blast
  then have B ≠ {} using assms
    by (metis affine_hull_eq_empty ex_in_conv)
  then obtain c where c ∈ B by auto
  then have c ∈ S
    by (metis aff affine_hull_eq ‹affine S› hull_inc)
  have xy: x − c = y − a ⟷ y = x + 1 ∗R (a − c) for x y c and a::'a
    by (auto simp: algebra_simps)
  have ∗: (λx. x − c) ` S = (λx. x − a) ` S
    using assms ‹c ∈ S›
    by (auto simp: image_iff xy; metis mem_affine_3_minus pth_1)
  have affS: affine hull S = S
    by (simp add: ‹affine S›)
  have aff_dim S = of_nat (card B) − 1
    using card by simp
  also have ... = dim ((λx. x − c) ` B)
```

```
    using affine_independent_card_dim_diffs [OF ind ‹c ∈ B›]
    by (simp add: affine_independent_card_dim_diffs [OF ind ‹c ∈ B›])
  also have ... = dim ((λx. x − c) ‘ (affine hull B))
    by (simp add: diffs_affine_hull_span ‹c ∈ B›)
  also have ... = dim ((λx. x − a) ‘ S)
    by (simp add: affS aff ∗)
  finally show ?thesis .
qed

lemma aff_dim_linear_image_le:
  assumes linear f
    shows aff_dim(f ‘ S) ≤ aff_dim S
proof −
  have aff_dim (f ‘ T) ≤ aff_dim T if affine T for T
  proof (cases T = {})
    case True then show ?thesis by (simp add: aff_dim_geq)
  next
    case False
    then obtain a where a ∈ T by auto
    have 1: ((λx. x − f a) ‘ f ‘ T) = {x − f a |x. x ∈ f ‘ T}
      by auto
    have 2: {x − f a| x. x ∈ f ‘ T} = f ‘ ((λx. x − a) ‘ T)
      by (force simp: linear_diff [OF assms])
    have aff_dim (f ‘ T) = int (dim {x − f a |x. x ∈ f ‘ T})
       by (simp add: ‹a ∈ T› hull_inc aff_dim_eq_dim [of f a] 1 cong: im-
age_cong_simp)
    also have ... = int (dim (f ‘ ((λx. x − a) ‘ T)))
      by (force simp: linear_diff [OF assms] 2)
    also have ... ≤ int (dim ((λx. x − a) ‘ T))
      by (simp add: dim_image_le [OF assms])
    also have ... ≤ aff_dim T
      by (simp add: aff_dim_dim_affine_diffs [symmetric] ‹a ∈ T› ‹affine T›)
    finally show ?thesis .
  qed
  then
  have aff_dim (f ‘ (affine hull S)) ≤ aff_dim (affine hull S)
    using affine_affine_hull [of S] by blast
  then show ?thesis
    using affine_hull_linear_image assms linear_conv_bounded_linear by fast-
force
qed

lemma aff_dim_injective_linear_image [simp]:
  assumes linear f inj f
    shows aff_dim (f ‘ S) = aff_dim S
proof (rule antisym)
  show aff_dim (f ‘ S) ≤ aff_dim S
    by (simp add: aff_dim_linear_image_le assms(1))
next
```

```
  obtain g where linear g g ∘ f = id
    using assms(1) assms(2) linear_injective_left_inverse by blast
  then have aff_dim S ≤ aff_dim(g ‘ f ‘ S)
    by (simp add: image_comp)
  also have ... ≤ aff_dim (f ‘ S)
    by (simp add: ‹linear g› aff_dim_linear_image_le)
  finally show aff_dim S ≤ aff_dim (f ‘ S) .
qed


lemma choose_affine_subset:
  assumes affine S −1 ≤ d and dle: d ≤ aff_dim S
  obtains T where affine T T ⊆ S aff_dim T = d
proof (cases d = −1 ∨ S={})
  case True with assms show ?thesis
    by (metis aff_dim_empty affine_empty bot.extremum that eq_iff)
next
  case False
  with assms obtain a where a ∈ S 0 ≤ d by auto
  with assms have ss: subspace ((+) (− a) ‘ S)
    by (simp add: affine_diffs_subspace_subtract cong: image_cong_simp)
  have nat d ≤ dim ((+) (− a) ‘ S)
     by (metis aff_dim_subspace aff_dim_translation_eq dle nat_int nat_mono
ss)
  then obtain T where subspace T and Tsb: T ⊆ span ((+) (− a) ‘ S)
                  and Tdim: dim T = nat d
    using choose_subspace_of_subspace [of nat d (+) (− a) ‘ S] by blast
  then have affine T
    using subspace_affine by blast
  then have affine ((+) a ‘ T)
    by (metis affine_hull_eq affine_hull_translation)
  moreover have (+) a ‘ T ⊆ S
  proof −
    have T ⊆ (+) (− a) ‘ S
      by (metis (no_types) span_eq_iff Tsb ss)
    then show (+) a ‘ T ⊆ S
      using add_ac by auto
  qed
  moreover have aff_dim ((+) a ‘ T) = d
   by (simp add: aff_dim_subspace Tdim ‹0 ≤ d› ‹subspace T› aff_dim_translation_eq)
  ultimately show ?thesis
    by (rule that)
qed
```

5.0.19 Paracompactness

```
proposition paracompact:
  fixes S :: 'a :: {metric_space,second_countable_topology} set
  assumes S ⊆ ⋃𝒞 and opC: ⋀T. T ∈ 𝒞 ⟹ open T
```

```
  obtains C' where S ⊆ ⋃ C'
              and ⋀U. U ∈ C' ⟹ open U ∧ (∃ T. T ∈ C ∧ U ⊆ T)
              and ⋀x. x ∈ S
                   ⟹ ∃ V. open V ∧ x ∈ V ∧ finite {U. U ∈ C' ∧ (U ∩ V ≠ {})}
proof (cases S = {})
  case True with that show ?thesis by blast
next
  case False
  have ∃ T U. x ∈ U ∧ open U ∧ closure U ⊆ T ∧ T ∈ C if x ∈ S for x
  proof −
    obtain T where x ∈ T T ∈ C open T
      using assms ‹x ∈ S› by blast
    then obtain e where e > 0 cball x e ⊆ T
      by (force simp: open_contains_cball)
    then show ?thesis
      by (meson open_ball ‹T ∈ C› ball_subset_cball centre_in_ball closed_cball
closure_minimal dual_order.trans)
  qed
  then obtain F G where Gin: x ∈ G x and oG: open (G x)
    and clos: closure (G x) ⊆ F x and Fin: F x ∈ C
  if x ∈ S for x
    by metis
  then obtain F where F ⊆ G ‘ S countable F ⋃F = ⋃(G ‘ S)
    using Lindelof [of G ‘ S] by (metis image_iff)
  then obtain K where K: K ⊆ S countable K and eq: ⋃(G ‘ K) = ⋃(G ‘ S)
    by (metis countable_subset_image)
  with False Gin have K ≠ {} by force
  then obtain a :: nat ⇒ 'a where range a = K
    by (metis range_from_nat_into ‹countable K›)
  then have odif: ⋀n. open (F (a n) − ⋃{closure (G (a m)) |m. m < n})
    using ‹K ⊆ S› Fin opC by (fastforce simp add:)
  let ?C = range (λn. F(a n) − ⋃{closure(G(a m)) |m. m < n})
  have enum_S: ∃ n. x ∈ F(a n) ∧ x ∈ G(a n) if x ∈ S for x
  proof −
    have ∃ y ∈ K. x ∈ G y using eq that Gin by fastforce
    then show ?thesis
      using clos K ‹range a = K› closure_subset by blast
  qed
  show ?thesis
  proof
    show S ⊆ Union ?C
    proof
      fix x assume x ∈ S
      define n where n ≡ LEAST n. x ∈ F(a n)
      have n: x ∈ F(a n)
        using enum_S [OF ‹x ∈ S›] by (force simp: n_def intro: LeastI)
      have notn: x ∉ F(a m) if m < n for m
        using that not_less_Least by (force simp: n_def)
      then have x ∉ ⋃{closure (G (a m)) |m. m < n}
```

```
      using n ‹K ⊆ S› ‹range a = K› clos notn by fastforce
    with n show x ∈ Union ?C
      by blast
  qed
  show ⋀U. U ∈ ?C ⟹ open U ∧ (∃ T. T ∈ 𝒞 ∧ U ⊆ T)
    using Fin ‹K ⊆ S› ‹range a = K› by (auto simp: odif)
  show ∃ V. open V ∧ x ∈ V ∧ finite {U. U ∈ ?C ∧ (U ∩ V ≠ {})} if x ∈ S
for x
  proof −
    obtain n where n: x ∈ F(a n) x ∈ G(a n)
      using ‹x ∈ S› enum_S by auto
     have {U ∈ ?C. U ∩ G (a n) ≠ {}} ⊆ (λn. F(a n) − ⋃{closure(G(a m))
|m. m < n}) ‘ atMost n
    proof clarsimp
      fix k assume (F (a k) − ⋃{closure (G (a m)) |m. m < k}) ∩ G (a n) ≠
{}
      then have k ≤ n
        by auto (metis closure_subset not_le subsetCE)
      then show F (a k) − ⋃{closure (G (a m)) |m. m < k}
            ∈ (λn. F (a n) − ⋃{closure (G (a m)) |m. m < n}) ‘ {..n}
        by force
    qed
     moreover have finite ((λn. F(a n) − ⋃{closure(G(a m)) |m. m < n}) ‘
atMost n)
      by force
    ultimately have ∗: finite {U ∈ ?C. U ∩ G (a n) ≠ {}}
      using finite_subset by blast
    have a n ∈ S
      using ‹K ⊆ S› ‹range a = K› by blast
    then show ?thesis
      by (blast intro: oG n ∗)
  qed
 qed
qed

corollary paracompact_closedin:
 fixes S :: 'a :: {metric_space,second_countable_topology} set
 assumes cin: closedin (top_of_set U) S
    and oin: ⋀T. T ∈ 𝒞 ⟹ openin (top_of_set U) T
    and S ⊆ ⋃𝒞
 obtains 𝒞' where S ⊆ ⋃ 𝒞'
           and ⋀V. V ∈ 𝒞' ⟹ openin (top_of_set U) V ∧ (∃ T. T ∈ 𝒞 ∧ V
⊆ T)
           and ⋀x. x ∈ U
                 ⟹ ∃ V. openin (top_of_set U) V ∧ x ∈ V ∧
                      finite {X. X ∈ 𝒞' ∧ (X ∩ V ≠ {})}
proof −
 have ∃ Z. open Z ∧ (T = U ∩ Z) if T ∈ 𝒞 for T
   using oin [OF that] by (auto simp: openin_open)
```

then obtain F **where** *opF*: *open* $(F\ T)$ **and** *intF*: $U \cap F\ T = T$ **if** $T \in \mathcal{C}$ **for** T
by *metis*
obtain K **where** K: *closed* K $U \cap K = S$
using *cin* **by** (*auto simp*: *closedin_closed*)
have *1*: $U \subseteq \bigcup(\mathit{insert}\ (-\ K)\ (F\ `\ \mathcal{C}))$
by *clarsimp* (*metis Int_iff Union_iff* ‹$U \cap K = S$› ‹$S \subseteq \bigcup\mathcal{C}$› *subsetD intF*)
have *2*: $\bigwedge T.\ T \in \mathit{insert}\ (-\ K)\ (F\ `\ \mathcal{C}) \Longrightarrow \mathit{open}\ T$
using ‹*closed K*› **by** (*auto simp*: *opF*)
obtain $\mathcal{D}$ **where** $U \subseteq \bigcup\mathcal{D}$
and *D1*: $\bigwedge U.\ U \in \mathcal{D} \Longrightarrow \mathit{open}\ U \wedge (\exists T.\ T \in \mathit{insert}\ (-\ K)\ (F\ `\ \mathcal{C}) \wedge U \subseteq T)$
and *D2*: $\bigwedge x.\ x \in U \Longrightarrow \exists V.\ \mathit{open}\ V \wedge x \in V \wedge \mathit{finite}\ \{U \in \mathcal{D}.\ U \cap V \neq \{\}\}$
by (*blast intro*: *paracompact* [*OF 1 2*])
let *?C* = $\{U \cap V\ |V.\ V \in \mathcal{D} \wedge (V \cap K \neq \{\})\}$
show *?thesis*
proof (*rule_tac* $\mathcal{C}' = \{U \cap V\ |V.\ V \in \mathcal{D} \wedge (V \cap K \neq \{\})\}$ **in** *that*)
show $S \subseteq \bigcup$ *?C*
using ‹$U \cap K = S$› ‹$U \subseteq \bigcup\mathcal{D}$› K **by** (*blast dest*!: *subsetD*)
show $\bigwedge V.\ V \in$ *?C* $\Longrightarrow$ *openin* (*top_of_set U*) $V \wedge (\exists T.\ T \in \mathcal{C} \wedge V \subseteq T)$
using *D1 intF* **by** *fastforce*
have $*$: $\{X.\ (\exists V.\ X = U \cap V \wedge V \in \mathcal{D} \wedge V \cap K \neq \{\}) \wedge X \cap (U \cap V) \neq \{\}\} \subseteq$
$(\lambda x.\ U \cap x)\ `\ \{U \in \mathcal{D}.\ U \cap V \neq \{\}\}$ **for** V
by *blast*
show $\exists V.$ *openin* (*top_of_set U*) $V \wedge x \in V \wedge \mathit{finite}\ \{X \in$ *?C*$.\ X \cap V \neq \{\}\}$
if $x \in U$ **for** x
proof −
from *D2* [*OF that*] **obtain** V **where** *open* V $x \in V$ *finite* $\{U \in \mathcal{D}.\ U \cap V \neq \{\}\}$
by *auto*
with $*$ **show** *?thesis*
by (*rule_tac* x=$U \cap V$ **in** *exI*) (*auto intro*: *that finite_subset* [*OF* $*$])
qed
qed
qed

corollary *paracompact_closed*:
fixes S :: *'a* :: {*metric_space*,*second_countable_topology*} *set*
assumes *closed S*
and *opC*: $\bigwedge T.\ T \in \mathcal{C} \Longrightarrow \mathit{open}\ T$
and $S \subseteq \bigcup\mathcal{C}$
obtains $\mathcal{C}'$ **where** $S \subseteq \bigcup\mathcal{C}'$
and $\bigwedge U.\ U \in \mathcal{C}' \Longrightarrow \mathit{open}\ U \wedge (\exists T.\ T \in \mathcal{C} \wedge U \subseteq T)$
and $\bigwedge x.\ \exists V.\ \mathit{open}\ V \wedge x \in V \wedge$
$\mathit{finite}\ \{U.\ U \in \mathcal{C}' \wedge (U \cap V \neq \{\})\}$
by (*rule paracompact_closedin* [*of UNIV S* $\mathcal{C}$]) (*auto simp*: *assms*)

5.0.20 Closed-graph characterization of continuity

lemma *continuous_closed_graph_gen*:
fixes $T :: 'b::real\_normed\_vector\ set$
assumes *contf*: *continuous_on S f* **and** *fim*: $f\ `\ S \subseteq T$
shows *closedin* (*top_of_set* $(S \times T)$) $((\lambda x.\ Pair\ x\ (f\ x))\ `\ S)$
proof −
have *eq*: $((\lambda x.\ Pair\ x\ (f\ x))\ `\ S) = (S \times T \cap (\lambda z.\ (f \circ fst)z - snd\ z) -` \{0\})$
using *fim* **by** *auto*
show *?thesis*
unfolding *eq*
by (*intro continuous_intros continuous_closedin_preimage continuous_on_subset* [*OF contf*]) *auto*
qed

lemma *continuous_closed_graph_eq*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes *compact T* **and** *fim*: $f \in S \to T$
shows *continuous_on S f* $\longleftrightarrow$
closedin (*top_of_set* $(S \times T)$) $((\lambda x.\ Pair\ x\ (f\ x))\ `\ S)$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* **if** *?rhs*
proof (*clarsimp simp add*: *continuous_on_closed_gen* [*OF fim*])
fix *U*
assume *U*: *closedin* (*top_of_set T*) *U*
have *eq*: $(S \cap f -` U) = fst\ `\ (((\lambda x.\ Pair\ x\ (f\ x))\ `\ S) \cap (S \times U))$
by (*force simp*: *image_iff*)
show *closedin* (*top_of_set S*) $(S \cap f -` U)$
by (*simp add*: *U closedin_Int closedin_Times closed_map_fst* [*OF* ‹*compact T*›] *that eq*)
qed
with *continuous_closed_graph_gen assms* **show** *?thesis* **by** *blast*
qed

lemma *continuous_closed_graph*:
fixes $f :: 'a::topological\_space \Rightarrow 'b::real\_normed\_vector$
assumes *closed S* **and** *contf*: *continuous_on S f*
shows *closed* $((\lambda x.\ Pair\ x\ (f\ x))\ `\ S)$
proof (*rule closedin_closed_trans*)
show *closedin* (*top_of_set* $(S \times UNIV)$) $((\lambda x.\ (x, f\ x))\ `\ S)$
by (*rule continuous_closed_graph_gen* [*OF contf subset_UNIV*])
qed (*simp add*: ‹*closed S*› *closed_Times*)

lemma *continuous_from_closed_graph*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes *compact T* **and** *fim*: $f \in S \to T$ **and** *clo*: *closed* $((\lambda x.\ Pair\ x\ (f\ x))\ `\ S)$
shows *continuous_on S f*
using *fim clo*

by (*auto intro*: *closed_subset simp*: *continuous_closed_graph_eq* [*OF* ‹*compact T*› *fim*])

lemma *continuous_on_Un_local_open*:
assumes *opS*: *openin* (*top_of_set* (*S* ∪ *T*)) *S*
and *opT*: *openin* (*top_of_set* (*S* ∪ *T*)) *T*
and *contf*: *continuous_on S f* **and** *contg*: *continuous_on T f*
shows *continuous_on* (*S* ∪ *T*) *f*
using *pasting_lemma* [*of* {*S*,*T*} *top_of_set* (*S* ∪ *T*) *id euclidean* λ*i*. *f f*] *contf contg opS opT*
by (*simp add*: *subtopology_subtopology*) (*metis inf.absorb2 openin_imp_subset*)

lemma *continuous_on_cases_local_open*:
assumes *opS*: *openin* (*top_of_set* (*S* ∪ *T*)) *S*
and *opT*: *openin* (*top_of_set* (*S* ∪ *T*)) *T*
and *contf*: *continuous_on S f* **and** *contg*: *continuous_on T g*
and *fg*: ⋀*x*. *x* ∈ *S* ∧ ¬*P x* ∨ *x* ∈ *T* ∧ *P x* ⟹ *f x* = *g x*
shows *continuous_on* (*S* ∪ *T*) (λ*x*. *if P x then f x else g x*)
proof −
have ⋀*x*. *x* ∈ *S* ⟹ (*if P x then f x else g x*) = *f x* ⋀*x*. *x* ∈ *T* ⟹ (*if P x then f x else g x*) = *g x*
by (*simp_all add*: *fg*)
then have *continuous_on S* (λ*x*. *if P x then f x else g x*) *continuous_on T* (λ*x*. *if P x then f x else g x*)
by (*simp_all add*: *contf contg cong*: *continuous_on_cong*)
then show *?thesis*
by (*rule continuous_on_Un_local_open* [*OF opS opT*])
qed

5.0.21 The union of two collinear segments is another segment

proposition *in_convex_hull_exchange*:
fixes *a* :: *'a*::*euclidean_space*
assumes *a*: *a* ∈ *convex hull S* **and** *xS*: *x* ∈ *convex hull S*
obtains *b* **where** *b* ∈ *S x* ∈ *convex hull* (*insert a* (*S* − {*b*}))
proof (*cases a* ∈ *S*)
case *True*
with *xS insert_Diff that* **show** *?thesis* **by** *fastforce*
next
case *False*
show *?thesis*
proof (*cases finite S* ∧ *card S* ≤ *Suc* (*DIM*(*'a*)))
case *True*
then obtain *u* **where** *u0*: ⋀*i*. *i* ∈ *S* ⟹ 0 ≤ *u i* **and** *u1*: *sum u S* = 1
and *ua*: (∑ *i*∈*S*. *u i* ∗*R i*) = *a*
using *a* **by** (*auto simp*: *convex_hull_finite*)
obtain *v* **where** *v0*: ⋀*i*. *i* ∈ *S* ⟹ 0 ≤ *v i* **and** *v1*: *sum v S* = 1

```
          and vx: (∑ i∈S. v i *R i) = x
    using True xS by (auto simp: convex_hull_finite)
  show ?thesis
  proof (cases ∃ b. b ∈ S ∧ v b = 0)
    case True
    then obtain b where b: b ∈ S v b = 0
      by blast
    show ?thesis
    proof
      have fin: finite (insert a (S - {b}))
        using sum.infinite v1 by fastforce
      show x ∈ convex hull insert a (S - {b})
        unfolding convex_hull_finite [OF fin] mem_Collect_eq
      proof (intro conjI exI ballI)
        have (∑ x ∈ insert a (S - {b}). if x = a then 0 else v x) =
              (∑ x ∈ S - {b}. if x = a then 0 else v x)
          using fin by (force intro: sum.mono_neutral_right)
        also have ... = (∑ x ∈ S - {b}. v x)
          using b False by (auto intro!: sum.cong split: if_split_asm)
        also have ... = (∑ x∈S. v x)
          by (metis ‹v b = 0› diff_zero sum.infinite sum_diff1 u1 zero_neq_one)
        finally show (∑ x∈insert a (S - {b}). if x = a then 0 else v x) = 1
          by (simp add: v1)
        show ⋀x. x ∈ insert a (S - {b}) ⟹ 0 ≤ (if x = a then 0 else v x)
          by (auto simp: v0)
        have (∑ x ∈ insert a (S - {b}). (if x = a then 0 else v x) *R x) =
              (∑ x ∈ S - {b}. (if x = a then 0 else v x) *R x)
          using fin by (force intro: sum.mono_neutral_right)
        also have ... = (∑ x ∈ S - {b}. v x *R x)
          using b False by (auto intro!: sum.cong split: if_split_asm)
        also have ... = (∑ x∈S. v x *R x)
          by (metis (no_types, lifting) b(2) diff_zero fin finite.emptyI finite_Diff2
finite_insert scale_eq_0_iff sum_diff1)
        finally show (∑ x∈insert a (S - {b}). (if x = a then 0 else v x) *R x) =
x
          by (simp add: vx)
      qed
    qed (rule ‹b ∈ S›)
  next
    case False
    have le_Max: u i / v i ≤ Max ((λi. u i / v i) ‘ S) if i ∈ S for i
      by (simp add: True that)
    have Max ((λi. u i / v i) ‘ S) ∈ (λi. u i / v i) ‘ S
      using True v1 by (auto intro: Max_in)
    then obtain b where b ∈ S and beq: Max ((λb. u b / v b) ‘ S) = u b / v b
      by blast
    then have 0 ≠ u b / v b
      using le_Max beq divide_le_0_iff le_numeral_extra(2) sum_nonpos u1
      by (metis False eq_iff v0)
```

```
    then have  0 < u b 0 < v b
      using False ‹b ∈ S› u0 v0 by force+
    have fin: finite (insert a (S − {b}))
      using sum.infinite v1 by fastforce
    show ?thesis
    proof
      show x ∈ convex hull insert a (S − {b})
        unfolding convex_hull_finite [OF fin] mem_Collect_eq
      proof (intro conjI exI ballI)
        have (∑ x ∈ insert a (S − {b}). if x=a then v b / u b else v x − (v b / u
b) ∗ u x) =
                v b / u b + (∑ x ∈ S − {b}. v x − (v b / u b) ∗ u x)
          using ‹a ∉ S› ‹b ∈ S› True
          by (auto intro!: sum.cong split: if_split_asm)
        also have ... = v b / u b + (∑ x ∈ S − {b}. v x) − (v b / u b) ∗ (∑ x ∈
S − {b}. u x)
          by (simp add: Groups_Big.sum_subtractf sum_distrib_left)
        also have ... = (∑ x∈S. v x)
            using ‹0 < u b› True  by (simp add: Groups_Big.sum_diff1 u1
field_simps)
        finally show sum (λx. if x=a then v b / u b else v x − (v b / u b) ∗ u x)
(insert a (S − {b})) = 1
          by (simp add: v1)
        show 0 ≤ (if i = a then v b / u b else v i − v b / u b ∗ u i)
          if i ∈ insert a (S − {b}) for i
          using ‹0 < u b› ‹0 < v b› v0 [of i] le_Max [of i] beq that False
          by (auto simp: field_simps split: if_split_asm)
        have (∑ x∈insert a (S − {b}). (if x=a then v b / u b else v x − v b / u b
∗ u x) ∗R x) =
                (v b / u b) ∗R a + (∑ x∈S − {b}. (v x − v b / u b ∗ u x) ∗R x)
          using ‹a ∉ S› ‹b ∈ S› True  by (auto intro!: sum.cong split: if_split_asm)
          also have ... = (v b / u b) ∗R a + (∑ x ∈ S − {b}. v x ∗R x) − (v b / u
b) ∗R (∑ x ∈ S − {b}. u x ∗R x)
            by (simp add: Groups_Big.sum_subtractf scaleR_left_diff_distrib
sum_distrib_left scale_sum_right)
          also have ... = (∑ x∈S. v x ∗R x)
              using ‹0 < u b› True  by (simp add: ua vx Groups_Big.sum_diff1
algebra_simps)
          finally
          show (∑ x∈insert a (S − {b}). (if x=a then v b / u b else v x − v b / u
b ∗ u x) ∗R x) = x
            by (simp add: vx)
        qed
      qed (rule ‹b ∈ S›)
    qed
  next
    case False
    obtain T where finite T T ⊆ S and caT: card T ≤ Suc (DIM('a)) and xT:
x ∈ convex hull T
```

```
      using xS by (auto simp: caratheodory [of S])
    with False obtain b where b: b ∈ S b ∉ T
      by (metis antisym subsetI)
    show ?thesis
    proof
      show x ∈ convex hull insert a (S − {b})
        using ‹T ⊆ S› b by (blast intro: subsetD [OF hull_mono xT])
    qed (rule ‹b ∈ S›)
  qed
qed

lemma convex_hull_exchange_Union:
  fixes a :: 'a::euclidean_space
  assumes a ∈ convex hull S
  shows convex hull S = (⋃b ∈ S. convex hull (insert a (S − {b})))) (is ?lhs =
?rhs)
proof
  show ?lhs ⊆ ?rhs
    by (blast intro: in_convex_hull_exchange [OF assms])
  show ?rhs ⊆ ?lhs
  proof clarify
    fix x b
    assumeb ∈ S x ∈ convex hull insert a (S − {b})
    then show x ∈ convex hull S if b ∈ S
      by (metis (no_types) that assms order_refl hull_mono hull_redundant in-
sert_Diff_single insert_subset subsetCE)
  qed
qed

lemma Un_closed_segment:
  fixes a :: 'a::euclidean_space
  assumes b ∈ closed_segment a c
    shows closed_segment a b ∪ closed_segment b c = closed_segment a c
proof (cases c = a)
  case True
  with assms show ?thesis by simp
next
  case False
  with assms have convex hull {a, b} ∪ convex hull {b, c} = (⋃ba∈{a, c}. convex
hull insert b ({a, c} − {ba}))
    by (auto simp: insert_Diff_if insert_commute)
  then show ?thesis
    using convex_hull_exchange_Union
    by (metis assms segment_convex_hull)
qed

lemma Un_open_segment:
  fixes a :: 'a::euclidean_space
  assumes b ∈ open_segment a c
```

shows *open_segment a b* ∪ {*b*} ∪ *open_segment b c* = *open_segment a c* (**is** *?lhs* = *?rhs*)
proof −
have *b*: *b* ∈ *closed_segment a c*
by (*simp add*: *assms open_closed_segment*)
have ∗: *?rhs* ⊆ *insert b* (*open_segment a b* ∪ *open_segment b c*)
if {*b*,*c*,*a*} ∪ *open_segment a b* ∪ *open_segment b c* = {*c*,*a*} ∪ *?rhs*
proof −
have *insert a* (*insert c* (*insert b* (*open_segment a b* ∪ *open_segment b c*))) = *insert a* (*insert c* (*?rhs*))
using *that* **by** (*simp add*: *insert_commute*)
then show *?thesis*
by (*metis* (*no_types*) *Diff_cancel Diff_eq_empty_iff Diff_insert2 open_segment_def*)
qed
show *?thesis*
proof
show *?lhs* ⊆ *?rhs*
by (*simp add*: *assms b subset_open_segment*)
show *?rhs* ⊆ *?lhs*
using *Un_closed_segment* [*OF b*] ∗
by (*simp add*: *closed_segment_eq_open insert_commute*)
qed
qed

5.0.22 Covering an open set by a countable chain of compact sets

proposition *open_Union_compact_subsets*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *open S*
obtains *C* **where** ⋀*n*. *compact*(*C n*) ⋀*n*. *C n* ⊆ *S*
⋀*n*. *C n* ⊆ *interior*(*C*(*Suc n*))
⋃(*range C*) = *S*
⋀*K*. ⟦*compact K*; *K* ⊆ *S*⟧ ⟹ ∃ *N*. ∀ *n*≥*N*. *K* ⊆ (*C n*)
proof (*cases S* = {})
case *True*
then show *?thesis*
by (*rule_tac C* = λ*n*. {} **in** *that*) *auto*
next
case *False*
then obtain *a* **where** *a* ∈ *S*
by *auto*
let *?C* = λ*n*. *cball a* (*real n*) − (⋃ *x* ∈ −*S*. ⋃ *e* ∈ *ball 0* (*1* / *real*(*Suc n*)). {*x* + *e*})
have ∃ *N*. ∀ *n*≥*N*. *K* ⊆ (*f n*)
if ⋀*n*. *compact*(*f n*) **and** *sub_int*: ⋀*n*. *f n* ⊆ *interior* (*f*(*Suc n*))
and *eq*: ⋃(*range f*) = *S* **and** *compact K K* ⊆ *S* **for** *f K*
proof −
have ∗: ∀ *n*. *f n* ⊆ (⋃ *n*. *interior* (*f n*))

```
    by (meson Sup_upper2 UNIV_I ‹⋀n. f n ⊆ interior (f (Suc n))› image_iff)
  have mono: ⋀m n. m ≤ n ⟹ f m ⊆ f n
    by (meson dual_order.trans interior_subset lift_Suc_mono_le sub_int)
  obtain I where finite I and I: K ⊆ (⋃i∈I. interior (f i))
  proof (rule compactE_image [OF ‹compact K›])
    show K ⊆ (⋃n. interior (f n))
      using ‹K ⊆ S› ‹⋃(f ` UNIV) = S› * by blast
  qed auto
  { fix n
    assume n: Max I ≤ n
    have (⋃i∈I. interior (f i)) ⊆ f n
     by (rule UN_least) (meson dual_order.trans interior_subset mono I Max_ge
[OF ‹finite I›] n)
    then have K ⊆ f n
      using I by auto
  }
  then show ?thesis
    by blast
 qed
 moreover have ∃f. (∀n. compact(f n)) ∧ (∀n. (f n) ⊆ S) ∧ (∀n. (f n) ⊆
interior(f(Suc n))) ∧
                  ((⋃(range f) = S))
 proof (intro exI conjI allI)
  show ⋀n. compact (?C n)
    by (auto simp: compact_diff open_sums)
  show ⋀n. ?C n ⊆ S
    by auto
  show ?C n ⊆ interior (?C (Suc n)) for n
  proof (simp add: interior_diff, rule Diff_mono)
    show cball a (real n) ⊆ ball a (1 + real n)
      by (simp add: cball_subset_ball_iff)
    have cl: closed (⋃x∈- S. ⋃e∈cball 0 (1 / (2 + real n)). {x + e})
      using assms by (auto intro: closed_compact_sums)
    have closure (⋃x∈- S. ⋃y∈ball 0 (1 / (2 + real n)). {x + y})
        ⊆ (⋃x ∈ -S. ⋃e ∈ cball 0 (1 / (2 + real n)). {x + e})
      by (intro closure_minimal UN_mono ball_subset_cball order_refl cl)
    also have ... ⊆ (⋃x ∈ -S. ⋃y∈ball 0 (1 / (1 + real n)). {x + y})
      by (simp add: cball_subset_ball_iff field_split_simps UN_mono)
    finally show closure (⋃x∈- S. ⋃y∈ball 0 (1 / (2 + real n)). {x + y})
                 ⊆ (⋃x ∈ -S. ⋃y∈ball 0 (1 / (1 + real n)). {x + y}) .
  qed
  have S ⊆ ⋃ (range ?C)
  proof
    fix x
    assume x: x ∈ S
    then obtain e where e > 0 and e: ball x e ⊆ S
      using assms open_contains_ball by blast
    then obtain N1 where N1 > 0 and N1: real N1 > 1/e
      using reals_Archimedean2
```

by (*metis divide_less_0_iff less_eq_real_def neq0_conv not_le of_nat_0 of_nat_1 of_nat_less_0_iff*)
obtain *N2* **where** *N2*: $norm(x - a) \leq real\ N2$
by (*meson real_arch_simple*)
have *N12*: $inverse((N1 + N2) + 1) \leq inverse(N1)$
using ‹$N1 > 0$› **by** (*auto simp*: *field_split_simps*)
have $x \neq y + z$ **if** $y \notin S$ $norm\ z < 1 / (1 + (real\ N1 + real\ N2))$ **for** $y\ z$
proof −
have $e * real\ N1 < e * (1 + (real\ N1 + real\ N2))$
by (*simp add*: ‹$0 < e$›)
then have $1 / (1 + (real\ N1 + real\ N2)) < e$
using *N1* ‹$e > 0$›
by (*metis divide_less_eq less_trans mult.commute of_nat_add of_nat_less_0_iff of_nat_Suc*)
then have $x - z \in ball\ x\ e$
using *that* **by** *simp*
then have $x - z \in S$
using *e* **by** *blast*
with *that* **show** *?thesis*
by *auto*
qed
with *N2* **show** $x \in \bigcup (range\ ?C)$
by (*rule_tac a = N1+N2* **in** *UN_I*) (*auto simp*: *dist_norm norm_minus_commute*)
qed
then show $\bigcup (range\ ?C) = S$ **by** *auto*
qed
ultimately show *?thesis*
using *that* **by** *metis*
qed

5.0.23 Orthogonal complement

definition *orthogonal_comp* (_$^\perp$ [*80*] *80*)
where *orthogonal_comp W* ≡ {*x*. ∀ *y* ∈ *W*. *orthogonal y x*}

proposition *subspace_orthogonal_comp*: *subspace* ($W^\perp$)
unfolding *subspace_def orthogonal_comp_def orthogonal_def*
by (*auto simp*: *inner_right_distrib*)

lemma *orthogonal_comp_anti_mono*:
assumes $A \subseteq B$
shows $B^\perp \subseteq A^\perp$
proof
fix *x* **assume** *x*: $x \in B^\perp$
show $x \in orthogonal\_comp\ A$ **using** *x* **unfolding** *orthogonal_comp_def*
by (*simp add*: *orthogonal_def*, *metis assms in_mono*)
qed

lemma *orthogonal_comp_null* [*simp*]: $\{0\}^\perp = UNIV$

by (*auto simp*: *orthogonal_comp_def orthogonal_def*)

lemma *orthogonal_comp_UNIV* [*simp*]: $UNIV^\perp = \{0\}$
unfolding *orthogonal_comp_def orthogonal_def*
by *auto* (*use inner_eq_zero_iff* **in** *blast*)

lemma *orthogonal_comp_subset*: $U \subseteq U^{\perp\perp}$
by (*auto simp*: *orthogonal_comp_def orthogonal_def inner_commute*)

lemma *subspace_sum_minimal*:
assumes $S \subseteq U$ $T \subseteq U$ *subspace U*
shows $S + T \subseteq U$
proof
fix *x*
assume $x \in S + T$
then obtain *xs xt* **where** $xs \in S$ $xt \in T$ $x = xs+xt$
by (*meson set_plus_elim*)
then show $x \in U$
by (*meson assms subsetCE subspace_add*)
qed

proposition *subspace_sum_orthogonal_comp*:
fixes $U :: 'a :: euclidean\_space\ set$
assumes *subspace U*
shows $U + U^\perp = UNIV$
proof −
obtain *B* **where** $B \subseteq U$
and *ortho*: *pairwise orthogonal B* $\bigwedge x.\ x \in B \Longrightarrow norm\ x = 1$
and *independent B card B* = *dim U span B* = *U*
using *orthonormal_basis_subspace* [*OF assms*] **by** *metis*
then have *finite B*
by (*simp add*: *indep_card_eq_dim_span*)
have $*$: $\forall x{\in}B.\ \forall y{\in}B.\ x \cdot y = (if\ x{=}y\ then\ 1\ else\ 0)$
using *ortho norm_eq_1* **by** (*auto simp*: *orthogonal_def pairwise_def*)
{ fix *v*
let *?u* = $\sum b{\in}B.\ (v \cdot b) *_R b$
have $v = ?u + (v - ?u)$
by *simp*
moreover have $?u \in U$
by (*metis* (*no_types*, *lifting*) ‹*span B* = *U*› *assms subspace_sum span_base span_mul*)
moreover have $(v - ?u) \in U^\perp$
proof (*clarsimp simp*: *orthogonal_comp_def orthogonal_def*)
fix *y*
assume $y \in U$
with ‹*span B* = *U*› *span_finite* [*OF* ‹*finite B*›]
obtain *u* **where** *u*: $y = (\sum b{\in}B.\ u\ b *_R b)$
by *auto*
have $b \cdot (v - ?u) = 0$ **if** $b \in B$ **for** *b*

```
      using that ‹finite B›
      by (simp add: ∗ algebra_simps inner_sum_right if_distrib [of (∗)v for v]
inner_commute cong: if_cong)
    then show y · (v − ?u) = 0
      by (simp add: u inner_sum_left)
  qed
  ultimately have v ∈ U + U^⊥
    using set_plus_intro by fastforce
 } then show ?thesis
  by auto
qed

lemma orthogonal_Int_0:
 assumes subspace U
 shows U ∩ U^⊥ = {0}
 using orthogonal_comp_def orthogonal_self
 by (force simp: assms subspace_0 subspace_orthogonal_comp)

lemma orthogonal_comp_self:
 fixes U :: 'a :: euclidean_space set
 assumes subspace U
 shows U^⊥⊥ = U
proof
 have ssU': subspace (U^⊥)
  by (simp add: subspace_orthogonal_comp)
 have u ∈ U if u ∈ U^⊥⊥ for u
 proof −
  obtain v w where u = v+w v ∈ U w ∈ U^⊥
    using subspace_sum_orthogonal_comp [OF assms] set_plus_elim by blast
  then have u−v ∈ U^⊥
    by simp
  moreover have v ∈ U^⊥⊥
    using ‹v ∈ U› orthogonal_comp_subset by blast
  then have u−v ∈ U^⊥⊥
    by (simp add: subspace_diff subspace_orthogonal_comp that)
  ultimately have u−v = 0
    using orthogonal_Int_0 ssU' by blast
  with ‹v ∈ U› show ?thesis
    by auto
 qed
 then show U^⊥⊥ ⊆ U
  by auto
qed (use orthogonal_comp_subset in auto)

lemma ker_orthogonal_comp_adjoint:
 fixes f :: 'm::euclidean_space ⇒ 'n::euclidean_space
 assumes linear f
 shows f −‘ {0} = (range (adjoint f))^⊥
proof −
```

have $\bigwedge x.\ [\![\forall\, y.\ y \cdot f\, x = 0]\!] \Longrightarrow f\, x = 0$
using *assms inner_commute all_zero_iff* **by** *metis*
then show *?thesis*
using *assms*
by (*auto simp*: *orthogonal_comp_def orthogonal_def adjoint_works inner_commute*)
qed

5.0.24 A non-injective linear function maps into a hyperplane.

lemma *linear_surj_adj_imp_inj*:
fixes f :: $'m$::*euclidean_space* ⇒ $'n$::*euclidean_space*
assumes *linear f surj* (*adjoint f*)
shows *inj f*
proof −
have $\exists\, x.\ y = adjoint\ f\ x$ **for** y
using *assms* **by** (*simp add*: *surjD*)
then show *inj f*
using *assms* **unfolding** *inj_on_def image_def*
by (*metis* (*no_types*) *adjoint_works euclidean_eqI*)
qed

— https://mathonline.wikidot.com/injectivity-and-surjectivity-of-the-adjoint-of-a-linear-map
lemma *surj_adjoint_iff_inj* [*simp*]:
fixes f :: $'m$::*euclidean_space* ⇒ $'n$::*euclidean_space*
assumes *linear f*
shows *surj* (*adjoint f*) ⟷ *inj f*
proof
assume *surj* (*adjoint f*)
then show *inj f*
by (*simp add*: *assms linear_surj_adj_imp_inj*)
next
assume *inj f*
have $f -` \{0\} = \{0\}$
using *assms* ‹*inj f*› *linear_0 linear_injective_0* **by** *fastforce*
moreover have $f -` \{0\} = range\ (adjoint\ f)^{\perp}$
by (*intro ker_orthogonal_comp_adjoint assms*)
ultimately have $range\ (adjoint\ f)^{\perp\perp} = UNIV$
by (*metis orthogonal_comp_null*)
then show *surj* (*adjoint f*)
using *adjoint_linear* ‹*linear f*›
by (*subst* (*asm*) *orthogonal_comp_self*)
(*simp add*: *adjoint_linear linear_subspace_image*)
qed

lemma *inj_adjoint_iff_surj* [*simp*]:
fixes f :: $'m$::*euclidean_space* ⇒ $'n$::*euclidean_space*
assumes *linear f*
shows *inj* (*adjoint f*) ⟷ *surj f*

proof
assume *inj* (*adjoint f*)
have (*adjoint f*) −‘ {*0*} = {*0*}
by (*metis* ‹*inj* (*adjoint f*)› *adjoint_linear assms surj_adjoint_iff_inj ker_orthogonal_comp_adjoint orthogonal_comp_UNIV*)
then have (*range*(*f*))$^{\perp}$ = {*0*}
by (*metis* (*no_types*, *opaque_lifting*) *adjoint_adjoint adjoint_linear assms ker_orthogonal_comp_adjoint set_zero*)
then show *surj f*
by (*metis* ‹*inj* (*adjoint f*)› *adjoint_adjoint adjoint_linear assms surj_adjoint_iff_inj*)
next
assume *surj f*
then have *range f* = (*adjoint f* −‘ {*0*})$^{\perp}$
by (*simp add*: *adjoint_adjoint adjoint_linear assms ker_orthogonal_comp_adjoint*)
then have {*0*} = *adjoint f* −‘ {*0*}
using ‹*surj f*› *adjoint_adjoint adjoint_linear assms ker_orthogonal_comp_adjoint*
by *force*
then show *inj* (*adjoint f*)
by (*simp add*: ‹*surj f*› *adjoint_adjoint adjoint_linear assms linear_surj_adj_imp_inj*)
qed

lemma *linear_singular_into_hyperplane*:
fixes *f* :: *'n*::*euclidean_space* ⇒ *'n*
assumes *linear f*
shows ¬ *inj f* ⟷ (∃ *a*. *a* ≠ *0* ∧ (∀ *x*. *a* ∙ *f x* = *0*)) (**is** _ = *?rhs*)
proof
assume ¬*inj f*
then show *?rhs*
using *all_zero_iff*
by (*metis* (*no_types*, *opaque_lifting*) *adjoint_clauses*(*2*) *adjoint_linear assms linear_injective_0 linear_injective_imp_surjective linear_surj_adj_imp_inj*)
next
assume *?rhs*
then show ¬*inj f*
by (*metis assms linear_injective_isomorphism all_zero_iff*)
qed

lemma *linear_singular_image_hyperplane*:
fixes *f* :: *'n*::*euclidean_space* ⇒ *'n*
assumes *linear f* ¬*inj f*
obtains *a* **where** *a* ≠ *0* ⋀*S*. *f* ‘ *S* ⊆ {*x*. *a* ∙ *x* = *0*}
using *assms* **by** (*fastforce simp add*: *linear_singular_into_hyperplane*)

end

5.1 Path-Connectedness

theory *Path_Connected*
imports

Starlike
T1_Spaces
begin

5.1.1 Paths and Arcs

definition *path* :: (*real* ⇒ *'a*::*topological_space*) ⇒ *bool*
where *path g* ≡ *continuous_on* {*0..1*} *g*

definition *pathstart* :: (*real* ⇒ *'a*::*topological_space*) ⇒ *'a*
where *pathstart g* ≡ *g 0*

definition *pathfinish* :: (*real* ⇒ *'a*::*topological_space*) ⇒ *'a*
where *pathfinish g* ≡ *g 1*

definition *path_image* :: (*real* ⇒ *'a*::*topological_space*) ⇒ *'a set*
where *path_image g* ≡ *g* ‘ {*0 .. 1*}

definition *reversepath* :: (*real* ⇒ *'a*::*topological_space*) ⇒ *real* ⇒ *'a*
where *reversepath g* ≡ (λ*x*. *g*(*1* − *x*))

definition *joinpaths* :: (*real* ⇒ *'a*::*topological_space*) ⇒ (*real* ⇒ *'a*) ⇒ *real* ⇒ *'a*
(**infixr** *+++* *75*)
where *g1 +++ g2* ≡ (λ*x*. *if x* ≤ *1/2 then g1* (*2* ∗ *x*) *else g2* (*2* ∗ *x* − *1*))

definition *loop_free* :: (*real* ⇒ *'a*::*topological_space*) ⇒ *bool*
where *loop_free g* ≡ ∀ *x*∈{*0..1*}. ∀ *y*∈{*0..1*}. *g x* = *g y* ⟶ *x* = *y* ∨ *x* = *0* ∧ *y* = *1* ∨ *x* = *1* ∧ *y* = *0*

definition *simple_path* :: (*real* ⇒ *'a*::*topological_space*) ⇒ *bool*
where *simple_path g* ≡ *path g* ∧ *loop_free g*

definition *arc* :: (*real* ⇒ *'a* :: *topological_space*) ⇒ *bool*
where *arc g* ≡ *path g* ∧ *inj_on g* {*0..1*}

5.1.2 Invariance theorems

lemma *path_eq*: *path p* ⟹ (⋀*t*. *t* ∈ {*0..1*} ⟹ *p t* = *q t*) ⟹ *path q*
using *continuous_on_eq path_def* **by** *blast*

lemma *path_continuous_image*: *path g* ⟹ *continuous_on* (*path_image g*) *f* ⟹ *path*(*f* ∘ *g*)
unfolding *path_def path_image_def*
using *continuous_on_compose* **by** *blast*

lemma *continuous_on_translation_eq*:
fixes *g* :: *'a* :: *real_normed_vector* ⇒ *'b* :: *real_normed_vector*
shows *continuous_on A* ((+) *a* ∘ *g*) = *continuous_on A g*
proof −
have *g*: *g* = (λ*x*. −*a* + *x*) ∘ ((λ*x*. *a* + *x*) ∘ *g*)

by (*rule ext*) *simp*
show *?thesis*
by (*metis* (*no_types*, *opaque_lifting*) *g continuous_on_compose homeomorphism_def homeomorphism_translation*)
qed

lemma *path_translation_eq*:
fixes *g* :: *real* ⇒ *'a* :: *real_normed_vector*
shows *path*((λ*x*. *a* + *x*) ∘ *g*) = *path g*
using *continuous_on_translation_eq path_def* **by** *blast*

lemma *path_linear_image_eq*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *linear f inj f*
shows *path*(*f* ∘ *g*) = *path g*
proof −
from *linear_injective_left_inverse* [*OF assms*]
obtain *h* **where** *h*: *linear h h* ∘ *f* = *id*
by *blast*
with *assms* **show** *?thesis*
by (*metis comp_assoc id_comp linear_continuous_on linear_linear path_continuous_image*)
qed

lemma *pathstart_translation*: *pathstart*((λ*x*. *a* + *x*) ∘ *g*) = *a* + *pathstart g*
by (*simp add*: *pathstart_def*)

lemma *pathstart_linear_image_eq*: *linear f* ⟹ *pathstart*(*f* ∘ *g*) = *f*(*pathstart g*)
by (*simp add*: *pathstart_def*)

lemma *pathfinish_translation*: *pathfinish*((λ*x*. *a* + *x*) ∘ *g*) = *a* + *pathfinish g*
by (*simp add*: *pathfinish_def*)

lemma *pathfinish_linear_image*: *linear f* ⟹ *pathfinish*(*f* ∘ *g*) = *f*(*pathfinish g*)
by (*simp add*: *pathfinish_def*)

lemma *path_image_translation*: *path_image*((λ*x*. *a* + *x*) ∘ *g*) = (λ*x*. *a* + *x*) ‘ (*path_image g*)
by (*simp add*: *image_comp path_image_def*)

lemma *path_image_linear_image*: *linear f* ⟹ *path_image*(*f* ∘ *g*) = *f* ‘ (*path_image g*)
by (*simp add*: *image_comp path_image_def*)

lemma *reversepath_translation*: *reversepath*((λ*x*. *a* + *x*) ∘ *g*) = (λ*x*. *a* + *x*) ∘ *reversepath g*
by (*rule ext*) (*simp add*: *reversepath_def*)

lemma *reversepath_linear_image*: *linear f* ⟹ *reversepath*(*f* ∘ *g*) = *f* ∘ *reversepath g*

by (*rule ext*) (*simp add*: *reversepath_def*)

lemma *joinpaths_translation*:
((λ*x*. *a* + *x*) ∘ *g1*) +++ ((λ*x*. *a* + *x*) ∘ *g2*) = (λ*x*. *a* + *x*) ∘ (*g1* +++ *g2*)
by (*rule ext*) (*simp add*: *joinpaths_def*)

lemma *joinpaths_linear_image*: *linear f* ⟹ (*f* ∘ *g1*) +++ (*f* ∘ *g2*) = *f* ∘ (*g1* +++ *g2*)
by (*rule ext*) (*simp add*: *joinpaths_def*)

lemma *loop_free_translation_eq*:
fixes *g* :: *real* ⇒ *'a*::*euclidean_space*
shows *loop_free*((λ*x*. *a* + *x*) ∘ *g*) = *loop_free g*
by (*simp add*: *loop_free_def*)

lemma *simple_path_translation_eq*:
fixes *g* :: *real* ⇒ *'a*::*euclidean_space*
shows *simple_path*((λ*x*. *a* + *x*) ∘ *g*) = *simple_path g*
by (*simp add*: *simple_path_def loop_free_translation_eq path_translation_eq*)

lemma *loop_free_linear_image_eq*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *linear f inj f*
shows *loop_free*(*f* ∘ *g*) = *loop_free g*
using *assms inj_on_eq_iff* [*of f*] **by** (*auto simp*: *loop_free_def*)

lemma *simple_path_linear_image_eq*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *linear f inj f*
shows *simple_path*(*f* ∘ *g*) = *simple_path g*
using *assms*
by (*simp add*: *loop_free_linear_image_eq path_linear_image_eq simple_path_def*)

lemma *arc_translation_eq*:
fixes *g* :: *real* ⇒ *'a*::*euclidean_space*
shows *arc*((λ*x*. *a* + *x*) ∘ *g*) ⟷ *arc g*
by (*auto simp*: *arc_def inj_on_def path_translation_eq*)

lemma *arc_linear_image_eq*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *linear f inj f*
shows *arc*(*f* ∘ *g*) = *arc g*
using *assms inj_on_eq_iff* [*of f*]
by (*auto simp*: *arc_def inj_on_def path_linear_image_eq*)

5.1.3 Basic lemmas about paths

lemma *path_of_real*: *path complex_of_real*
unfolding *path_def* **by** (*intro continuous_intros*)

lemma *path_const*: *path* (λ*t*. *a*) **for** *a*::′*a*::*real_normed_vector*
unfolding *path_def* **by** (*intro continuous_intros*)

lemma *path_minus*: *path g* ⟹ *path* (λ*t*. − *g t*) **for** *g*::*real*⇒′*a*::*real_normed_vector*
unfolding *path_def* **by** (*intro continuous_intros*)

lemma *path_add*: ⟦*path f*; *path g*⟧ ⟹ *path* (λ*t*. *f t* + *g t*) **for** *f*::*real*⇒′*a*::*real_normed_vector*
unfolding *path_def* **by** (*intro continuous_intros*)

lemma *path_diff*: ⟦*path f*; *path g*⟧ ⟹ *path* (λ*t*. *f t* − *g t*) **for** *f*::*real*⇒′*a*::*real_normed_vector*
unfolding *path_def* **by** (*intro continuous_intros*)

lemma *path_mult*: ⟦*path f*; *path g*⟧ ⟹ *path* (λ*t*. *f t* ∗ *g t*) **for** *f*::*real*⇒′*a*::*real_normed_field*
unfolding *path_def* **by** (*intro continuous_intros*)

lemma *pathin_iff_path_real* [*simp*]: *pathin euclideanreal g* ⟷ *path g*
by (*simp add*: *pathin_def path_def*)

lemma *continuous_on_path*: *path f* ⟹ *t* ⊆ {*0..1*} ⟹ *continuous_on t f*
using *continuous_on_subset path_def* **by** *blast*

lemma *inj_on_imp_loop_free*: *inj_on g* {*0..1*} ⟹ *loop_free g*
by (*simp add*: *inj_onD loop_free_def*)

lemma *arc_imp_simple_path*: *arc g* ⟹ *simple_path g*
by (*simp add*: *arc_def inj_on_imp_loop_free simple_path_def*)

lemma *arc_imp_path*: *arc g* ⟹ *path g*
using *arc_def* **by** *blast*

lemma *arc_imp_inj_on*: *arc g* ⟹ *inj_on g* {*0..1*}
by (*auto simp*: *arc_def*)

lemma *simple_path_imp_path*: *simple_path g* ⟹ *path g*
using *simple_path_def* **by** *blast*

lemma *loop_free_cases*: *loop_free g* ⟹ *inj_on g* {*0..1*} ∨ *pathfinish g* = *pathstart g*
by (*force simp*: *inj_on_def loop_free_def pathfinish_def pathstart_def*)

lemma *simple_path_cases*: *simple_path g* ⟹ *arc g* ∨ *pathfinish g* = *pathstart g*
using *arc_def loop_free_cases simple_path_def* **by** *blast*

lemma *simple_path_imp_arc*: *simple_path g* ⟹ *pathfinish g* ≠ *pathstart g* ⟹ *arc g*
using *simple_path_cases* **by** *auto*

lemma *arc_distinct_ends*: *arc g* ⟹ *pathfinish g* ≠ *pathstart g*

unfolding *arc_def inj_on_def pathfinish_def pathstart_def*
by *fastforce*

lemma *arc_simple_path*: *arc g* ⟷ *simple_path g* ∧ *pathfinish g* ≠ *pathstart g*
using *arc_distinct_ends arc_imp_simple_path simple_path_cases* **by** *blast*

lemma *simple_path_eq_arc*: *pathfinish g* ≠ *pathstart g* ⟹ (*simple_path g* = *arc g*)
by (*simp add*: *arc_simple_path*)

lemma *path_image_const* [*simp*]: *path_image* (λ*t*. *a*) = {*a*}
by (*force simp*: *path_image_def*)

lemma *path_image_nonempty* [*simp*]: *path_image g* ≠ {}
unfolding *path_image_def image_is_empty box_eq_empty*
by *auto*

lemma *pathstart_in_path_image*[*intro*]: *pathstart g* ∈ *path_image g*
unfolding *pathstart_def path_image_def*
by *auto*

lemma *pathfinish_in_path_image*[*intro*]: *pathfinish g* ∈ *path_image g*
unfolding *pathfinish_def path_image_def*
by *auto*

lemma *connected_path_image*[*intro*]: *path g* ⟹ *connected* (*path_image g*)
unfolding *path_def path_image_def*
using *connected_continuous_image connected_Icc* **by** *blast*

lemma *compact_path_image*[*intro*]: *path g* ⟹ *compact* (*path_image g*)
unfolding *path_def path_image_def*
using *compact_continuous_image connected_Icc* **by** *blast*

lemma *reversepath_reversepath*[*simp*]: *reversepath* (*reversepath g*) = *g*
unfolding *reversepath_def*
by *auto*

lemma *pathstart_reversepath*[*simp*]: *pathstart* (*reversepath g*) = *pathfinish g*
unfolding *pathstart_def reversepath_def pathfinish_def*
by *auto*

lemma *pathfinish_reversepath*[*simp*]: *pathfinish* (*reversepath g*) = *pathstart g*
unfolding *pathstart_def reversepath_def pathfinish_def*
by *auto*

lemma *reversepath_o*: *reversepath g* = *g* ∘ (−)*1*
by (*auto simp*: *reversepath_def*)

lemma *pathstart_join*[*simp*]: *pathstart* (*g1* +++ *g2*) = *pathstart g1*

unfolding *pathstart_def joinpaths_def pathfinish_def*
by *auto*

lemma *pathfinish_join*[*simp*]: *pathfinish* (*g1* +++ *g2*) = *pathfinish g2*
unfolding *pathstart_def joinpaths_def pathfinish_def*
by *auto*

lemma *path_image_reversepath*[*simp*]: *path_image* (*reversepath g*) = *path_image g*
proof −
have ∗: ⋀*g*. *path_image* (*reversepath g*) ⊆ *path_image g*
unfolding *path_image_def subset_eq reversepath_def Ball_def image_iff*
by *force*
show *?thesis*
using ∗[*of g*] ∗[*of reversepath g*]
unfolding *reversepath_reversepath*
by *auto*
qed

lemma *path_reversepath* [*simp*]: *path* (*reversepath g*) ⟷ *path g*
proof −
have ∗: ⋀*g*. *path g* ⟹ *path* (*reversepath g*)
by (*metis cancel_comm_monoid_add_class.diff_cancel continuous_on_compose*

continuous_on_op_minus diff_zero image_diff_atLeastAtMost path_def reversepath_o)
then show *?thesis* **by** *force*
qed

lemma *arc_reversepath*:
assumes *arc g* **shows** *arc*(*reversepath g*)
proof −
have *injg*: *inj_on g* {*0..1*}
using *assms*
by (*simp add*: *arc_def*)
have ∗∗: ⋀*x y*::*real*. *1*−*x* = *1*−*y* ⟹ *x* = *y*
by *simp*
show *?thesis*
using *assms* **by** (*clarsimp simp*: *arc_def intro*!: *inj_onI*) (*simp add*: *inj_onD reversepath_def* ∗∗)
qed

lemma *loop_free_reversepath*:
assumes *loop_free g* **shows** *loop_free*(*reversepath g*)
using *assms* **by** (*simp add*: *reversepath_def loop_free_def Ball_def*) (*smt* (*verit*))

lemma *simple_path_reversepath*: *simple_path g* ⟹ *simple_path* (*reversepath g*)
by (*simp add*: *loop_free_reversepath simple_path_def*)

lemmas *reversepath_simps* =
path_reversepath path_image_reversepath pathstart_reversepath pathfinish_reversepath

lemma *path_join*[*simp*]:
assumes *pathfinish g1 = pathstart g2*
shows *path* (*g1 +++ g2*) ⟷ *path g1* ∧ *path g2*
unfolding *path_def pathfinish_def pathstart_def*
proof *safe*
assume *cont*: *continuous_on* {*0..1*} (*g1 +++ g2*)
have *g1*: *continuous_on* {*0..1*} *g1* ⟷ *continuous_on* {*0..1*} ((*g1 +++ g2*) ∘ (λ*x*. *x* / *2*))
by (*intro continuous_on_cong refl*) (*auto simp*: *joinpaths_def*)
have *g2*: *continuous_on* {*0..1*} *g2* ⟷ *continuous_on* {*0..1*} ((*g1 +++ g2*) ∘ (λ*x*. *x* / *2* + *1/2*))
using *assms*
by (*intro continuous_on_cong refl*) (*auto simp*: *joinpaths_def pathfinish_def pathstart_def*)
show *continuous_on* {*0..1*} *g1* **and** *continuous_on* {*0..1*} *g2*
unfolding *g1 g2*
by (*auto intro*!: *continuous_intros continuous_on_subset*[*OF cont*] *simp del*: *o_apply*)
next
assume *g1g2*: *continuous_on* {*0..1*} *g1 continuous_on* {*0..1*} *g2*
have *01*: {*0 .. 1*} = {*0..1/2*} ∪ {*1/2 .. 1*::*real*}
by *auto*
{
fix *x* :: *real*
assume *0* ≤ *x* **and** *x* ≤ *1*
then have *x* ∈ (λ*x*. *x* ∗ *2*) ‘ {*0..1 / 2*}
by (*intro image_eqI*[**where** *x*=*x/2*]) *auto*
}
note *1* = *this*
{
fix *x* :: *real*
assume *0* ≤ *x* **and** *x* ≤ *1*
then have *x* ∈ (λ*x*. *x* ∗ *2* − *1*) ‘ {*1 / 2..1*}
by (*intro image_eqI*[**where** *x*=*x/2* + *1/2*]) *auto*
}
note *2* = *this*
show *continuous_on* {*0..1*} (*g1 +++ g2*)
using *assms*
unfolding *joinpaths_def 01*
apply (*intro continuous_on_cases closed_atLeastAtMost g1g2*[*THEN continuous_on_compose2*] *continuous_intros*)
apply (*auto simp*: *field_simps pathfinish_def pathstart_def intro*!: *1 2*)
done
qed

5.1.4 Path Images

lemma *bounded_path_image*: *path g* $\Longrightarrow$ *bounded*(*path_image g*)
by (*simp add*: *compact_imp_bounded compact_path_image*)

lemma *closed_path_image*:
fixes *g* :: *real* $\Rightarrow$ *'a*::*t2_space*
shows *path g* $\Longrightarrow$ *closed*(*path_image g*)
by (*metis compact_path_image compact_imp_closed*)

lemma *connected_simple_path_image*: *simple_path g* $\Longrightarrow$ *connected*(*path_image g*)
by (*metis connected_path_image simple_path_imp_path*)

lemma *compact_simple_path_image*: *simple_path g* $\Longrightarrow$ *compact*(*path_image g*)
by (*metis compact_path_image simple_path_imp_path*)

lemma *bounded_simple_path_image*: *simple_path g* $\Longrightarrow$ *bounded*(*path_image g*)
by (*metis bounded_path_image simple_path_imp_path*)

lemma *closed_simple_path_image*:
fixes *g* :: *real* $\Rightarrow$ *'a*::*t2_space*
shows *simple_path g* $\Longrightarrow$ *closed*(*path_image g*)
by (*metis closed_path_image simple_path_imp_path*)

lemma *connected_arc_image*: *arc g* $\Longrightarrow$ *connected*(*path_image g*)
by (*metis connected_path_image arc_imp_path*)

lemma *compact_arc_image*: *arc g* $\Longrightarrow$ *compact*(*path_image g*)
by (*metis compact_path_image arc_imp_path*)

lemma *bounded_arc_image*: *arc g* $\Longrightarrow$ *bounded*(*path_image g*)
by (*metis bounded_path_image arc_imp_path*)

lemma *closed_arc_image*:
fixes *g* :: *real* $\Rightarrow$ *'a*::*t2_space*
shows *arc g* $\Longrightarrow$ *closed*(*path_image g*)
by (*metis closed_path_image arc_imp_path*)

lemma *path_image_join_subset*: *path_image* (*g1* +++ *g2*) $\subseteq$ *path_image g1* $\cup$ *path_image g2*
unfolding *path_image_def joinpaths_def*
by *auto*

lemma *subset_path_image_join*:
assumes *path_image g1* $\subseteq$ *s* **and** *path_image g2* $\subseteq$ *s*
shows *path_image* (*g1* +++ *g2*) $\subseteq$ *s*
using *path_image_join_subset*[*of g1 g2*] **and** *assms*
by *auto*

lemma *path_image_join*:
assumes *pathfinish g1 = pathstart g2*
shows *path_image(g1 +++ g2) = path_image g1 ∪ path_image g2*
proof −
have *path_image g1 ⊆ path_image (g1 +++ g2)*
proof (*clarsimp simp*: *path_image_def joinpaths_def*)
fix *u*::*real*
assume *0 ≤ u u ≤ 1*
then show *g1 u ∈ (λx. g1 (2 ∗ x)) ‘ ({0..1} ∩ {x. x ∗ 2 ≤ 1})*
by (*rule_tac x*=*u*/*2* **in** *image_eqI*) *auto*
qed
moreover
have §: *g2 u ∈ (λx. g2 (2 ∗ x − 1)) ‘ ({0..1} ∩ {x. ¬ x ∗ 2 ≤ 1})*
if *0 < u u ≤ 1* **for** *u*
using *that assms*
by (*rule_tac x*=(*u*+*1*)/*2* **in** *image_eqI*) (*auto simp*: *field_simps pathfinish_def pathstart_def*)
have *g2 0 ∈ (λx. g1 (2 ∗ x)) ‘ ({0..1} ∩ {x. x ∗ 2 ≤ 1})*
using *assms*
by (*rule_tac x*=*1*/*2* **in** *image_eqI*) (*auto simp*: *pathfinish_def pathstart_def*)
then have *path_image g2 ⊆ path_image (g1 +++ g2)*
by (*auto simp*: *path_image_def joinpaths_def intro*!: §)
ultimately show *?thesis*
using *path_image_join_subset* **by** *blast*
qed

lemma *not_in_path_image_join*:
assumes *x ∉ path_image g1* **and** *x ∉ path_image g2*
shows *x ∉ path_image (g1 +++ g2)*
using *assms* **and** *path_image_join_subset*[*of g1 g2*]
by *auto*

lemma *pathstart_compose*: *pathstart*(*f ∘ p*) = *f*(*pathstart p*)
by (*simp add*: *pathstart_def*)

lemma *pathfinish_compose*: *pathfinish*(*f ∘ p*) = *f*(*pathfinish p*)
by (*simp add*: *pathfinish_def*)

lemma *path_image_compose*: *path_image* (*f ∘ p*) = *f ‘* (*path_image p*)
by (*simp add*: *image_comp path_image_def*)

lemma *path_compose_join*: *f ∘* (*p +++ q*) = (*f ∘ p*) *+++* (*f ∘ q*)
by (*rule ext*) (*simp add*: *joinpaths_def*)

lemma *path_compose_reversepath*: *f ∘ reversepath p* = *reversepath*(*f ∘ p*)
by (*rule ext*) (*simp add*: *reversepath_def*)

lemma *joinpaths_eq*:
(⋀*t*. *t ∈ {0..1}* ⟹ *p t* = *p*′ *t*) ⟹

$(\bigwedge t.\ t \in \{0..1\} \Longrightarrow q\ t = q'\ t)$
$\Longrightarrow\ t \in \{0..1\} \Longrightarrow (p +++ q)\ t = (p' +++ q')\ t$
by (*auto simp*: *joinpaths_def*)

lemma *loop_free_inj_on*: *loop_free g* $\Longrightarrow$ *inj_on g* $\{0<..<1\}$
by (*force simp*: *inj_on_def loop_free_def*)

lemma *simple_path_inj_on*: *simple_path g* $\Longrightarrow$ *inj_on g* $\{0<..<1\}$
using *loop_free_inj_on simple_path_def* **by** *auto*

5.1.5 Simple paths with the endpoints removed

lemma *simple_path_endless*:
assumes *simple_path c*
shows *path_image c* − {*pathstart c*,*pathfinish c*} = *c* ‘ $\{0<..<1\}$ (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs*
using *less_eq_real_def* **by** (*auto simp*: *path_image_def pathstart_def pathfinish_def*)
show *?rhs* ⊆ *?lhs*
using *assms*
apply (*simp add*: *image_subset_iff path_image_def pathstart_def pathfinish_def simple_path_def loop_free_def Ball_def*)
by (*smt* (*verit*))
qed

lemma *connected_simple_path_endless*:
assumes *simple_path c*
shows *connected*(*path_image c* − {*pathstart c*,*pathfinish c*})
proof −
have *continuous_on* $\{0<..<1\}$ *c*
using *assms* **by** (*simp add*: *simple_path_def continuous_on_path path_def subset_iff*)
then have *connected* (*c* ‘ $\{0<..<1\}$)
using *connected_Ioo connected_continuous_image* **by** *blast*
then show *?thesis*
using *assms* **by** (*simp add*: *simple_path_endless*)
qed

lemma *nonempty_simple_path_endless*:
simple_path c $\Longrightarrow$ *path_image c* − {*pathstart c*,*pathfinish c*} ≠ {}
by (*simp add*: *simple_path_endless*)

5.1.6 The operations on paths

lemma *path_image_subset_reversepath*: *path_image*(*reversepath g*) ≤ *path_image g*
by *simp*

lemma *path_imp_reversepath*: *path g* $\Longrightarrow$ *path*(*reversepath g*)
by *simp*

lemma *half_bounded_equal*: $1 \leq x * 2 \Longrightarrow x * 2 \leq 1 \longleftrightarrow x = (1/2::real)$
by *simp*

lemma *continuous_on_joinpaths*:
assumes *continuous_on* $\{0..1\}$ *g1 continuous_on* $\{0..1\}$ *g2 pathfinish g1* = *pathstart g2*
shows *continuous_on* $\{0..1\}$ (*g1* +++ *g2*)
using *assms path_def path_join* **by** *blast*

lemma *path_join_imp*: $[\![$*path g1*; *path g2*; *pathfinish g1* = *pathstart g2*$]\!] \Longrightarrow$ *path*(*g1* +++ *g2*)
by *simp*

lemma *arc_join*:
assumes *arc g1 arc g2*
pathfinish g1 = *pathstart g2*
path_image g1 $\cap$ *path_image g2* $\subseteq$ {*pathstart g2*}
shows *arc*(*g1* +++ *g2*)
proof −
have *injg1*: *inj_on g1* $\{0..1\}$
using *assms*
by (*simp add*: *arc_def*)
have *injg2*: *inj_on g2* $\{0..1\}$
using *assms*
by (*simp add*: *arc_def*)
have *g11*: *g1 1* = *g2 0*
and *sb*: *g1* ‘ $\{0..1\}$ $\cap$ *g2* ‘ $\{0..1\}$ $\subseteq$ {*g2 0*}
using *assms*
by (*simp_all add*: *arc_def pathfinish_def pathstart_def path_image_def*)
{ fix *x* **and** *y*::*real*
assume *xy*: *g2* (2 ∗ *x* − 1) = *g1* (2 ∗ *y*) $x \leq 1$ $0 \leq y$ $y * 2 \leq 1$ ¬ $x * 2 \leq 1$
then have *g1* (2 ∗ *y*) = *g2 0*
using *sb* **by** *force*
then have *False*
using *xy inj_onD injg2* **by** *fastforce*
} note ∗ = *this*
have *inj_on* (*g1* +++ *g2*) $\{0..1\}$
using *inj_onD* [*OF injg1*] *inj_onD* [*OF injg2*] ∗
by (*simp add*: *inj_on_def joinpaths_def Ball_def*) (*smt* (*verit*))
then show *?thesis*
using *arc_def assms path_join_imp* **by** *blast*
qed

lemma *simple_path_join_loop*:
assumes *arc g1 arc g2*
pathfinish g1 = *pathstart g2* *pathfinish g2* = *pathstart g1*

```
        path_image g1 ∩ path_image g2 ⊆ {pathstart g1, pathstart g2}
      shows simple_path(g1 +++ g2)
proof −
  have injg1: inj_on g1 {0..1} and injg2: inj_on g2 {0..1}
    using assms by (auto simp add: arc_def)
  have g12: g1 1 = g2 0
   and g21: g2 1 = g1 0
   and sb:  g1 ‘ {0..1} ∩ g2 ‘ {0..1} ⊆ {g1 0, g2 0}
    using assms
    by (simp_all add: arc_def pathfinish_def pathstart_def path_image_def)
  { fix x and y::real
    assume g2_eq: g2 (2 ∗ x − 1) = g1 (2 ∗ y)
      and xyI: x ≠ 1 ∨ y ≠ 0
      and xy: x ≤ 1 0 ≤ y  y ∗ 2 ≤ 1 ¬ x ∗ 2 ≤ 1
    then consider g1 (2 ∗ y) = g1 0 | g1 (2 ∗ y) = g2 0
      using sb by force
    then have False
    proof cases
      case 1
      then have y = 0
        using xy g2_eq by (auto dest!: inj_onD [OF injg1])
      then show ?thesis
        using xy g2_eq xyI by (auto dest: inj_onD [OF injg2] simp flip: g21)
    next
      case 2
      then have 2∗x = 1
        using g2_eq g12 inj_onD [OF injg2] atLeastAtMost_iff xy(1) xy(4) by
fastforce
      with xy show False by auto
    qed
  } note ∗ = this
  have loop_free(g1 +++ g2)
    using inj_onD [OF injg1] inj_onD [OF injg2] ∗
    by (simp add: loop_free_def joinpaths_def Ball_def) (smt (verit))
  then show ?thesis
    by (simp add: arc_imp_path assms simple_path_def)
qed

lemma reversepath_joinpaths:
    pathfinish g1 = pathstart g2 ⟹ reversepath(g1 +++ g2) = reversepath g2
+++ reversepath g1
  unfolding reversepath_def pathfinish_def pathstart_def joinpaths_def
  by (rule ext) (auto simp: mult.commute)
```

5.1.7 Some reversed and "if and only if" versions of joining theorems

```
lemma path_join_path_ends:
  fixes g1 :: real ⇒ 'a::metric_space
```

assumes *path*(*g1 +++ g2*) *path g2*
shows *pathfinish g1 = pathstart g2*
proof (*rule ccontr*)
define *e* **where** *e = dist* (*g1 1*) (*g2 0*)
assume *Neg*: *pathfinish g1 ≠ pathstart g2*
then have *0 < dist* (*pathfinish g1*) (*pathstart g2*)
by *auto*
then have *e > 0*
by (*metis e_def pathfinish_def pathstart_def*)
then have ∀ *e*>*0*. ∃ *d*>*0*. ∀ *x′*∈{*0..1*}. *dist x′ 0 < d* ⟶ *dist* (*g2 x′*) (*g2 0*) < *e*
using ‹*path g2*› *atLeastAtMost_iff zero_le_one* **unfolding** *path_def continuous_on_iff*
by *blast*
then obtain *d1* **where** *d1 > 0*
and *d1*: ⋀*x′*. ⟦*x′*∈{*0..1*}; *norm x′ < d1*⟧ ⟹ *dist* (*g2 x′*) (*g2 0*) < *e*/*2*
by (*metis* ‹*0 < e*› *half_gt_zero_iff norm_conv_dist*)
obtain *d2* **where** *d2 > 0*
and *d2*: ⋀*x′*. ⟦*x′*∈{*0..1*}; *dist x′* (*1*/*2*) < *d2*⟧
⟹ *dist* ((*g1 +++ g2*) *x′*) (*g1 1*) < *e*/*2*
using *assms*(*1*) ‹*e > 0*› **unfolding** *path_def continuous_on_iff*
apply (*drule_tac x*=*1*/*2* **in** *bspec*, *simp*)
apply (*drule_tac x*=*e*/*2* **in** *spec*, *force simp*: *joinpaths_def*)
done
have *int01_1*: *min* (*1*/*2*) (*min d1 d2*) / *2* ∈ {*0..1*}
using ‹*d1 > 0*› ‹*d2 > 0*› **by** (*simp add*: *min_def*)
have *dist1*: *norm* (*min* (*1 / 2*) (*min d1 d2*) / *2*) < *d1*
using ‹*d1 > 0*› ‹*d2 > 0*› **by** (*simp add*: *min_def dist_norm*)
have *int01_2*: *1*/*2* + *min* (*1*/*2*) (*min d1 d2*) / *4* ∈ {*0..1*}
using ‹*d1 > 0*› ‹*d2 > 0*› **by** (*simp add*: *min_def*)
have *dist2*: *dist* (*1 / 2 + min* (*1 / 2*) (*min d1 d2*) / *4*) (*1 / 2*) < *d2*
using ‹*d1 > 0*› ‹*d2 > 0*› **by** (*simp add*: *min_def dist_norm*)
have [*simp*]: ¬ *min* (*1 / 2*) (*min d1 d2*) ≤ *0*
using ‹*d1 > 0*› ‹*d2 > 0*› **by** (*simp add*: *min_def*)
have *dist* (*g2* (*min* (*1 / 2*) (*min d1 d2*) / *2*)) (*g1 1*) < *e*/*2*
dist (*g2* (*min* (*1 / 2*) (*min d1 d2*) / *2*)) (*g2 0*) < *e*/*2*
using *d1* [*OF int01_1 dist1*] *d2* [*OF int01_2 dist2*] **by** (*simp_all add*: *joinpaths_def*)
then have *dist* (*g1 1*) (*g2 0*) < *e*/*2* + *e*/*2*
using *dist_triangle_half_r e_def* **by** *blast*
then show *False*
by (*simp add*: *e_def* [*symmetric*])
qed

lemma *path_join_eq* [*simp*]:
fixes *g1* :: *real* ⇒ *′a*::*metric_space*
assumes *path g1 path g2*
shows *path*(*g1 +++ g2*) ⟷ *pathfinish g1 = pathstart g2*
using *assms* **by** (*metis path_join_path_ends path_join_imp*)

lemma *simple_path_joinE*:
assumes *simple_path*(*g1 +++ g2*) **and** *pathfinish g1 = pathstart g2*
obtains *arc g1 arc g2*
path_image g1 ∩ path_image g2 ⊆ {pathstart g1, pathstart g2}
proof −
have ∗: ⋀*x y*. ⟦*0 ≤ x*; *x ≤ 1*; *0 ≤ y*; *y ≤ 1*; (*g1 +++ g2*) *x* = (*g1 +++ g2*) *y*⟧
⟹ *x* = *y* ∨ *x* = *0* ∧ *y* = *1* ∨ *x* = *1* ∧ *y* = *0*
using *assms* **by** (*simp add*: *simple_path_def loop_free_def*)
have *path g1*
using *assms path_join simple_path_imp_path* **by** *blast*
moreover have *inj_on g1 {0..1}*
proof (*clarsimp simp*: *inj_on_def*)
fix *x y*
assume *g1 x* = *g1 y 0 ≤ x x ≤ 1 0 ≤ y y ≤ 1*
then show *x* = *y*
using ∗ [*of x/2 y/2*] **by** (*simp add*: *joinpaths_def split_ifs*)
qed
ultimately have *arc g1*
using *assms* **by** (*simp add*: *arc_def*)
have [*simp*]: *g2 0* = *g1 1*
using *assms* **by** (*metis pathfinish_def pathstart_def*)
have *path g2*
using *assms path_join simple_path_imp_path* **by** *blast*
moreover have *inj_on g2 {0..1}*
proof (*clarsimp simp*: *inj_on_def*)
fix *x y*
assume *g2 x* = *g2 y 0 ≤ x x ≤ 1 0 ≤ y y ≤ 1*
then show *x* = *y*
using ∗ [*of* (*x*+*1*) / *2* (*y*+*1*) / *2*]
by (*force simp*: *joinpaths_def split_ifs field_split_simps*)
qed
ultimately have *arc g2*
using *assms* **by** (*simp add*: *arc_def*)
have *g2 y* = *g1 0* ∨ *g2 y* = *g1 1*
if *g1 x* = *g2 y 0 ≤ x x ≤ 1 0 ≤ y y ≤ 1* **for** *x y*
using ∗ [*of x* / *2* (*y* + *1*) / *2*] *that*
by (*auto simp*: *joinpaths_def split_ifs field_split_simps*)
then have *path_image g1 ∩ path_image g2 ⊆ {pathstart g1, pathstart g2}*
by (*fastforce simp*: *pathstart_def pathfinish_def path_image_def*)
with ‹*arc g1*› ‹*arc g2*› **show** *?thesis* **using** *that* **by** *blast*
qed

lemma *simple_path_join_loop_eq*:
assumes *pathfinish g2 = pathstart g1 pathfinish g1 = pathstart g2*
shows *simple_path*(*g1 +++ g2*) ⟷
arc g1 ∧ arc g2 ∧ path_image g1 ∩ path_image g2 ⊆ {pathstart g1,
pathstart g2}
by (*metis assms simple_path_joinE simple_path_join_loop*)

lemma *arc_join_eq*:
  assumes *pathfinish g1 = pathstart g2*
    shows *arc(g1 +++ g2) ⟷*
          arc g1 ∧ arc g2 ∧ path_image g1 ∩ path_image g2 ⊆ {pathstart g2}
          (**is** *?lhs = ?rhs*)
proof
  assume *?lhs* **then show** *?rhs*
    using *reversepath_simps assms*
   by (*smt* (*verit*, *ccfv_threshold*) *Int_commute arc_distinct_ends arc_imp_simple_path arc_reversepath*
        in_mono insertE pathfinish_join reversepath_joinpaths simple_path_joinE subsetI)
next
  assume *?rhs* **then show** *?lhs*
    using *assms*
    by (*fastforce simp*: *pathfinish_def pathstart_def intro*!: *arc_join*)
qed

lemma *arc_join_eq_alt*:
       pathfinish g1 = pathstart g2
       ⟹ (*arc(g1 +++ g2) ⟷*
            arc g1 ∧ arc g2 ∧ path_image g1 ∩ path_image g2 = {pathstart g2})
using *pathfinish_in_path_image* **by** (*fastforce simp*: *arc_join_eq*)

5.1.8 The joining of paths is associative

lemma *path_assoc*:
    ⟦*pathfinish p = pathstart q*; *pathfinish q = pathstart r*⟧
     ⟹ path(p +++ (q +++ r)) ⟷ path((p +++ q) +++ r)
by *simp*

lemma *simple_path_assoc*:
  assumes *pathfinish p = pathstart q pathfinish q = pathstart r*
    shows *simple_path (p +++ (q +++ r)) ⟷ simple_path ((p +++ q) +++ r)*
proof (*cases pathstart p = pathfinish r*)
  case *True* **show** *?thesis*
  proof
    assume *simple_path (p +++ q +++ r)*
    with *assms True* **show** *simple_path ((p +++ q) +++ r)*
    by (*fastforce simp add*: *simple_path_join_loop_eq arc_join_eq path_image_join*
                  dest: *arc_distinct_ends* [*of r*])
  next
    assume *0*: *simple_path ((p +++ q) +++ r)*
    with *assms True* **have** *q*: *pathfinish r ∉ path_image q*
      using *arc_distinct_ends*
    by (*fastforce simp add*: *simple_path_join_loop_eq arc_join_eq path_image_join*)
    have *pathstart r ∉ path_image p*
      using *assms*

by (*metis 0 IntI arc_distinct_ends arc_join_eq_alt empty_iff insert_iff pathfinish_in_path_image pathfinish_join simple_path_joinE*)
with *assms 0 q True* **show** *simple_path* (*p +++ q +++ r*)
by (*auto simp*: *simple_path_join_loop_eq arc_join_eq path_image_join dest*!: *subsetD* [*OF _ IntI*])
qed
next
case *False*
{ fix *x* :: *'a*
assume *a*: *path_image p ∩ path_image q ⊆ {pathstart q}*
(path_image p ∪ path_image q) ∩ path_image r ⊆ {pathstart r}
x ∈ path_image p x ∈ path_image r
have *pathstart r ∈ path_image q*
by (*metis assms*(*2*) *pathfinish_in_path_image*)
with *a* **have** *x = pathstart q*
by *blast*
}
with *False assms* **show** *?thesis*
by (*auto simp*: *simple_path_eq_arc simple_path_join_loop_eq arc_join_eq path_image_join*)
qed

lemma *arc_assoc*:
⟦*pathfinish p = pathstart q*; *pathfinish q = pathstart r*⟧
⟹ *arc*(*p +++ (q +++ r*)) ⟷ *arc*((*p +++ q*) *+++ r*)
by (*simp add*: *arc_simple_path simple_path_assoc*)

Symmetry and loops

lemma *path_sym*:
⟦*pathfinish p = pathstart q*; *pathfinish q = pathstart p*⟧ ⟹ *path*(*p +++ q*) ⟷ *path*(*q +++ p*)
by *auto*

lemma *simple_path_sym*:
⟦*pathfinish p = pathstart q*; *pathfinish q = pathstart p*⟧
⟹ *simple_path*(*p +++ q*) ⟷ *simple_path*(*q +++ p*)
by (*metis* (*full_types*) *inf_commute insert_commute simple_path_joinE simple_path_join_loop*)

lemma *path_image_sym*:
⟦*pathfinish p = pathstart q*; *pathfinish q = pathstart p*⟧
⟹ *path_image*(*p +++ q*) = *path_image*(*q +++ p*)
by (*simp add*: *path_image_join sup_commute*)

5.1.9 Subpath

definition *subpath* :: *real* ⇒ *real* ⇒ (*real* ⇒ *'a*) ⇒ *real* ⇒ *'a*::*real_normed_vector*
where *subpath a b g* ≡ λ*x*. *g*((*b − a*) ∗ *x* + *a*)

lemma *path_image_subpath_gen*:

fixes $g :: \_ \Rightarrow 'a$::*real_normed_vector*
shows *path_image*(*subpath u v g*) = *g* ‘ (*closed_segment u v*)
by (*auto simp add*: *closed_segment_real_eq path_image_def subpath_def*)

lemma *path_image_subpath*:
fixes *g* :: *real* ⇒ $'a$::*real_normed_vector*
shows *path_image*(*subpath u v g*) = (*if u* ≤ *v then g* ‘ {*u*..*v*} *else g* ‘ {*v*..*u*})
by (*simp add*: *path_image_subpath_gen closed_segment_eq_real_ivl*)

lemma *path_image_subpath_commute*:
fixes *g* :: *real* ⇒ $'a$::*real_normed_vector*
shows *path_image*(*subpath u v g*) = *path_image*(*subpath v u g*)
by (*simp add*: *path_image_subpath_gen closed_segment_eq_real_ivl*)

lemma *path_subpath* [*simp*]:
fixes *g* :: *real* ⇒ $'a$::*real_normed_vector*
assumes *path g u* ∈ {*0*..*1*} *v* ∈ {*0*..*1*}
shows *path*(*subpath u v g*)
proof −
have *continuous_on* {*u*..*v*} *g continuous_on* {*v*..*u*} *g*
using *assms continuous_on_path* **by** *fastforce*+
then have *continuous_on* {*0*..*1*} (*g* ∘ (λ*x*. ((*v*−*u*) ∗ *x*+ *u*)))
by (*intro continuous_intros*; *simp add*: *image_affinity_atLeastAtMost* [**where** *c*=*u*])
then show *?thesis*
by (*simp add*: *path_def subpath_def*)
qed

lemma *pathstart_subpath* [*simp*]: *pathstart*(*subpath u v g*) = *g*(*u*)
by (*simp add*: *pathstart_def subpath_def*)

lemma *pathfinish_subpath* [*simp*]: *pathfinish*(*subpath u v g*) = *g*(*v*)
by (*simp add*: *pathfinish_def subpath_def*)

lemma *subpath_trivial* [*simp*]: *subpath 0 1 g* = *g*
by (*simp add*: *subpath_def*)

lemma *subpath_reversepath*: *subpath 1 0 g* = *reversepath g*
by (*simp add*: *reversepath_def subpath_def*)

lemma *reversepath_subpath*: *reversepath*(*subpath u v g*) = *subpath v u g*
by (*simp add*: *reversepath_def subpath_def algebra_simps*)

lemma *subpath_translation*: *subpath u v* ((λ*x*. *a* + *x*) ∘ *g*) = (λ*x*. *a* + *x*) ∘ *subpath u v g*
by (*rule ext*) (*simp add*: *subpath_def*)

lemma *subpath_image*: *subpath u v* (*f* ∘ *g*) = *f* ∘ *subpath u v g*
by (*rule ext*) (*simp add*: *subpath_def*)

lemma *affine_ineq*:
fixes x :: *'a*::*linordered_idom*
assumes $x \leq 1$ $v \leq u$
shows $v + x * u \leq u + x * v$
proof −
have $(1-x)*(u-v) \geq 0$
using *assms* **by** *auto*
then show *?thesis*
by (*simp add*: *algebra_simps*)
qed

lemma *sum_le_prod1*:
fixes *a*::*real* **shows** $[\![a \leq 1; b \leq 1]\!] \Longrightarrow a + b \leq 1 + a * b$
by (*metis add.commute affine_ineq mult.right_neutral*)

lemma *simple_path_subpath_eq*:
simple_path(*subpath u v g*) $\longleftrightarrow$
path(*subpath u v g*) $\wedge$ $u \neq v$ $\wedge$
$(\forall x\ y.\ x \in$ *closed_segment u v* $\wedge$ $y \in$ *closed_segment u v* $\wedge$ $g\ x = g\ y$
$\longrightarrow x = y \vee x = u \wedge y = v \vee x = v \wedge y = u)$
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then have *p*: *path* $(\lambda x.\ g\ ((v - u) * x + u))$
and *sim*: $(\bigwedge x\ y.\ [\![x \in \{0..1\}; y \in \{0..1\}; g\ ((v - u) * x + u) = g\ ((v - u) * y + u)]\!]$
$\Longrightarrow x = y \vee x = 0 \wedge y = 1 \vee x = 1 \wedge y = 0)$
by (*auto simp*: *simple_path_def loop_free_def subpath_def*)
{ fix *x y*
assume $x \in$ *closed_segment u v* $y \in$ *closed_segment u v* $g\ x = g\ y$
then have $x = y \vee x = u \wedge y = v \vee x = v \wedge y = u$
using *sim* [*of* $(x-u)/(v-u)$ $(y-u)/(v-u)$] *p*
by (*auto split*: *if_split_asm simp add*: *closed_segment_real_eq image_affinity_atLeastAtMost*)
(*simp_all add*: *field_split_simps*)
} moreover
have *path*(*subpath u v g*) $\wedge$ $u \neq v$
using *sim* [*of 1/3 2/3*] *p*
by (*auto simp*: *subpath_def*)
ultimately show *?rhs*
by *metis*
next
assume *?rhs*
then
have *d1*: $\bigwedge x\ y.\ [\![g\ x = g\ y; u \leq x; x \leq v; u \leq y; y \leq v]\!] \Longrightarrow x = y \vee x = u \wedge y = v \vee x = v \wedge y = u$
and *d2*: $\bigwedge x\ y.\ [\![g\ x = g\ y; v \leq x; x \leq u; v \leq y; y \leq u]\!] \Longrightarrow x = y \vee x = u \wedge y = v \vee x = v \wedge y = u$
and *ne*: $u < v \vee v < u$

and *psp*: *path* (*subpath u v g*)
by (*auto simp*: *closed_segment_real_eq image_affinity_atLeastAtMost*)
have [*simp*]: $\bigwedge x.\ u + x * v = v + x * u \longleftrightarrow u{=}v \vee x{=}1$
by *algebra*
show *?lhs* **using** *psp ne*
unfolding *simple_path_def loop_free_def subpath_def*
by (*fastforce simp add*: *algebra_simps affine_ineq mult_left_mono crossproduct_eq dest*: *d1 d2*)
qed

lemma *arc_subpath_eq*:
$arc(subpath\ u\ v\ g) \longleftrightarrow path(subpath\ u\ v\ g) \wedge u{\neq}v \wedge inj\_on\ g\ (closed\_segment\ u\ v)$
by (*smt* (*verit*, *best*) *arc_simple_path closed_segment_commute ends_in_segment*(*2*) *inj_on_def pathfinish_subpath pathstart_subpath simple_path_subpath_eq*)

lemma *simple_path_subpath*:
assumes *simple_path g* $u \in \{0..1\}$ $v \in \{0..1\}$ $u \neq v$
shows $simple\_path(subpath\ u\ v\ g)$
using *assms*
apply (*simp add*: *simple_path_subpath_eq simple_path_imp_path*)
apply (*simp add*: *simple_path_def loop_free_def closed_segment_real_eq image_affinity_atLeastAtMost*, *fastforce*)
done

lemma *arc_simple_path_subpath*:
$[\![simple\_path\ g;\ u \in \{0..1\};\ v \in \{0..1\};\ g\ u \neq g\ v]\!] \Longrightarrow arc(subpath\ u\ v\ g)$
by (*force intro*: *simple_path_subpath simple_path_imp_arc*)

lemma *arc_subpath_arc*:
$[\![arc\ g;\ u \in \{0..1\};\ v \in \{0..1\};\ u \neq v]\!] \Longrightarrow arc(subpath\ u\ v\ g)$
by (*meson arc_def arc_imp_simple_path arc_simple_path_subpath inj_onD*)

lemma *arc_simple_path_subpath_interior*:
$[\![simple\_path\ g;\ u \in \{0..1\};\ v \in \{0..1\};\ u \neq v;\ |u{-}v| < 1]\!] \Longrightarrow arc(subpath\ u\ v\ g)$
by (*force simp*: *simple_path_def loop_free_def intro*: *arc_simple_path_subpath*)

lemma *path_image_subpath_subset*:
$[\![u \in \{0..1\};\ v \in \{0..1\}]\!] \Longrightarrow path\_image(subpath\ u\ v\ g) \subseteq path\_image\ g$
by (*metis atLeastAtMost_iff atLeastatMost_subset_iff path_image_def path_image_subpath subset_image_iff*)

lemma *join_subpaths_middle*: $subpath\ (0)\ ((1\ /\ 2))\ p +{+}{+} subpath\ ((1\ /\ 2))\ 1\ p = p$
by (*rule ext*) (*simp add*: *joinpaths_def subpath_def field_split_simps*)

5.1.10 There is a subpath to the frontier

lemma *subpath_to_frontier_explicit*:
  fixes S :: *'a::metric_space set*
  assumes g: *path g* **and** *pathfinish* $g \notin S$
  obtains u **where** $0 \le u\ u \le 1$
    $\bigwedge x.\ 0 \le x \wedge x < u \Longrightarrow g\ x \in \mathit{interior}\ S$
    $(g\ u \notin \mathit{interior}\ S)\ (u = 0 \vee g\ u \in \mathit{closure}\ S)$
proof −
 have *gcon*: *continuous_on* $\{0..1\}$ *g*
  using *g* **by** (*simp add*: *path_def*)
 moreover have *bounded* $(\{u.\ g\ u \in \mathit{closure}\ (-S)\} \cap \{0..1\})$
  using *compact_eq_bounded_closed* **by** *fastforce*
 ultimately have *com*: *compact* $(\{0..1\} \cap \{u.\ g\ u \in \mathit{closure}\ (-S)\})$
  using *closed_vimage_Int*
  by (*metis* (*full_types*) *Int_commute closed_atLeastAtMost closed_closure compact_eq_bounded_closed vimage_def*)
 have $1 \in \{u.\ g\ u \in \mathit{closure}\ (-S)\}$
  using *assms* **by** (*simp add*: *pathfinish_def closure_def*)
 then have *dis*: $\{0..1\} \cap \{u.\ g\ u \in \mathit{closure}\ (-S)\} \neq \{\}$
  using *atLeastAtMost_iff zero_le_one* **by** *blast*
 then obtain u **where** $0 \le u\ u \le 1$ **and** *gu*: $g\ u \in \mathit{closure}\ (-S)$
    and *umin*: $\bigwedge t.\ [\![0 \le t;\ t \le 1;\ g\ t \in \mathit{closure}\ (-S)]\!] \Longrightarrow u \le t$
  using *compact_attains_inf* [*OF com dis*] **by** *fastforce*
 then have *umin'*: $\bigwedge t.\ [\![0 \le t;\ t \le 1;\ t < u]\!] \Longrightarrow\ g\ t \in S$
  using *closure_def* **by** *fastforce*
 have §: $g\ u \in \mathit{closure}\ S$ **if** $u \neq 0$
 proof −
  have $u > 0$ **using** *that* ‹$0 \le u$› **by** *auto*
  { fix *e*::*real* **assume** $e > 0$
   obtain d **where** $d{>}0$ **and** d: $\bigwedge x'.\ [\![x' \in \{0..1\};\ \mathit{dist}\ x'\ u \le d]\!] \Longrightarrow \mathit{dist}\ (g\ x')\ (g\ u) < e$
    using *continuous_onE* [*OF gcon* _ ‹$e > 0$›] ‹$0 \le$ _› ‹_ ≤ 1› *atLeastAtMost_iff* **by** *auto*
   have ∗: *dist* $(\mathit{max}\ 0\ (u - d\ /\ 2))\ u \le d$
    using ‹$0 \le u$› ‹$u \le 1$› ‹$d > 0$› **by** (*simp add*: *dist_real_def*)
   have $\exists y{\in}S.\ \mathit{dist}\ y\ (g\ u) < e$
    using ‹$0 < u$› ‹$u \le 1$› ‹$d > 0$›
    by (*force intro*: *d* [*OF* _ ∗] *umin'*)
  }
  then show *?thesis*
   by (*simp add*: *frontier_def closure_approachable*)
 qed
 show *?thesis*
 proof
  show $\bigwedge x.\ 0 \le x \wedge x < u \Longrightarrow g\ x \in \mathit{interior}\ S$
   using ‹$u \le 1$› *interior_closure umin* **by** *fastforce*
  show $g\ u \notin \mathit{interior}\ S$
   by (*simp add*: *gu interior_closure*)
 qed (*use* ‹$0 \le u$› ‹$u \le 1$› § **in** *auto*)

qed

lemma *subpath_to_frontier_strong*:
  assumes *g*: *path g* **and** *pathfinish g* ∉ *S*
  obtains *u* **where** *0* ≤ *u u* ≤ *1 g u* ∉ *interior S*
    u = *0* ∨ (∀*x*. *0* ≤ *x* ∧ *x* < *1* ⟶ *subpath 0 u g x* ∈ *interior S*) ∧ *g u* ∈ *closure S*
proof −
  obtain *u* **where** *0* ≤ *u u* ≤ *1*
    and *gxin*: ⋀*x*. *0* ≤ *x* ∧ *x* < *u* ⟹ *g x* ∈ *interior S*
    and *gunot*: (*g u* ∉ *interior S*) **and** *u0*: (*u* = *0* ∨ *g u* ∈ *closure S*)
    using *subpath_to_frontier_explicit* [*OF assms*] **by** *blast*
  show *?thesis*
  proof
    show *g u* ∉ *interior S*
      using *gunot* **by** *blast*
  qed (*use* ‹*0* ≤ *u*› ‹*u* ≤ *1*› *u0* **in** ‹(*force simp*: *subpath_def gxin*)+›)
qed

lemma *subpath_to_frontier*:
  assumes *g*: *path g* **and** *g0*: *pathstart g* ∈ *closure S* **and** *g1*: *pathfinish g* ∉ *S*
  obtains *u* **where** *0* ≤ *u u* ≤ *1 g u* ∈ *frontier S path_image*(*subpath 0 u g*) − {*g u*} ⊆ *interior S*
proof −
  obtain *u* **where** *0* ≤ *u u* ≤ *1*
    and *notin*: *g u* ∉ *interior S*
    and *disj*: *u* = *0* ∨
      (∀*x*. *0* ≤ *x* ∧ *x* < *1* ⟶ *subpath 0 u g x* ∈ *interior S*) ∧ *g u* ∈ *closure S*
      (**is** _ ∨ *?P*)
    using *subpath_to_frontier_strong* [*OF g g1*] **by** *blast*
  show *?thesis*
  proof
    show *g u* ∈ *frontier S*
      by (*metis DiffI disj frontier_def g0 notin pathstart_def*)
    show *path_image* (*subpath 0 u g*) − {*g u*} ⊆ *interior S*
      using *disj*
    proof
      assume *u* = *0*
      then show *?thesis*
        by (*simp add*: *path_image_subpath*)
    next
      assume *P*: *?P*
      show *?thesis*
      proof (*clarsimp simp add*: *path_image_subpath_gen*)
        fix *y*
        assume *y*: *y* ∈ *closed_segment 0 u g y* ∉ *interior S*
        with ‹*0* ≤ *u*› **have** *0* ≤ *y y* ≤ *u*
          by (*auto simp*: *closed_segment_eq_real_ivl split*: *if_split_asm*)

```
      then have y=u ∨ subpath 0 u g (y/u) ∈ interior S
        using P less_eq_real_def by force
      then show g y = g u
        using y by (auto simp: subpath_def split: if_split_asm)
    qed
  qed
 qed (use ‹0 ≤ u› ‹u ≤ 1› in auto)
qed

lemma exists_path_subpath_to_frontier:
  fixes S :: 'a::real_normed_vector set
  assumes path g pathstart g ∈ closure S pathfinish g ∉ S
  obtains h where path h pathstart h = pathstart g path_image h ⊆ path_image
g
                 path_image h − {pathfinish h} ⊆ interior S
                 pathfinish h ∈ frontier S
proof −
 obtain u where u: 0 ≤ u u ≤ 1 g u ∈ frontier S (path_image(subpath 0 u g) −
{g u}) ⊆ interior S
  using subpath_to_frontier [OF assms] by blast
 show ?thesis
 proof
  show path_image (subpath 0 u g) ⊆ path_image g
    by (simp add: path_image_subpath_subset u)
  show pathstart (subpath 0 u g) = pathstart g
    by (metis pathstart_def pathstart_subpath)
 qed (use assms u in ‹auto simp: path_image_subpath›)
qed

lemma exists_path_subpath_to_frontier_closed:
  fixes S :: 'a::real_normed_vector set
  assumes S: closed S and g: path g and g0: pathstart g ∈ S and g1: pathfinish
g ∉ S
  obtains h where path h pathstart h = pathstart g path_image h ⊆ path_image
g ∩ S
                 pathfinish h ∈ frontier S
 by (smt (verit, del_insts) Diff_iff Int_iff S closure_closed exists_path_subpath_to_frontier

    frontier_def g g0 g1 interior_subset singletonD subset_eq)
```

5.1.11 Shift Path to Start at Some Given Point

```
definition shiftpath :: real ⇒ (real ⇒ 'a::topological_space) ⇒ real ⇒ 'a
 where shiftpath a f = (λx. if (a + x) ≤ 1 then f (a + x) else f (a + x − 1))

lemma shiftpath_alt_def: shiftpath a f = (λx. if x ≤ 1−a then f (a + x) else f
(a + x − 1))
 by (auto simp: shiftpath_def)
```

lemma *pathstart_shiftpath*: $a \leq 1 \Longrightarrow$ *pathstart* (*shiftpath a g*) = *g a*
unfolding *pathstart_def shiftpath_def* **by** *auto*

lemma *pathfinish_shiftpath*:
assumes $0 \leq a$
and *pathfinish g* = *pathstart g*
shows *pathfinish* (*shiftpath a g*) = *g a*
using *assms*
unfolding *pathstart_def pathfinish_def shiftpath_def*
by *auto*

lemma *endpoints_shiftpath*:
assumes *pathfinish g* = *pathstart g*
and $a \in \{0 \mathinner{..} 1\}$
shows *pathfinish* (*shiftpath a g*) = *g a*
and *pathstart* (*shiftpath a g*) = *g a*
using *assms*
by (*auto intro*!: *pathfinish_shiftpath pathstart_shiftpath*)

lemma *closed_shiftpath*:
assumes *pathfinish g* = *pathstart g*
and $a \in \{0..1\}$
shows *pathfinish* (*shiftpath a g*) = *pathstart* (*shiftpath a g*)
using *endpoints_shiftpath*[*OF assms*]
by *auto*

lemma *path_shiftpath*:
assumes *path g*
and *pathfinish g* = *pathstart g*
and $a \in \{0..1\}$
shows *path* (*shiftpath a g*)
proof −
have ∗: $\{0 \mathinner{..} 1\} = \{0 \mathinner{..} 1-a\} \cup \{1-a \mathinner{..} 1\}$
using *assms*(*3*) **by** *auto*
have ∗∗: $\bigwedge x.\ x + a = 1 \Longrightarrow g\ (x + a - 1) = g\ (x + a)$
by (*smt* (*verit*, *best*) *assms*(*2*) *pathfinish_def pathstart_def*)
show *?thesis*
unfolding *path_def shiftpath_def* ∗
proof (*rule continuous_on_closed_Un*)
have *contg*: *continuous_on* $\{0..1\}$ *g*
using ‹*path g*› *path_def* **by** *blast*
show *continuous_on* $\{0..1-a\}$ (λ*x*. *if a* + *x* ≤ *1 then g* (*a* + *x*) *else g* (*a* + *x* − *1*))
proof (*rule continuous_on_eq*)
show *continuous_on* $\{0..1-a\}$ (*g* ∘ (+) *a*)
by (*intro continuous_intros continuous_on_subset* [*OF contg*]) (*use* ‹*a* ∈ $\{0..1\}$› **in** *auto*)
qed *auto*
show *continuous_on* $\{1-a..1\}$ (λ*x*. *if a* + *x* ≤ *1 then g* (*a* + *x*) *else g* (*a* + *x*

```
− 1))
    proof (rule continuous_on_eq)
      show continuous_on {1−a..1} (g ∘ (+) (a − 1))
        by (intro continuous_intros continuous_on_subset [OF contg]) (use ‹a ∈
{0..1}› in auto)
    qed (auto simp: ∗∗ add.commute add_diff_eq)
  qed auto
qed

lemma shiftpath_shiftpath:
  assumes pathfinish g = pathstart g
    and a ∈ {0..1}
    and x ∈ {0..1}
  shows shiftpath (1 − a) (shiftpath a g) x = g x
  using assms
  unfolding pathfinish_def pathstart_def shiftpath_def
  by auto

lemma path_image_shiftpath:
  assumes a: a ∈ {0..1}
    and pathfinish g = pathstart g
  shows path_image (shiftpath a g) = path_image g
proof −
  { fix x
    assume g: g 1 = g 0 x ∈ {0..1::real} and gne: ⋀y. y∈{0..1} ∩ {x. ¬ a + x
≤ 1} ⟹ g x ≠ g (a + y − 1)
    then have ∃ y∈{0..1} ∩ {x. a + x ≤ 1}. g x = g (a + y)
    proof (cases a ≤ x)
      case False
      then show ?thesis
        apply (rule_tac x=1 + x − a in bexI)
        using g gne[of 1 + x − a] a by (force simp: field_simps)+
    next
      case True
      then show ?thesis
        using g a  by (rule_tac x=x − a in bexI) (auto simp: field_simps)
    qed
  }
  then show ?thesis
    using assms
    unfolding shiftpath_def path_image_def pathfinish_def pathstart_def
    by (auto simp: image_iff)
qed

lemma loop_free_shiftpath:
  assumes loop_free g pathfinish g = pathstart g and a: 0 ≤ a a ≤ 1
    shows loop_free (shiftpath a g)
  unfolding loop_free_def
proof (intro conjI impI ballI)
```

show $x = y \vee x = 0 \wedge y = 1 \vee x = 1 \wedge y = 0$
if $x \in \{0..1\}$ $y \in \{0..1\}$ *shiftpath a g x* = *shiftpath a g y* **for** *x y*
using *that a assms* **unfolding** *shiftpath_def loop_free_def*
by (*smt* (*verit*, *ccfv_threshold*) *atLeastAtMost_iff*)
qed

lemma *simple_path_shiftpath*:
assumes *simple_path g pathfinish g* = *pathstart g* **and** *a*: $0 \leq a$ $a \leq 1$
shows *simple_path* (*shiftpath a g*)
using *assms loop_free_shiftpath path_shiftpath simple_path_def* **by** *fastforce*

5.1.12 Straight-Line Paths

definition *linepath* :: *'a*::*real_normed_vector* ⇒ *'a* ⇒ *real* ⇒ *'a*
where *linepath a b* = (λ*x*. (1 − *x*) ∗$_R$ *a* + *x* ∗$_R$ *b*)

lemma *pathstart_linepath*[*simp*]: *pathstart* (*linepath a b*) = *a*
unfolding *pathstart_def linepath_def*
by *auto*

lemma *pathfinish_linepath*[*simp*]: *pathfinish* (*linepath a b*) = *b*
unfolding *pathfinish_def linepath_def*
by *auto*

lemma *linepath_inner*: *linepath a b x* • *v* = *linepath* (*a* • *v*) (*b* • *v*) *x*
by (*simp add*: *linepath_def algebra_simps*)

lemma *Re_linepath'*: *Re* (*linepath a b x*) = *linepath* (*Re a*) (*Re b*) *x*
by (*simp add*: *linepath_def*)

lemma *Im_linepath'*: *Im* (*linepath a b x*) = *linepath* (*Im a*) (*Im b*) *x*
by (*simp add*: *linepath_def*)

lemma *linepath_0'*: *linepath a b 0* = *a*
by (*simp add*: *linepath_def*)

lemma *linepath_1'*: *linepath a b 1* = *b*
by (*simp add*: *linepath_def*)

lemma *continuous_linepath_at*[*intro*]: *continuous* (*at x*) (*linepath a b*)
unfolding *linepath_def*
by (*intro continuous_intros*)

lemma *continuous_on_linepath* [*intro*,*continuous_intros*]: *continuous_on s* (*linepath a b*)
using *continuous_linepath_at*
by (*auto intro*!: *continuous_at_imp_continuous_on*)

lemma *path_linepath*[*iff*]: *path* (*linepath a b*)

unfolding *path_def*
by (*rule continuous_on_linepath*)

lemma *path_image_linepath*[*simp*]: *path_image* (*linepath a b*) = *closed_segment a b*
unfolding *path_image_def segment linepath_def*
by *auto*

lemma *reversepath_linepath*[*simp*]: *reversepath* (*linepath a b*) = *linepath b a*
unfolding *reversepath_def linepath_def*
by *auto*

lemma *linepath_0* [*simp*]: *linepath 0 b x* = *x* $*_R$ *b*
by (*simp add*: *linepath_def*)

lemma *linepath_cnj*: *cnj* (*linepath a b x*) = *linepath* (*cnj a*) (*cnj b*) *x*
by (*simp add*: *linepath_def*)

lemma *arc_linepath*:
assumes *a* ≠ *b* **shows** [*simp*]: *arc* (*linepath a b*)
proof −
{
fix *x y* :: *real*
assume *x* $*_R$ *b* + *y* $*_R$ *a* = *x* $*_R$ *a* + *y* $*_R$ *b*
then have (*x* − *y*) $*_R$ *a* = (*x* − *y*) $*_R$ *b*
by (*simp add*: *algebra_simps*)
with *assms* **have** *x* = *y*
by *simp*
}
then show *?thesis*
unfolding *arc_def inj_on_def*
by (*fastforce simp*: *algebra_simps linepath_def*)
qed

lemma *simple_path_linepath*[*intro*]: *a* ≠ *b* ⟹ *simple_path* (*linepath a b*)
by (*simp add*: *arc_imp_simple_path*)

lemma *linepath_trivial* [*simp*]: *linepath a a x* = *a*
by (*simp add*: *linepath_def real_vector.scale_left_diff_distrib*)

lemma *linepath_refl*: *linepath a a* = (λ*x*. *a*)
by *auto*

lemma *subpath_refl*: *subpath a a g* = *linepath* (*g a*) (*g a*)
by (*simp add*: *subpath_def linepath_def algebra_simps*)

lemma *linepath_of_real*: (*linepath* (*of_real a*) (*of_real b*) *x*) = *of_real* ((*1* − *x*)∗*a* + *x*∗*b*)
by (*simp add*: *scaleR_conv_of_real linepath_def*)

lemma *of_real_linepath*: *of_real* (*linepath a b x*) = *linepath* (*of_real a*) (*of_real b*) *x*
by (*metis linepath_of_real mult.right_neutral of_real_def real_scaleR_def*)

lemma *inj_on_linepath*:
assumes $a \neq b$ **shows** *inj_on* (*linepath a b*) $\{0..1\}$
using *arc_imp_inj_on arc_linepath assms* **by** *blast*

lemma *linepath_le_1*:
fixes *a*::'*a*::*linordered_idom* **shows** $[\![a \leq 1; b \leq 1; 0 \leq u; u \leq 1]\!] \Longrightarrow (1 - u) * a + u * b \leq 1$
using *mult_left_le* [*of a* 1−*u*] *mult_left_le* [*of b u*] **by** *auto*

lemma *linepath_in_path*:
shows $x \in \{0..1\} \Longrightarrow$ *linepath a b x* ∈ *closed_segment a b*
by (*auto simp*: *segment linepath_def*)

lemma *linepath_image_01*: *linepath a b* ‘ $\{0..1\}$ = *closed_segment a b*
by (*auto simp*: *segment linepath_def*)

lemma *linepath_in_convex_hull*:
fixes *x*::*real*
assumes *a* ∈ *convex hull S*
and *b* ∈ *convex hull S*
and 0≤*x x*≤1
shows *linepath a b x* ∈ *convex hull S*
by (*meson assms atLeastAtMost_iff convex_contains_segment convex_convex_hull linepath_in_path subset_eq*)

lemma *Re_linepath*: *Re*(*linepath* (*of_real a*) (*of_real b*) *x*) = (1 − *x*)∗*a* + *x*∗*b*
by (*simp add*: *linepath_def*)

lemma *Im_linepath*: *Im*(*linepath* (*of_real a*) (*of_real b*) *x*) = 0
by (*simp add*: *linepath_def*)

lemma *bounded_linear_linepath*:
assumes *bounded_linear f*
shows *f* (*linepath a b x*) = *linepath* (*f a*) (*f b*) *x*
proof −
interpret *f*: *bounded_linear f* **by** *fact*
show *?thesis* **by** (*simp add*: *linepath_def f.add f.scale*)
qed

lemma *bounded_linear_linepath'*:
assumes *bounded_linear f*
shows *f* ∘ *linepath a b* = *linepath* (*f a*) (*f b*)
using *bounded_linear_linepath*[*OF assms*] **by** (*simp add*: *fun_eq_iff*)

lemma *linepath_cnj'*: *cnj* ∘ *linepath a b* = *linepath* (*cnj a*) (*cnj b*)
by (*simp add*: *linepath_def fun_eq_iff*)

lemma *differentiable_linepath* [*intro*]: *linepath a b differentiable at x within A*
by (*auto simp*: *linepath_def*)

lemma *has_vector_derivative_linepath_within*:
(*linepath a b has_vector_derivative* (*b* − *a*)) (*at x within S*)
by (*force intro*: *derivative_eq_intros simp add*: *linepath_def has_vector_derivative_def algebra_simps*)

5.1.13 Segments via convex hulls

lemma *segments_subset_convex_hull*:
closed_segment a b ⊆ (*convex hull* {*a*,*b*,*c*})
closed_segment a c ⊆ (*convex hull* {*a*,*b*,*c*})
closed_segment b c ⊆ (*convex hull* {*a*,*b*,*c*})
closed_segment b a ⊆ (*convex hull* {*a*,*b*,*c*})
closed_segment c a ⊆ (*convex hull* {*a*,*b*,*c*})
closed_segment c b ⊆ (*convex hull* {*a*,*b*,*c*})
by (*auto simp*: *segment_convex_hull linepath_of_real elim*!: *rev_subsetD* [*OF _ hull_mono*])

lemma *midpoints_in_convex_hull*:
assumes *x* ∈ *convex hull s y* ∈ *convex hull s*
shows *midpoint x y* ∈ *convex hull s*
using *assms closed_segment_subset_convex_hull csegment_midpoint_subset* **by** *blast*

lemma *not_in_interior_convex_hull_3*:
fixes *a* :: *complex*
shows *a* ∉ *interior*(*convex hull* {*a*,*b*,*c*})
b ∉ *interior*(*convex hull* {*a*,*b*,*c*})
c ∉ *interior*(*convex hull* {*a*,*b*,*c*})
by (*auto simp*: *card_insert_le_m1 not_in_interior_convex_hull*)

lemma *midpoint_in_closed_segment* [*simp*]: *midpoint a b* ∈ *closed_segment a b*
using *midpoints_in_convex_hull segment_convex_hull* **by** *blast*

lemma *midpoint_in_open_segment* [*simp*]: *midpoint a b* ∈ *open_segment a b* ⟷ *a* ≠ *b*
by (*simp add*: *open_segment_def*)

lemma *continuous_IVT_local_extremum*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *real*
assumes *contf*: *continuous_on* (*closed_segment a b*) *f*
and *ab*: *a* ≠ *b f a* = *f b*
obtains *z* **where** *z* ∈ *open_segment a b*
(∀ *w* ∈ *closed_segment a b*. (*f w*) ≤ (*f z*)) ∨

$$(\forall\, w \in \mathit{closed\_segment}\ a\ b.\ (f\ z) \le (f\ w))$$

proof −
obtain c **where** $c \in \mathit{closed\_segment}\ a\ b$ **and** c: $\bigwedge y.\ y \in \mathit{closed\_segment}\ a\ b \Longrightarrow f\ y \le f\ c$
using *continuous_attains_sup* [*of closed_segment a b f*] *contf* **by** *auto*
moreover
obtain d **where** $d \in \mathit{closed\_segment}\ a\ b$ **and** d: $\bigwedge y.\ y \in \mathit{closed\_segment}\ a\ b \Longrightarrow f\ d \le f\ y$
using *continuous_attains_inf* [*of closed_segment a b f*] *contf* **by** *auto*
ultimately show *?thesis*
by (*smt* (*verit*) *UnE ab closed_segment_eq_open empty_iff insert_iff midpoint_in_open_segment that*)
qed

An injective map into R is also an open map w.r.T. the universe, and conversely.

proposition *injective_eq_1d_open_map_UNIV*:
fixes f :: *real* ⇒ *real*
assumes *contf*: *continuous_on S f* **and** S: *is_interval S*
shows *inj_on f S* ⟷ (∀ T. *open T* ∧ $T \subseteq S$ ⟶ *open*(*f* ‘ T))
(**is** *?lhs* = *?rhs*)
proof *safe*
fix T
assume *injf*: *?lhs* **and** *open T* **and** $T \subseteq S$
have ∃ U. *open U* ∧ $f\ x \in U$ ∧ $U \subseteq f$ ‘ T **if** $x \in T$ **for** x
proof −
obtain δ **where** $\delta > 0$ **and** δ: *cball x δ* ⊆ T
using ‹*open T*› ‹$x \in T$› *open_contains_cball_eq* **by** *blast*
show *?thesis*
proof (*intro exI conjI*)
have *closed_segment* (*x*−*δ*) (*x*+*δ*) = {*x*−*δ*..*x*+*δ*}
using ‹0 < *δ*› **by** (*auto simp*: *closed_segment_eq_real_ivl*)
also have ... ⊆ S
using *δ* ‹$T \subseteq S$› **by** (*auto simp*: *dist_norm subset_eq*)
finally have *f* ‘ (*open_segment* (*x*−*δ*) (*x*+*δ*)) = *open_segment* (*f* (*x*−*δ*)) (*f* (*x*+*δ*))
using *continuous_injective_image_open_segment_1*
by (*metis continuous_on_subset* [*OF contf*] *inj_on_subset* [*OF injf*])
then show *open* (*f* ‘ {*x*−*δ*<..<*x*+*δ*})
using ‹0 < *δ*› **by** (*simp add*: *open_segment_eq_real_ivl*)
show *f x* ∈ *f* ‘ {*x* − *δ*<..<*x* + *δ*}
by (*auto simp*: ‹*δ* > 0›)
show *f* ‘ {*x* − *δ*<..<*x* + *δ*} ⊆ *f* ‘ *T*
using *δ* **by** (*auto simp*: *dist_norm subset_iff*)
qed
qed
with *open_subopen* **show** *open* (*f* ‘ *T*)
by *blast*
next

assume *R*: *?rhs*
have *False* **if** *xy*: $x \in S$ $y \in S$ **and** $f\ x = f\ y$ $x \neq y$ **for** $x\ y$
proof −
have *open* (*f* ‘ *open_segment x y*)
using *R*
by (*metis S convex_contains_open_segment is_interval_convex open_greaterThanLessThan open_segment_eq_real_ivl xy*)
moreover
have *continuous_on* (*closed_segment x y*) *f*
by (*meson S closed_segment_subset contf continuous_on_subset is_interval_convex that*)
then obtain ξ **where** $\xi \in$ *open_segment x y*
and ξ: $(\forall w \in$ *closed_segment x y*. $(f\ w) \leq (f\ \xi)) \vee$
$(\forall w \in$ *closed_segment x y*. $(f\ \xi) \leq (f\ w))$
using *continuous_IVT_local_extremum* [*of x y f*] ‹$f\ x = f\ y$› ‹$x \neq y$› **by** *blast*
ultimately obtain *e* **where** *e>0* **and** *e*: $\bigwedge u$. *dist u* $(f\ \xi) < e \Longrightarrow u \in f$ ‘ *open_segment x y*
using *open_dist* **by** (*metis image_eqI*)
have *fin*: $f\ \xi + (e/2) \in f$ ‘ *open_segment x y* $f\ \xi - (e/2) \in f$ ‘ *open_segment x y*
using *e* [*of* $f\ \xi + (e/2)$] *e* [*of* $f\ \xi - (e/2)$] ‹$e > 0$› **by** (*auto simp*: *dist_norm*)
show *?thesis*
using ξ ‹$0 < e$› *fin open_closed_segment* **by** *fastforce*
qed
then show *?lhs*
by (*force simp*: *inj_on_def*)
qed

5.1.14 Bounding a point away from a path

lemma *not_on_path_ball*:
fixes *g* :: *real* ⇒ *'a*::*heine_borel*
assumes *path g*
and *z*: $z \notin$ *path_image g*
shows $\exists e > 0$. *ball z e* $\cap$ *path_image g* $= \{\}$
proof −
have *closed* (*path_image g*)
by (*simp add*: ‹*path g*› *closed_path_image*)
then obtain *a* **where** $a \in$ *path_image g* $\forall y \in$ *path_image g*. *dist z a* $\leq$ *dist z y*
by (*auto intro*: *distance_attains_inf*[*OF _ path_image_nonempty, of g z*])
then show *?thesis*
by (*rule_tac x=dist z a* **in** *exI*) (*use dist_commute z* **in** *auto*)
qed

lemma *not_on_path_cball*:
fixes *g* :: *real* ⇒ *'a*::*heine_borel*
assumes *path g*
and $z \notin$ *path_image g*
shows $\exists e{>}0$. *cball z e* $\cap$ (*path_image g*) $= \{\}$

by (*smt* (*verit*, *ccfv_threshold*) *open_ball assms centre_in_ball inf.orderE inf_assoc inf_bot_right not_on_path_ball open_contains_cball_eq*)

5.1.15 Path component

Original formalization by Tom Hales

definition *path_component S x y* ≡
(∃ *g*. *path g* ∧ *path_image g* ⊆ *S* ∧ *pathstart g* = *x* ∧ *pathfinish g* = *y*)

abbreviation
path_component_set S x ≡ *Collect* (*path_component S x*)

lemmas *path_defs* = *path_def pathstart_def pathfinish_def path_image_def path_component_def*

lemma *path_component_mem*:
assumes *path_component S x y*
shows *x* ∈ *S* **and** *y* ∈ *S*
using *assms*
unfolding *path_defs*
by *auto*

lemma *path_component_refl*:
assumes *x* ∈ *S*
shows *path_component S x x*
using *assms*
unfolding *path_defs*
by (*metis* (*full_types*) *assms continuous_on_const image_subset_iff path_image_def*)

lemma *path_component_refl_eq*: *path_component S x x* ⟷ *x* ∈ *S*
by (*auto intro*!: *path_component_mem path_component_refl*)

lemma *path_component_sym*: *path_component S x y* ⟹ *path_component S y x*
unfolding *path_component_def*
by (*metis* (*no_types*) *path_image_reversepath path_reversepath pathfinish_reversepath pathstart_reversepath*)

lemma *path_component_trans*:
assumes *path_component S x y* **and** *path_component S y z*
shows *path_component S x z*
using *assms*
unfolding *path_component_def*
by (*metis path_join pathfinish_join pathstart_join subset_path_image_join*)

lemma *path_component_of_subset*: *S* ⊆ *T* ⟹ *path_component S x y* ⟹ *path_component T x y*
unfolding *path_component_def* **by** *auto*

lemma *path_component_linepath*:
fixes *S* :: *'a*::*real_normed_vector set*

shows *closed_segment a b* $\subseteq$ *S* $\Longrightarrow$ *path_component S a b*
unfolding *path_component_def* **by** *fastforce*

Path components as sets

lemma *path_component_set*:
path_component_set S x =
{*y*. ($\exists$ *g*. *path g* $\wedge$ *path_image g* $\subseteq$ *S* $\wedge$ *pathstart g* = *x* $\wedge$ *pathfinish g* = *y*)}
by (*auto simp*: *path_component_def*)

lemma *path_component_subset*: *path_component_set S x* $\subseteq$ *S*
by (*auto simp*: *path_component_mem*(*2*))

lemma *path_component_eq_empty*: *path_component_set S x* = {} $\longleftrightarrow$ *x* $\notin$ *S*
using *path_component_mem path_component_refl_eq*
by *fastforce*

lemma *path_component_mono*:
S $\subseteq$ *T* $\Longrightarrow$ (*path_component_set S x*) $\subseteq$ (*path_component_set T x*)
by (*simp add*: *Collect_mono path_component_of_subset*)

lemma *path_component_eq*:
y $\in$ *path_component_set S x* $\Longrightarrow$ *path_component_set S y* = *path_component_set S x*
by (*metis* (*no_types*, *lifting*) *Collect_cong mem_Collect_eq path_component_sym path_component_trans*)

5.1.16 Path connectedness of a space

definition *path_connected S* $\longleftrightarrow$
($\forall$ *x*$\in$*S*. $\forall$ *y*$\in$*S*. $\exists$ *g*. *path g* $\wedge$ *path_image g* $\subseteq$ *S* $\wedge$ *pathstart g* = *x* $\wedge$ *pathfinish g* = *y*)

lemma *path_connectedin_iff_path_connected_real* [*simp*]:
path_connectedin euclideanreal S $\longleftrightarrow$ *path_connected S*
by (*simp add*: *path_connectedin path_connected_def path_defs image_subset_iff_funcset*)

lemma *path_connected_component*: *path_connected S* $\longleftrightarrow$ ($\forall$ *x*$\in$*S*. $\forall$ *y*$\in$*S*. *path_component S x y*)
unfolding *path_connected_def path_component_def* **by** *auto*

lemma *path_connected_component_set*: *path_connected S* $\longleftrightarrow$ ($\forall$ *x*$\in$*S*. *path_component_set S x* = *S*)
unfolding *path_connected_component path_component_subset*
using *path_component_mem* **by** *blast*

lemma *path_component_maximal*:
$[\![x \in T; \textit{path\_connected } T; T \subseteq S]\!] \Longrightarrow T \subseteq (\textit{path\_component\_set } S\ x)$
by (*metis path_component_mono path_connected_component_set*)

lemma *convex_imp_path_connected*:
fixes S :: *'a*::*real_normed_vector set*
assumes *convex S*
shows *path_connected S*
unfolding *path_connected_def*
using *assms convex_contains_segment* **by** *fastforce*

lemma *path_connected_UNIV* [*iff*]: *path_connected* (*UNIV* :: *'a*::*real_normed_vector set*)
by (*simp add*: *convex_imp_path_connected*)

lemma *path_component_UNIV*: *path_component_set UNIV x* = (*UNIV* :: *'a*::*real_normed_vector set*)
using *path_connected_component_set* **by** *auto*

lemma *path_connected_imp_connected*:
assumes *path_connected S*
shows *connected S*
proof (*rule connectedI*)
fix *e1 e2*
assume *as*: *open e1 open e2 S* ⊆ *e1* ∪ *e2 e1* ∩ *e2* ∩ *S* = {} *e1* ∩ *S* ≠ {} *e2* ∩ *S* ≠ {}
then obtain *x1 x2* **where** *obt*:*x1* ∈ *e1* ∩ *S x2* ∈ *e2* ∩ *S*
by *auto*
then obtain *g* **where** *g*: *path g path_image g* ⊆ *S* **and** *pg*: *pathstart g* = *x1 pathfinish g* = *x2*
using *assms*[*unfolded path_connected_def*,*rule_format*,*of x1 x2*] **by** *auto*
have ∗: *connected* {*0..1*::*real*}
by (*auto intro*!: *convex_connected*)
have {*0..1*} ⊆ {*x* ∈ {*0..1*}. *g x* ∈ *e1*} ∪ {*x* ∈ {*0..1*}. *g x* ∈ *e2*}
using *as*(*3*) *g*(*2*)[*unfolded path_defs*] **by** *blast*
moreover have {*x* ∈ {*0..1*}. *g x* ∈ *e1*} ∩ {*x* ∈ {*0..1*}. *g x* ∈ *e2*} = {}
using *as*(*4*) *g*(*2*)[*unfolded path_defs*]
unfolding *subset_eq*
by *auto*
moreover have {*x* ∈ {*0..1*}. *g x* ∈ *e1*} ≠ {} ∧ {*x* ∈ {*0..1*}. *g x* ∈ *e2*} ≠ {}
by (*smt* (*verit*, *ccfv_threshold*) *IntE atLeastAtMost_iff empty_iff pg mem_Collect_eq obt pathfinish_def pathstart_def*)
ultimately show *False*
using ∗[*unfolded connected_local not_ex*, *rule_format*,
of {*0..1*} ∩ *g* −‘ *e1* {*0..1*} ∩ *g* −‘ *e2*]
using *continuous_openin_preimage_gen*[*OF g*(*1*)[*unfolded path_def*] *as*(*1*)]
using *continuous_openin_preimage_gen*[*OF g*(*1*)[*unfolded path_def*] *as*(*2*)]
by *auto*
qed

lemma *open_path_component*:
fixes S :: *'a*::*real_normed_vector set*

assumes *open S*
shows *open* (*path_component_set S x*)
unfolding *open_contains_ball*
by (*metis assms centre_in_ball convex_ball convex_imp_path_connected equals0D openE*
path_component_eq path_component_eq_empty path_component_maximal)

lemma *open_non_path_component*:
fixes *S* :: *'a*::*real_normed_vector set*
assumes *open S*
shows *open* (*S* − *path_component_set S x*)
unfolding *open_contains_ball*
proof
fix *y*
assume *y*: *y* ∈ *S* − *path_component_set S x*
then obtain *e* **where** *e*: *e* > *0 ball y e* ⊆ *S*
using *assms openE* **by** *auto*
show ∃ *e*>*0*. *ball y e* ⊆ *S* − *path_component_set S x*
proof (*intro exI conjI subsetI DiffI notI*)
show ⋀*x*. *x* ∈ *ball y e* ⟹ *x* ∈ *S*
using *e* **by** *blast*
show *False* **if** *z* ∈ *ball y e z* ∈ *path_component_set S x* **for** *z*
by (*metis* (*no_types*, *lifting*) *Diff_iff centre_in_ball convex_ball convex_imp_path_connected*
path_component_eq path_component_maximal subsetD that y e)
qed (*use e* **in** *auto*)
qed

lemma *connected_open_path_connected*:
fixes *S* :: *'a*::*real_normed_vector set*
assumes *open S*
and *connected S*
shows *path_connected S*
unfolding *path_connected_component_set*
proof (*rule*, *rule*, *rule path_component_subset*, *rule*)
fix *x y*
assume *x* ∈ *S* **and** *y* ∈ *S*
show *y* ∈ *path_component_set S x*
proof (*rule ccontr*)
assume ¬ *?thesis*
moreover have *path_component_set S x* ∩ *S* ≠ {}
using ‹*x* ∈ *S*› *path_component_eq_empty path_component_subset*[*of S x*]
by *auto*
ultimately
show *False*
using ‹*y* ∈ *S*› *open_non_path_component*[*OF* ‹*open S*›] *open_path_component*[*OF* ‹*open S*›]
using ‹*connected S*›[*unfolded connected_def not_ex*, *rule_format*,
of path_component_set S x S − *path_component_set S x*]

```
    by auto
  qed
qed

lemma path_connected_continuous_image:
  assumes contf: continuous_on S f
    and path_connected S
  shows path_connected (f ‘ S)
  unfolding path_connected_def
proof clarsimp
  fix x y
  assume x: x ∈ S and y: y ∈ S
  with ‹path_connected S›
  show ∃g. path g ∧ path_image g ⊆ f ‘ S ∧ pathstart g = f x ∧ pathfinish g = f y
    unfolding path_defs path_connected_def
    using continuous_on_subset[OF contf]
   by (smt (verit, ccfv_threshold) continuous_on_compose2 image_eqI image_subset_iff)
qed

lemma path_connected_translationI:
  fixes a :: 'a :: topological_group_add
  assumes path_connected S shows path_connected ((λx. a + x) ‘ S)
  by (intro path_connected_continuous_image assms continuous_intros)

lemma path_connected_translation:
  fixes a :: 'a :: topological_group_add
  shows path_connected ((λx. a + x) ‘ S) = path_connected S
proof −
  have ∀x y. (+) (x::'a) ‘ (+) (0 − x) ‘ y = y
    by (simp add: image_image)
  then show ?thesis
    by (metis (no_types) path_connected_translationI)
qed

lemma path_connected_segment [simp]:
    fixes a :: 'a::real_normed_vector
    shows path_connected (closed_segment a b)
  by (simp add: convex_imp_path_connected)

lemma path_connected_open_segment [simp]:
    fixes a :: 'a::real_normed_vector
    shows path_connected (open_segment a b)
  by (simp add: convex_imp_path_connected)

lemma homeomorphic_path_connectedness:
  S homeomorphic T ⟹ path_connected S ⟷ path_connected T
 unfolding homeomorphic_def homeomorphism_def by (metis path_connected_continuous_image)

lemma path_connected_empty [simp]: path_connected {}
```

unfolding *path_connected_def* **by** *auto*

lemma *path_connected_singleton* [*simp*]: *path_connected* {*a*}
unfolding *path_connected_def pathstart_def pathfinish_def path_image_def*
using *path_def* **by** *fastforce*

lemma *path_connected_Un*:
assumes *path_connected S*
and *path_connected T*
and $S \cap T \neq \{\}$
shows *path_connected* $(S \cup T)$
unfolding *path_connected_component*
proof (*intro ballI*)
fix *x y*
assume *x*: $x \in S \cup T$ **and** *y*: $y \in S \cup T$
from *assms* **obtain** *z* **where** *z*: $z \in S$ $z \in T$
by *auto*
with *x y* **show** *path_component* $(S \cup T)$ *x y*
by (*smt* (*verit*) *assms*(*1*,*2*) *in_mono mem_Collect_eq path_component_eq path_component_maximal*
sup.bounded_iff sup.cobounded2 sup_ge1)
qed

lemma *path_connected_UNION*:
assumes $\bigwedge i.\ i \in A \Longrightarrow$ *path_connected* $(S\ i)$
and $\bigwedge i.\ i \in A \Longrightarrow z \in S\ i$
shows *path_connected* $(\bigcup i{\in}A.\ S\ i)$
unfolding *path_connected_component*
proof *clarify*
fix *x i y j*
assume $*$: $i \in A$ $x \in S\ i$ $j \in A$ $y \in S\ j$
then have *path_component* $(S\ i)$ *x z* **and** *path_component* $(S\ j)$ *z y*
using *assms* **by** (*simp_all add*: *path_connected_component*)
then have *path_component* $(\bigcup i{\in}A.\ S\ i)$ *x z* **and** *path_component* $(\bigcup i{\in}A.\ S\ i)$ *z y*
using $*$(*1*,*3*) **by** (*meson SUP_upper path_component_of_subset*)+
then show *path_component* $(\bigcup i{\in}A.\ S\ i)$ *x y*
by (*rule path_component_trans*)
qed

lemma *path_component_path_image_pathstart*:
assumes *p*: *path p* **and** *x*: $x \in$ *path_image p*
shows *path_component* (*path_image p*) (*pathstart p*) *x*
proof −
obtain *y* **where** *x*: $x = p\ y$ **and** *y*: $0 \le y\ y \le 1$
using *x* **by** (*auto simp*: *path_image_def*)
show *?thesis*
unfolding *path_component_def*
proof (*intro exI conjI*)

```
    have continuous_on ((*) y ` {0..1}) p
      by (simp add: continuous_on_path image_mult_atLeastAtMost_if p y)
    then have continuous_on {0..1} (p ∘ ((*) y))
      using continuous_on_compose continuous_on_mult_const by blast
    then show path (λu. p (y * u))
      by (simp add: path_def)
    show path_image (λu. p (y * u)) ⊆ path_image p
      using y mult_le_one by (fastforce simp: path_image_def image_iff)
  qed (auto simp: pathstart_def pathfinish_def x)
qed

lemma path_connected_path_image: path p ⟹ path_connected(path_image p)
  unfolding path_connected_component
 by (meson path_component_path_image_pathstart path_component_sym path_component_trans)

lemma path_connected_path_component [simp]:
  path_connected (path_component_set S x)
proof (clarsimp simp: path_connected_def)
  fix y z
  assume pa: path_component S x y path_component S x z
  then have pae: path_component_set S x = path_component_set S y
    using path_component_eq by auto
  obtain g where g: path g ∧ path_image g ⊆ S ∧ pathstart g = y ∧ pathfinish g
= z
     using pa path_component_sym path_component_trans path_component_def
by metis
  then have path_image g ⊆ path_component_set S x
    using pae path_component_maximal path_connected_path_image by blast
  then show ∃g. path g ∧ path_image g ⊆ path_component_set S x ∧
                pathstart g = y ∧ pathfinish g = z
    using g by blast
qed

lemma path_component:
  path_component S x y ⟷ (∃t. path_connected t ∧ t ⊆ S ∧ x ∈ t ∧ y ∈ t)
    (is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
   by (metis path_component_def path_connected_path_image pathfinish_in_path_image
pathstart_in_path_image)
next
  assume ?rhs then show ?lhs
    by (meson path_component_of_subset path_connected_component)
qed

lemma path_component_path_component [simp]:
   path_component_set (path_component_set S x) x = path_component_set S x
proof (cases x ∈ S)
  case True show ?thesis
```

by (*metis True mem_Collect_eq path_component_refl path_connected_component_set path_connected_path_component*)
next
case *False* **then show** *?thesis*
by (*metis False empty_iff path_component_eq_empty*)
qed

lemma *path_component_subset_connected_component*:
(*path_component_set S x*) ⊆ (*connected_component_set S x*)
proof (*cases x ∈ S*)
case *True* **show** *?thesis*
by (*simp add*: *True connected_component_maximal path_component_refl path_component_subset path_connected_imp_connected*)
next
case *False* **then show** *?thesis*
using *path_component_eq_empty* **by** *auto*
qed

5.1.17 Lemmas about path-connectedness

lemma *path_connected_linear_image*:
fixes *f* :: *'a*::*real_normed_vector* ⇒ *'b*::*real_normed_vector*
assumes *path_connected S bounded_linear f*
shows *path_connected*(*f* ‘ *S*)
by (*auto simp*: *linear_continuous_on assms path_connected_continuous_image*)

lemma *is_interval_path_connected*: *is_interval S* ⟹ *path_connected S*
by (*simp add*: *convex_imp_path_connected is_interval_convex*)

lemma *path_connected_Ioi*[*simp*]: *path_connected* {*a*<..} **for** *a* :: *real*
by (*simp add*: *convex_imp_path_connected*)

lemma *path_connected_Ici*[*simp*]: *path_connected* {*a*..} **for** *a* :: *real*
by (*simp add*: *convex_imp_path_connected*)

lemma *path_connected_Iio*[*simp*]: *path_connected* {..<*a*} **for** *a* :: *real*
by (*simp add*: *convex_imp_path_connected*)

lemma *path_connected_Iic*[*simp*]: *path_connected* {..*a*} **for** *a* :: *real*
by (*simp add*: *convex_imp_path_connected*)

lemma *path_connected_Ioo*[*simp*]: *path_connected* {*a*<..<*b*} **for** *a b* :: *real*
by (*simp add*: *convex_imp_path_connected*)

lemma *path_connected_Ioc*[*simp*]: *path_connected* {*a*<..*b*} **for** *a b* :: *real*
by (*simp add*: *convex_imp_path_connected*)

lemma *path_connected_Ico*[*simp*]: *path_connected* {*a*..<*b*} **for** *a b* :: *real*
by (*simp add*: *convex_imp_path_connected*)

lemma *path_connectedin_path_image*:
assumes *pathin X g* **shows** *path_connectedin X* (*g* ‘ ({*0..1*}))
unfolding *pathin_def*
proof (*rule path_connectedin_continuous_map_image*)
show *continuous_map* (*subtopology euclideanreal* {*0..1*}) *X g*
using *assms pathin_def* **by** *blast*
qed (*auto simp*: *is_interval_1 is_interval_path_connected*)

lemma *path_connected_space_subconnected*:
path_connected_space X ⟷
(∀ *x* ∈ *topspace X*. ∀ *y* ∈ *topspace X*. ∃ *S*. *path_connectedin X S* ∧ *x* ∈ *S* ∧ *y* ∈ *S*)
by (*metis path_connectedin path_connectedin_topspace path_connected_space_def*)

lemma *connectedin_path_image*: *pathin X g* ⟹ *connectedin X* (*g* ‘ ({*0..1*}))
by (*simp add*: *path_connectedin_imp_connectedin path_connectedin_path_image*)

lemma *compactin_path_image*: *pathin X g* ⟹ *compactin X* (*g* ‘ ({*0..1*}))
unfolding *pathin_def*
by (*rule image_compactin* [*of top_of_set* {*0..1*}]) *auto*

lemma *linear_homeomorphism_image*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *linear f inj f*
obtains *g* **where** *homeomorphism* (*f* ‘ *S*) *S g f*
proof −
obtain *g* **where** *linear g g* ∘ *f* = *id*
using *assms linear_injective_left_inverse* **by** *blast*
then have *homeomorphism* (*f* ‘ *S*) *S g f*
using *assms* **unfolding** *homeomorphism_def*
by (*auto simp*: *eq_id_iff* [*symmetric*] *image_comp linear_conv_bounded_linear linear_continuous_on*)
then show *thesis* **..**
qed

lemma *linear_homeomorphic_image*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *linear f inj f*
shows *S homeomorphic f* ‘ *S*
by (*meson homeomorphic_def homeomorphic_sym linear_homeomorphism_image* [*OF assms*])

lemma *path_connected_Times*:
assumes *path_connected s path_connected t*
shows *path_connected* (*s* × *t*)
proof (*simp add*: *path_connected_def Sigma_def*, *clarify*)
fix *x1 y1 x2 y2*

assume *x1* ∈ *s* *y1* ∈ *t* *x2* ∈ *s* *y2* ∈ *t*
obtain *g* **where** *path g* **and** *g*: *path_image g* ⊆ *s* **and** *gs*: *pathstart g* = *x1* **and** *gf*: *pathfinish g* = *x2*
using ‹*x1* ∈ *s*› ‹*x2* ∈ *s*› *assms* **by** (*force simp*: *path_connected_def*)
obtain *h* **where** *path h* **and** *h*: *path_image h* ⊆ *t* **and** *hs*: *pathstart h* = *y1* **and** *hf*: *pathfinish h* = *y2*
using ‹*y1* ∈ *t*› ‹*y2* ∈ *t*› *assms* **by** (*force simp*: *path_connected_def*)
have *path* (λ*z*. (*x1*, *h z*))
using ‹*path h*›
unfolding *path_def*
by (*intro continuous_intros continuous_on_compose2* [**where** *g* = *Pair* _]; *force*)
moreover have *path* (λ*z*. (*g z*, *y2*))
using ‹*path g*›
unfolding *path_def*
by (*intro continuous_intros continuous_on_compose2* [**where** *g* = *Pair* _]; *force*)
ultimately have *1*: *path* ((λ*z*. (*x1*, *h z*)) +++ (λ*z*. (*g z*, *y2*)))
by (*metis hf gs path_join_imp pathstart_def pathfinish_def*)
have *path_image* ((λ*z*. (*x1*, *h z*)) +++ (λ*z*. (*g z*, *y2*))) ⊆ *path_image* (λ*z*. (*x1*, *h z*)) ∪ *path_image* (λ*z*. (*g z*, *y2*))
by (*rule Path_Connected.path_image_join_subset*)
also have ... ⊆ (⋃*x*∈*s*. ⋃*x1*∈*t*. {(*x*, *x1*)})
using *g h* ‹*x1* ∈ *s*› ‹*y2* ∈ *t*› **by** (*force simp*: *path_image_def*)
finally have *2*: *path_image* ((λ*z*. (*x1*, *h z*)) +++ (λ*z*. (*g z*, *y2*))) ⊆ (⋃*x*∈*s*. ⋃*x1*∈*t*. {(*x*, *x1*)}) **.**
show ∃*g*. *path g* ∧ *path_image g* ⊆ (⋃*x*∈*s*. ⋃*x1*∈*t*. {(*x*, *x1*)}) ∧
pathstart g = (*x1*, *y1*) ∧ *pathfinish g* = (*x2*, *y2*)
using *1 2 gf hs*
by (*metis* (*no_types*, *lifting*) *pathfinish_def pathfinish_join pathstart_def pathstart_join*)
qed

lemma *is_interval_path_connected_1*:
fixes *s* :: *real set*
shows *is_interval s* ⟷ *path_connected s*
using *is_interval_connected_1 is_interval_path_connected path_connected_imp_connected*
by *blast*

5.1.18 Path components

lemma *Union_path_component* [*simp*]:
Union {*path_component_set S x* |*x*. *x* ∈ *S*} = *S*
using *path_component_subset path_component_refl* **by** *blast*

lemma *path_component_disjoint*:
disjnt (*path_component_set S a*) (*path_component_set S b*) ⟷
(*a* ∉ *path_component_set S b*)
unfolding *disjnt_iff*

using *path_component_sym path_component_trans* **by** *blast*

lemma *path_component_eq_eq*:
path_component S x = path_component S y ⟷
(x ∉ S) ∧ (y ∉ S) ∨ x ∈ S ∧ y ∈ S ∧ path_component S x y
(**is** *?lhs = ?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*metis* (*no_types*) *path_component_mem*(*1*) *path_component_refl*)
next
assume *?rhs* **then show** *?lhs*
proof
assume *x ∉ S ∧ y ∉ S* **then show** *?lhs*
by (*metis Collect_empty_eq_bot path_component_eq_empty*)
next
assume *S*: *x ∈ S ∧ y ∈ S ∧ path_component S x y* **show** *?lhs*
by (*rule ext*) (*metis S path_component_trans path_component_sym*)
qed
qed

lemma *path_component_unique*:
assumes *x ∈ c c ⊆ S path_connected c*
⋀c′. ⟦x ∈ c′; c′ ⊆ S; path_connected c′⟧ ⟹ c′ ⊆ c
shows *path_component_set S x = c*
(**is** *?lhs = ?rhs*)
proof
show *?lhs ⊆ ?rhs*
using *assms*
by (*metis mem_Collect_eq path_component_refl path_component_subset path_connected_path_component subsetD*)
qed (*simp add*: *assms path_component_maximal*)

lemma *path_component_intermediate_subset*:
path_component_set u a ⊆ t ∧ t ⊆ u
⟹ path_component_set t a = path_component_set u a
by (*metis* (*no_types*) *path_component_mono path_component_path_component subset_antisym*)

lemma *complement_path_component_Union*:
fixes *x* :: *'a* :: *topological_space*
shows *S − path_component_set S x =*
⋃({*path_component_set S y*| *y*. *y ∈ S*} − {*path_component_set S x*})
proof −
have ∗: (*⋀x*. *x ∈ S − {a} ⟹ disjnt a x*) *⟹ ⋃S − a = ⋃*(*S − {a}*)
for *a*::*'a set* **and** *S*
by (*auto simp*: *disjnt_def*)
have *⋀y*. *y ∈* {*path_component_set S x* |*x*. *x ∈ S*} − {*path_component_set S x*}
⟹ disjnt (*path_component_set S x*) *y*

using *path_component_disjoint path_component_eq* **by** *fastforce*
then have $\bigcup \{path\_component\_set\ S\ x\ |x.\ x \in S\} - path\_component\_set\ S\ x$
$=$
$\bigcup (\{path\_component\_set\ S\ y\ |y.\ y \in S\} - \{path\_component\_set\ S\ x\})$
by (*meson* $*$)
then show *?thesis* **by** *simp*
qed

5.1.19 Path components

definition *path_component_of*
where $path\_component\_of\ X\ x\ y \equiv \exists g.\ pathin\ X\ g \wedge g\ 0 = x \wedge g\ 1 = y$

abbreviation *path_component_of_set*
where $path\_component\_of\_set\ X\ x \equiv Collect\ (path\_component\_of\ X\ x)$

definition $path\_components\_of :: 'a\ topology \Rightarrow 'a\ set\ set$
where $path\_components\_of\ X \equiv path\_component\_of\_set\ X\ `\ topspace\ X$

lemma *pathin_canon_iff*: $pathin\ (top\_of\_set\ T)\ g \longleftrightarrow path\ g \wedge g \in \{0..1\} \to T$
by (*simp add*: *path_def pathin_def image_subset_iff_funcset*)

lemma *path_component_of_canon_iff* [*simp*]:
$path\_component\_of\ (top\_of\_set\ T)\ a\ b \longleftrightarrow path\_component\ T\ a\ b$
by (*simp add*: *path_component_of_def pathin_canon_iff path_defs image_subset_iff_funcset*)

lemma *path_component_in_topspace*:
$path\_component\_of\ X\ x\ y \Longrightarrow x \in topspace\ X \wedge y \in topspace\ X$
by (*auto simp*: *path_component_of_def pathin_def continuous_map_def*)

lemma *path_component_of_refl*:
$path\_component\_of\ X\ x\ x \longleftrightarrow x \in topspace\ X$
by (*metis path_component_in_topspace path_component_of_def pathin_const*)

lemma *path_component_of_sym*:
assumes *path_component_of X x y*
shows *path_component_of X y x*
using *assms*
apply (*clarsimp simp*: *path_component_of_def pathin_def*)
apply (*rule_tac* $x{=}g \circ (\lambda t.\ 1 - t)$ **in** *exI*)
apply (*auto intro*!: *continuous_map_compose simp*: *continuous_map_in_subtopology continuous_on_op_minus*)
done

lemma *path_component_of_sym_iff*:
$path\_component\_of\ X\ x\ y \longleftrightarrow path\_component\_of\ X\ y\ x$
by (*metis path_component_of_sym*)

lemma *continuous_map_cases_le*:
assumes *contp*: *continuous_map X euclideanreal p*
and *contq*: *continuous_map X euclideanreal q*
and *contf*: *continuous_map* (*subtopology X* {*x*. *x* ∈ *topspace X* ∧ *p x* ≤ *q x*}) *Y f*
and *contg*: *continuous_map* (*subtopology X* {*x*. *x* ∈ *topspace X* ∧ *q x* ≤ *p x*}) *Y g*
and *fg*: ⋀*x*. ⟦*x* ∈ *topspace X*; *p x* = *q x*⟧ ⟹ *f x* = *g x*
shows *continuous_map X Y* (λ*x*. *if p x* ≤ *q x then f x else g x*)
proof −
have *continuous_map X Y* (λ*x*. *if q x* − *p x* ∈ {*0*..} *then f x else g x*)
proof (*rule continuous_map_cases_function*)
show *continuous_map X euclideanreal* (λ*x*. *q x* − *p x*)
by (*intro contp contq continuous_intros*)
show *continuous_map* (*subtopology X* {*x* ∈ *topspace X*. *q x* − *p x* ∈ *euclideanreal closure_of* {*0*..}}) *Y f*
by (*simp add*: *contf*)
show *continuous_map* (*subtopology X* {*x* ∈ *topspace X*. *q x* − *p x* ∈ *euclideanreal closure_of* (*topspace euclideanreal* − {*0*..})}) *Y g*
by (*simp add*: *contg flip*: *Compl_eq_Diff_UNIV*)
qed (*auto simp*: *fg*)
then show *?thesis*
by *simp*
qed

lemma *continuous_map_cases_lt*:
assumes *contp*: *continuous_map X euclideanreal p*
and *contq*: *continuous_map X euclideanreal q*
and *contf*: *continuous_map* (*subtopology X* {*x*. *x* ∈ *topspace X* ∧ *p x* ≤ *q x*}) *Y f*
and *contg*: *continuous_map* (*subtopology X* {*x*. *x* ∈ *topspace X* ∧ *q x* ≤ *p x*}) *Y g*
and *fg*: ⋀*x*. ⟦*x* ∈ *topspace X*; *p x* = *q x*⟧ ⟹ *f x* = *g x*
shows *continuous_map X Y* (λ*x*. *if p x* < *q x then f x else g x*)
proof −
have *continuous_map X Y* (λ*x*. *if q x* − *p x* ∈ {*0*<..} *then f x else g x*)
proof (*rule continuous_map_cases_function*)
show *continuous_map X euclideanreal* (λ*x*. *q x* − *p x*)
by (*intro contp contq continuous_intros*)
show *continuous_map* (*subtopology X* {*x* ∈ *topspace X*. *q x* − *p x* ∈ *euclideanreal closure_of* {*0*<..}}) *Y f*
by (*simp add*: *contf*)
show *continuous_map* (*subtopology X* {*x* ∈ *topspace X*. *q x* − *p x* ∈ *euclideanreal closure_of* (*topspace euclideanreal* − {*0*<..})}) *Y g*
by (*simp add*: *contg flip*: *Compl_eq_Diff_UNIV*)
qed (*auto simp*: *fg*)
then show *?thesis*
by *simp*
qed

lemma *path_component_of_trans*:
assumes *path_component_of X x y* **and** *path_component_of X y z*
shows *path_component_of X x z*
unfolding *path_component_of_def pathin_def*
proof −
let *?T01 = top_of_set {0..1::real}*
obtain *g1 g2* **where** *g1*: *continuous_map ?T01 X g1 x = g1 0 y = g1 1*
and *g2*: *continuous_map ?T01 X g2 g2 0 = g1 1 z = g2 1*
using *assms* **unfolding** *path_component_of_def pathin_def* **by** *blast*
let *?g = λx. if x ≤ 1/2 then (g1 ∘ (λt. 2 ∗ t)) x else (g2 ∘ (λt. 2 ∗ t −1)) x*
show *∃ g. continuous_map ?T01 X g ∧ g 0 = x ∧ g 1 = z*
proof (*intro exI conjI*)
show *continuous_map* (*subtopology euclideanreal {0..1}*) *X ?g*
proof (*intro continuous_map_cases_le continuous_map_compose*, *force*, *force*)
show *continuous_map* (*subtopology ?T01 {x ∈ topspace ?T01. x ≤ 1/2}*) *?T01* ((∗) *2*)
by (*auto simp*: *continuous_map_in_subtopology continuous_map_from_subtopology*)
have *continuous_map*
(*subtopology* (*top_of_set {0..1}*) {*x. 0 ≤ x ∧ x ≤ 1 ∧ 1 ≤ x ∗ 2*})
euclideanreal (*λt. 2 ∗ t − 1*)
by (*intro continuous_intros*) (*force intro*: *continuous_map_from_subtopology*)
then show *continuous_map* (*subtopology ?T01 {x ∈ topspace ?T01. 1/2 ≤ x}*) *?T01* (*λt. 2 ∗ t − 1*)
by (*force simp*: *continuous_map_in_subtopology*)
show (*g1 ∘* (∗) *2*) *x* = (*g2 ∘* (*λt. 2 ∗ t − 1*)) *x* **if** *x ∈ topspace ?T01 x = 1/2* **for** *x*
using *that* **by** (*simp add*: *g2*(*2*) *mult.commute continuous_map_from_subtopology*)
qed (*auto simp*: *g1 g2*)
qed (*auto simp*: *g1 g2*)
qed

lemma *path_component_of_mono*:
⟦*path_component_of* (*subtopology X S*) *x y*; *S ⊆ T*⟧ ⟹ *path_component_of* (*subtopology X T*) *x y*
unfolding *path_component_of_def*
by (*metis subsetD pathin_subtopology*)

lemma *path_component_of*:
path_component_of X x y ⟷ (*∃ T. path_connectedin X T ∧ x ∈ T ∧ y ∈ T*)
(**is** *?lhs = ?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*metis atLeastAtMost_iff image_eqI order_refl path_component_of_def path_connectedin_path_image zero_le_one*)
next
assume *?rhs* **then show** *?lhs*
by (*metis path_component_of_def path_connectedin*)
qed

lemma *path_component_of_set*:
path_component_of X x y ⟷ (∃ *g*. *pathin X g* ∧ *g 0* = *x* ∧ *g 1* = *y*)
by (*auto simp*: *path_component_of_def*)

lemma *path_component_of_subset_topspace*:
Collect(*path_component_of X x*) ⊆ *topspace X*
using *path_component_in_topspace* **by** *fastforce*

lemma *path_component_of_eq_empty*:
Collect(*path_component_of X x*) = {} ⟷ (*x* ∉ *topspace X*)
using *path_component_in_topspace path_component_of_refl* **by** *fastforce*

lemma *path_connected_space_iff_path_component*:
path_connected_space X ⟷ (∀ *x* ∈ *topspace X*. ∀ *y* ∈ *topspace X*. *path_component_of X x y*)
by (*simp add*: *path_component_of path_connected_space_subconnected*)

lemma *path_connected_space_imp_path_component_of*:
⟦*path_connected_space X*; *a* ∈ *topspace X*; *b* ∈ *topspace X*⟧
⟹ *path_component_of X a b*
by (*simp add*: *path_connected_space_iff_path_component*)

lemma *path_connected_space_path_component_set*:
path_connected_space X ⟷ (∀ *x* ∈ *topspace X*. *Collect*(*path_component_of X x*) = *topspace X*)
using *path_component_of_subset_topspace path_connected_space_iff_path_component*
by *fastforce*

lemma *path_component_of_maximal*:
⟦*path_connectedin X s*; *x* ∈ *s*⟧ ⟹ *s* ⊆ *Collect*(*path_component_of X x*)
using *path_component_of* **by** *fastforce*

lemma *path_component_of_equiv*:
path_component_of X x y ⟷ *x* ∈ *topspace X* ∧ *y* ∈ *topspace X* ∧ *path_component_of X x* = *path_component_of X y*
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
apply (*simp add*: *fun_eq_iff path_component_in_topspace*)
apply (*meson path_component_of_sym path_component_of_trans*)
done
qed (*simp add*: *path_component_of_refl*)

lemma *path_component_of_disjoint*:
disjnt (*Collect* (*path_component_of X x*)) (*Collect* (*path_component_of X y*)) ⟷
~(*path_component_of X x y*)

by (*force simp*: *disjnt_def path_component_of_eq_empty path_component_of_equiv*)

lemma *path_component_of_eq*:
path_component_of X x = path_component_of X y ⟷
(x ∉ topspace X) ∧ (y ∉ topspace X) ∨
x ∈ topspace X ∧ y ∈ topspace X ∧ path_component_of X x y
by (*metis Collect_empty_eq_bot path_component_of_eq_empty path_component_of_equiv*)

lemma *path_component_of_aux*:
path_component_of X x y
⟹ path_component_of (subtopology X (Collect (path_component_of X x))) x y
by (*meson path_component_of path_component_of_maximal path_connectedin_subtopology*)

lemma *path_connectedin_path_component_of*:
path_connectedin X (Collect (path_component_of X x))
proof −
have *topspace (subtopology X (path_component_of_set X x)) = path_component_of_set X x*
by (*meson path_component_of_subset_topspace topspace_subtopology_subset*)
then have *path_connected_space (subtopology X (path_component_of_set X x))*
by (*metis* (*full_types*) *path_component_of_aux mem_Collect_eq path_component_of_equiv path_connected_space_iff_path_component*)
then show *?thesis*
by (*simp add*: *path_component_of_subset_topspace path_connectedin_def*)
qed

lemma *path_connectedin_euclidean* [*simp*]:
path_connectedin euclidean S ⟷ path_connected S
by (*auto simp*: *path_connectedin_def path_connected_space_iff_path_component path_connected_component*)

lemma *path_connected_space_euclidean_subtopology* [*simp*]:
path_connected_space(subtopology euclidean S) ⟷ path_connected S
using *path_connectedin_topspace* **by** *force*

lemma *Union_path_components_of*:
⋃(path_components_of X) = topspace X
by (*auto simp*: *path_components_of_def path_component_of_equiv*)

lemma *path_components_of_maximal*:
⟦C ∈ path_components_of X; path_connectedin X S; ~ disjnt C S⟧ ⟹ S ⊆ C
by (*smt* (*verit*, *ccfv_SIG*) *disjnt_iff imageE mem_Collect_eq path_component_of_equiv*
path_component_of_maximal path_components_of_def)

lemma *pairwise_disjoint_path_components_of*:
pairwise disjnt (path_components_of X)

by (*auto simp*: *path_components_of_def pairwise_def path_component_of_disjoint path_component_of_equiv*)

lemma *complement_path_components_of_Union*:
C ∈ *path_components_of X* ⟹ *topspace X* − *C* = ⋃(*path_components_of X* − {*C*})
by (*metis Union_path_components_of bot.extremum ccpo_Sup_singleton diff_Union_pairwise_disjoint insert_subsetI pairwise_disjoint_path_components_of*)

lemma *nonempty_path_components_of*:
assumes *C* ∈ *path_components_of X* **shows** *C* ≠ {}
by (*metis assms imageE path_component_of_eq_empty path_components_of_def*)

lemma *path_components_of_subset*: *C* ∈ *path_components_of X* ⟹ *C* ⊆ *topspace X*
by (*auto simp*: *path_components_of_def path_component_of_equiv*)

lemma *path_connectedin_path_components_of*:
C ∈ *path_components_of X* ⟹ *path_connectedin X C*
by (*auto simp*: *path_components_of_def path_connectedin_path_component_of*)

lemma *path_component_in_path_components_of*:
Collect (*path_component_of X a*) ∈ *path_components_of X* ⟷ *a* ∈ *topspace X*
by (*metis imageI nonempty_path_components_of path_component_of_eq_empty path_components_of_def*)

lemma *path_connectedin_Union*:
assumes 𝒜: ⋀*S*. *S* ∈ 𝒜 ⟹ *path_connectedin X S* ⋂𝒜 ≠ {}
shows *path_connectedin X* (⋃𝒜)
proof −
obtain *a* **where** ⋀*S*. *S* ∈ 𝒜 ⟹ *a* ∈ *S*
using *assms* **by** *blast*
then have ⋀*x*. *x* ∈ *topspace* (*subtopology X* (⋃𝒜)) ⟹ *path_component_of* (*subtopology X* (⋃𝒜)) *a x*
by *simp* (*meson Union_upper* 𝒜 *path_component_of path_connectedin_subtopology*)
then show *?thesis*
using 𝒜 **unfolding** *path_connectedin_def*
by (*metis Sup_le_iff path_component_of_equiv path_connected_space_iff_path_component*)
qed

lemma *path_connectedin_Un*:
⟦*path_connectedin X S*; *path_connectedin X T*; *S* ∩ *T* ≠ {}⟧
⟹ *path_connectedin X* (*S* ∪ *T*)
by (*blast intro*: *path_connectedin_Union* [*of* {*S*,*T*}, *simplified*])

lemma *path_connected_space_iff_components_eq*:
path_connected_space X ⟷

(∀ C ∈ path_components_of X. ∀ C′ ∈ path_components_of X. C = C′)
unfolding path_components_of_def
proof (intro iffI ballI)
assume ∀ C ∈ path_component_of_set X ‘ topspace X.
∀ C′ ∈ path_component_of_set X ‘ topspace X. C = C′
then show path_connected_space X
using path_component_of_refl path_connected_space_iff_path_component **by**
fastforce
qed (auto simp: path_connected_space_path_component_set)

lemma path_components_of_eq_empty:
path_components_of X = {} ⟷ X = trivial_topology
by (metis image_is_empty path_components_of_def subtopology_eq_discrete_topology_empty)

lemma path_components_of_empty_space:
path_components_of trivial_topology = {}
by (simp add: path_components_of_eq_empty)

lemma path_components_of_subset_singleton:
path_components_of X ⊆ {S} ⟷
path_connected_space X ∧ (topspace X = {} ∨ topspace X = S)
proof (cases topspace X = {})
case True
then show ?thesis
by (auto simp: path_components_of_empty_space path_connected_space_topspace_empty)
next
case False
have (path_components_of X = {S}) ⟷ (path_connected_space X ∧ topspace
X = S)
by (metis False Set.set_insert ex_in_conv insert_iff path_component_in_path_components_of

path_connected_space_iff_components_eq path_connected_space_path_component_set)
with False **show** ?thesis
by (simp add: path_components_of_eq_empty subset_singleton_iff)
qed

lemma path_connected_space_iff_components_subset_singleton:
path_connected_space X ⟷ (∃ a. path_components_of X ⊆ {a})
by (simp add: path_components_of_subset_singleton)

lemma path_components_of_eq_singleton:
path_components_of X = {S} ⟷ path_connected_space X ∧ topspace X ≠
{} ∧ S = topspace X
by (metis cSup_singleton insert_not_empty path_components_of_subset_singleton
subset_singleton_iff)

lemma path_components_of_path_connected_space:
path_connected_space X ⟹ path_components_of X = (if topspace X = {}
then {} else {topspace X})

by (*simp add*: *path_components_of_eq_empty path_components_of_eq_singleton*)

lemma *path_component_subset_connected_component_of*:
path_component_of_set X x ⊆ *connected_component_of_set X x*
proof (*cases x* ∈ *topspace X*)
case *True*
then show *?thesis*
by (*simp add*: *connected_component_of_maximal path_component_of_refl path_connectedin_imp_connectedin path_connectedin_path_component_of*)
next
case *False*
then show *?thesis*
using *path_component_of_eq_empty* **by** *fastforce*
qed

lemma *exists_path_component_of_superset*:
assumes *S*: *path_connectedin X S* **and** *ne*: *topspace X* ≠ {}
obtains *C* **where** *C* ∈ *path_components_of X S* ⊆ *C*
by (*metis S ne ex_in_conv path_component_in_path_components_of path_component_of_maximal path_component_of_subset_topspace subset_eq that*)

lemma *path_component_of_eq_overlap*:
path_component_of X x = *path_component_of X y* ⟷
(*x* ∉ *topspace X*) ∧ (*y* ∉ *topspace X*) ∨
Collect (*path_component_of X x*) ∩ *Collect* (*path_component_of X y*) ≠ {}
by (*metis disjnt_def empty_iff inf_bot_right mem_Collect_eq path_component_of_disjoint path_component_of_eq path_component_of_eq_empty*)

lemma *path_component_of_nonoverlap*:
Collect (*path_component_of X x*) ∩ *Collect* (*path_component_of X y*) = {}
⟷
(*x* ∉ *topspace X*) ∨ (*y* ∉ *topspace X*) ∨
path_component_of X x ≠ *path_component_of X y*
by (*metis inf.idem path_component_of_eq_empty path_component_of_eq_overlap*)

lemma *path_component_of_overlap*:
Collect (*path_component_of X x*) ∩ *Collect* (*path_component_of X y*) ≠ {}
⟷
x ∈ *topspace X* ∧ *y* ∈ *topspace X* ∧ *path_component_of X x* = *path_component_of X y*
by (*meson path_component_of_nonoverlap*)

lemma *path_components_of_disjoint*:
⟦*C* ∈ *path_components_of X*; *C′* ∈ *path_components_of X*⟧ ⟹ *disjnt C C′* ⟷ *C* ≠ *C′*
by (*auto simp*: *path_components_of_def path_component_of_disjoint path_component_of_equiv*)

lemma *path_components_of_overlap*:
⟦*C* ∈ *path_components_of X*; *C′* ∈ *path_components_of X*⟧ ⟹ *C* ∩ *C′* ≠ {}

$\longleftrightarrow$ $C = C'$
by (*auto simp*: *path_components_of_def path_component_of_equiv*)

lemma *path_component_of_unique*:
$[\![x \in C$; *path_connectedin X C*; $\bigwedge C'$. $[\![x \in C'$; *path_connectedin X C'*$]\!] \Longrightarrow C' \subseteq C]\!]$
$\Longrightarrow$ *Collect* (*path_component_of X x*) = *C*
by (*meson subsetD eq_iff path_component_of_maximal path_connectedin_path_component_of*)

lemma *path_component_of_discrete_topology* [*simp*]:
Collect (*path_component_of* (*discrete_topology U*) *x*) = (*if x* $\in$ *U then* {*x*} *else* {})
proof −
have $\bigwedge C'$. $[\![x \in C'$; *path_connectedin* (*discrete_topology U*) *C'*$]\!] \Longrightarrow C' \subseteq \{x\}$
by (*metis path_connectedin_discrete_topology subsetD singletonD*)
then have *x* $\in$ *U* $\Longrightarrow$ *Collect* (*path_component_of* (*discrete_topology U*) *x*) = {*x*}
by (*simp add*: *path_component_of_unique*)
then show *?thesis*
using *path_component_in_topspace* **by** *fastforce*
qed

lemma *path_component_of_discrete_topology_iff* [*simp*]:
path_component_of (*discrete_topology U*) *x y* $\longleftrightarrow$ *x* $\in$ *U* $\wedge$ *y*=*x*
by (*metis empty_iff insertI1 mem_Collect_eq path_component_of_discrete_topology singletonD*)

lemma *path_components_of_discrete_topology* [*simp*]:
path_components_of (*discrete_topology U*) = (λx. {*x*}) ‘ *U*
by (*auto simp*: *path_components_of_def image_def fun_eq_iff*)

lemma *homeomorphic_map_path_component_of*:
assumes *f*: *homeomorphic_map X Y f* **and** *x*: *x* $\in$ *topspace X*
shows *Collect* (*path_component_of Y* (*f x*)) = *f* ‘ *Collect*(*path_component_of X x*)
proof −
obtain *g* **where** *g*: *homeomorphic_maps X Y f g*
using *f homeomorphic_map_maps* **by** *blast*
show *?thesis*
proof
have *Collect* (*path_component_of Y* (*f x*)) $\subseteq$ *topspace Y*
by (*simp add*: *path_component_of_subset_topspace*)
moreover have *g* ‘ *Collect*(*path_component_of Y* (*f x*)) $\subseteq$ *Collect* (*path_component_of X* (*g* (*f x*)))
using *f g x* **unfolding** *homeomorphic_maps_def*
by (*metis image_Collect_subsetI image_eqI mem_Collect_eq path_component_of_equiv path_component_of_maximal*
path_connectedin_continuous_map_image path_connectedin_path_component_of)
ultimately show *Collect* (*path_component_of Y* (*f x*)) $\subseteq$ *f* ‘ *Collect* (*path_component_of*

X x)
using g x unfolding homeomorphic_maps_def continuous_map_def image_iff subset_iff
by metis
show f ‘ Collect (path_component_of X x) ⊆ Collect (path_component_of Y (f x))
proof (rule path_component_of_maximal)
show path_connectedin Y (f ‘ Collect (path_component_of X x))
by (meson f homeomorphic_map_path_connectedness_eq path_connectedin_path_component_of)
qed (simp add: path_component_of_refl x)
qed
qed

lemma homeomorphic_map_path_components_of:
assumes homeomorphic_map X Y f
shows path_components_of Y = (image f) ‘ (path_components_of X)
unfolding path_components_of_def homeomorphic_imp_surjective_map [OF assms, symmetric]
using assms homeomorphic_map_path_component_of by fastforce

5.1.20 Paths and path-connectedness

lemma path_connected_space_quotient_map_image:
⟦quotient_map X Y q; path_connected_space X⟧ ⟹ path_connected_space Y
by (metis path_connectedin_continuous_map_image path_connectedin_topspace quotient_imp_continuous_map quotient_imp_surjective_map)

lemma path_connected_space_retraction_map_image:
⟦retraction_map X Y r; path_connected_space X⟧ ⟹ path_connected_space Y
using path_connected_space_quotient_map_image retraction_imp_quotient_map
by blast

lemma path_connected_space_prod_topology:
path_connected_space(prod_topology X Y) ⟷
topspace(prod_topology X Y) = {} ∨ path_connected_space X ∧ path_connected_space Y
proof (cases topspace(prod_topology X Y) = {})
case True
then show ?thesis
using path_connected_space_topspace_empty by force
next
case False
have path_connected_space (prod_topology X Y)
if X: path_connected_space X and Y: path_connected_space Y
proof (clarsimp simp: path_connected_space_def)
fix x y x′ y′
assume x ∈ topspace X and y ∈ topspace Y and x′ ∈ topspace X and y′ ∈ topspace Y
obtain f where pathin X f f 0 = x f 1 = x′

by (*meson X ‹x ∈ topspace X› ‹x′ ∈ topspace X› path_connected_space_def*)
obtain *g* **where** *pathin Y g g 0 = y g 1 = y′*
by (*meson Y ‹y ∈ topspace Y› ‹y′ ∈ topspace Y› path_connected_space_def*)
show ∃*h*. *pathin* (*prod_topology X Y*) *h* ∧ *h 0* = (*x*,*y*) ∧ *h 1* = (*x′*,*y′*)
proof (*intro exI conjI*)
show *pathin* (*prod_topology X Y*) (λ*t*. (*f t*, *g t*))
using ‹*pathin X f*› ‹*pathin Y g*› **by** (*simp add*: *continuous_map_paired pathin_def*)
show (λ*t*. (*f t*, *g t*)) *0* = (*x*, *y*)
using ‹*f 0* = *x*› ‹*g 0* = *y*› **by** *blast*
show (λ*t*. (*f t*, *g t*)) *1* = (*x′*, *y′*)
using ‹*f 1* = *x′*› ‹*g 1* = *y′*› **by** *blast*
qed
qed
then show *?thesis*
by (*metis False path_connected_space_quotient_map_image prod_topology_trivial1 prod_topology_trivial2*
quotient_map_fst quotient_map_snd topspace_discrete_topology)
qed

lemma *path_connectedin_Times*:
path_connectedin (*prod_topology X Y*) (*S* × *T*) ⟷
S = {} ∨ *T* = {} ∨ *path_connectedin X S* ∧ *path_connectedin Y T*
by (*auto simp add*: *path_connectedin_def subtopology_Times path_connected_space_prod_topology*)

5.1.21 Path components

lemma *path_component_of_subtopology_eq*:
path_component_of (*subtopology X U*) *x* = *path_component_of X x* ⟷ *path_component_of_set X x* ⊆ *U*
(**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⟹ *?rhs*
by (*metis path_connectedin_path_component_of path_connectedin_subtopology*)
next
show *?rhs* ⟹ *?lhs*
unfolding *fun_eq_iff*
by (*metis path_connectedin_subtopology path_component_of path_component_of_aux path_component_of_mono*)
qed

lemma *path_components_of_subtopology*:
assumes *C* ∈ *path_components_of X C* ⊆ *U*
shows *C* ∈ *path_components_of* (*subtopology X U*)
using *assms path_component_of_refl path_component_of_subtopology_eq topspace_subtopology*
by (*smt* (*verit*) *imageE path_component_in_path_components_of path_components_of_def*)

lemma *path_imp_connected_component_of*:
path_component_of X x y ⟹ *connected_component_of X x y*

by (*metis in_mono mem_Collect_eq path_component_subset_connected_component_of*)

lemma *finite_path_components_of_finite*:
finite(*topspace X*) ⟹ *finite*(*path_components_of X*)
by (*simp add*: *Union_path_components_of finite_UnionD*)

lemma *path_component_of_continuous_image*:
⟦*continuous_map X X' f*; *path_component_of X x y*⟧ ⟹ *path_component_of X' (f x) (f y)*
by (*meson image_eqI path_component_of path_connectedin_continuous_map_image*)

lemma *path_component_of_pair* [*simp*]:
path_component_of_set (*prod_topology X Y*) (*x*,*y*) =
path_component_of_set X x × *path_component_of_set Y y* (**is** *?lhs* = *?rhs*)
proof (*cases ?lhs* = {})
case *True*
then show *?thesis*
by (*metis Sigma_empty1 Sigma_empty2 mem_Sigma_iff path_component_of_eq_empty topspace_prod_topology*)
next
case *False*
then have *path_component_of X x x path_component_of Y y y*
using *path_component_of_eq_empty path_component_of_refl* **by** *fastforce*+
moreover
have *path_connectedin* (*prod_topology X Y*) (*path_component_of_set X x* × *path_component_of_set Y y*)
by (*metis path_connectedin_Times path_connectedin_path_component_of*)
moreover have *path_component_of X x a path_component_of Y y b*
if (*x*, *y*) ∈ *C'* (*a*,*b*) ∈ *C'* **and** *path_connectedin* (*prod_topology X Y*) *C'* **for** *C' a b*
by (*smt* (*verit*, *best*) *that continuous_map_fst continuous_map_snd fst_conv snd_conv path_component_of path_component_of_continuous_image*)+
ultimately show *?thesis*
by (*intro path_component_of_unique*) *auto*
qed

lemma *path_components_of_prod_topology*:
path_components_of (*prod_topology X Y*) =
(λ(*C*,*D*). *C* × *D*) ‘ (*path_components_of X* × *path_components_of Y*)
by (*force simp add*: *image_iff path_components_of_def*)

lemma *path_components_of_prod_topology'*:
path_components_of (*prod_topology X Y*) =
{*C* × *D* |*C D*. *C* ∈ *path_components_of X* ∧ *D* ∈ *path_components_of Y*}
by (*auto simp*: *path_components_of_prod_topology*)

lemma *path_component_of_product_topology*:
path_component_of_set (*product_topology X I*) *f* =
(*if f* ∈ *extensional I then PiE I* (λ*i*. *path_component_of_set* (*X i*) (*f i*)) *else*

```
{})
    (is ?lhs = ?rhs)
proof (cases path_component_of_set (product_topology X I) f = {})
  case True
  then show ?thesis
     by (smt (verit) PiE_eq_empty_iff PiE_iff path_component_of_eq_empty
topspace_product_topology)
next
  case False
  then have [simp]: f ∈ extensional I
   by (auto simp: path_component_of_eq_empty PiE_iff path_component_of_equiv)
  show ?thesis
  proof (intro path_component_of_unique)
    show f ∈ ?rhs
     using False path_component_of_eq_empty path_component_of_refl by force
    show path_connectedin (product_topology X I) (if f ∈ extensional I then Π_E
i∈I. path_component_of_set (X i) (f i) else {})
     by (simp add: path_connectedin_PiE path_connectedin_path_component_of)
    fix C'
    assume f ∈ C' and C': path_connectedin (product_topology X I) C'
    show C' ⊆ ?rhs
    proof −
      have C' ⊆ extensional I
        using PiE_def C' path_connectedin_subset_topspace by fastforce
      with ‹f ∈ C'› C' show ?thesis
        apply (clarsimp simp: PiE_iff subset_iff)
      by (smt (verit, ccfv_threshold) continuous_map_product_projection path_component_of
path_component_of_continuous_image)
    qed
  qed
qed

lemma path_components_of_product_topology:
  path_components_of (product_topology X I) =
    {PiE I B |B. ∀ i ∈ I. B i ∈ path_components_of(X i)}  (is ?lhs=?rhs)
proof
  show ?lhs ⊆ ?rhs
    apply (simp add: path_components_of_def image_subset_iff)
   by (smt (verit, best) PiE_iff image_eqI path_component_of_product_topology)
next
  show ?rhs ⊆ ?lhs
  proof
    fix F
    assume F ∈ ?rhs
    then obtain B where B: F = Pi_E I B
      and ∀ i∈I. ∃ x∈topspace (X i). B i = path_component_of_set (X i) x
      by (force simp add: path_components_of_def image_iff)
    then obtain f where ftop: ⋀i. i ∈ I ⟹ f i ∈ topspace (X i)
      and BF: ⋀i. i ∈ I ⟹ B i = path_component_of_set (X i) (f i)
```

```
    by metis
  then have F = path_component_of_set (product_topology X I) (restrict f I)
   by (metis (mono_tags, lifting) B PiE_cong path_component_of_product_topology
restrict_apply' restrict_extensional)
  then show F ∈ ?lhs
    by (simp add: ftop path_component_in_path_components_of)
 qed
qed
```

5.1.22 Sphere is path-connected

```
lemma path_connected_punctured_universe:
 assumes 2 ≤ DIM('a::euclidean_space)
 shows path_connected (− {a::'a})
proof −
 let ?A = {x::'a. ∃ i∈Basis. x · i < a · i}
 let ?B = {x::'a. ∃ i∈Basis. a · i < x · i}

 have A: path_connected ?A
   unfolding Collect_bex_eq
 proof (rule path_connected_UNION)
   fix i :: 'a
   assume i ∈ Basis
   then show (∑ i∈Basis. (a · i − 1)*R i) ∈ {x::'a. x · i < a · i}
     by simp
   show path_connected {x. x · i < a · i}
     using convex_imp_path_connected [OF convex_halfspace_lt, of i a · i]
     by (simp add: inner_commute)
 qed
 have B: path_connected ?B
   unfolding Collect_bex_eq
 proof (rule path_connected_UNION)
   fix i :: 'a
   assume i ∈ Basis
   then show (∑ i∈Basis. (a · i + 1) *R i) ∈ {x::'a. a · i < x · i}
     by simp
   show path_connected {x. a · i < x · i}
     using convex_imp_path_connected [OF convex_halfspace_gt, of a · i i]
     by (simp add: inner_commute)
 qed
 obtain S :: 'a set where S ⊆ Basis and card S = Suc (Suc 0)
  using obtain_subset_with_card_n[OF assms] by (force simp add: eval_nat_numeral)
 then obtain b0 b1 :: 'a where b0 ∈ Basis and b1 ∈ Basis and b0 ≠ b1
   unfolding card_Suc_eq by auto
 then have a + b0 − b1 ∈ ?A ∩ ?B
   by (auto simp: inner_simps inner_Basis)
 then have ?A ∩ ?B ≠ {}
   by fast
 with A B have path_connected (?A ∪ ?B)
```

by (*rule path_connected_Un*)
also have *?A ∪ ?B = {x. ∃i∈Basis. x · i ≠ a · i}*
unfolding *neq_iff bex_disj_distrib Collect_disj_eq* **..**
also have *... = {x. x ≠ a}*
unfolding *euclidean_eq_iff* [**where** *'a='a*]
by (*simp add*: *Bex_def*)
also have *... = − {a}*
by *auto*
finally show *?thesis* **.**
qed

corollary *connected_punctured_universe*:
2 ≤ DIM('N::euclidean_space) ⟹ connected(− {a::'N})
by (*simp add*: *path_connected_punctured_universe path_connected_imp_connected*)

proposition *path_connected_sphere*:
fixes *a :: 'a :: euclidean_space*
assumes *2 ≤ DIM('a)*
shows *path_connected(sphere a r)*
proof (*cases r 0::real rule*: *linorder_cases*)
case *less*
then show *?thesis*
by *simp*
next
case *equal*
then show *?thesis*
by *simp*
next
case *greater*
then have *eq*: *(sphere (0::'a) r) = (λx. (r / norm x) *R x) ‘ (− {0::'a})*
by (*force simp*: *image_iff split*: *if_split_asm*)
have *continuous_on (− {0::'a}) (λx. (r / norm x) *R x)*
by (*intro continuous_intros*) *auto*
then have *path_connected ((λx. (r / norm x) *R x) ‘ (− {0::'a}))*
by (*intro path_connected_continuous_image path_connected_punctured_universe assms*)
with *eq* **have** *path_connected (sphere (0::'a) r)*
by *auto*
then have *path_connected((+) a ‘ (sphere (0::'a) r))*
by (*simp add*: *path_connected_translation*)
then show *?thesis*
by (*metis add.right_neutral sphere_translation*)
qed

lemma *connected_sphere*:
fixes *a :: 'a :: euclidean_space*
assumes *2 ≤ DIM('a)*
shows *connected(sphere a r)*
using *path_connected_sphere* [*OF assms*]

by (*simp add*: *path_connected_imp_connected*)

corollary *path_connected_complement_bounded_convex*:
fixes $S :: 'a :: euclidean\_space\ set$
assumes *bounded S convex S* **and** *2*: $2 \leq DIM('a)$
shows *path_connected* $(-S)$
proof (*cases* $S = \{\}$)
case *True* **then show** *?thesis*
using *convex_imp_path_connected* **by** *auto*
next
case *False*
then obtain *a* **where** $a \in S$ **by** *auto*
have § [*rule_format*]: $\forall y{\in}S.\ \forall u.\ 0 \leq u \wedge u \leq 1 \longrightarrow (1 - u) *_R a + u *_R y \in S$
using ‹*convex S*› ‹$a \in S$› **by** (*simp add*: *convex_alt*)
{ fix *x y* **assume** $x \notin S\ y \notin S$
then have $x \neq a\ y \neq a$ **using** ‹$a \in S$› **by** *auto*
then have *bxy*: *bounded*(*insert x* (*insert y S*))
by (*simp add*: ‹*bounded S*›)
then obtain *B*::*real* **where** *B*: $0 < B$ **and** *Bx*: *norm* $(a - x) < B$ **and** *By*: *norm* $(a - y) < B$
and $S \subseteq ball\ a\ B$
using *bounded_subset_ballD* [*OF bxy*, *of a*] **by** (*auto simp*: *dist_norm*)
define *C* **where** $C = B\ /\ norm(x - a)$
let *?Cxa* $= a + C *_R (x - a)$
{ fix *u*
assume *u*: $(1 - u) *_R x + u *_R$ *?Cxa* $\in S$ **and** $0 \leq u\ u \leq 1$
have *CC*: $1 \leq 1 + (C - 1) * u$
using ‹$x \neq a$› ‹$0 \leq u$› *Bx*
by (*auto simp add*: *C_def norm_minus_commute*)
have $*$: $\bigwedge v.\ (1 - u) *_R x + u *_R (a + v *_R (x - a)) = a + (1 + (v - 1) * u) *_R (x - a)$
by (*simp add*: *algebra_simps*)
have $a + ((1\ /\ (1 + C * u - u)) *_R x + ((u\ /\ (1 + C * u - u)) *_R a + (C * u\ /\ (1 + C * u - u)) *_R x)) =$
$(1 + (u\ /\ (1 + C * u - u))) *_R a + ((1\ /\ (1 + C * u - u)) + (C * u\ /\ (1 + C * u - u))) *_R x$
by (*simp add*: *algebra_simps*)
also have $\ldots = (1 + (u\ /\ (1 + C * u - u))) *_R a + (1 + (u\ /\ (1 + C * u - u))) *_R x$
using *CC* **by** (*simp add*: *field_simps*)
also have $\ldots = x + (1 + (u\ /\ (1 + C * u - u))) *_R a + (u\ /\ (1 + C * u - u)) *_R x$
by (*simp add*: *algebra_simps*)
also have $\ldots = x + ((1\ /\ (1 + C * u - u)) *_R a +$
$((u\ /\ (1 + C * u - u)) *_R x + (C * u\ /\ (1 + C * u - u)) *_R a))$
using *CC* **by** (*simp add*: *field_simps*) (*simp add*: *add_divide_distrib scaleR_add_left*)

```
    finally have xeq: (1 - 1 / (1 + (C - 1) * u)) *R a + (1 / (1 + (C - 1)
* u)) *R (a + (1 + (C - 1) * u) *R (x - a)) = x
      by (simp add: algebra_simps)
    have False
      using § [of a + (1 + (C - 1) * u) *R (x - a) 1 / (1 + (C - 1) * u)]
      using u ‹x ≠ a› ‹x ∉ S› ‹0 ≤ u› CC
      by (auto simp: xeq *)
  }
  then have pcx: path_component (- S) x ?Cxa
    by (force simp: closed_segment_def intro!: path_component_linepath)
  define D where D = B / norm(y - a)  — massive duplication with the proof
above
  let ?Dya = a + D *R (y - a)
  { fix u
    assume u: (1 - u) *R y + u *R ?Dya ∈ S and 0 ≤ u u ≤ 1
    have DD: 1 ≤ 1 + (D - 1) * u
      using ‹y ≠ a› ‹0 ≤ u› By
      by (auto simp add: D_def norm_minus_commute)
    have *: ⋀v. (1 - u) *R y + u *R (a + v *R (y - a)) = a + (1 + (v - 1)
* u) *R (y - a)
      by (simp add: algebra_simps)
    have a + ((1 / (1 + D * u - u)) *R y + ((u / (1 + D * u - u)) *R a +
(D * u / (1 + D * u - u)) *R y)) =
          (1 + (u / (1 + D * u - u))) *R a + ((1 / (1 + D * u - u)) + (D * u
/ (1 + D * u - u))) *R y
      by (simp add: algebra_simps)
    also have ... = (1 + (u / (1 + D * u - u))) *R a + (1 + (u / (1 + D *
u - u))) *R y
      using DD by (simp add: field_simps)
    also have ... = y + (1 + (u / (1 + D * u - u))) *R a + (u / (1 + D * u
- u)) *R y
      by (simp add: algebra_simps)
    also have ... = y + ((1 / (1 + D * u - u)) *R a +
          ((u / (1 + D * u - u)) *R y + (D * u / (1 + D * u - u)) *R a))
        using DD by (simp add: field_simps) (simp add: add_divide_distrib
scaleR_add_left)
    finally have xeq: (1 - 1 / (1 + (D - 1) * u)) *R a + (1 / (1 + (D - 1)
* u)) *R (a + (1 + (D - 1) * u) *R (y - a)) = y
      by (simp add: algebra_simps)
    have False
      using § [of a + (1 + (D - 1) * u) *R (y - a) 1 / (1 + (D - 1) * u)]
      using u ‹y ≠ a› ‹y ∉ S› ‹0 ≤ u› DD
      by (auto simp: xeq *)
  }
  then have pdy: path_component (- S) y ?Dya
    by (force simp: closed_segment_def intro!: path_component_linepath)
  have pyx: path_component (- S) ?Dya ?Cxa
  proof (rule path_component_of_subset)
    show sphere a B ⊆ - S
```

```
      using ‹S ⊆ ball a B› by (force simp: ball_def dist_norm norm_minus_commute)
      have aB: ?Dya ∈ sphere a B ?Cxa ∈ sphere a B
        using ‹x ≠ a› using ‹y ≠ a› B by (auto simp: dist_norm C_def D_def)
      then show path_component (sphere a B) ?Dya ?Cxa
        using path_connected_sphere [OF 2] path_connected_component by blast
    qed
    have path_component (− S) x y
      by (metis path_component_trans path_component_sym pcx pdy pyx)
  }
  then show ?thesis
    by (auto simp: path_connected_component)
qed

lemma connected_complement_bounded_convex:
    fixes S :: 'a :: euclidean_space set
    assumes bounded S convex S 2 ≤ DIM('a)
      shows  connected (− S)
 using path_connected_complement_bounded_convex [OF assms] path_connected_imp_connected
by blast

lemma connected_diff_ball:
    fixes S :: 'a :: euclidean_space set
    assumes connected S cball a r ⊆ S 2 ≤ DIM('a)
      shows connected (S − ball a r)
proof (rule connected_diff_open_from_closed [OF ball_subset_cball])
  show connected (cball a r − ball a r)
    using assms connected_sphere by (auto simp: cball_diff_eq_sphere)
qed (auto simp: assms dist_norm)

proposition connected_open_delete:
  assumes open S connected S and 2: 2 ≤ DIM('N::euclidean_space)
    shows connected(S − {a::'N})
proof (cases a ∈ S)
  case True
  with ‹open S› obtain ε where ε > 0 and ε: cball a ε ⊆ S
    using open_contains_cball_eq by blast
  define b where b ≡ a + ε *_R (SOME i. i ∈ Basis)
  have dist a b = ε
    by (simp add: b_def dist_norm SOME_Basis ‹0 < ε› less_imp_le)
  with ε have b ∈ ⋂{S − ball a r |r. 0 < r ∧ r < ε}
    by auto
  then have nonemp: (⋂{S − ball a r |r. 0 < r ∧ r < ε}) = {} ⟹ False
    by auto
  have con: ⋀r. r < ε ⟹ connected (S − ball a r)
    using ε by (force intro: connected_diff_ball [OF ‹connected S› _ 2])
  have x ∈ ⋃{S − ball a r |r. 0 < r ∧ r < ε} if x ∈ S − {a} for x
    using that ‹0 < ε›
    by (intro UnionI [of S − ball a (min ε (dist a x) / 2)]) auto
  then have S − {a} = ⋃{S − ball a r | r. 0 < r ∧ r < ε}
```

by *auto*
then show *?thesis*
by (*auto intro*: *connected_Union con dest*!: *nonemp*)
next
case *False* **then show** *?thesis*
by (*simp add*: ‹*connected S*›)
qed

corollary *path_connected_open_delete*:
assumes *open S connected S* **and** *2*: *2* ≤ *DIM*('*N*::*euclidean_space*)
shows *path_connected*(*S* − {*a*::'*N*})
by (*simp add*: *assms connected_open_delete connected_open_path_connected open_delete*)

corollary *path_connected_punctured_ball*:
2 ≤ *DIM*('*N*::*euclidean_space*) ⟹ *path_connected*(*ball a r* − {*a*::'*N*})
by (*simp add*: *path_connected_open_delete*)

corollary *connected_punctured_ball*:
2 ≤ *DIM*('*N*::*euclidean_space*) ⟹ *connected*(*ball a r* − {*a*::'*N*})
by (*simp add*: *connected_open_delete*)

corollary *connected_open_delete_finite*:
fixes *S T*::'*a*::*euclidean_space set*
assumes *S*: *open S connected S* **and** *2*: *2* ≤ *DIM*('*a*) **and** *finite T*
shows *connected*(*S* − *T*)
using ‹*finite T*› *S*
proof (*induct T*)
case *empty*
show *?case* **using** ‹*connected S*› **by** *simp*
next
case (*insert x T*)
then have *connected* (*S*−*T*)
by *auto*
moreover have *open* (*S* − *T*)
using *finite_imp_closed*[*OF* ‹*finite T*›] ‹*open S*› **by** *auto*
ultimately have *connected* (*S* − *T* − {*x*})
using *connected_open_delete*[*OF _ _ 2*] **by** *auto*
thus *?case* **by** (*metis Diff_insert*)
qed

lemma *sphere_1D_doubleton_zero*:
assumes *1*: *DIM*('*a*) = *1* **and** *r* > *0*
obtains *x y*::'*a*::*euclidean_space*
where *sphere 0 r* = {*x*,*y*} ∧ *dist x y* = *2*∗*r*
proof −
obtain *b*::'*a* **where** *b*: *Basis* = {*b*}
using *1 card_1_singletonE* **by** *blast*
show *?thesis*

```
  proof (intro that conjI)
    have x = norm x *R b ∨ x = - norm x *R b if r = norm x for x
    proof -
      have xb: (x • b) *R b = x
        using euclidean_representation [of x, unfolded b] by force
      then have norm ((x • b) *R b) = norm x
        by simp
      with b have |x • b| = norm x
        using norm_Basis by (simp add: b)
      with xb show ?thesis
        by (metis (mono_tags, opaque_lifting) abs_eq_iff abs_norm_cancel)
    qed
    with ‹r > 0› b show sphere 0 r = {r *R b, - r *R b}
      by (force simp: sphere_def dist_norm)
    have dist (r *R b) (- r *R b) = norm (r *R b + r *R b)
      by (simp add: dist_norm)
    also have ... = norm ((2*r) *R b)
      by (metis mult_2 scaleR_add_left)
    also have ... = 2*r
      using ‹r > 0› b norm_Basis by fastforce
    finally show dist (r *R b) (- r *R b) = 2*r .
  qed
qed

lemma sphere_1D_doubleton:
  fixes a :: 'a :: euclidean_space
  assumes DIM('a) = 1 and r > 0
  obtains x y where sphere a r = {x,y} ∧ dist x y = 2*r
   using sphere_1D_doubleton_zero [OF assms] dist_add_cancel image_empty
image_insert
  by (metis (no_types, opaque_lifting) add.right_neutral sphere_translation)

lemma psubset_sphere_Compl_connected:
  fixes S :: 'a::euclidean_space set
  assumes S: S ⊂ sphere a r and 0 < r and 2: 2 ≤ DIM('a)
  shows connected(- S)
proof -
  have S ⊆ sphere a r
    using S by blast
  obtain b where dist a b = r and b ∉ S
    using S mem_sphere by blast
  have CS: - S = {x. dist a x ≤ r ∧ (x ∉ S)} ∪ {x. r ≤ dist a x ∧ (x ∉ S)}
    by auto
  have {x. dist a x ≤ r ∧ x ∉ S} ∩ {x. r ≤ dist a x ∧ x ∉ S} ≠ {}
    using ‹b ∉ S› ‹dist a b = r› by blast
  moreover have connected {x. dist a x ≤ r ∧ x ∉ S}
    using assms
    by (force intro: connected_intermediate_closure [of ball a r])
  moreover have connected {x. r ≤ dist a x ∧ x ∉ S}
```

```
  proof (rule connected_intermediate_closure [of − cball a r])
    show {x. r ≤ dist a x ∧ x ∉ S} ⊆ closure (− cball a r)
     using interior_closure by (force intro: connected_complement_bounded_convex)
  qed (use assms connected_complement_bounded_convex in auto)
  ultimately show ?thesis
    by (simp add: CS connected_Un)
qed
```

5.1.23 Every annulus is a connected set

```
lemma path_connected_2DIM_I:
  fixes a :: 'N::euclidean_space
  assumes 2: 2 ≤ DIM('N) and pc: path_connected {r. 0 ≤ r ∧ P r}
  shows path_connected {x. P(norm(x − a))}
proof −
  have {x. P(norm(x − a))} = (+) a ‘ {x. P(norm x)}
    by force
  moreover have path_connected {x::'N. P(norm x)}
  proof −
    let ?D = {x. 0 ≤ x ∧ P x} × sphere (0::'N) 1
    have x ∈ (λz. fst z ∗R snd z) ‘ ?D
      if P (norm x) for x::'N
    proof (cases x=0)
      case True
      with that show ?thesis
        apply (simp add: image_iff)
       by (metis (no_types) mem_sphere_0 order_refl vector_choose_size zero_le_one)
    next
      case False
      with that show ?thesis
        by (rule_tac x=(norm x, x /R norm x) in image_eqI) auto
    qed
    then have ∗: {x::'N. P(norm x)} = (λz. fst z ∗R snd z) ‘ ?D
      by auto
    have continuous_on ?D (λz:: real×'N. fst z ∗R snd z)
      by (intro continuous_intros)
    moreover have path_connected ?D
      by (metis path_connected_Times [OF pc] path_connected_sphere 2)
    ultimately show ?thesis
      by (simp add: ∗ path_connected_continuous_image)
  qed
  ultimately show ?thesis
    using path_connected_translation by metis
qed

proposition path_connected_annulus:
  fixes a :: 'N::euclidean_space
  assumes 2 ≤ DIM('N)
  shows path_connected {x. r1 < norm(x − a) ∧ norm(x − a) < r2}
```

path_connected $\{x.\ r1 < norm(x - a) \wedge norm(x - a) \leq r2\}$
path_connected $\{x.\ r1 \leq norm(x - a) \wedge norm(x - a) < r2\}$
path_connected $\{x.\ r1 \leq norm(x - a) \wedge norm(x - a) \leq r2\}$
by (*auto simp*: *is_interval_def intro*!: *is_interval_convex convex_imp_path_connected path_connected_2DIM_I* [*OF assms*])

proposition *connected_annulus*:
fixes a :: $'N$::*euclidean_space*
assumes $2 \leq DIM('N$::*euclidean_space*$)$
shows *connected* $\{x.\ r1 < norm(x - a) \wedge norm(x - a) < r2\}$
connected $\{x.\ r1 < norm(x - a) \wedge norm(x - a) \leq r2\}$
connected $\{x.\ r1 \leq norm(x - a) \wedge norm(x - a) < r2\}$
connected $\{x.\ r1 \leq norm(x - a) \wedge norm(x - a) \leq r2\}$
by (*auto simp*: *path_connected_annulus* [*OF assms*] *path_connected_imp_connected*)

5.1.24 Relations between components and path components

lemma *open_connected_component*:
fixes S :: $'a$::*real_normed_vector set*
assumes *open S*
shows *open* (*connected_component_set S x*)
proof (*clarsimp simp*: *open_contains_ball*)
fix y
assume *xy*: *connected_component S x y*
then obtain e **where** $e{>}0$ *ball y e* $\subseteq S$
using *assms connected_component_in openE* **by** *blast*
then show $\exists e{>}0.$ *ball y e* $\subseteq$ *connected_component_set S x*
by (*metis xy centre_in_ball connected_ball connected_component_eq_eq connected_component_in connected_component_maximal*)
qed

corollary *open_components*:
fixes S :: $'a$::*real_normed_vector set*
shows $[\![$*open u*; $S \in$ *components u*$]\!] \Longrightarrow$ *open S*
by (*simp add*: *components_iff*) (*metis open_connected_component*)

lemma *in_closure_connected_component*:
fixes S :: $'a$::*real_normed_vector set*
assumes *x*: $x \in S$ **and** *S*: *open S*
shows $x \in$ *closure* (*connected_component_set S y*) $\longleftrightarrow$ $x \in$ *connected_component_set S y*
proof −
have *x islimpt connected_component_set S y* $\Longrightarrow$ *connected_component S y x*
by (*metis* (*no_types*, *lifting*) *S connected_component_eq connected_component_refl islimptE mem_Collect_eq open_connected_component x*)
then show *?thesis*
by (*auto simp*: *closure_def*)
qed

lemma *connected_disjoint_Union_open_pick*:
assumes *pairwise disjnt B*
$\bigwedge S.\ S \in A \Longrightarrow$ *connected* $S \wedge S \neq \{\}$
$\bigwedge S.\ S \in B \Longrightarrow$ *open* S
$\bigcup A \subseteq \bigcup B$
$S \in A$
obtains T **where** $T \in B\ S \subseteq T\ S \cap \bigcup(B - \{T\}) = \{\}$
proof −
have $S \subseteq \bigcup B$ *connected* $S\ S \neq \{\}$
using *assms* ‹$S \in A$› **by** *blast*+
then obtain T **where** $T \in B\ S \cap T \neq \{\}$
by (*metis Sup_inf_eq_bot_iff inf.absorb_iff2 inf_commute*)
have *1*: *open* T **by** (*simp add*: ‹$T \in B$› *assms*)
have *2*: *open* $(\bigcup(B-\{T\}))$ **using** *assms* **by** *blast*
have *3*: $S \subseteq T \cup \bigcup(B - \{T\})$ **using** ‹$S \subseteq \bigcup B$› **by** *blast*
have $T \cap \bigcup(B - \{T\}) = \{\}$ **using** ‹$T \in B$› ‹*pairwise disjnt B*›
by (*auto simp*: *pairwise_def disjnt_def*)
then have *4*: $T \cap \bigcup(B - \{T\}) \cap S = \{\}$ **by** *auto*
from *connectedD* [*OF* ‹*connected S*› *1 2 4 3*]
have $S \cap \bigcup(B-\{T\}) = \{\}$
by (*auto simp*: *Int_commute* ‹$S \cap T \neq \{\}$›)
with ‹$T \in B$› *3 that* **show** *?thesis*
by (*metis IntI UnE empty_iff subsetD subsetI*)
qed

lemma *connected_disjoint_Union_open_subset*:
assumes *A*: *pairwise disjnt A* **and** *B*: *pairwise disjnt B*
and *SA*: $\bigwedge S.\ S \in A \Longrightarrow$ *open* $S \wedge$ *connected* $S \wedge S \neq \{\}$
and *SB*: $\bigwedge S.\ S \in B \Longrightarrow$ *open* $S \wedge$ *connected* $S \wedge S \neq \{\}$
and *eq* [*simp*]: $\bigcup A = \bigcup B$
shows $A \subseteq B$
proof
fix S
assume $S \in A$
obtain T **where** $T \in B\ S \subseteq T\ S \cap \bigcup(B - \{T\}) = \{\}$
using *SA SB* ‹$S \in A$› *connected_disjoint_Union_open_pick* [*OF B, of A*] *eq order_refl* **by** *blast*
moreover obtain S' **where** $S' \in A\ T \subseteq S'\ T \cap \bigcup(A - \{S'\}) = \{\}$
using *SA SB* ‹$T \in B$› *connected_disjoint_Union_open_pick* [*OF A, of B*] *eq order_refl* **by** *blast*
ultimately have $S' = S$
by (*metis A Int_subset_iff SA* ‹$S \in A$› *disjnt_def inf.orderE pairwise_def*)
with ‹$T \subseteq S'$› **have** $T \subseteq S$ **by** *simp*
with ‹$S \subseteq T$› **have** $S = T$ **by** *blast*
with ‹$T \in B$› **show** $S \in B$ **by** *simp*
qed

lemma *connected_disjoint_Union_open_unique*:
assumes *A*: *pairwise disjnt A* **and** *B*: *pairwise disjnt B*

and *SA*: $\bigwedge S.\ S \in A \Longrightarrow$ *open* $S \wedge$ *connected* $S \wedge S \neq \{\}$
and *SB*: $\bigwedge S.\ S \in B \Longrightarrow$ *open* $S \wedge$ *connected* $S \wedge S \neq \{\}$
and *eq* [*simp*]: $\bigcup A = \bigcup B$
shows $A = B$
by (*metis subset_antisym connected_disjoint_Union_open_subset assms*)

proposition *components_open_unique*:
fixes $S :: 'a$::*real_normed_vector set*
assumes *pairwise disjnt* A $\bigcup A = S$
$\bigwedge X.\ X \in A \Longrightarrow$ *open* $X \wedge$ *connected* $X \wedge X \neq \{\}$
shows *components* $S = A$
proof −
have *open* S **using** *assms* **by** *blast*
show *?thesis*
proof (*rule connected_disjoint_Union_open_unique*)
show *disjoint* (*components* S)
by (*simp add*: *components_eq disjnt_def pairwise_def*)
qed (*use* ‹*open* S› **in** ‹*simp_all add*: *assms open_components in_components_connected in_components_nonempty*›)
qed

5.1.25 Existence of unbounded components

lemma *cobounded_unbounded_component*:
fixes $S :: 'a :: euclidean\_space\ set$
assumes *bounded* $(-S)$
shows $\exists x.\ x \in S \wedge \neg$ *bounded* (*connected_component_set* S x)
proof −
obtain i::$'a$ **where** i: $i \in Basis$
using *nonempty_Basis* **by** *blast*
obtain B **where** B: $B>0$ $-S \subseteq$ *ball 0 B*
using *bounded_subset_ballD* [*OF assms*, *of 0*] **by** *auto*
then have $*$: $\bigwedge x.\ B \leq$ *norm* $x \Longrightarrow x \in S$
by (*force simp*: *ball_def dist_norm*)
have *unbounded_inner*: $\neg$ *bounded* $\{x.\ inner\ i\ x \geq B\}$
proof (*clarsimp simp*: *bounded_def dist_norm*)
fix e x
show $\exists y.\ B \leq i \cdot y \wedge \neg$ *norm* $(x - y) \leq e$
using i
by (*rule_tac x*=$x + (max\ B\ e + 1 + |i \cdot x|) *_R i$ **in** *exI*) (*auto simp*: *inner_right_distrib*)
qed
have §: $\bigwedge x.\ B \leq i \cdot x \Longrightarrow x \in S$
using $*$ *Basis_le_norm* [*OF i*] **by** (*metis abs_ge_self inner_commute order_trans*)
have $\{x.\ B \leq i \cdot x\} \subseteq$ *connected_component_set* S $(B *_R i)$
by (*intro connected_component_maximal*) (*auto simp*: i *intro*: *convex_connected convex_halfspace_ge* [*of B*] §)
then have $\neg$ *bounded* (*connected_component_set* S $(B *_R i)$)

```
    using bounded_subset unbounded_inner by blast
  moreover have B *_R i ∈ S
    by (rule *) (simp add: norm_Basis [OF i])
  ultimately show ?thesis
    by blast
qed

lemma cobounded_unique_unbounded_component:
    fixes S :: 'a :: euclidean_space set
    assumes bs: bounded (−S) and 2 ≤ DIM('a)
        and bo: ¬ bounded(connected_component_set S x)
                ¬ bounded(connected_component_set S y)
      shows connected_component_set S x = connected_component_set S y
proof −
  obtain i::'a where i: i ∈ Basis
    using nonempty_Basis by blast
  obtain B where B: B>0 −S ⊆ ball 0 B
    using bounded_subset_ballD [OF bs, of 0] by auto
  then have *: ⋀x. B ≤ norm x ⟹ x ∈ S
    by (force simp: ball_def dist_norm)
  obtain x' where x': connected_component S x x' norm x' > B
    using B(1) bo(1) bounded_pos by force
  obtain y' where y': connected_component S y y' norm y' > B
    using B(1) bo(2) bounded_pos by force
  have x'y': connected_component S x' y'
    unfolding connected_component_def
  proof (intro exI conjI)
    show connected (− ball 0 B :: 'a set)
      using assms by (auto intro: connected_complement_bounded_convex)
  qed (use x' y' dist_norm * in auto)
  show ?thesis
      using x' y' x'y'
      by (metis connected_component_eq mem_Collect_eq)
qed

lemma cobounded_unbounded_components:
    fixes S :: 'a :: euclidean_space set
    shows bounded (−S) ⟹ ∃ c. c ∈ components S ∧ ¬bounded c
  by (metis cobounded_unbounded_component components_def imageI)

lemma cobounded_unique_unbounded_components:
    fixes S :: 'a :: euclidean_space set
    shows ⟦bounded (− S); c ∈ components S; ¬ bounded c; c' ∈ components S;
¬ bounded c'; 2 ≤ DIM('a)⟧ ⟹ c' = c
  unfolding components_iff
  by (metis cobounded_unique_unbounded_component)

lemma cobounded_has_bounded_component:
  fixes S :: 'a :: euclidean_space set
```

assumes *bounded* $(-S)$ $\neg$ *connected* S $2 \leq$ *DIM*$('a)$
obtains C **where** $C \in$ *components* S *bounded* C
by (*meson cobounded_unique_unbounded_components connected_eq_connected_components_eq assms*)

5.1.26 The *inside* and *outside* of a Set

The inside comprises the points in a bounded connected component of the set's complement. The outside comprises the points in unbounded connected component of the complement.

definition *inside* **where**
inside $S \equiv \{x.\ (x \notin S) \wedge$ *bounded*(*connected_component_set* $(-S)\ x)\}$

definition *outside* **where**
outside $S \equiv -S \cap \{x.\ \neg$ *bounded*(*connected_component_set* $(-S)\ x)\}$

lemma *outside*: *outside* $S = \{x.\ \neg$ *bounded*(*connected_component_set* $(-S)\ x)\}$
by (*auto simp*: *outside_def*) (*metis Compl_iff bounded_empty connected_component_eq_empty*)

lemma *inside_no_overlap* [*simp*]: *inside* $S \cap S = \{\}$
by (*auto simp*: *inside_def*)

lemma *outside_no_overlap* [*simp*]:
outside $S \cap S = \{\}$
by (*auto simp*: *outside_def*)

lemma *inside_Int_outside* [*simp*]: *inside* $S \cap$ *outside* $S = \{\}$
by (*auto simp*: *inside_def outside_def*)

lemma *inside_Un_outside* [*simp*]: *inside* $S \cup$ *outside* $S = (-S)$
by (*auto simp*: *inside_def outside_def*)

lemma *inside_eq_outside*:
inside $S =$ *outside* $S \longleftrightarrow S =$ *UNIV*
by (*auto simp*: *inside_def outside_def*)

lemma *inside_outside*: *inside* $S = (-(S \cup$ *outside* $S))$
by (*force simp*: *inside_def outside*)

lemma *outside_inside*: *outside* $S = (-(S \cup$ *inside* $S))$
by (*auto simp*: *inside_outside*) (*metis IntI equals0D outside_no_overlap*)

lemma *union_with_inside*: $S \cup$ *inside* $S = -$ *outside* S
by (*auto simp*: *inside_outside*) (*simp add*: *outside_inside*)

lemma *union_with_outside*: $S \cup$ *outside* $S = -$ *inside* S
by (*simp add*: *inside_outside*)

lemma *outside_mono*: $S \subseteq T \Longrightarrow$ *outside* $T \subseteq$ *outside* S

```
  by (auto simp: outside bounded_subset connected_component_mono)

lemma inside_mono: S ⊆ T ⟹ inside S − T ⊆ inside T
  by (auto simp: inside_def bounded_subset connected_component_mono)

lemma segment_bound_lemma:
  fixes u::real
  assumes x ≥ B y ≥ B 0 ≤ u u ≤ 1
  shows (1 − u) * x + u * y ≥ B
  by (smt (verit) assms convex_bound_le ge_iff_diff_ge_0 minus_add_distrib
      mult_minus_right neg_le_iff_le)

lemma cobounded_outside:
  fixes S :: 'a :: real_normed_vector set
  assumes bounded S shows bounded (− outside S)
proof −
  obtain B where B: B>0 S ⊆ ball 0 B
    using bounded_subset_ballD [OF assms, of 0] by auto
  { fix x::'a and C::real
    assume Bno: B ≤ norm x and C: 0 < C
    have ∃ y. connected_component (− S) x y ∧ norm y > C
    proof (cases x = 0)
      case True with B Bno show ?thesis by force
    next
      case False
      have closed_segment x (((B + C) / norm x) *R x) ⊆ − ball 0 B
      proof
        fix w
        assume w ∈ closed_segment x (((B + C) / norm x) *R x)
        then obtain u where
          w: w = (1 − u + u * (B + C) / norm x) *R x 0 ≤ u u ≤ 1
       by (auto simp add: closed_segment_def real_vector_class.scaleR_add_left
[symmetric])
        with False B C have B ≤ (1 − u) * norm x + u * (B + C)
          using segment_bound_lemma [of B norm x B + C u] Bno
          by simp
        with False B C show w ∈ − ball 0 B
          using distrib_right [of _ _ norm x]
          by (simp add: ball_def w not_less)
      qed
      also have ... ⊆ −S
        by (simp add: B)
      finally have ∃ T. connected T ∧ T ⊆ − S ∧ x ∈ T ∧ ((B + C) / norm x)
*R x ∈ T
        by (rule_tac x=closed_segment x (((B+C)/norm x) *R x) in exI) simp
      with False B
      show ?thesis
      by (rule_tac x=((B+C)/norm x) *R x in exI) (simp add: connected_component_def)
    qed
```

```
  }
  then show ?thesis
    apply (simp add: outside_def assms)
    apply (rule bounded_subset [OF bounded_ball [of 0 B]])
    apply (force simp: dist_norm not_less bounded_pos)
    done
qed

lemma unbounded_outside:
    fixes S :: 'a::{real_normed_vector, perfect_space} set
    shows bounded S ⟹ ¬ bounded(outside S)
  using cobounded_imp_unbounded cobounded_outside by blast

lemma bounded_inside:
    fixes S :: 'a::{real_normed_vector, perfect_space} set
    shows bounded S ⟹ bounded(inside S)
  by (simp add: bounded_Int cobounded_outside inside_outside)

lemma connected_outside:
    fixes S :: 'a::euclidean_space set
    assumes bounded S 2 ≤ DIM('a)
    shows connected(outside S)
  apply (clarsimp simp add: connected_iff_connected_component outside)
  apply (rule_tac S=connected_component_set (- S) x in connected_component_of_subset)
  apply (metis (no_types) assms cobounded_unbounded_component cobounded_unique_unbounded_component
connected_component_eq_eq connected_component_idemp double_complement mem_Collect_eq)
  by (simp add: Collect_mono connected_component_eq)

lemma outside_connected_component_lt:
  outside S = {x. ∀ B. ∃ y. B < norm(y) ∧ connected_component (- S) x y}
proof -
  have ⋀x B. x ∈ outside S ⟹ ∃ y. B < norm y ∧ connected_component (- S)
x y
    by (metis boundedI linorder_not_less mem_Collect_eq outside)
  moreover
  have ⋀x. ∀ B. ∃ y. B < norm y ∧ connected_component (- S) x y ⟹ x ∈
outside S
    by (metis bounded_pos linorder_not_less mem_Collect_eq outside)
  ultimately show ?thesis by auto
qed

lemma outside_connected_component_le:
  outside S = {x. ∀ B. ∃ y. B ≤ norm(y) ∧ connected_component (- S) x y}
  apply (simp add: outside_connected_component_lt Set.set_eq_iff)
  by (meson gt_ex leD le_less_linear less_imp_le order.trans)

lemma not_outside_connected_component_lt:
    fixes S :: 'a::euclidean_space set
    assumes S: bounded S and 2 ≤ DIM('a)
```

shows $-$ (*outside* S) = $\{x.\ \forall B.\ \exists y.\ B < norm(y) \wedge \neg$ *connected_component* $(-S)\ x\ y\}$
proof $-$
obtain B::*real* **where** B: $0 < B$ **and** *Bno*: $\bigwedge x.\ x \in S \Longrightarrow$ *norm* $x \leq B$
using S [*simplified bounded_pos*] **by** *auto*
have *cyz*: *connected_component* $(-S)\ y\ z$
if *yz*: $B <$ *norm* z $B <$ *norm* y **for** y::$'a$ **and** z::$'a$
proof $-$
have *connected_component* $(-$ *cball* $0\ B)\ y\ z$
using *assms yz*
by (*force simp*: *dist_norm intro*: *connected_componentI* [*OF* _ *subset_refl*] *connected_complement_bounded_convex*)
then show *?thesis*
by (*metis connected_component_of_subset Bno Compl_anti_mono mem_cball_0 subset_iff*)
qed
show *?thesis*
apply (*auto simp*: *outside bounded_pos*)
apply (*metis Compl_iff bounded_iff cobounded_imp_unbounded mem_Collect_eq not_le*)
by (*metis B connected_component_trans cyz not_le*)
qed

lemma *not_outside_connected_component_le*:
fixes S :: $'a$::*euclidean_space set*
assumes S: *bounded* S $2 \leq DIM('a)$
shows $-$ (*outside* S) = $\{x.\ \forall B.\ \exists y.\ B \leq norm(y) \wedge \neg$ *connected_component* $(-S)\ x\ y\}$
apply (*auto intro*: *less_imp_le simp*: *not_outside_connected_component_lt* [*OF assms*])
by (*meson gt_ex less_le_trans*)

lemma *inside_connected_component_lt*:
fixes S :: $'a$::*euclidean_space set*
assumes S: *bounded* S $2 \leq DIM('a)$
shows *inside* S = $\{x.\ (x \notin S) \wedge (\forall B.\ \exists y.\ B < norm(y) \wedge \neg$ *connected_component* $(-S)\ x\ y)\}$
by (*auto simp*: *inside_outside not_outside_connected_component_lt* [*OF assms*])

lemma *inside_connected_component_le*:
fixes S :: $'a$::*euclidean_space set*
assumes S: *bounded* S $2 \leq DIM('a)$
shows *inside* S = $\{x.\ (x \notin S) \wedge (\forall B.\ \exists y.\ B \leq norm(y) \wedge \neg$ *connected_component* $(-S)\ x\ y)\}$
by (*auto simp*: *inside_outside not_outside_connected_component_le* [*OF assms*])

lemma *inside_subset*:
assumes *connected* U **and** $\neg$ *bounded* U **and** $T \cup U = -S$
shows *inside* $S \subseteq T$

using *bounded_subset* [*of connected_component_set* (− *S*) _ *U*] *assms*
by (*metis* (*no_types*, *lifting*) *ComplI Un_iff connected_component_maximal inside_def mem_Collect_eq subsetI*)

lemma *frontier_not_empty*:
fixes *S* :: *'a* :: *real_normed_vector set*
shows ⟦*S* ≠ {}; *S* ≠ *UNIV*⟧ ⟹ *frontier S* ≠ {}
using *connected_Int_frontier* [*of UNIV S*] **by** *auto*

lemma *frontier_eq_empty*:
fixes *S* :: *'a* :: *real_normed_vector set*
shows *frontier S* = {} ⟷ *S* = {} ∨ *S* = *UNIV*
using *frontier_UNIV frontier_empty frontier_not_empty* **by** *blast*

lemma *frontier_of_connected_component_subset*:
fixes *S* :: *'a*::*real_normed_vector set*
shows *frontier*(*connected_component_set S x*) ⊆ *frontier S*
proof −
{ fix *y*
assume *y1*: *y* ∈ *closure* (*connected_component_set S x*)
and *y2*: *y* ∉ *interior* (*connected_component_set S x*)
have *y* ∈ *closure S*
using *y1 closure_mono connected_component_subset* **by** *blast*
moreover have *z* ∈ *interior* (*connected_component_set S x*)
if *0* < *e ball y e* ⊆ *interior S dist y z* < *e* **for** *e z*
proof −
have *ball y e* ⊆ *connected_component_set S y*
using *connected_component_maximal that interior_subset*
by (*metis centre_in_ball connected_ball subset_trans*)
then show *?thesis*
using *y1* **apply** (*simp add*: *closure_approachable open_contains_ball_eq* [*OF open_interior*])
by (*metis connected_component_eq dist_commute mem_Collect_eq mem_ball mem_interior subsetD* ‹*0* < *e*› *y2*)
qed
then have *y* ∉ *interior S*
using *y2* **by** (*force simp*: *open_contains_ball_eq* [*OF open_interior*])
ultimately have *y* ∈ *frontier S*
by (*auto simp*: *frontier_def*)
}
then show *?thesis* **by** (*auto simp*: *frontier_def*)
qed

lemma *frontier_Union_subset_closure*:
fixes *F* :: *'a*::*real_normed_vector set set*
shows *frontier*(⋃ *F*) ⊆ *closure*(⋃ *t* ∈ *F*. *frontier t*)
proof −
have ∃ *y*∈*F*. ∃ *y*∈*frontier y*. *dist y x* < *e*
if *T* ∈ *F y* ∈ *T dist y x* < *e*

```
        x ∉ interior (⋃F) 0 < e for x y e T
  proof (cases x ∈ T)
    case True with that show ?thesis
      by (metis Diff_iff Sup_upper closure_subset contra_subsetD dist_self fron-
tier_def interior_mono)
  next
    case False
    have §: closed_segment x y ∩ T ≠ {} closed_segment x y − T ≠ {}
      using ‹y ∈ T› False by blast+
    obtain c where c ∈ closed_segment x y c ∈ frontier T
      using False connected_Int_frontier [OF connected_segment §] by auto
     with that show ?thesis
      by (smt (verit) dist_norm segment_bound1)
  qed
  then show ?thesis
    by (fastforce simp add: frontier_def closure_approachable)
qed

lemma frontier_Union_subset:
  fixes F :: 'a::real_normed_vector set set
  shows finite F ⟹ frontier(⋃F) ⊆ (⋃t ∈ F. frontier t)
 by (metis closed_UN closure_closed frontier_Union_subset_closure frontier_closed)

lemma frontier_of_components_subset:
  fixes S :: 'a::real_normed_vector set
  shows C ∈ components S ⟹ frontier C ⊆ frontier S
 by (metis Path_Connected.frontier_of_connected_component_subset components_iff)

lemma frontier_of_components_closed_complement:
  fixes S :: 'a::real_normed_vector set
  shows ⟦closed S; C ∈ components (− S)⟧ ⟹ frontier C ⊆ S
  using frontier_complement frontier_of_components_subset frontier_subset_eq
by blast

lemma frontier_minimal_separating_closed:
  fixes S :: 'a::real_normed_vector set
  assumes closed S
      and nconn: ¬ connected(− S)
      and C: C ∈ components (− S)
      and conn: ⋀T. ⟦closed T; T ⊂ S⟧ ⟹ connected(− T)
    shows frontier C = S
proof (rule ccontr)
  assume frontier C ≠ S
  then have frontier C ⊂ S
    using frontier_of_components_closed_complement [OF ‹closed S› C] by blast
  then have connected(− (frontier C))
    by (simp add: conn)
  have ¬ connected(− (frontier C))
    unfolding connected_def not_not
```

```
  proof (intro exI conjI)
    show open C
      using C ‹closed S› open_components by blast
    show open (− closure C)
      by blast
    show C ∩ − closure C ∩ − frontier C = {}
      using closure_subset by blast
    show C ∩ − frontier C ≠ {}
      using C ‹open C› components_eq frontier_disjoint_eq by fastforce
    show − frontier C ⊆ C ∪ − closure C
      by (simp add: ‹open C› closed_Compl frontier_closures)
    then show − closure C ∩ − frontier C ≠ {}
      by (metis C Compl_Diff_eq Un_Int_eq(4) Un_commute ‹frontier C ⊂ S›
‹open C› compl_le_compl_iff frontier_def in_components_subset interior_eq leD
sup_bot.right_neutral)
  qed
  then show False
    using ‹connected (− frontier C)› by blast
qed

lemma connected_component_UNIV [simp]:
    fixes x :: 'a::real_normed_vector
    shows connected_component_set UNIV x = UNIV
using connected_iff_eq_connected_component_set [of UNIV::'a set] connected_UNIV
by auto

lemma connected_component_eq_UNIV:
    fixes x :: 'a::real_normed_vector
    shows connected_component_set s x = UNIV ⟷ s = UNIV
  using connected_component_in connected_component_UNIV by blast

lemma components_UNIV [simp]: components UNIV = {UNIV :: 'a::real_normed_vector
set}
  by (auto simp: components_eq_sing_iff)

lemma interior_inside_frontier:
    fixes S :: 'a::real_normed_vector set
    assumes bounded S
      shows interior S ⊆ inside (frontier S)
proof −
  { fix x y
    assume x: x ∈ interior S and y: y ∉ S
       and cc: connected_component (− frontier S) x y
    have connected_component_set (− frontier S) x ∩ frontier S ≠ {}
    proof (rule connected_Int_frontier; simp add: set_eq_iff)
      show ∃u. connected_component (− frontier S) x u ∧ u ∈ S
        by (meson cc connected_component_in connected_component_refl_eq in-
terior_subset subsetD x)
      show ∃u. connected_component (− frontier S) x u ∧ u ∉ S
```

```
      using y cc by blast
    qed
    then have bounded (connected_component_set (− frontier S) x)
      using connected_component_in by auto
  }
  then show ?thesis
    using bounded_subset [OF assms]
    by (metis (no_types, lifting) Diff_iff frontier_def inside_def mem_Collect_eq
subsetI)
qed

lemma inside_empty [simp]: inside {} = ({} :: 'a :: {real_normed_vector, per-
fect_space} set)
  by (simp add: inside_def)

lemma outside_empty [simp]: outside {} = (UNIV :: 'a :: {real_normed_vector,
perfect_space} set)
  using inside_empty inside_Un_outside by blast

lemma inside_same_component:
   ⟦connected_component (− S) x y; x ∈ inside S⟧ ⟹ y ∈ inside S
  using connected_component_eq connected_component_in
  by (fastforce simp add: inside_def)

lemma outside_same_component:
   ⟦connected_component (− S) x y; x ∈ outside S⟧ ⟹ y ∈ outside S
  using connected_component_eq connected_component_in
  by (fastforce simp add: outside_def)

lemma convex_in_outside:
  fixes S :: 'a :: {real_normed_vector, perfect_space} set
  assumes S: convex S and z: z ∉ S
    shows z ∈ outside S
proof (cases S={})
  case True then show ?thesis by simp
next
  case False then obtain a where a ∈ S by blast
  with z have zna: z ≠ a by auto
  { assume bounded (connected_component_set (− S) z)
   with bounded_pos_less obtain B where B>0 and B: ⋀x. connected_component
(− S) z x ⟹ norm x < B
      by (metis mem_Collect_eq)
    define C where C = (B + 1 + norm z) / norm (z−a)
    have C > 0
     using ‹0 < B› zna by (simp add: C_def field_split_simps add_strict_increasing)
    have |norm (z + C *R (z−a)) − norm (C *R (z−a))| ≤ norm z
      by (metis add_diff_cancel norm_triangle_ineq3)
    moreover have norm (C *R (z−a)) > norm z + B
      using zna ‹B>0› by (simp add: C_def le_max_iff_disj)
```

```
    ultimately have C: norm (z + C *R (z−a)) > B by linarith
    { fix u::real
     assume u: 0≤u u≤1 and ins: (1 − u) *R z + u *R (z + C *R (z − a)) ∈ S
      then have Cpos: 1 + u * C > 0
        by (meson ‹0 < C› add_pos_nonneg less_eq_real_def zero_le_mult_iff
zero_less_one)
      then have *: (1 / (1 + u * C)) *R z + (u * C / (1 + u * C)) *R z = z
        by (simp add: scaleR_add_left [symmetric] field_split_simps)
      then have False
        using convexD_alt [OF S ‹a ∈ S› ins, of 1/(u*C + 1)] ‹C>0› ‹z ∉ S›
Cpos u
        by (simp add: * field_split_simps)
    } note contra = this
    have connected_component (− S) z (z + C *R (z−a))
    proof (rule connected_componentI [OF connected_segment])
      show closed_segment z (z + C *R (z − a)) ⊆ − S
        using contra by (force simp add: closed_segment_def)
    qed auto
    then have False
      using zna B [of z + C *R (z−a)] C
      by (auto simp: field_split_simps max_mult_distrib_right)
  }
  then show ?thesis
    by (auto simp: outside_def z)
qed

lemma outside_convex:
  fixes S :: 'a :: {real_normed_vector, perfect_space} set
  assumes convex S
    shows outside S = − S
  by (metis ComplD assms convex_in_outside equalityI inside_Un_outside subsetI
sup.cobounded2)

lemma outside_singleton [simp]:
  fixes x :: 'a :: {real_normed_vector, perfect_space}
  shows outside {x} = −{x}
  by (auto simp: outside_convex)

lemma inside_convex:
  fixes S :: 'a :: {real_normed_vector, perfect_space} set
  shows convex S ⟹ inside S = {}
  by (simp add: inside_outside outside_convex)

lemma inside_singleton [simp]:
  fixes x :: 'a :: {real_normed_vector, perfect_space}
  shows inside {x} = {}
  by (auto simp: inside_convex)

lemma outside_subset_convex:
```

```
  fixes S :: 'a :: {real_normed_vector, perfect_space} set
  shows ⟦convex T; S ⊆ T⟧ ⟹ - T ⊆ outside S
  using outside_convex outside_mono by blast

lemma outside_Un_outside_Un:
  fixes S :: 'a::real_normed_vector set
  assumes S ∩ outside(T ∪ U) = {}
  shows outside(T ∪ U) ⊆ outside(T ∪ S)
proof
  fix x
  assume x: x ∈ outside (T ∪ U)
  have Y ⊆ - S if connected Y Y ⊆ - T Y ⊆ - U x ∈ Y u ∈ Y for u Y
  proof -
    have Y ⊆ connected_component_set (- (T ∪ U)) x
      by (simp add: connected_component_maximal that)
    also have ... ⊆ outside(T ∪ U)
      by (metis (mono_tags, lifting) Collect_mono mem_Collect_eq outside out-
side_same_component x)
    finally have Y ⊆ outside(T ∪ U) .
    with assms show ?thesis by auto
  qed
  with x show x ∈ outside (T ∪ S)
    by (simp add: outside_connected_component_lt connected_component_def)
meson
qed

lemma outside_frontier_misses_closure:
    fixes S :: 'a::real_normed_vector set
    assumes bounded S
    shows  outside(frontier S) ⊆ - closure S
  using assms frontier_def interior_inside_frontier outside_inside by fastforce

lemma outside_frontier_eq_complement_closure:
  fixes S :: 'a :: {real_normed_vector, perfect_space} set
    assumes bounded S convex S
      shows outside(frontier S) = - closure S
by (metis Diff_subset assms convex_closure frontier_def outside_frontier_misses_closure
          outside_subset_convex subset_antisym)

lemma inside_frontier_eq_interior:
     fixes S :: 'a :: {real_normed_vector, perfect_space} set
     shows ⟦bounded S; convex S⟧ ⟹ inside(frontier S) = interior S
  apply (simp add: inside_outside outside_frontier_eq_complement_closure)
  using closure_subset interior_subset
  apply (auto simp: frontier_def)
  done

lemma open_inside:
    fixes S :: 'a::real_normed_vector set
```

assumes *closed S*
shows *open* (*inside S*)
proof −
{ fix *x* **assume** *x*: *x* ∈ *inside S*
have *open* (*connected_component_set* (− *S*) *x*)
using *assms open_connected_component* **by** *blast*
then obtain *e* **where** *e*: *e*>*0* **and** *e*: ⋀*y*. *dist y x* < *e* ⟶ *connected_component* (− *S*) *x y*
using *dist_not_less_zero*
apply (*simp add*: *open_dist*)
by (*metis* (*no_types*, *lifting*) *Compl_iff connected_component_refl_eq inside_def mem_Collect_eq x*)
then have ∃ *e*>*0*. *ball x e* ⊆ *inside S*
by (*metis e dist_commute inside_same_component mem_ball subsetI x*)
}
then show *?thesis*
by (*simp add*: *open_contains_ball*)
qed

lemma *open_outside*:
fixes *S* :: *'a*::*real_normed_vector set*
assumes *closed S*
shows *open* (*outside S*)
proof −
{ fix *x* **assume** *x*: *x* ∈ *outside S*
have *open* (*connected_component_set* (− *S*) *x*)
using *assms open_connected_component* **by** *blast*
then obtain *e* **where** *e*: *e*>*0* **and** *e*: ⋀*y*. *dist y x* < *e* ⟶ *connected_component* (− *S*) *x y*
using *dist_not_less_zero x*
by (*auto simp add*: *open_dist outside_def intro*: *connected_component_refl*)
then have ∃ *e*>*0*. *ball x e* ⊆ *outside S*
by (*metis e dist_commute outside_same_component mem_ball subsetI x*)
}
then show *?thesis*
by (*simp add*: *open_contains_ball*)
qed

lemma *closure_inside_subset*:
fixes *S* :: *'a*::*real_normed_vector set*
assumes *closed S*
shows *closure*(*inside S*) ⊆ *S* ∪ *inside S*
by (*metis assms closure_minimal open_closed open_outside sup.cobounded2 union_with_inside*)

lemma *frontier_inside_subset*:
fixes *S* :: *'a*::*real_normed_vector set*
assumes *closed S*
shows *frontier*(*inside S*) ⊆ *S*
using *assms closure_inside_subset frontier_closures frontier_disjoint_eq open_inside*

by *fastforce*

lemma *closure_outside_subset*:
  fixes S :: *'a*::*real_normed_vector set*
  assumes *closed S*
    shows *closure*(*outside S*) ⊆ S ∪ *outside S*
 by (*metis assms closed_open closure_minimal inside_outside open_inside sup_ge2*)

lemma *closed_path_image_Un_inside*:
 fixes *g* :: *real* ⇒ *'a* :: *real_normed_vector*
 assumes *path g*
 shows *closed* (*path_image g* ∪ *inside* (*path_image g*))
 by (*simp add*: *assms closed_Compl closed_path_image open_outside union_with_inside*)

lemma *frontier_outside_subset*:
 fixes S :: *'a*::*real_normed_vector set*
 assumes *closed S*
 shows *frontier*(*outside S*) ⊆ S
 unfolding *frontier_def*
 by (*metis Diff_subset_conv assms closure_outside_subset interior_eq open_outside sup_aci*(*1*))

lemma *inside_complement_unbounded_connected_empty*:
    ⟦*connected* (− S); ¬ *bounded* (− S)⟧ ⟹ *inside S* = {}
 using *inside_subset* **by** *blast*

lemma *inside_bounded_complement_connected_empty*:
  fixes S :: *'a*::{*real_normed_vector*, *perfect_space*} *set*
  shows ⟦*connected* (− S); *bounded S*⟧ ⟹ *inside S* = {}
 by (*metis inside_complement_unbounded_connected_empty cobounded_imp_unbounded*)

lemma *inside_inside*:
  assumes S ⊆ *inside T*
  shows *inside S* − T ⊆ *inside T*
unfolding *inside_def*
proof *clarify*
 fix x
 assume x: x ∉ T x ∉ S **and** *bo*: *bounded* (*connected_component_set* (− S) x)
 show *bounded* (*connected_component_set* (− T) x)
 proof (*cases S* ∩ *connected_component_set* (− T) x = {})
  case *True* **then show** *?thesis*
  by (*metis bounded_subset* [*OF bo*] *compl_le_compl_iff connected_component_idemp connected_component_mono disjoint_eq_subset_Compl double_compl*)
 next
  case *False*
  then obtain y **where** y: y ∈ S y ∈ *connected_component_set* (− T) x
   by (*meson disjoint_iff*)
  then have *bounded* (*connected_component_set* (− T) y)
   using *assms* [*unfolded inside_def*] **by** *blast*

```
    with y show ?thesis
      by (metis connected_component_eq)
  qed
qed

lemma inside_inside_subset: inside(inside S) ⊆ S
  using inside_inside union_with_outside by fastforce

lemma inside_outside_intersect_connected:
      ⟦connected T; inside S ∩ T ≠ {}; outside S ∩ T ≠ {}⟧ ⟹ S ∩ T ≠ {}
  apply (simp add: inside_def outside_def ex_in_conv [symmetric] disjoint_eq_subset_Compl,
clarify)
  by (metis (no_types, opaque_lifting) Compl_anti_mono connected_component_eq
connected_component_maximal contra_subsetD double_compl)

lemma outside_bounded_nonempty:
  fixes S :: 'a :: {real_normed_vector, perfect_space} set
  assumes bounded S shows outside S ≠ {}
  using assms unbounded_outside by force

lemma outside_compact_in_open:
    fixes S :: 'a :: {real_normed_vector,perfect_space} set
    assumes S: compact S and T: open T and S ⊆ T T ≠ {}
      shows outside S ∩ T ≠ {}
proof −
  have outside S ≠ {}
    by (simp add: compact_imp_bounded outside_bounded_nonempty S)
  with assms obtain a b where a: a ∈ outside S and b: b ∈ T by auto
  show ?thesis
  proof (cases a ∈ T)
    case True with a show ?thesis by blast
  next
    case False
      have front: frontier T ⊆ − S
        using ‹S ⊆ T› frontier_disjoint_eq T by auto
      { fix γ
        assume path γ and pimg_sbs: path_image γ − {pathfinish γ} ⊆ interior
(− T)
           and pf: pathfinish γ ∈ frontier T and ps: pathstart γ = a
        define c where c = pathfinish γ
        have c ∈ −S unfolding c_def using front pf by blast
        moreover have open (−S) using S compact_imp_closed by blast
        ultimately obtain ε::real where ε > 0 and ε: cball c ε ⊆ −S
          using open_contains_cball[of −S] S by blast
        then obtain d where d ∈ T and d: dist d c < ε
          using closure_approachable [of c T] pf unfolding c_def
          by (metis Diff_iff frontier_def)
        then have d ∈ −S using ε
              using dist_commute by (metis contra_subsetD mem_cball not_le
```

```
not_less_iff_gr_or_eq)
      have pimg_sbs_cos: path_image γ ⊆ −S
        using ‹c ∈ − S› ‹S ⊆ T› c_def interior_subset pimg_sbs by fastforce
      have closed_segment c d ≤ cball c ε
        by (metis ‹0 < ε› centre_in_cball closed_segment_subset convex_cball d
dist_commute less_eq_real_def mem_cball)
      with ε have closed_segment c d ⊆ −S by blast
      moreover have con_gcd: connected (path_image γ ∪ closed_segment c d)
      by (rule connected_Un) (auto simp: c_def ‹path γ› connected_path_image)
      ultimately have connected_component (− S) a d
        unfolding connected_component_def using pimg_sbs_cos ps by blast
      then have outside S ∩ T ≠ {}
          using outside_same_component [OF _ a] by (metis IntI ‹d ∈ T›
empty_iff)
    } note ∗ = this
    have pal: pathstart (linepath a b) ∈ closure (− T)
      by (auto simp: False closure_def)
    show ?thesis
     by (rule exists_path_subpath_to_frontier [OF path_linepath pal _ ∗]) (auto
simp: b)
  qed
qed

lemma inside_inside_compact_connected:
   fixes S :: 'a :: euclidean_space set
   assumes S: closed S and T: compact T and connected T S ⊆ inside T
     shows inside S ⊆ inside T
proof (cases inside T = {})
  case True with assms show ?thesis by auto
next
  case False
  consider DIM('a) = 1 | DIM('a) ≥ 2
    using antisym not_less_eq_eq by fastforce
  then show ?thesis
  proof cases
    case 1 then show ?thesis
          using connected_convex_1_gen assms False inside_convex by blast
  next
    case 2
    have bounded S
    using assms by (meson bounded_inside bounded_subset compact_imp_bounded)
    then have coms: compact S
      by (simp add: S compact_eq_bounded_closed)
    then have bst: bounded (S ∪ T)
      by (simp add: compact_imp_bounded T)
    then obtain r where 0 < r and r: S ∪ T ⊆ ball 0 r
      using bounded_subset_ballD by blast
    have outst: outside S ∩ outside T ≠ {}
    by (metis bounded_Un bounded_subset bst cobounded_outside disjoint_eq_subset_Compl
```

unbounded_outside)
  have $S \cap T = \{\}$ **using** *assms*
   by (*metis disjoint_iff_not_equal inside_no_overlap subsetCE*)
  moreover have *outside* $S \cap$ *inside* $T \neq \{\}$
   by (*meson False assms*(*4*) *compact_eq_bounded_closed coms open_inside outside_compact_in_open T*)
  ultimately have *inside* $S \cap T = \{\}$
   using *inside_outside_intersect_connected* [*OF* ‹*connected T*›, *of S*]
  by (*metis 2 compact_eq_bounded_closed coms connected_outside inf.commute inside_outside_intersect_connected outst*)
  then show *?thesis*
   using *inside_inside* [*OF* ‹$S \subseteq$ *inside T*›] **by** *blast*
 qed
qed

lemma *connected_with_inside*:
  fixes $S :: 'a ::$ *real_normed_vector set*
  assumes *S*: *closed S* **and** *cons*: *connected S*
   shows *connected*($S \cup$ *inside S*)
proof (*cases* $S \cup$ *inside* $S = UNIV$)
 case *True* **with** *assms* **show** *?thesis* **by** *auto*
next
 case *False*
 then obtain *b* **where** *b*: $b \notin S$ $b \notin$ *inside S* **by** *blast*
 have $*$: $\exists y\ T.\ y \in S \wedge$ *connected* $T \wedge a \in T \wedge y \in T \wedge T \subseteq (S \cup$ *inside S*)
  if $a \in S \cup$ *inside S* **for** *a*
  using *that*
 proof
  assume $a \in S$ **then show** *?thesis*
   using *cons* **by** *blast*
 next
  assume *a*: $a \in$ *inside S*
  then have *ain*: $a \in$ *closure* (*inside S*)
   by (*simp add*: *closure_def*)
  obtain *h* **where** *h*: *path h pathstart h* $= a$
        path_image h $-$ {*pathfinish h*} $\subseteq$ *interior* (*inside S*)
        pathfinish h $\in$ *frontier* (*inside S*)
   using *ain b*
  by (*metis exists_path_subpath_to_frontier path_linepath pathfinish_linepath pathstart_linepath*)
  moreover
  have *h1S*: *pathfinish h* $\in S$
   using *S h frontier_inside_subset* **by** *blast*
  moreover
  have *path_image h* $\subseteq S \cup$ *inside S*
   using *IntD1 S h1S h interior_eq open_inside* **by** *fastforce*
  ultimately show *?thesis* **by** *blast*
 qed
 show *?thesis*

```
    apply (simp add: connected_iff_connected_component)
    apply (clarsimp simp add: connected_component_def dest!: ∗)
    subgoal for x y u u' T t'
      by (rule_tac x = S ∪ T ∪ t' in exI) (auto intro!: connected_Un cons)
    done
qed
```

The proof is virtually the same as that above.

```
lemma connected_with_outside:
    fixes S :: 'a :: real_normed_vector set
    assumes S: closed S and cons: connected S
      shows connected(S ∪ outside S)
proof (cases S ∪ outside S = UNIV)
  case True with assms show ?thesis by auto
next
  case False
  then obtain b where b: b ∉ S b ∉ outside S by blast
  have ∗: ∃ y T. y ∈ S ∧ connected T ∧ a ∈ T ∧ y ∈ T ∧ T ⊆ (S ∪ outside S)
if a ∈ (S ∪ outside S) for a
  using that proof
    assume a ∈ S then show ?thesis
      by (rule_tac x=a in exI, rule_tac x={a} in exI, simp)
  next
    assume a: a ∈ outside S
    then have ain: a ∈ closure (outside S)
      by (simp add: closure_def)
    obtain h where h: path h pathstart h = a
                      path_image h − {pathfinish h} ⊆ interior (outside S)
                      pathfinish h ∈ frontier (outside S)
      using ain b
     by (metis exists_path_subpath_to_frontier path_linepath pathfinish_linepath
pathstart_linepath)
    moreover
    have h1S: pathfinish h ∈ S
      using S frontier_outside_subset h(4) by blast
    moreover
    have path_image h ⊆ S ∪ outside S
      using IntD1 S h1S h interior_eq open_outside by fastforce
    ultimately show ?thesis
      by blast
  qed
  show ?thesis
    apply (simp add: connected_iff_connected_component)
    apply (clarsimp simp add: connected_component_def dest!: ∗)
    subgoal for x y u u' T t'
      by (rule_tac x=(S ∪ T ∪ t') in exI) (auto intro!: connected_Un cons)
    done
qed
```

lemma *inside_inside_eq_empty* [*simp*]:
  fixes $S :: 'a :: \{real\_normed\_vector, perfect\_space\}$ *set*
  assumes *S*: *closed S* **and** *cons*: *connected S*
  shows *inside* (*inside S*) = {}
proof −
 have *connected* (− *inside S*)
  by (*metis S connected_with_outside cons union_with_outside*)
 then show *?thesis*
  by (*metis bounded_Un inside_complement_unbounded_connected_empty unbounded_outside union_with_outside*)
qed

lemma *inside_in_components*:
  inside S ∈ *components* (− *S*) ⟷ *connected*(*inside S*) ∧ *inside S* ≠ {} (**is** *?lhs* = *?rhs*)
proof
 assume *R*: *?rhs*
 then have ⋀*x*. ⟦*x* ∈ *S*; *x* ∈ *inside S*⟧ ⟹ ¬ *connected* (*inside S*)
  by (*simp add*: *inside_outside*)
 with *R* **show** *?lhs*
  unfolding *in_components_maximal*
  by (*auto intro*: *inside_same_component connected_componentI*)
qed (*simp add*: *in_components_maximal*)

The proof is like that above.

lemma *outside_in_components*:
  outside S ∈ *components* (− *S*) ⟷ *connected*(*outside S*) ∧ *outside S* ≠ {} (**is** *?lhs* = *?rhs*)
proof
 assume *R*: *?rhs*
 then have ⋀*x*. ⟦*x* ∈ *S*; *x* ∈ *outside S*⟧ ⟹ ¬ *connected* (*outside S*)
  by (*meson disjoint_iff outside_no_overlap*)
 with *R* **show** *?lhs*
  unfolding *in_components_maximal*
  by (*auto intro*: *outside_same_component connected_componentI*)
qed (*simp add*: *in_components_maximal*)

lemma *bounded_unique_outside*:
 fixes $S :: 'a :: euclidean\_space$ *set*
 assumes *bounded S DIM*('*a*) ≥ *2*
 shows (*c* ∈ *components* (− *S*) ∧ ¬ *bounded c*) ⟷ *c* = *outside S*
 using *assms*
  by (*metis cobounded_unique_unbounded_components connected_outside double_compl outside_bounded_nonempty*
  outside_in_components unbounded_outside)

5.1.27 Condition for an open map's image to contain a ball

proposition *ball_subset_open_map_image*:

fixes $f :: 'a::heine\_borel \Rightarrow 'b :: \{real\_normed\_vector,heine\_borel\}$
assumes *contf*: *continuous_on* (*closure S*) *f*
and *oint*: *open* (*f ‘ interior S*)
and *le_no*: $\bigwedge z.\ z \in frontier\ S \Longrightarrow r \leq norm(f\ z - f\ a)$
and *bounded S a* $\in$ *S 0* $< r$
shows *ball* (*f a*) *r* $\subseteq$ *f ‘ S*
proof (*cases f ‘ S = UNIV*)
case *True* **then show** *?thesis* **by** *simp*
next
case *False*
then have *closed* (*frontier* (*f ‘ S*)) *frontier* (*f ‘ S*) $\neq \{\}$
using ‹*a* $\in$ *S*› **by** (*auto simp*: *frontier_eq_empty*)
then obtain *w* **where** *w*: *w* $\in$ *frontier* (*f ‘ S*)
and *dw_le*: $\bigwedge y.\ y \in frontier\ (f\ `\ S) \Longrightarrow norm\ (f\ a - w) \leq norm\ (f\ a - y)$
by (*auto simp add*: *dist_norm intro*: *distance_attains_inf* [*of frontier*(*f ‘ S*) *f a*])
then obtain ξ **where** ξ: $\bigwedge n.\ \xi\ n \in f\ `\ S$ **and** *tendsw*: $\xi \longrightarrow w$
by (*metis Diff_iff frontier_def closure_sequential*)
then have $\bigwedge n.\ \exists x \in S.\ \xi\ n = f\ x$ **by** *force*
then obtain *z* **where** *zs*: $\bigwedge n.\ z\ n \in S$ **and** *fz*: $\bigwedge n.\ \xi\ n = f\ (z\ n)$
by *metis*
then obtain *y K* **where** *y*: *y* $\in$ *closure S* **and** *strict_mono* (*K* :: *nat* $\Rightarrow$ *nat*)
and *Klim*: (*z* $\circ$ *K*) $\longrightarrow$ *y*
using ‹*bounded S*›
unfolding *compact_closure* [*symmetric*] *compact_def* **by** (*meson closure_subset subset_iff*)
then have *ftendsw*: $((\lambda n.\ f\ (z\ n)) \circ K) \longrightarrow w$
by (*metis LIMSEQ_subseq_LIMSEQ fun.map_cong0 fz tendsw*)
have *zKs*: $\bigwedge n.\ (z \circ K)\ n \in S$ **by** (*simp add*: *zs*)
have *fz*: $f \circ z = \xi\ \ (\lambda n.\ f\ (z\ n)) = \xi$
using *fz* **by** *auto*
then have $(\xi \circ K) \longrightarrow f\ y$
by (*metis* (*no_types*) *Klim zKs y contf comp_assoc continuous_on_closure_sequentially*)
with *fz* **have** *wy*: *w* = *f y* **using** *fz LIMSEQ_unique ftendsw* **by** *auto*
have *r* $\leq$ *norm* (*f y* $-$ *f a*)
proof (*rule le_no*)
show *y* $\in$ *frontier S*
using *w wy oint* **by** (*force simp*: *imageI image_mono interiorI interior_subset frontier_def y*)
qed
then have $\bigwedge y.\ [\![norm\ (f\ a - y) < r;\ y \in frontier\ (f\ `\ S)]\!] \Longrightarrow False$
by (*metis dw_le norm_minus_commute not_less order_trans wy*)
then have *ball* (*f a*) *r* $\cap$ *frontier* (*f ‘ S*) $= \{\}$
by (*metis disjoint_iff_not_equal dist_norm mem_ball*)
moreover
have *ball* (*f a*) *r* $\cap$ *f ‘ S* $\neq \{\}$
using ‹*a* $\in$ *S*› ‹*0* $<$ *r*› *centre_in_ball* **by** *blast*
ultimately show *?thesis*
by (*meson connected_Int_frontier connected_ball diff_shunt_var*)

qed

Special characterizations of classes of functions into and out of R.

lemma *Hausdorff_space_euclidean* [*simp*]: *Hausdorff_space* (*euclidean* :: *'a*::*metric_space topology*)
proof −
  have ∃ *U V*. *open U* ∧ *open V* ∧ *x* ∈ *U* ∧ *y* ∈ *V* ∧ *disjnt U V*
    if *x* ≠ *y* **for** *x y* :: *'a*
  proof (*intro exI conjI*)
    let *?r* = *dist x y* / *2*
    have [*simp*]: *?r* > *0*
      by (*simp add*: *that*)
    show *open* (*ball x ?r*) *open* (*ball y ?r*) *x* ∈ (*ball x ?r*) *y* ∈ (*ball y ?r*)
      by (*auto simp add*: *that*)
    show *disjnt* (*ball x ?r*) (*ball y ?r*)
      unfolding *disjnt_def* **by** (*simp add*: *disjoint_ballI*)
  qed
  then show *?thesis*
    by (*simp add*: *Hausdorff_space_def*)
qed

proposition *embedding_map_into_euclideanreal*:
  assumes *path_connected_space X*
  shows *embedding_map X euclideanreal f* ⟷
        continuous_map X euclideanreal f ∧ *inj_on f* (*topspace X*)
  proof *safe*
  show *continuous_map X euclideanreal f*
    if *embedding_map X euclideanreal f*
    using *continuous_map_in_subtopology homeomorphic_imp_continuous_map that*
    unfolding *embedding_map_def* **by** *blast*
  show *inj_on f* (*topspace X*)
    if *embedding_map X euclideanreal f*
    using *that homeomorphic_imp_injective_map*
    unfolding *embedding_map_def* **by** *blast*
  show *embedding_map X euclideanreal f*
    if *cont*: *continuous_map X euclideanreal f* **and** *inj*: *inj_on f* (*topspace X*)
  proof −
    obtain *g* **where** *gf*: ⋀*x*. *x* ∈ *topspace X* ⟹ *g* (*f x*) = *x*
      using *inv_into_f_f* [*OF inj*] **by** *auto*
    show *?thesis*
    unfolding *embedding_map_def homeomorphic_map_maps homeomorphic_maps_def*
    proof (*intro exI conjI*)
      show *continuous_map X* (*top_of_set* (*f* ‘ *topspace X*)) *f*
        by (*simp add*: *cont continuous_map_in_subtopology*)
      let *?S* = *f* ‘ *topspace X*
      have *eq*: {*x* ∈ *?S*. *g x* ∈ *U*} = *f* ‘ *U* **if** *openin X U* **for** *U*
        using *openin_subset* [*OF that*] **by** (*auto simp*: *gf*)

```
    have 1: g ‘ ?S ⊆ topspace X
      using eq by blast
    have openin (top_of_set ?S) {x ∈ ?S. g x ∈ T}
      if openin X T for T
    proof −
      have T ⊆ topspace X
        by (simp add: openin_subset that)
      have RR: ∀ x ∈ ?S ∩ g −‘ T. ∃ d>0. ∀ x′ ∈ ?S ∩ ball x d. g x′ ∈ T
      proof (clarsimp simp add: gf)
        have pcS: path_connectedin euclidean ?S
       using assms cont path_connectedin_continuous_map_image path_connectedin_topspace
by blast
        show ∃ d>0. ∀ x′∈f ‘ topspace X ∩ ball (f x) d. g x′ ∈ T
          if x ∈ T for x
        proof −
          have x: x ∈ topspace X
            using ‹T ⊆ topspace X› ‹x ∈ T› by blast
          obtain u v d where 0 < d u ∈ topspace X v ∈ topspace X
                       and sub_fuv: ?S ∩ {f x − d .. f x + d} ⊆ {f u..f v}
          proof (cases ∃ u ∈ topspace X. f u < f x)
            case True
            then obtain u where u: u ∈ topspace X f u < f x ..
            show ?thesis
            proof (cases ∃ v ∈ topspace X. f x < f v)
              case True
              then obtain v where v: v ∈ topspace X f x < f v ..
              show ?thesis
              proof
                let ?d = min (f x − f u) (f v − f x)
                show 0 < ?d
                  by (simp add: ‹f u < f x› ‹f x < f v›)
                show f ‘ topspace X ∩ {f x − ?d..f x + ?d} ⊆ {f u..f v}
                  by fastforce
              qed (auto simp: u v)
            next
              case False
              show ?thesis
              proof
                let ?d = f x − f u
                show 0 < ?d
                  by (simp add: u)
                show f ‘ topspace X ∩ {f x − ?d..f x + ?d} ⊆ {f u..f x}
                  using x u False by auto
              qed (auto simp: x u)
            qed
          next
            case False
            note no_u = False
            show ?thesis
```

```
      proof (cases ∃v ∈ topspace X. f x < f v)
        case True
        then obtain v where v: v ∈ topspace X f x < f v ..
        show ?thesis
        proof
          let ?d = f v − f x
          show 0 < ?d
            by (simp add: v)
          show f ‘ topspace X ∩ {f x − ?d..f x + ?d} ⊆ {f x..f v}
            using False by auto
        qed (auto simp: x v)
      next
        case False
        show ?thesis
        proof
          show f ‘ topspace X ∩ {f x − 1..f x + 1} ⊆ {f x..f x}
            using False no_u by fastforce
        qed (auto simp: x)
      qed
    qed
    then obtain h where pathin X h h 0 = u h 1 = v
      using assms unfolding path_connected_space_def by blast
    obtain C where compactin X C connectedin X C u ∈ C v ∈ C
    proof
      show compactin X (h ‘ {0..1})
        using that by (simp add: ‹pathin X h› compactin_path_image)
      show connectedin X (h ‘ {0..1})
        using ‹pathin X h› connectedin_path_image by blast
    qed (use ‹h 0 = u› ‹h 1 = v› in auto)
   have continuous_map (subtopology euclideanreal (?S ∩ {f x − d .. f x +
d})) (subtopology X C) g
    proof (rule continuous_inverse_map)
      show compact_space (subtopology X C)
        using ‹compactin X C› compactin_subspace by blast
      show continuous_map (subtopology X C) euclideanreal f
        by (simp add: cont continuous_map_from_subtopology)
      have {f u .. f v} ⊆ f ‘ topspace (subtopology X C)
      proof (rule connected_contains_Icc)
        show connected (f ‘ topspace (subtopology X C))
          using connectedin_continuous_map_image [OF cont]
                by (simp add: ‹compactin X C› ‹connectedin X C› com-
pactin_subset_topspace inf_absorb2)
        show f u ∈ f ‘ topspace (subtopology X C)
          by (simp add: ‹u ∈ C› ‹u ∈ topspace X›)
        show f v ∈ f ‘ topspace (subtopology X C)
          by (simp add: ‹v ∈ C› ‹v ∈ topspace X›)
      qed
    then show f ‘ topspace X ∩ {f x − d..f x + d} ⊆ f ‘ topspace (subtopology
X C)
```

```
            using sub_fuv by blast
          qed (auto simp: gf)
         then have contg: continuous_map (subtopology euclideanreal (?S ∩ {f x
− d .. f x + d})) X g
           using continuous_map_in_subtopology by blast
         have ∃ e>0. ∀ x ∈ ?S ∩ {f x − d .. f x + d} ∩ ball (f x) e. g x ∈ T
           using openin_continuous_map_preimage [OF contg ‹openin X T›] x
‹x ∈ T› ‹0 < d›
           unfolding openin_euclidean_subtopology_iff
           by (force simp: gf dist_commute)
         then obtain e where e > 0 ∧ (∀ x∈f ‘ topspace X ∩ {f x − d..f x +
d} ∩ ball (f x) e. g x ∈ T)
           by metis
         with ‹0 < d› have min d e > 0 ∀ u. u ∈ topspace X ⟶ |f x − f u| <
min d e ⟶ u ∈ T
           using dist_real_def gf by force+
         then show ?thesis
           by (metis (full_types) Int_iff dist_real_def image_iff mem_ball gf)
       qed
     qed
     then obtain d where d: ⋀r. r ∈ ?S ∩ g −‘ T ⟹
           d r > 0 ∧ (∀ x ∈ ?S ∩ ball r (d r). g x ∈ T)
       by metis
     show ?thesis
       unfolding openin_subtopology
     proof (intro exI conjI)
      show {x ∈ ?S. g x ∈ T} = (⋃ r ∈ ?S ∩ g −‘ T. ball r (d r)) ∩ f ‘ topspace
X
         using d by (auto simp: gf)
     qed auto
   qed
   then show continuous_map (top_of_set ?S) X g
     by (simp add: 1 continuous_map)
  qed (auto simp: gf)
 qed
qed
```

An injective function into R is a homeomorphism and so an open map.

```
lemma injective_into_1d_eq_homeomorphism:
 fixes f :: 'a::topological_space ⇒ real
 assumes f: continuous_on S f and S: path_connected S
 shows inj_on f S ⟷ (∃ g. homeomorphism S (f ‘ S) f g)
proof
 show ∃ g. homeomorphism S (f ‘ S) f g
   if inj_on f S
 proof −
   have embedding_map (top_of_set S) euclideanreal f
```

using *that embedding_map_into_euclideanreal* [*of top_of_set S f*] *assms* **by** *auto*
then show *?thesis*
by (*simp add*: *embedding_map_def*) (*metis all_closedin_homeomorphic_image f homeomorphism_injective_closed_map that*)
qed
qed (*metis homeomorphism_def inj_onI*)

lemma *injective_into_1d_imp_open_map*:
fixes *f* :: *'a*::*topological_space* ⇒ *real*
assumes *continuous_on S f path_connected S inj_on f S openin* (*subtopology euclidean S*) *T*
shows *openin* (*subtopology euclidean* (*f* ‘ *S*)) (*f* ‘ *T*)
using *assms homeomorphism_imp_open_map injective_into_1d_eq_homeomorphism*
by *blast*

lemma *homeomorphism_into_1d*:
fixes *f* :: *'a*::*topological_space* ⇒ *real*
assumes *path_connected S continuous_on S f f* ‘ *S* = *T inj_on f S*
shows ∃ *g*. *homeomorphism S T f g*
using *assms injective_into_1d_eq_homeomorphism* **by** *blast*

5.1.28 Rectangular paths

definition *rectpath* **where**
rectpath a1 a3 = (*let a2* = *Complex* (*Re a3*) (*Im a1*); *a4* = *Complex* (*Re a1*) (*Im a3*)
in linepath a1 a2 +++ linepath a2 a3 +++ linepath a3 a4 +++ linepath a4 a1)

lemma *path_rectpath* [*simp*, *intro*]: *path* (*rectpath a b*)
by (*simp add*: *Let_def rectpath_def*)

lemma *pathstart_rectpath* [*simp*]: *pathstart* (*rectpath a1 a3*) = *a1*
by (*simp add*: *rectpath_def Let_def*)

lemma *pathfinish_rectpath* [*simp*]: *pathfinish* (*rectpath a1 a3*) = *a1*
by (*simp add*: *rectpath_def Let_def*)

lemma *simple_path_rectpath* [*simp*, *intro*]:
assumes *Re a1* ≠ *Re a3 Im a1* ≠ *Im a3*
shows *simple_path* (*rectpath a1 a3*)
unfolding *rectpath_def Let_def* **using** *assms*
by (*intro simple_path_join_loop arc_join arc_linepath*)
(*auto simp*: *complex_eq_iff path_image_join closed_segment_same_Re closed_segment_same_Im*)

lemma *path_image_rectpath*:
assumes *Re a1* ≤ *Re a3 Im a1* ≤ *Im a3*
shows *path_image* (*rectpath a1 a3*) =

$\{z.\ Re\ z \in \{Re\ a1,\ Re\ a3\} \wedge Im\ z \in \{Im\ a1..Im\ a3\}\} \cup$
$\{z.\ Im\ z \in \{Im\ a1,\ Im\ a3\} \wedge Re\ z \in \{Re\ a1..Re\ a3\}\}$ (**is** *?lhs = ?rhs*)
proof −
define *a2 a4* **where** *a2 = Complex* (*Re a3*) (*Im a1*) **and** *a4 = Complex* (*Re a1*) (*Im a3*)
have *?lhs = closed_segment a1 a2 ∪ closed_segment a2 a3 ∪ closed_segment a4 a3 ∪ closed_segment a1 a4*
by (*simp_all add*: *rectpath_def Let_def path_image_join closed_segment_commute a2_def a4_def Un_assoc*)
also have ... = *?rhs* **using** *assms*
by (*auto simp*: *rectpath_def Let_def path_image_join a2_def a4_def closed_segment_same_Re closed_segment_same_Im closed_segment_eq_real_ivl*)
finally show *?thesis* **.**
qed

lemma *path_image_rectpath_subset_cbox*:
assumes *Re a ≤ Re b Im a ≤ Im b*
shows *path_image* (*rectpath a b*) ⊆ *cbox a b*
using *assms* **by** (*auto simp*: *path_image_rectpath in_cbox_complex_iff*)

lemma *path_image_rectpath_inter_box*:
assumes *Re a ≤ Re b Im a ≤ Im b*
shows *path_image* (*rectpath a b*) ∩ *box a b* = {}
using *assms* **by** (*auto simp*: *path_image_rectpath in_box_complex_iff*)

lemma *path_image_rectpath_cbox_minus_box*:
assumes *Re a ≤ Re b Im a ≤ Im b*
shows *path_image* (*rectpath a b*) = *cbox a b* − *box a b*
using *assms* **by** (*auto simp*: *path_image_rectpath in_cbox_complex_iff in_box_complex_iff*)

end

5.2 Neighbourhood bases and Locally path-connected spaces

theory *Locally*
imports
Path_Connected Function_Topology Sum_Topology
begin

5.2.1 Neighbourhood Bases

Useful for "local" properties of various kinds

definition *neighbourhood_base_at* **where**
neighbourhood_base_at x P X ≡
∀ *W*. *openin X W ∧ x ∈ W*
⟶ (∃ *U V*. *openin X U ∧ P V ∧ x ∈ U ∧ U ⊆ V ∧ V ⊆ W*)

definition *neighbourhood_base_of* **where**
neighbourhood_base_of P X ≡
∀ *x* ∈ *topspace X*. *neighbourhood_base_at x P X*

lemma *neighbourhood_base_of*:
neighbourhood_base_of P X ⟷
(∀ *W x*. *openin X W* ∧ *x* ∈ *W*
⟶ (∃ *U V*. *openin X U* ∧ *P V* ∧ *x* ∈ *U* ∧ *U* ⊆ *V* ∧ *V* ⊆ *W*))
unfolding *neighbourhood_base_at_def neighbourhood_base_of_def*
using *openin_subset* **by** *blast*

lemma *neighbourhood_base_at_mono*:
⟦*neighbourhood_base_at x P X*; ⋀*S*. ⟦*P S*; *x* ∈ *S*⟧ ⟹ *Q S*⟧ ⟹ *neighbourhood_base_at x Q X*
unfolding *neighbourhood_base_at_def*
by (*meson subset_eq*)

lemma *neighbourhood_base_of_mono*:
⟦*neighbourhood_base_of P X*; ⋀*S*. *P S* ⟹ *Q S*⟧ ⟹ *neighbourhood_base_of Q X*
unfolding *neighbourhood_base_of_def*
using *neighbourhood_base_at_mono* **by** *force*

lemma *open_neighbourhood_base_at*:
(⋀*S*. ⟦*P S*; *x* ∈ *S*⟧ ⟹ *openin X S*)
⟹ *neighbourhood_base_at x P X* ⟷ (∀ *W*. *openin X W* ∧ *x* ∈ *W* ⟶ (∃ *U*. *P U* ∧ *x* ∈ *U* ∧ *U* ⊆ *W*))
unfolding *neighbourhood_base_at_def*
by (*meson subsetD*)

lemma *open_neighbourhood_base_of*:
(∀ *S*. *P S* ⟶ *openin X S*)
⟹ *neighbourhood_base_of P X* ⟷ (∀ *W x*. *openin X W* ∧ *x* ∈ *W* ⟶ (∃ *U*. *P U* ∧ *x* ∈ *U* ∧ *U* ⊆ *W*))
by (*smt* (*verit*) *neighbourhood_base_of subsetD*)

lemma *neighbourhood_base_of_open_subset*:
⟦*neighbourhood_base_of P X*; *openin X S*⟧
⟹ *neighbourhood_base_of P* (*subtopology X S*)
by (*smt* (*verit*, *ccfv_SIG*) *neighbourhood_base_of openin_open_subtopology subset_trans*)

lemma *neighbourhood_base_of_topology_base*:
openin X = *arbitrary union_of* (λ*W*. *W* ∈ ℬ)
⟹ *neighbourhood_base_of P X* ⟷
(∀ *W x*. *W* ∈ ℬ ∧ *x* ∈ *W* ⟶ (∃ *U V*. *openin X U* ∧ *P V* ∧ *x* ∈ *U* ∧ *U* ⊆ *V* ∧ *V* ⊆ *W*))
by (*smt* (*verit*, *del_insts*) *neighbourhood_base_of openin_topology_base_unique subset_trans*)

lemma *neighbourhood_base_at_unlocalized*:
assumes $\bigwedge S\ T.\ [\![P\ S;\ openin\ X\ T;\ x \in T;\ T \subseteq S]\!] \Longrightarrow P\ T$
shows *neighbourhood_base_at x P X*
$\longleftrightarrow (x \in topspace\ X \longrightarrow (\exists\ U\ V.\ openin\ X\ U \wedge P\ V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq topspace\ X))$
(**is** *?lhs = ?rhs*)
proof
assume *R*: *?rhs* **show** *?lhs*
unfolding *neighbourhood_base_at_def*
proof *clarify*
fix *W*
assume *openin X W x ∈ W*
then have *x ∈ topspace X*
using *openin_subset* **by** *blast*
with *R* **obtain** *U V* **where** *openin X U P V x ∈ U U ⊆ V V ⊆ topspace X*
by *blast*
then show $\exists\ U\ V.$ *openin X U ∧ P V ∧ x ∈ U ∧ U ⊆ V ∧ V ⊆ W*
by (*metis IntI ‹openin X W› ‹x ∈ W› assms inf_le1 inf_le2 openin_Int*)
qed
qed (*simp add*: *neighbourhood_base_at_def*)

lemma *neighbourhood_base_at_with_subset*:
$[\![$*openin X U*; *x ∈ U*$]\!]$
$\Longrightarrow$ (*neighbourhood_base_at x P X* $\longleftrightarrow$ *neighbourhood_base_at x* ($\lambda T.\ T \subseteq U \wedge P\ T$) *X*)
unfolding *neighbourhood_base_at_def* **by** (*metis IntI Int_subset_iff openin_Int*)

lemma *neighbourhood_base_of_with_subset*:
neighbourhood_base_of P X $\longleftrightarrow$ *neighbourhood_base_of* ($\lambda t.\ t \subseteq topspace\ X \wedge P\ t$) *X*
using *neighbourhood_base_at_with_subset*
by (*fastforce simp add*: *neighbourhood_base_of_def*)

5.2.2 Locally path-connected spaces

definition *weakly_locally_path_connected_at*
where *weakly_locally_path_connected_at x X* ≡ *neighbourhood_base_at x* (*path_connectedin X*) *X*

definition *locally_path_connected_at*
where *locally_path_connected_at x X* ≡
neighbourhood_base_at x ($\lambda U.$ *openin X U ∧ path_connectedin X U*) *X*

definition *locally_path_connected_space*
where *locally_path_connected_space X* ≡ *neighbourhood_base_of* (*path_connectedin X*) *X*

lemma *locally_path_connected_space_alt*:

locally_path_connected_space X ⟷ *neighbourhood_base_of* (λ*U*. *openin X U* ∧ *path_connectedin X U*) *X*
(**is** *?P* = *?Q*)
and *locally_path_connected_space_eq_open_path_component_of*:
locally_path_connected_space X ⟷
(∀ *U x*. *openin X U* ∧ *x* ∈ *U* ⟶ *openin X* (*Collect* (*path_component_of* (*subtopology X U*) *x*)))
(**is** *?P* = *?R*)
proof −
have *?P* **if** *?Q*
using *locally_path_connected_space_def neighbourhood_base_of_mono that*
by *auto*
moreover have *?R* **if** *P*: *?P*
proof −
show *?thesis*
proof *clarify*
fix *U y*
assume *openin X U y* ∈ *U*
have ∃ *T*. *openin X T* ∧ *x* ∈ *T* ∧ *T* ⊆ *Collect* (*path_component_of* (*subtopology X U*) *y*)
if *path_component_of* (*subtopology X U*) *y x* **for** *x*

proof −
have *x* ∈ *U*
using *path_component_of_equiv that topspace_subtopology* **by** *fastforce*
then have ∃ *Ua*. *openin X Ua* ∧ (∃ *V*. *path_connectedin X V* ∧ *x* ∈ *Ua* ∧ *Ua* ⊆ *V* ∧ *V* ⊆ *U*)
using *P*
by (*simp add*: ‹*openin X U*› *locally_path_connected_space_def neighbourhood_base_of*)
then show *?thesis*
by (*metis dual_order.trans path_component_of_equiv path_component_of_maximal path_connectedin_subtopology subset_iff that*)
qed
then show *openin X* (*Collect* (*path_component_of* (*subtopology X U*) *y*))
using *openin_subopen* **by** *force*
qed
qed
moreover have *?Q* **if** *?R*
by (*smt* (*verit*) *mem_Collect_eq open_neighbourhood_base_of openin_subset path_component_of_refl*
path_connectedin_path_component_of path_connectedin_subtopology that topspace_subtopology_subset)
ultimately show *?P* = *?Q* *?P* = *?R*
by *blast*+
qed

lemma *locally_path_connected_space*:
locally_path_connected_space X

$\longleftrightarrow$ ($\forall$ *V x*. *openin X V* $\wedge$ *x* $\in$ *V* $\longrightarrow$ ($\exists$ *U*. *openin X U* $\wedge$ *path_connectedin X U* $\wedge$ *x* $\in$ *U* $\wedge$ *U* $\subseteq$ *V*))
by (*simp add*: *locally_path_connected_space_alt open_neighbourhood_base_of*)

lemma *locally_path_connected_space_open_path_components*:
locally_path_connected_space X $\longleftrightarrow$
($\forall$ *U C*. *openin X U* $\wedge$ *C* $\in$ *path_components_of*(*subtopology X U*) $\longrightarrow$ *openin X C*)
apply (*simp add*: *locally_path_connected_space_eq_open_path_component_of path_components_of_def*)
by (*smt* (*verit*, *ccfv_threshold*) *Int_iff image_iff openin_subset subset_iff*)

lemma *openin_path_component_of_locally_path_connected_space*:
locally_path_connected_space X $\Longrightarrow$ *openin X* (*Collect* (*path_component_of X x*))
using *locally_path_connected_space_eq_open_path_component_of openin_subopen path_component_of_eq_empty*
by *fastforce*

lemma *openin_path_components_of_locally_path_connected_space*:
$[\![$*locally_path_connected_space X*; *C* $\in$ *path_components_of X*$]\!] \Longrightarrow$ *openin X C*
using *locally_path_connected_space_open_path_components* **by** *force*

lemma *closedin_path_components_of_locally_path_connected_space*:
$[\![$*locally_path_connected_space X*; *C* $\in$ *path_components_of X*$]\!] \Longrightarrow$ *closedin X C*
unfolding *closedin_def*
by (*metis Diff_iff complement_path_components_of_Union openin_clauses*(3) *openin_closedin_eq*
openin_path_components_of_locally_path_connected_space)

lemma *closedin_path_component_of_locally_path_connected_space*:
assumes *locally_path_connected_space X*
shows *closedin X* (*Collect* (*path_component_of X x*))
proof (*cases x* $\in$ *topspace X*)
case *True*
then show *?thesis*
by (*simp add*: *assms closedin_path_components_of_locally_path_connected_space path_component_in_path_components_of*)
next
case *False*
then show *?thesis*
by (*metis closedin_empty path_component_of_eq_empty*)
qed

lemma *weakly_locally_path_connected_at*:
weakly_locally_path_connected_at x X $\longleftrightarrow$
($\forall$ *V*. *openin X V* $\wedge$ *x* $\in$ *V*

```
        ⟶ (∃ U. openin X U ∧ x ∈ U ∧ U ⊆ V ∧
              (∀ y ∈ U. ∃ C. path_connectedin X C ∧ C ⊆ V ∧ x ∈ C ∧ y ∈ C)))
      (is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
   by (smt (verit) neighbourhood_base_at_def subset_iff weakly_locally_path_connected_at_def)
next
  have ∗: ∃ V. path_connectedin X V ∧ U ⊆ V ∧ V ⊆ W
    if (∀ y∈U. ∃ C. path_connectedin X C ∧ C ⊆ W ∧ x ∈ C ∧ y ∈ C)
    for W U
  proof (intro exI conjI)
    let ?V = (Collect (path_component_of (subtopology X W) x))
      show path_connectedin X (Collect (path_component_of (subtopology X W)
x))
      by (meson path_connectedin_path_component_of path_connectedin_subtopology)
      show U ⊆ ?V
        by (auto simp: path_component_of path_connectedin_subtopology that)
      show ?V ⊆ W
      by (meson path_connectedin_path_component_of path_connectedin_subtopology)
    qed
  assume ?rhs
  then show ?lhs
   unfolding weakly_locally_path_connected_at_def neighbourhood_base_at_def
by (metis ∗)
qed

lemma locally_path_connected_space_im_kleinen:
   locally_path_connected_space X ⟷
     (∀ V x. openin X V ∧ x ∈ V
           ⟶ (∃ U. openin X U ∧
                 x ∈ U ∧ U ⊆ V ∧
                 (∀ y ∈ U. ∃ C. path_connectedin X C ∧
                       C ⊆ V ∧ x ∈ C ∧ y ∈ C)))
       (is ?lhs = ?rhs)
proof
  show ?lhs ⟹ ?rhs
    by (metis locally_path_connected_space)
  assume ?rhs
  then show ?lhs
    unfolding locally_path_connected_space_def neighbourhood_base_of
   by (metis neighbourhood_base_at_def weakly_locally_path_connected_at weakly_locally_path_connected_at_de
qed

lemma locally_path_connected_space_open_subset:
   ⟦locally_path_connected_space X; openin X S⟧
       ⟹ locally_path_connected_space (subtopology X S)
   by (smt (verit, best) locally_path_connected_space openin_open_subtopology
path_connectedin_subtopology subset_trans)
```

lemma *locally_path_connected_space_quotient_map_image*:
assumes *f*: *quotient_map X Y f* **and** *X*: *locally_path_connected_space X*
shows *locally_path_connected_space Y*
unfolding *locally_path_connected_space_open_path_components*
proof *clarify*
fix *V C*
assume *V*: *openin Y V* **and** *c*: *C ∈ path_components_of (subtopology Y V)*
then have *sub*: *C ⊆ topspace Y*
using *path_connectedin_path_components_of path_connectedin_subset_topspace path_connectedin_subtopology* **by** *blast*
have *openin X {x ∈ topspace X. f x ∈ C}*
proof (*subst openin_subopen*, *clarify*)
fix *x*
assume *x*: *x ∈ topspace X* **and** *f x ∈ C*
let *?T = Collect (path_component_of (subtopology X {z ∈ topspace X. f z ∈ V}) x)*
show *∃ T. openin X T ∧ x ∈ T ∧ T ⊆ {x ∈ topspace X. f x ∈ C}*
proof (*intro exI conjI*)
have *∗*: *∃ U. openin X U ∧ ?T ∈ path_components_of (subtopology X U)*
proof (*intro exI conjI*)
show *openin X ({z ∈ topspace X. f z ∈ V})*
using *V f openin_subset quotient_map_def* **by** *fastforce*
have *x ∈ topspace (subtopology X {z ∈ topspace X. f z ∈ V})*
using *‹f x ∈ C› c path_components_of_subset x* **by** *force*
then show *Collect (path_component_of (subtopology X {z ∈ topspace X. f z ∈ V}) x) ∈ path_components_of (subtopology X {z ∈ topspace X. f z ∈ V})*
by (*meson path_component_in_path_components_of*)
qed
with *X* **show** *openin X ?T*
using *locally_path_connected_space_open_path_components* **by** *blast*
show *x*: *x ∈ ?T*
using *∗ nonempty_path_components_of path_component_of_eq path_component_of_eq_empty* **by** *fastforce*
have *f ‘ ?T ⊆ C*
proof (*rule path_components_of_maximal*)
show *C ∈ path_components_of (subtopology Y V)*
by (*simp add*: *c*)
show *path_connectedin (subtopology Y V) (f ‘ ?T)*
proof −
have *quotient_map (subtopology X {a ∈ topspace X. f a ∈ V}) (subtopology Y V) f*
by (*simp add*: *V f quotient_map_restriction*)
then show *?thesis*
by (*meson path_connectedin_continuous_map_image path_connectedin_path_component_of quotient_imp_continuous_map*)
qed
show *¬ disjnt C (f ‘ ?T)*
by (*metis* (*no_types*, *lifting*) *‹f x ∈ C› x disjnt_iff image_eqI*)

```
      qed
      then show ?T ⊆ {x ∈ topspace X. f x ∈ C}
        by (force simp: path_component_of_equiv)
    qed
  qed
  then show openin Y C
    using f sub by (simp add: quotient_map_def)
qed

lemma homeomorphic_locally_path_connected_space:
  assumes X homeomorphic_space Y
  shows locally_path_connected_space X ⟷ locally_path_connected_space Y
  using assms
   unfolding homeomorphic_space_def homeomorphic_map_def homeomorphic_maps_map
    by (metis locally_path_connected_space_quotient_map_image)

lemma locally_path_connected_space_retraction_map_image:
   ⟦retraction_map X Y r; locally_path_connected_space X⟧
        ⟹ locally_path_connected_space Y
 using Abstract_Topology.retraction_imp_quotient_map locally_path_connected_space_quotient_map_image
by blast

lemma locally_path_connected_space_euclideanreal: locally_path_connected_space
euclideanreal
  unfolding locally_path_connected_space_def neighbourhood_base_of
  proof clarsimp
  fix W and x :: real
  assume open W x ∈ W
  then obtain e where e > 0 and e: ⋀x'. |x' − x| < e ⟶ x' ∈ W
    by (auto simp: open_real)
  then show ∃ U. open U ∧ (∃ V. path_connected V ∧ x ∈ U ∧ U ⊆ V ∧ V ⊆
W)
   by (force intro!: convex_imp_path_connected exI [where x = {x−e<..<x+e}])
qed

lemma locally_path_connected_space_discrete_topology:
   locally_path_connected_space (discrete_topology U)
  using locally_path_connected_space_open_path_components by fastforce

lemma path_component_eq_connected_component_of:
  assumes locally_path_connected_space X
  shows path_component_of_set X x = connected_component_of_set X x
proof (cases x ∈ topspace X)
  case True
  have path_component_of_set X x ⊆ connected_component_of_set X x
    by (simp add: path_component_subset_connected_component_of)
  moreover have closedin X (path_component_of_set X x)
   by (simp add: assms closedin_path_component_of_locally_path_connected_space)
  moreover have openin X (path_component_of_set X x)
```

by (*simp add*: *assms openin_path_component_of_locally_path_connected_space*)
moreover have *path_component_of_set X x* ≠ {}
by (*meson True path_component_of_eq_empty*)
ultimately show *?thesis*
using *connectedin_connected_component_of* [*of X x*] **unfolding** *connectedin_def*
by (*metis closedin_subset_topspace connected_space_clopen_in subset_openin_subtopology topspace_subtopology_subset*)
next
case *False*
then show *?thesis*
using *connected_component_of_eq_empty path_component_of_eq_empty* **by** *fastforce*
qed

lemma *path_components_eq_connected_components_of*:
locally_path_connected_space X ⟹ (*path_components_of X* = *connected_components_of X*)
by (*simp add*: *path_components_of_def connected_components_of_def image_def path_component_eq_connected_component_of*)

lemma *path_connected_eq_connected_space*:
locally_path_connected_space X
⟹ *path_connected_space X* ⟷ *connected_space X*
by (*metis connected_components_of_subset_sing path_components_eq_connected_components_of path_components_of_subset_singleton*)

lemma *locally_path_connected_space_product_topology*:
locally_path_connected_space(*product_topology X I*) ⟷
(*product_topology X I*) = *trivial_topology* ∨
finite {*i*. *i* ∈ *I* ∧ ~*path_connected_space*(*X i*)} ∧
(∀ *i* ∈ *I*. *locally_path_connected_space*(*X i*))
(**is** *?lhs* ⟷ *?empty* ∨ *?rhs*)
proof (*cases ?empty*)
case *True*
then show *?thesis*
by (*simp add*: *locally_path_connected_space_def neighbourhood_base_of openin_closedin_eq*)
next
case *False*
then obtain *z* **where** *z*: *z* ∈ (Π$_E$ *i*∈*I*. *topspace* (*X i*))
using *discrete_topology_unique_derived_set* **by** (*fastforce iff*: *null_topspace_iff_trivial*)
have *?rhs* **if** *L*: *?lhs*
proof −
obtain *U C* **where** *U*: *openin* (*product_topology X I*) *U*
and *V*: *path_connectedin* (*product_topology X I*) *C*
and *z* ∈ *U U* ⊆ *C* **and** *Csub*: *C* ⊆ (Π$_E$ *i*∈*I*. *topspace* (*X i*))
by (*metis L locally_path_connected_space openin_topspace topspace_product_topology z*)
then obtain *V* **where** *finV*: *finite* {*i* ∈ *I*. *V i* ≠ *topspace* (*X i*)}

```
    and XV: ⋀i. i∈I ⟹ openin (X i) (V i) and z ∈ Pi_E I V and subU: Pi_E
I V ⊆ U
    by (force simp: openin_product_topology_alt)
  show ?thesis
  proof (intro conjI ballI)
    have path_connected_space (X i) if i ∈ I V i = topspace (X i) for i
    proof −
      have pc: path_connectedin (X i) ((λx. x i) ‘ C)
     by (metis V continuous_map_product_projection path_connectedin_continuous_map_image
that(1))
      moreover have ((λx. x i) ‘ C) = topspace (X i)
      proof
        show (λx. x i) ‘ C ⊆ topspace (X i)
          by (simp add: pc path_connectedin_subset_topspace)
        have V i ⊆ (λx. x i) ‘ (Π_E i∈I. V i)
          by (metis ‹z ∈ Pi_E I V› empty_iff image_projection_PiE order_refl
that(1))
        also have ... ⊆ (λx. x i) ‘ U
          using subU by blast
        finally show topspace (X i) ⊆ (λx. x i) ‘ C
          using ‹U ⊆ C› that by blast
      qed
      ultimately show ?thesis
        by (simp add: path_connectedin_topspace)
    qed
    then have {i ∈ I. ¬ path_connected_space (X i)} ⊆ {i ∈ I. V i ≠ topspace
(X i)}
      by blast
    with finV show finite {i ∈ I. ¬ path_connected_space (X i)}
      using finite_subset by blast
  next
    show locally_path_connected_space (X i) if i ∈ I for i
      by (meson False L locally_path_connected_space_quotient_map_image
quotient_map_product_projection that)
  qed
 qed
 moreover have ?lhs if R: ?rhs
 proof (clarsimp simp add: locally_path_connected_space_def neighbourhood_base_of)
  fix F z
  assume openin (product_topology X I) F and z ∈ F
  then obtain W where finW: finite {i ∈ I. W i ≠ topspace (X i)}
        and opeW: ⋀i. i ∈ I ⟹ openin (X i) (W i) and z ∈ Pi_E I W Pi_E I
W ⊆ F
    by (auto simp: openin_product_topology_alt)
  have ∀ i ∈ I. ∃ U C. openin (X i) U ∧ path_connectedin (X i) C ∧ z i ∈ U ∧
U ⊆ C ∧ C ⊆ W i ∧
                    (W i = topspace (X i) ∧
                     path_connected_space (X i) ⟶ U = topspace (X i) ∧ C =
topspace (X i))
```

```
        (is ∀ i ∈ I. ?Φ i)
    proof
      fix i assume i ∈ I
      have locally_path_connected_space (X i)
        by (simp add: R ‹i ∈ I›)
      moreover have ∗:openin (X i) (W i)  z i ∈ W i
        using ‹z ∈ Pi_E I W› opeW ‹i ∈ I› by auto
      ultimately obtain U C where UC: openin (X i) U path_connectedin (X i)
C z i ∈ U U ⊆ C C ⊆ W i
        using ‹i ∈ I› by (force simp: locally_path_connected_space_def neighbour-
hood_base_of)
      show ?Φ i
        by (metis UC ∗ openin_subset path_connectedin_topspace)
    qed
    then obtain U C where
      ∗: ⋀i. i ∈ I ⟹ openin (X i) (U i) ∧ path_connectedin (X i) (C i) ∧ z i ∈
(U i) ∧ (U i) ⊆ (C i) ∧ (C i) ⊆ W i ∧
                    (W i = topspace (X i) ∧ path_connected_space (X i)
                     ⟶ (U i) = topspace (X i) ∧ (C i) = topspace (X i))
      by metis
    let ?A = {i ∈ I. ¬ path_connected_space (X i)} ∪ {i ∈ I. W i ≠ topspace (X
i)}
    have {i ∈ I. U i ≠ topspace (X i)} ⊆ ?A
      by (clarsimp simp add: ∗)
    moreover have finite ?A
      by (simp add: that finW)
    ultimately have finite {i ∈ I. U i ≠ topspace (X i)}
      using finite_subset by auto
    with ∗ have openin (product_topology X I) (Pi_E I U)
      by (simp add: openin_PiE_gen)
    then show ∃ U. openin (product_topology X I) U ∧
              (∃ V. path_connectedin (product_topology X I) V ∧ z ∈ U ∧ U ⊆ V
∧ V ⊆ F)
      by (metis (no_types, opaque_lifting) subsetI ‹z ∈ Pi_E I W› ‹Pi_E I W ⊆ F›
∗ path_connectedin_PiE
          PiE_iff PiE_mono order.trans)
  qed
  ultimately show ?thesis
    using False by blast
qed

lemma locally_path_connected_is_realinterval:
  assumes is_interval S
  shows locally_path_connected_space(subtopology euclideanreal S)
  unfolding locally_path_connected_space_def
proof (clarsimp simp add: neighbourhood_base_of openin_subtopology_alt)
  fix a U
  assume a ∈ S and a ∈ U and open U
  then obtain r where r > 0 and r: ball a r ⊆ U
```

```
    by (metis open_contains_ball_eq)
  show ∃ W. open W ∧ (∃ V. path_connectedin (top_of_set S) V ∧ a ∈ W ∧ S
∩ W ⊆ V ∧ V ⊆ S ∧ V ⊆ U)
  proof (intro exI conjI)
    show path_connectedin (top_of_set S) (S ∩ ball a r)
    by (simp add: assms is_interval_Int is_interval_ball_real is_interval_path_connected
path_connectedin_subtopology)
    show a ∈ ball a r
      by (simp add: ‹0 < r›)
  qed (use ‹0 < r› r in auto)
qed

lemma locally_path_connected_real_interval:
 locally_path_connected_space (subtopology euclideanreal{a..b})
  locally_path_connected_space (subtopology euclideanreal{a<..<b})
  using locally_path_connected_is_realinterval
  by (auto simp add: is_interval_1)

lemma locally_path_connected_space_prod_topology:
   locally_path_connected_space (prod_topology X Y) ⟷
      (prod_topology X Y) = trivial_topology ∨
    locally_path_connected_space X ∧ locally_path_connected_space Y (is ?lhs=?rhs)
proof (cases (prod_topology X Y) = trivial_topology)
  case True
  then show ?thesis
    using locally_path_connected_space_discrete_topology by force
next
  case False
  then have ne: X ≠ trivial_topology Y ≠ trivial_topology
    by simp_all
  show ?thesis
  proof
    assume ?lhs then show ?rhs
      by (meson locally_path_connected_space_quotient_map_image ne(1) ne(2)
quotient_map_fst quotient_map_snd)
  next
    assume ?rhs
   with False have X: locally_path_connected_space X and Y: locally_path_connected_space
Y
      by auto
    show ?lhs
      unfolding locally_path_connected_space_def neighbourhood_base_of
    proof clarify
      fix UV x y
      assume UV: openin (prod_topology X Y) UV and (x,y) ∈ UV
      obtain A B where W12: openin X A ∧ openin Y B ∧ x ∈ A ∧ y ∈ B ∧ A
× B ⊆ UV
        using X Y by (metis UV ‹(x,y) ∈ UV› openin_prod_topology_alt)
      then obtain C D K L
```

where *openin X C path_connectedin X K x ∈ C C ⊆ K K ⊆ A*
openin Y D path_connectedin Y L y ∈ D D ⊆ L L ⊆ B
by (*metis X Y locally_path_connected_space*)
with *W12* ‹*openin X C*› ‹*openin Y D*›
show ∃ *U V*. *openin* (*prod_topology X Y*) *U* ∧ *path_connectedin* (*prod_topology X Y*) *V* ∧ (*x*, *y*) ∈ *U* ∧ *U* ⊆ *V* ∧ *V* ⊆ *UV*
apply (*rule_tac x*=*C* × *D* **in** *exI*)
apply (*rule_tac x*=*K* × *L* **in** *exI*)
apply (*fastforce simp*: *openin_prod_Times_iff path_connectedin_Times*)
done
qed
qed
qed

lemma *locally_path_connected_space_sum_topology*:
locally_path_connected_space(*sum_topology X I*) ⟷
(∀ *i* ∈ *I*. *locally_path_connected_space* (*X i*)) (**is** *?lhs*=*?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*smt* (*verit*) *homeomorphic_locally_path_connected_space locally_path_connected_space_open_s*
topological_property_of_sum_component)
next
assume *R*: *?rhs*
show *?lhs*
proof (*clarsimp simp add*: *locally_path_connected_space_def neighbourhood_base_of forall_openin_sum_topology imp_conjL*)
fix *W i x*
assume *ope*: ∀ *i*∈*I*. *openin* (*X i*) (*W i*)
and *i* ∈ *I* **and** *x* ∈ *W i*
then obtain *U V* **where** *U*: *openin* (*X i*) *U* **and** *V*: *path_connectedin* (*X i*) *V*
and *x* ∈ *U U* ⊆ *V V* ⊆ *W i*
by (*metis R* ‹*i* ∈ *I*› ‹*x* ∈ *W i*› *locally_path_connected_space*)
show ∃ *U*. *openin* (*sum_topology X I*) *U* ∧ (∃ *V*. *path_connectedin* (*sum_topology X I*) *V* ∧ (*i*, *x*) ∈ *U* ∧ *U* ⊆ *V* ∧ *V* ⊆ *Sigma I W*)
proof (*intro exI conjI*)
show *openin* (*sum_topology X I*) (*Pair i* ‘ *U*)
by (*meson U* ‹*i* ∈ *I*› *open_map_component_injection open_map_def*)
show *path_connectedin* (*sum_topology X I*) (*Pair i* ‘ *V*)
by (*meson V* ‹*i* ∈ *I*› *continuous_map_component_injection path_connectedin_continuous_map_i*
show *Pair i* ‘ *V* ⊆ *Sigma I W*
using ‹*V* ⊆ *W i*› ‹*i* ∈ *I*› **by** *force*
qed (*use* ‹*x* ∈ *U*› ‹*U* ⊆ *V*› **in** *auto*)
qed
qed

5.2.3 Locally connected spaces

definition *weakly_locally_connected_at*

where *weakly_locally_connected_at x X* ≡ *neighbourhood_base_at x* (*connectedin X*) *X*

definition *locally_connected_at*
where *locally_connected_at x X* ≡
neighbourhood_base_at x (λ*U*. *openin X U* ∧ *connectedin X U*) *X*

definition *locally_connected_space*
where *locally_connected_space X* ≡ *neighbourhood_base_of* (*connectedin X*) *X*

lemma *locally_connected_A*: (∀ *U x*. *openin X U* ∧ *x* ∈ *U* ⟶ *openin X* (*connected_component_of_set* (*subtopology X U*) *x*))
⟹ *neighbourhood_base_of* (λ*U*. *openin X U* ∧ *connectedin X U*) *X*
unfolding *neighbourhood_base_of*
by (*metis* (*full_types*) *connected_component_of_refl connectedin_connected_component_of connectedin_subtopology mem_Collect_eq openin_subset topspace_subtopology_subset*)

lemma *locally_connected_B*: *locally_connected_space X* ⟹
(∀ *U x*. *openin X U* ∧ *x* ∈ *U* ⟶ *openin X* (*connected_component_of_set* (*subtopology X U*) *x*))
unfolding *locally_connected_space_def neighbourhood_base_of*
apply (*erule all_forward*)
apply *clarify*
apply (*subst openin_subopen*)
by (*smt* (*verit*, *ccfv_threshold*) *Ball_Collect connected_component_of_def connected_component_of_equiv connectedin_subtopology in_mono mem_Collect_eq*)

lemma *locally_connected_C*: *neighbourhood_base_of* (λ*U*. *openin X U* ∧ *connectedin X U*) *X* ⟹ *locally_connected_space X*
using *locally_connected_space_def neighbourhood_base_of_mono* **by** *auto*

lemma *locally_connected_space_alt*:
locally_connected_space X ⟷ *neighbourhood_base_of* (λ*U*. *openin X U* ∧ *connectedin X U*) *X*
using *locally_connected_A locally_connected_B locally_connected_C* **by** *blast*

lemma *locally_connected_space_eq_open_connected_component_of*:
locally_connected_space X ⟷
(∀ *U x*. *openin X U* ∧ *x* ∈ *U*
⟶ *openin X* (*connected_component_of_set* (*subtopology X U*) *x*))
by (*meson locally_connected_A locally_connected_B locally_connected_C*)

lemma *locally_connected_space*:
locally_connected_space X ⟷
(∀ *V x*. *openin X V* ∧ *x* ∈ *V* ⟶ (∃ *U*. *openin X U* ∧ *connectedin X U* ∧ *x* ∈ *U* ∧ *U* ⊆ *V*))
by (*simp add*: *locally_connected_space_alt open_neighbourhood_base_of*)

lemma *locally_path_connected_imp_locally_connected_space*:
locally_path_connected_space X $\Longrightarrow$ *locally_connected_space X*
by (*simp add*: *locally_connected_space_def locally_path_connected_space_def neighbourhood_base_of_mono path_connectedin_imp_connectedin*)

lemma *locally_connected_space_open_connected_components*:
locally_connected_space X $\longleftrightarrow$
($\forall$ *U C*. *openin X U* $\wedge$ *C* $\in$ *connected_components_of*(*subtopology X U*) $\longrightarrow$ *openin X C*)
unfolding *connected_components_of_def locally_connected_space_eq_open_connected_component_*
by (*smt* (*verit*, *best*) *image_iff openin_subset topspace_subtopology_subset*)

lemma *openin_connected_component_of_locally_connected_space*:
locally_connected_space X $\Longrightarrow$ *openin X* (*connected_component_of_set X x*)
by (*metis connected_component_of_eq_empty locally_connected_B openin_empty openin_topspace subtopology_topspace*)

lemma *openin_connected_components_of_locally_connected_space*:
⟦*locally_connected_space X*; *C* $\in$ *connected_components_of X*⟧ $\Longrightarrow$ *openin X C*
by (*metis locally_connected_space_open_connected_components openin_topspace subtopology_topspace*)

lemma *weakly_locally_connected_at*:
weakly_locally_connected_at x X $\longleftrightarrow$
($\forall$ *V*. *openin X V* $\wedge$ *x* $\in$ *V*
$\longrightarrow$ ($\exists$ *U*. *openin X U* $\wedge$ *x* $\in$ *U* $\wedge$ *U* $\subseteq$ *V* $\wedge$
($\forall$ *y* $\in$ *U*. $\exists$ *C*. *connectedin X C* $\wedge$ *C* $\subseteq$ *V* $\wedge$ *x* $\in$ *C* $\wedge$ *y* $\in$ *C*))) (**is** *?lhs=?rhs*)
proof
assume *?lhs* **then show** *?rhs*
unfolding *neighbourhood_base_at_def weakly_locally_connected_at_def*
by (*meson subsetD subset_trans*)
next
assume *R*: *?rhs*
show *?lhs*
unfolding *neighbourhood_base_at_def weakly_locally_connected_at_def*
proof *clarify*
fix *V*
assume *openin X V* **and** *x* $\in$ *V*
then obtain *U* **where** *openin X U x* $\in$ *U U* $\subseteq$ *V*
and *U*: $\forall$ *y*$\in$*U*. $\exists$ *C*. *connectedin X C* $\wedge$ *C* $\subseteq$ *V* $\wedge$ *x* $\in$ *C* $\wedge$ *y* $\in$ *C*
using *R* **by** *force*
show $\exists$ *A B*. *openin X A* $\wedge$ *connectedin X B* $\wedge$ *x* $\in$ *A* $\wedge$ *A* $\subseteq$ *B* $\wedge$ *B* $\subseteq$ *V*
proof (*intro conjI exI*)
show *connectedin X* (*connected_component_of_set* (*subtopology X V*) *x*)
by (*meson connectedin_connected_component_of connectedin_subtopology*)
show *U* $\subseteq$ *connected_component_of_set* (*subtopology X V*) *x*

```
      using connected_component_of_maximal U
      by (simp add: connected_component_of_def connectedin_subtopology subsetI)
    show connected_component_of_set (subtopology X V) x ⊆ V
      using connected_component_of_subset_topspace by fastforce
  qed (auto simp: ‹x ∈ U› ‹openin X U›)
 qed
qed

lemma locally_connected_space_iff_weak:
 locally_connected_space X ⟷ (∀x∈topspace X. weakly_locally_connected_at x X)
 by (simp add: locally_connected_space_def neighbourhood_base_of_def weakly_locally_connected_at_def)

lemma locally_connected_space_im_kleinen:
  locally_connected_space X ⟷
   (∀V x. openin X V ∧ x ∈ V
       ⟶ (∃U. openin X U ∧ x ∈ U ∧ U ⊆ V ∧
              (∀y∈U. ∃C. connectedin X C ∧ C ⊆ V ∧ x ∈ C ∧ y ∈ C)))
 unfolding locally_connected_space_iff_weak weakly_locally_connected_at
 using openin_subset subsetD by fastforce

lemma locally_connected_space_open_subset:
 ⟦locally_connected_space X; openin X S⟧ ⟹ locally_connected_space (subtopology X S)
 unfolding locally_connected_space_def neighbourhood_base_of
 by (smt (verit) connectedin_subtopology openin_open_subtopology subset_trans)

lemma locally_connected_space_quotient_map_image:
 assumes X: locally_connected_space X and f: quotient_map X Y f
 shows locally_connected_space Y
 unfolding locally_connected_space_open_connected_components
proof clarify
 fix V C
 assume openin Y V and C: C ∈ connected_components_of (subtopology Y V)
 then have C ⊆ topspace Y
   using connected_components_of_subset by force
 have ope1: openin X {a ∈ topspace X. f a ∈ V}
  using ‹openin Y V› f openin_continuous_map_preimage quotient_imp_continuous_map
by blast
 define Vf where Vf ≡ {z ∈ topspace X. f z ∈ V}
 have openin X {x ∈ topspace X. f x ∈ C}
 proof (clarsimp simp: openin_subopen [where S = {x ∈ topspace X. f x ∈ C}])
   fix x
   assume x ∈ topspace X and f x ∈ C
   show ∃T. openin X T ∧ x ∈ T ∧ T ⊆ {x ∈ topspace X. f x ∈ C}
   proof (intro exI conjI)
     show openin X (connected_component_of_set (subtopology X Vf) x)
     by (metis Vf_def X connected_component_of_eq_empty locally_connected_B
```

ope1 openin_empty
openin_subset topspace_subtopology_subset)
show *x_in_conn*: $x \in$ *connected_component_of_set* (*subtopology X Vf*) *x*
using *C Vf_def* ‹*f x* ∈ *C*› ‹*x* ∈ *topspace X*› *connected_component_of_refl*
connected_components_of_subset **by** *fastforce*
have *connected_component_of_set* (*subtopology X Vf*) *x* ⊆ *topspace X* ∩ *Vf*
using *connected_component_of_subset_topspace* **by** *fastforce*
moreover
have *f* ‘ *connected_component_of_set* (*subtopology X Vf*) *x* ⊆ *C*
proof (*rule connected_components_of_maximal* [**where** *X* = *subtopology Y*
V])
show *C* ∈ *connected_components_of* (*subtopology Y V*)
by (*simp add*: *C*)
have §: *quotient_map* (*subtopology X Vf*) (*subtopology Y V*) *f*
by (*simp add*: *Vf_def* ‹*openin Y V*› *f quotient_map_restriction*)
then show *connectedin* (*subtopology Y V*) (*f* ‘ *connected_component_of_set*
(*subtopology X Vf*) *x*)
by (*metis connectedin_connected_component_of connectedin_continuous_map_image*
quotient_imp_continuous_map)
show ¬ *disjnt C* (*f* ‘ *connected_component_of_set* (*subtopology X Vf*) *x*)
using ‹*f x* ∈ *C*› *x_in_conn* **by** (*auto simp*: *disjnt_iff*)
qed
ultimately
show *connected_component_of_set* (*subtopology X Vf*) *x* ⊆ {*x* ∈ *topspace*
X. *f x* ∈ *C*}
by *blast*
qed
qed
then show *openin Y C*
using ‹*C* ⊆ *topspace Y*› *f quotient_map_def* **by** *fastforce*
qed

lemma *locally_connected_space_retraction_map_image*:
⟦*retraction_map X Y r*; *locally_connected_space X*⟧
⟹ *locally_connected_space Y*
using *locally_connected_space_quotient_map_image retraction_imp_quotient_map*
by *blast*

lemma *homeomorphic_locally_connected_space*:
X homeomorphic_space Y ⟹ *locally_connected_space X* ⟷ *locally_connected_space*
Y
by (*meson homeomorphic_map_def homeomorphic_space homeomorphic_space_sym*
locally_connected_space_quotient_map_image)

lemma *locally_connected_space_euclideanreal*: *locally_connected_space euclidean-*
real
by (*simp add*: *locally_path_connected_imp_locally_connected_space locally_path_connected_space_e*

lemma *locally_connected_is_realinterval*:
is_interval S $\Longrightarrow$ *locally_connected_space*(*subtopology euclideanreal S*)
by (*simp add*: *locally_path_connected_imp_locally_connected_space locally_path_connected_is_realinterval*)

lemma *locally_connected_real_interval*:
locally_connected_space (*subtopology euclideanreal*{*a*..*b*})
locally_connected_space (*subtopology euclideanreal*{*a*<..<*b*})
using *connected_Icc is_interval_connected_1 locally_connected_is_realinterval*
by *auto*

lemma *locally_connected_space_discrete_topology*:
locally_connected_space (*discrete_topology U*)
by (*simp add*: *locally_path_connected_imp_locally_connected_space locally_path_connected_space_discrete_top*

lemma *locally_path_connected_imp_locally_connected_at*:
locally_path_connected_at x X $\Longrightarrow$ *locally_connected_at x X*
by (*simp add*: *locally_connected_at_def locally_path_connected_at_def neighbourhood_base_at_mono path_connectedin_imp_connectedin*)

lemma *weakly_locally_path_connected_imp_weakly_locally_connected_at*:
weakly_locally_path_connected_at x X $\Longrightarrow$ *weakly_locally_connected_at x X*
by (*metis path_connectedin_imp_connectedin weakly_locally_connected_at weakly_locally_path_connected_at*)

lemma *interior_of_locally_connected_subspace_component*:
assumes *X*: *locally_connected_space X*
and *C*: *C* $\in$ *connected_components_of* (*subtopology X S*)
shows *X interior_of C* = *C* $\cap$ *X interior_of S*
proof −
obtain *Csub*: *C* $\subseteq$ *topspace X C* $\subseteq$ *S*
by (*meson C connectedin_connected_components_of connectedin_subset_topspace connectedin_subtopology*)
show *?thesis*
proof
show *X interior_of C* $\subseteq$ *C* $\cap$ *X interior_of S*
by (*simp add*: *Csub interior_of_mono interior_of_subset*)
have *eq*: *X interior_of S* = $\bigcup$ (*connected_components_of* (*subtopology X* (*X interior_of S*)))
by (*metis Union_connected_components_of interior_of_subset_topspace topspace_subtopology_subset*)
moreover have *C* $\cap$ *D* $\subseteq$ *X interior_of C*
if *D* $\in$ *connected_components_of* (*subtopology X* (*X interior_of S*)) **for** *D*
proof (*cases C* $\cap$ *D* = {})
case *False*
have *D* $\subseteq$ *X interior_of C*
proof (*rule interior_of_maximal*)
have *connectedin* (*subtopology X S*) *D*
by (*meson connectedin_connected_components_of connectedin_subtopology interior_of_subset subset_trans that*)

```
      then show D ⊆ C
        by (meson C False connected_components_of_maximal disjnt_def)
      show openin X D
     using X locally_connected_space_open_connected_components openin_interior_of
that by blast
    qed
    then show ?thesis
      by blast
  qed auto
  ultimately show C ∩ X interior_of S ⊆ X interior_of C
    by blast
 qed
qed


lemma frontier_of_locally_connected_subspace_component:
 assumes X: locally_connected_space X and closedin X S
   and C: C ∈ connected_components_of (subtopology X S)
 shows X frontier_of C = C ∩ X frontier_of S
proof −
 obtain Csub: C ⊆ topspace X C ⊆ S
  by (meson C connectedin_connected_components_of connectedin_subset_topspace
connectedin_subtopology)
 then have X closure_of C − X interior_of C = C ∩ X closure_of S − C ∩ X
interior_of S
   using assms
  apply (simp add: closure_of_closedin flip: interior_of_locally_connected_subspace_component)
  by (metis closedin_connected_components_of closedin_trans_full closure_of_eq
inf.orderE)
 then show ?thesis
   by (simp add: Diff_Int_distrib frontier_of_def)
qed


lemma locally_connected_space_prod_topology:
  locally_connected_space (prod_topology X Y) ⟷
    (prod_topology X Y) = trivial_topology ∨
    locally_connected_space X ∧ locally_connected_space Y (is ?lhs=?rhs)
proof (cases (prod_topology X Y) = trivial_topology)
 case True
 then show ?thesis
   using locally_connected_space_iff_weak by force
next
 case False
 then have ne: X ≠ trivial_topology Y ≠ trivial_topology
   by simp_all
 show ?thesis
 proof
   assume ?lhs then show ?rhs
```

by (*metis locally_connected_space_quotient_map_image ne quotient_map_fst quotient_map_snd*)
next
assume *?rhs*
with *False* **have** *X*: *locally_connected_space X* **and** *Y*: *locally_connected_space Y*
by *auto*
show *?lhs*
unfolding *locally_connected_space_def neighbourhood_base_of*
proof *clarify*
fix *UV x y*
assume *UV*: *openin* (*prod_topology X Y*) *UV* **and** (*x*,*y*) ∈ *UV*

obtain *A B* **where** *W12*: *openin X A* ∧ *openin Y B* ∧ *x* ∈ *A* ∧ *y* ∈ *B* ∧ *A* × *B* ⊆ *UV*
using *X Y* **by** (*metis UV* ‹(*x*,*y*) ∈ *UV*› *openin_prod_topology_alt*)
then obtain *C D K L*
where *openin X C connectedin X K x* ∈ *C C* ⊆ *K K* ⊆ *A*
openin Y D connectedin Y L y ∈ *D D* ⊆ *L L* ⊆ *B*
by (*metis X Y locally_connected_space*)
with *W12* ‹*openin X C*› ‹*openin Y D*›
show ∃ *U V*. *openin* (*prod_topology X Y*) *U* ∧ *connectedin* (*prod_topology X Y*) *V* ∧ (*x*, *y*) ∈ *U* ∧ *U* ⊆ *V* ∧ *V* ⊆ *UV*
apply (*rule_tac x*=*C* × *D* **in** *exI*)
apply (*rule_tac x*=*K* × *L* **in** *exI*)
apply (*auto simp*: *openin_prod_Times_iff connectedin_Times*)
done
qed
qed
qed

lemma *locally_connected_space_product_topology*:
locally_connected_space(*product_topology X I*) ⟷
(*product_topology X I*) = *trivial_topology* ∨
finite {*i*. *i* ∈ *I* ∧ ~*connected_space*(*X i*)} ∧
(∀ *i* ∈ *I*. *locally_connected_space*(*X i*))
(**is** *?lhs* ⟷ *?empty* ∨ *?rhs*)
proof (*cases ?empty*)
case *True*
then show *?thesis*
by (*simp add*: *locally_connected_space_def neighbourhood_base_of openin_closedin_eq*)
next
case *False*
then obtain *z* **where** *z*: *z* ∈ (Π$_E$ *i*∈*I*. *topspace* (*X i*))
using *discrete_topology_unique_derived_set* **by** (*fastforce iff*: *null_topspace_iff_trivial*)
have *?rhs* **if** *L*: *?lhs*
proof −
obtain *U C* **where** *U*: *openin* (*product_topology X I*) *U*

and V: *connectedin* (*product_topology X I*) C
and $z \in U$ $U \subseteq C$ **and** *Csub*: $C \subseteq (\Pi_E\ i{\in}I.\ topspace\ (X\ i))$
using L **apply** (*clarsimp simp add*: *locally_connected_space_def neighbourhood_base_of*)
by (*metis openin_topspace topspace_product_topology z*)
then obtain V **where** *finV*: *finite* $\{i \in I.\ V\ i \neq topspace\ (X\ i)\}$
and *XV*: $\bigwedge i.\ i{\in}I \Longrightarrow openin\ (X\ i)\ (V\ i)$ **and** $z \in Pi_E\ I\ V$ **and** *subU*: $Pi_E\ I\ V \subseteq U$
by (*force simp*: *openin_product_topology_alt*)
show *?thesis*
proof (*intro conjI ballI*)
have *connected_space* $(X\ i)$ **if** $i \in I$ $V\ i = topspace\ (X\ i)$ **for** i
proof −
have *pc*: *connectedin* $(X\ i)$ $((\lambda x.\ x\ i)\ `\ C)$
by (*metis V connectedin_continuous_map_image continuous_map_product_projection that*(*1*))
moreover have $((\lambda x.\ x\ i)\ `\ C) = topspace\ (X\ i)$
proof
show $(\lambda x.\ x\ i)\ `\ C \subseteq topspace\ (X\ i)$
by (*simp add*: *pc connectedin_subset_topspace*)
have $V\ i \subseteq (\lambda x.\ x\ i)\ `\ (\Pi_E\ i{\in}I.\ V\ i)$
by (*metis ‹z ∈ Pi_E I V› empty_iff image_projection_PiE order_refl that*(*1*))
also have $\ldots \subseteq (\lambda x.\ x\ i)\ `\ U$
using *subU* **by** *blast*
finally show $topspace\ (X\ i) \subseteq (\lambda x.\ x\ i)\ `\ C$
using $‹U \subseteq C›$ *that* **by** *blast*
qed
ultimately show *?thesis*
by (*simp add*: *connectedin_topspace*)
qed
then have $\{i \in I.\ \neg\ connected\_space\ (X\ i)\} \subseteq \{i \in I.\ V\ i \neq topspace\ (X\ i)\}$
by *blast*
with *finV* **show** *finite* $\{i \in I.\ \neg\ connected\_space\ (X\ i)\}$
using *finite_subset* **by** *blast*
next
show *locally_connected_space* $(X\ i)$ **if** $i \in I$ **for** i
by (*meson False L locally_connected_space_quotient_map_image quotient_map_product_projection that*)
qed
qed
moreover have *?lhs* **if** *R*: *?rhs*
proof (*clarsimp simp add*: *locally_connected_space_def neighbourhood_base_of*)
fix F z
assume *openin* (*product_topology X I*) F **and** $z \in F$
then obtain W **where** *finW*: *finite* $\{i \in I.\ W\ i \neq topspace\ (X\ i)\}$
and *opeW*: $\bigwedge i.\ i \in I \Longrightarrow openin\ (X\ i)\ (W\ i)$ **and** $z \in Pi_E\ I\ W\ Pi_E\ I\ W \subseteq F$
by (*auto simp*: *openin_product_topology_alt*)

have $\forall i \in I.\ \exists U\ C.\ openin\ (X\ i)\ U \wedge connectedin\ (X\ i)\ C \wedge z\ i \in U \wedge U \subseteq C \wedge C \subseteq W\ i \wedge$
$(W\ i = topspace\ (X\ i) \wedge connected\_space\ (X\ i) \longrightarrow U = topspace\ (X\ i) \wedge C = topspace\ (X\ i))$
(**is** $\forall i \in I.\ ?\Phi\ i$)
proof
fix i **assume** $i \in I$
have *locally_connected_space* $(X\ i)$
by (*simp add*: $R\ \langle i \in I\rangle$)
moreover have $*$: *openin* $(X\ i)\ (W\ i)\ z\ i \in W\ i$
using $\langle z \in Pi_E\ I\ W\rangle$ *opeW* $\langle i \in I\rangle$ **by** *auto*
ultimately obtain $U\ C$ **where** *openin* $(X\ i)\ U$ *connectedin* $(X\ i)\ C\ z\ i \in U\ U \subseteq C\ C \subseteq W\ i$
using $\langle i \in I\rangle$ **by** (*force simp*: *locally_connected_space_def neighbourhood_base_of*)
then show $?\Phi\ i$
by (*metis* $*$ *connectedin_topspace openin_subset*)
qed
then obtain $U\ C$ **where**
$*$: $\bigwedge i.\ i \in I \Longrightarrow openin\ (X\ i)\ (U\ i) \wedge connectedin\ (X\ i)\ (C\ i) \wedge z\ i \in (U\ i) \wedge (U\ i) \subseteq (C\ i) \wedge (C\ i) \subseteq W\ i \wedge$
$(W\ i = topspace\ (X\ i) \wedge connected\_space\ (X\ i) \longrightarrow (U\ i) = topspace\ (X\ i) \wedge (C\ i) = topspace\ (X\ i))$
by *metis*
let $?A = \{i \in I.\ \neg\ connected\_space\ (X\ i)\} \cup \{i \in I.\ W\ i \neq topspace\ (X\ i)\}$
have $\{i \in I.\ U\ i \neq topspace\ (X\ i)\} \subseteq ?A$
by (*clarsimp simp add*: $*$)
moreover have *finite* $?A$
by (*simp add*: *that finW*)
ultimately have *finite* $\{i \in I.\ U\ i \neq topspace\ (X\ i)\}$
using *finite_subset* **by** *auto*
then have *openin* $(product\_topology\ X\ I)\ (Pi_E\ I\ U)$
using $*$ **by** (*simp add*: *openin_PiE_gen*)
then show $\exists U.\ openin\ (product\_topology\ X\ I)\ U \wedge$
$(\exists V.\ connectedin\ (product\_topology\ X\ I)\ V \wedge z \in U \wedge U \subseteq V \wedge V \subseteq F)$
using $\langle z \in Pi_E\ I\ W\rangle$ $\langle Pi_E\ I\ W \subseteq F\rangle$ $*$
by (*metis* (*no_types*, *opaque_lifting*) *PiE_iff PiE_mono connectedin_PiE subset_iff*)
qed
ultimately show *?thesis*
using *False* **by** *blast*
qed

lemma *locally_connected_space_sum_topology*:
locally_connected_space$(sum\_topology\ X\ I) \longleftrightarrow$
$(\forall i \in I.\ locally\_connected\_space\ (X\ i))$ (**is** *?lhs*=*?rhs*)
proof

assume *?lhs* **then show** *?rhs*
by (*smt* (*verit*) *homeomorphic_locally_connected_space locally_connected_space_open_subset topological_property_of_sum_component*)
next
assume *R*: *?rhs*
show *?lhs*
proof (*clarsimp simp add*: *locally_connected_space_def neighbourhood_base_of forall_openin_sum_topology imp_conjL*)
fix *W i x*
assume *ope*: ∀ *i*∈*I*. *openin* (*X i*) (*W i*)
and *i* ∈ *I* **and** *x* ∈ *W i*
then obtain *U V* **where** *U*: *openin* (*X i*) *U* **and** *V*: *connectedin* (*X i*) *V*
and *x* ∈ *U U* ⊆ *V V* ⊆ *W i*
by (*metis R* ‹*i* ∈ *I*› ‹*x* ∈ *W i*› *locally_connected_space*)
show ∃ *U*. *openin* (*sum_topology X I*) *U* ∧ (∃ *V*. *connectedin* (*sum_topology X I*) *V* ∧ (*i*,*x*) ∈ *U* ∧ *U* ⊆ *V* ∧ *V* ⊆ *Sigma I W*)
proof (*intro exI conjI*)
show *openin* (*sum_topology X I*) (*Pair i* ‘ *U*)
by (*meson U* ‹*i* ∈ *I*› *open_map_component_injection open_map_def*)
show *connectedin* (*sum_topology X I*) (*Pair i* ‘ *V*)
by (*meson V* ‹*i* ∈ *I*› *continuous_map_component_injection connectedin_continuous_map_image*)
show *Pair i* ‘ *V* ⊆ *Sigma I W*
using ‹*V* ⊆ *W i*› ‹*i* ∈ *I*› **by** *force*
qed (*use* ‹*x* ∈ *U*› ‹*U* ⊆ *V*› **in** *auto*)
qed
qed

5.2.4 Dimension of a topological space

Basic definition of the small inductive dimension relation. Works in any topological space.

inductive *dimension_le* :: [*'a topology*, *int*] ⇒ *bool* (**infix** *dim'_le 50*)
where ⟦−*1* ≤ *n*;
⋀*V a*. ⟦*openin X V*; *a* ∈ *V*⟧ ⟹ ∃ *U*. *a* ∈ *U* ∧ *U* ⊆ *V* ∧ *openin X U* ∧ (*subtopology X* (*X frontier_of U*)) *dim_le* (*n*−*1*)⟧
⟹ *X dim_le* (*n*::*int*)

lemma *dimension_le_neighbourhood_base*:
X dim_le n ⟷
−*1* ≤ *n* ∧ *neighbourhood_base_of* (λ*U*. *openin X U* ∧ (*subtopology X* (*X frontier_of U*)) *dim_le* (*n*−*1*)) *X*
by (*smt* (*verit*, *best*) *dimension_le.simps open_neighbourhood_base_of*)

lemma *dimension_le_bound*: *X dim_le n* ⟹−*1* ≤ *n*
using *dimension_le.simps* **by** *blast*

lemma *dimension_le_mono* [*rule_format*]:
assumes *X dim_le m*

```
  shows m ≤ n ⟶ X dim_le n
  using assms
proof (induction arbitrary: n rule: dimension_le.induct)
qed (smt (verit) dimension_le.simps)

inductive_simps dim_le_minus2 [simp]: X dim_le −2

lemma dimension_le_eq_empty [simp]:
   X dim_le −1 ⟷ X = trivial_topology
proof
  show X dim_le (−1) ⟹ X = trivial_topology
    by (force intro: dimension_le.cases)
next
  show X = trivial_topology ⟹ X dim_le (−1)
    using dimension_le.simps openin_subset by fastforce
qed

lemma dimension_le_0_neighbourhood_base_of_clopen:
  X dim_le 0 ⟷ neighbourhood_base_of (λU. closedin X U ∧ openin X U) X
proof −
  have (subtopology X (X frontier_of U) dim_le −1) = closedin X U
      if openin X U for U
    using that clopenin_eq_frontier_of openin_subset
     by (fastforce simp add: subtopology_trivial_iff frontier_of_subset_topspace
Int_absorb1)
  then show ?thesis
    by (smt (verit, del_insts) dimension_le.simps open_neighbourhood_base_of)
qed

lemma dimension_le_subtopology:
  X dim_le n ⟹ subtopology X S dim_le n
proof (induction arbitrary: S rule: dimension_le.induct)
  case (1 n X)
  show ?case
  proof (intro dimension_le.intros)
    show − 1 ≤ n
      by (simp add: 1.hyps)
    fix U' a
    assume U': openin (subtopology X S) U' and a ∈ U'
    then obtain U where U: openin X U U' = U ∩ S
      by (meson openin_subtopology)
    then obtain V where a ∈ V V ⊆ U openin X V
      and subV: subtopology X (X frontier_of V) dim_le n−1
      and dimV: ⋀T. subtopology X (X frontier_of V ∩ T) dim_le n−1
      by (metis 1.IH Int_iff ‹a ∈ U'› subtopology_subtopology)
    show ∃ W. a ∈ W ∧ W ⊆ U' ∧ openin (subtopology X S) W ∧ subtopology
(subtopology X S) (subtopology X S frontier_of W) dim_le n−1
    proof (intro exI conjI)
      show a ∈ S ∩ V S ∩ V ⊆ U'
```

using ‹$U' = U \cap S$› ‹$a \in U'$› ‹$a \in V$› ‹$V \subseteq U$› **by** *blast+*
show *openin* (*subtopology X S*) ($S \cap V$)
by (*simp add*: ‹*openin X V*› *openin_subtopology_Int2*)
have $S \cap$ *subtopology X S frontier_of V* $\subseteq$ *X frontier_of V*
by (*simp add*: *frontier_of_subtopology_subset*)
then show *subtopology* (*subtopology X S*) (*subtopology X S frontier_of* ($S \cap$ *V*)) *dim_le n−1*
by (*metis dimV frontier_of_restrict inf.absorb_iff2 inf_left_idem subtopology_subtopology topspace_subtopology*)
qed
qed
qed

lemma *dimension_le_subtopologies*:
⟦*subtopology X T dim_le n*; $S \subseteq T$⟧ $\Longrightarrow$ (*subtopology X S*) *dim_le n*
by (*metis dimension_le_subtopology inf.absorb_iff2 subtopology_subtopology*)

lemma *dimension_le_eq_subtopology*:
(*subtopology X S*) *dim_le n* $\longleftrightarrow$
$-1 \le n \wedge$
($\forall$ *V a. openin X V* $\wedge$ $a \in V \wedge a \in S$
$\longrightarrow$ ($\exists$ *U*. $a \in U \wedge U \subseteq V \wedge$ *openin X U* $\wedge$
subtopology X (*subtopology X S frontier_of* ($S \cap U$)) *dim_le* (*n−1*)))
proof −
have ∗: ($\exists$ *T*. $a \in T \wedge T \cap S \subseteq V \cap S \wedge$ *openin X T* $\wedge$ *subtopology X* ($S \cap$ (*subtopology X S frontier_of* ($T \cap S$))) *dim_le n−1*)
$\longleftrightarrow$ ($\exists$ *U*. $a \in U \wedge U \subseteq V \wedge$ *openin X U* $\wedge$ *subtopology X* (*subtopology X S frontier_of* ($S \cap U$)) *dim_le n−1*)
if $a \in V$ $a \in S$ *openin X V* **for** *a V*
proof −
have $\exists$ *U*. $a \in U \wedge U \subseteq V \wedge$ *openin X U* $\wedge$ *subtopology X* (*subtopology X S frontier_of* ($S \cap U$)) *dim_le n−1*
if $a \in T$ **and** *sub*: $T \cap S \subseteq V \cap S$ **and** *openin X T*
and *dim*: *subtopology X* ($S \cap$ *subtopology X S frontier_of* ($T \cap S$)) *dim_le n−1*
for *T*
proof (*intro exI conjI*)
show *openin X* ($T \cap V$)
using ‹*openin X V*› ‹*openin X T*› **by** *blast*
show *subtopology X* (*subtopology X S frontier_of* ($S \cap$ ($T \cap V$))) *dim_le n−1*
by (*metis dim frontier_of_subset_subtopology inf.boundedE inf_absorb2 inf_assoc inf_commute sub*)
qed (*use* ‹$a \in V$› ‹$a \in T$› **in** *auto*)
moreover have $\exists$ *T*. $a \in T \wedge T \cap S \subseteq V \cap S \wedge$ *openin X T* $\wedge$ *subtopology X* ($S \cap$ *subtopology X S frontier_of* ($T \cap S$)) *dim_le n−1*
if $a \in U$ **and** $U \subseteq V$ **and** *openin X U*
and *dim*: *subtopology X* (*subtopology X S frontier_of* ($S \cap U$)) *dim_le n−1*

```
    for U
      by (metis that frontier_of_subset_subtopology inf_absorb2 inf_commute
inf_le1 le_inf_iff)
   ultimately show ?thesis
     by safe
 qed
 show ?thesis
  apply (simp add: dimension_le.simps [of _ n] subtopology_subtopology openin_subtopology
flip: ∗)
   by (safe; metis Int_iff inf_le2 le_inf_iff)
qed


lemma homeomorphic_space_dimension_le_aux:
 assumes X homeomorphic_space Y X dim_le of_nat n − 1
 shows Y dim_le of_nat n − 1
 using assms
proof (induction n arbitrary: X Y)
 case 0
 then show ?case
   by (simp add: dimension_le_eq_empty homeomorphic_empty_space)
next
 case (Suc n)
 then have X_dim_n: X dim_le n
   by simp
 show ?case
 proof (clarsimp simp add: dimension_le.simps [of Y n])
   fix V b
   assume openin Y V and b ∈ V
   obtain f g where fg: homeomorphic_maps X Y f g
     using ‹X homeomorphic_space Y› homeomorphic_space_def by blast
   then have openin X (g ‘ V)
   using ‹openin Y V› homeomorphic_map_openness_eq homeomorphic_maps_map
by blast
   then obtain U where g b ∈ U openin X U and gim: U ⊆ g ‘ V and sub:
subtopology X (X frontier_of U) dim_le int n − int 1
     using X_dim_n unfolding dimension_le.simps [of X n] by (metis ‹b ∈ V›
imageI of_nat_eq_1_iff)
   show ∃ U. b ∈ U ∧ U ⊆ V ∧ openin Y U ∧ subtopology Y (Y frontier_of U)
dim_le int n − 1
   proof (intro conjI exI)
     show b ∈ f ‘ U
       by (metis (no_types, lifting) ‹b ∈ V› ‹g b ∈ U› ‹openin Y V› fg homeo-
morphic_maps_map image_iff openin_subset subsetD)
     show f ‘ U ⊆ V
     by (smt (verit, ccfv_threshold) ‹openin Y V› fg gim homeomorphic_maps_map
image_iff openin_subset subset_iff)
     show openin Y (f ‘ U)
         using ‹openin X U› fg homeomorphic_map_openness_eq homeomor-
```

phic_maps_map **by** *blast*
show *subtopology Y (Y frontier_of f ‘ U) dim_le int n−1*
proof (*rule Suc.IH*)
have *homeomorphic_maps (subtopology X (X frontier_of U)) (subtopology Y (Y frontier_of f ‘ U)) f g*
using ‹*openin X U*› *fg*
by (*metis frontier_of_subset_topspace homeomorphic_map_frontier_of homeomorphic_maps_map homeomorphic_maps_subtopologies openin_subset topspace_subtopology topspace_subtopology_subset*)
then show *subtopology X (X frontier_of U) homeomorphic_space subtopology Y (Y frontier_of f ‘ U)*
using *homeomorphic_space_def* **by** *blast*
show *subtopology X (X frontier_of U) dim_le int n−1*
using *sub* **by** *fastforce*
qed
qed
qed
qed

lemma *homeomorphic_space_dimension_le*:
assumes *X homeomorphic_space Y*
shows *X dim_le n ⟷ Y dim_le n*
proof (*cases n ≥ −1*)
case *True*
then show *?thesis*
using *homeomorphic_space_dimension_le_aux* [*of _ _ nat(n+1)*]
by (*smt* (*verit*) *assms homeomorphic_space_sym nat_eq_iff*)
next
case *False*
then show *?thesis*
by (*metis dimension_le_bound*)
qed

lemma *dimension_le_retraction_map_image*:
⟦*retraction_map X Y r*; *X dim_le n*⟧ ⟹ *Y dim_le n*
by (*meson dimension_le_subtopology homeomorphic_space_dimension_le retraction_map_def retraction_maps_section_image2*)

lemma *dimension_le_discrete_topology* [*simp*]: (*discrete_topology U*) *dim_le 0*
using *dimension_le.simps dimension_le_eq_empty* **by** *fastforce*

end

5.3 Some Uncountable Sets

theory *Uncountable_Sets*
imports *Path_Connected Continuum_Not_Denumerable*
begin

lemma *uncountable_closed_segment*:
fixes *a* :: *'a*::*real_normed_vector*
assumes *a* ≠ *b* **shows** *uncountable* (*closed_segment a b*)
unfolding *path_image_linepath* [*symmetric*] *path_image_def*
using *inj_on_linepath* [*OF assms*] *uncountable_closed_interval* [*of 0 1*]
countable_image_inj_on **by** *auto*

lemma *uncountable_open_segment*:
fixes *a* :: *'a*::*real_normed_vector*
assumes *a* ≠ *b* **shows** *uncountable* (*open_segment a b*)
by (*simp add*: *assms open_segment_def uncountable_closed_segment uncountable_minus_countable*)

lemma *uncountable_convex*:
fixes *a* :: *'a*::*real_normed_vector*
assumes *convex S a* ∈ *S b* ∈ *S a* ≠ *b*
shows *uncountable S*
proof −
have *uncountable* (*closed_segment a b*)
by (*simp add*: *uncountable_closed_segment assms*)
then show *?thesis*
by (*meson assms convex_contains_segment countable_subset*)
qed

lemma *uncountable_ball*:
fixes *a* :: *'a*::*euclidean_space*
assumes *r* > *0*
shows *uncountable* (*ball a r*)
proof −
have *uncountable* (*open_segment a* (*a* + *r* ∗*R* (*SOME i*. *i* ∈ *Basis*)))
by (*metis Basis_zero SOME_Basis add_cancel_right_right assms less_le scale_eq_0_iff uncountable_open_segment*)
moreover have *open_segment a* (*a* + *r* ∗*R* (*SOME i*. *i* ∈ *Basis*)) ⊆ *ball a r*
using *assms* **by** (*auto simp*: *in_segment algebra_simps dist_norm SOME_Basis*)
ultimately show *?thesis*
by (*metis countable_subset*)
qed

lemma *ball_minus_countable_nonempty*:
assumes *countable* (*A* :: *'a* :: *euclidean_space set*) *r* > *0*
shows *ball z r* − *A* ≠ {}
proof
assume ∗: *ball z r* − *A* = {}
have *uncountable* (*ball z r* − *A*)
by (*intro uncountable_minus_countable assms uncountable_ball*)
thus *False* **by** (*subst* (*asm*) ∗) *auto*
qed

lemma *uncountable_cball*:

fixes $a :: 'a$::*euclidean_space*
assumes $r > 0$
shows *uncountable* (*cball a r*)
using *assms countable_subset uncountable_ball* **by** *auto*

lemma *pairwise_disjnt_countable*:
fixes $\mathcal{N} ::$ *nat set set*
assumes *pairwise disjnt* $\mathcal{N}$
shows *countable* $\mathcal{N}$
by (*simp add*: *assms countable_disjoint_open_subsets open_discrete*)

lemma *pairwise_disjnt_countable_Union*:
assumes *countable* ($\bigcup\mathcal{N}$) **and** *pwd*: *pairwise disjnt* $\mathcal{N}$
shows *countable* $\mathcal{N}$
proof −
obtain $f :: \_ \Rightarrow$ *nat* **where** *f*: *inj_on f* ($\bigcup\mathcal{N}$)
using *assms* **by** *blast*
then have *pairwise disjnt* ($\bigcup\ X \in \mathcal{N}.\ \{f\ `\ X\}$)
using *assms* **by** (*force simp*: *pairwise_def disjnt_inj_on_iff* [*OF f*])
then have *countable* ($\bigcup\ X \in \mathcal{N}.\ \{f\ `\ X\}$)
using *pairwise_disjnt_countable* **by** *blast*
then show *?thesis*
by (*meson pwd countable_image_inj_on disjoint_image f inj_on_image pairwise_disjnt_countable*)
qed

lemma *connected_uncountable*:
fixes $S :: 'a$::*metric_space set*
assumes *connected S a* $\in$ *S b* $\in$ *S a* $\neq$ *b* **shows** *uncountable S*
proof −
have *continuous_on S* (*dist a*)
by (*intro continuous_intros*)
then have *connected* (*dist a* ` *S*)
by (*metis connected_continuous_image* ‹*connected S*›)
then have *closed_segment 0* (*dist a b*) $\subseteq$ (*dist a* ` *S*)
by (*simp add*: *assms closed_segment_subset is_interval_connected_1 is_interval_convex*)
then have *uncountable* (*dist a* ` *S*)
by (*metis* ‹*a* $\neq$ *b*› *countable_subset dist_eq_0_iff uncountable_closed_segment*)
then show *?thesis*
by *blast*
qed

lemma *path_connected_uncountable*:
fixes $S :: 'a$::*metric_space set*
assumes *path_connected S a* $\in$ *S b* $\in$ *S a* $\neq$ *b* **shows** *uncountable S*
using *path_connected_imp_connected assms connected_uncountable* **by** *metis*

lemma *simple_path_image_uncountable*:
fixes $g ::$ *real* $\Rightarrow$ $'a$::*metric_space*

assumes *simple_path g*
shows *uncountable* (*path_image g*)
proof −
have *g 0* ∈ *path_image g g* (*1/2*) ∈ *path_image g*
by (*simp_all add*: *path_defs*)
moreover have *g 0* ≠ *g* (*1/2*)
using *assms* **by** (*fastforce simp add*: *simple_path_def loop_free_def*)
ultimately have ∀ *a*. ¬ *path_image g* ⊆ {*a*}
by *blast*
then show *?thesis*
using *assms connected_simple_path_image connected_uncountable* **by** *blast*
qed

lemma *arc_image_uncountable*:
fixes *g* :: *real* ⇒ *'a*::*metric_space*
assumes *arc g*
shows *uncountable* (*path_image g*)
by (*simp add*: *arc_imp_simple_path assms simple_path_image_uncountable*)

end

5.4 Homotopy of Maps

theory *Homotopy*
imports *Path_Connected Product_Topology Uncountable_Sets*
begin

definition *homotopic_with*
where
homotopic_with P X Y f g ≡
(∃ *h*. *continuous_map* (*prod_topology* (*top_of_set* {*0..1*::*real*}) *X*) *Y h* ∧
(∀ *x*. *h*(*0*, *x*) = *f x*) ∧
(∀ *x*. *h*(*1*, *x*) = *g x*) ∧
(∀ *t* ∈ {*0..1*}. *P*(λ*x*. *h*(*t*,*x*))))

p, *q* are functions $X \to Y$, and the property P restricts all intermediate maps. We often just want to require that P fixes some subset, but to include the case of a loop homotopy, it is convenient to have a general property P.

abbreviation *homotopic_with_canon* ::
[(*'a*::*topological_space* ⇒ *'b*::*topological_space*) ⇒ *bool*, *'a set*, *'b set*, *'a* ⇒ *'b*, *'a* ⇒ *'b*] ⇒ *bool*
where
homotopic_with_canon P S T p q ≡ *homotopic_with P* (*top_of_set S*) (*top_of_set T*) *p q*

lemma *split_01*: {*0..1*::*real*} = {*0..1/2*} ∪ {*1/2..1*}
by *force*

lemma *split_01_prod*: $\{0..1::real\} \times X = (\{0..1/2\} \times X) \cup (\{1/2..1\} \times X)$
by *force*

lemma *image_Pair_const*: $(\lambda x.\ (x,\ c))\ `\ A = A \times \{c\}$
by *auto*

lemma *fst_o_paired* [*simp*]: $fst \circ (\lambda(x,y).\ (f\ x\ y,\ g\ x\ y)) = (\lambda(x,y).\ f\ x\ y)$
by *auto*

lemma *snd_o_paired* [*simp*]: $snd \circ (\lambda(x,y).\ (f\ x\ y,\ g\ x\ y)) = (\lambda(x,y).\ g\ x\ y)$
by *auto*

lemma *continuous_on_o_Pair*: $[\![continuous\_on\ (T \times X)\ h;\ t \in T]\!] \Longrightarrow continuous\_on\ X\ (h \circ Pair\ t)$
by (*fast intro*: *continuous_intros elim*!: *continuous_on_subset*)

lemma *continuous_map_o_Pair*:
assumes *h*: *continuous_map* (*prod_topology X Y*) *Z h* **and** *t*: $t \in topspace\ X$
shows *continuous_map Y Z* $(h \circ Pair\ t)$
by (*intro continuous_map_compose* [*OF _ h*] *continuous_intros*; *simp add*: *t*)

5.4.1 Trivial properties

We often want to just localize the ending function equality or whatever.

proposition *homotopic_with*:
assumes $\bigwedge h\ k.\ (\bigwedge x.\ x \in topspace\ X \Longrightarrow h\ x = k\ x) \Longrightarrow (P\ h \longleftrightarrow P\ k)$
shows *homotopic_with P X Y p q* $\longleftrightarrow$
$(\exists h.\ continuous\_map\ (prod\_topology\ (subtopology\ euclideanreal\ \{0..1\})\ X)\ Y\ h \wedge$
$(\forall x \in topspace\ X.\ h(0,x) = p\ x) \wedge$
$(\forall x \in topspace\ X.\ h(1,x) = q\ x) \wedge$
$(\forall t \in \{0..1\}.\ P(\lambda x.\ h(t,\ x))))$
unfolding *homotopic_with_def*
apply (*rule iffI*, *blast*, *clarify*)
apply (*rule_tac* $x=\lambda(u,v).\ if\ v \in topspace\ X\ then\ h(u,v)\ else\ if\ u = 0\ then\ p\ v\ else\ q\ v$ **in** *exI*)
apply *simp*
by (*smt* (*verit*, *best*) *SigmaE assms case_prod_conv continuous_map_eq topspace_prod_topology*)

lemma *homotopic_with_mono*:
assumes *hom*: *homotopic_with P X Y f g*
and *Q*: $\bigwedge h.\ [\![continuous\_map\ X\ Y\ h;\ P\ h]\!] \Longrightarrow Q\ h$
shows *homotopic_with Q X Y f g*
using *hom* **unfolding** *homotopic_with_def*
by (*force simp*: *o_def dest*: *continuous_map_o_Pair intro*: *Q*)

lemma *homotopic_with_imp_continuous_maps*:
assumes *homotopic_with P X Y f g*

shows *continuous_map X Y f* ∧ *continuous_map X Y g*
proof −
obtain *h* :: *real* × *'a* ⇒ *'b*
where *conth*: *continuous_map* (*prod_topology* (*top_of_set* {*0..1*}) *X*) *Y h*
and *h*: ∀ *x*. *h* (*0*, *x*) = *f x* ∀ *x*. *h* (*1*, *x*) = *g x*
using *assms* **by** (*auto simp*: *homotopic_with_def*)
have ∗: *t* ∈ {*0..1*} ⟹ *continuous_map X Y* (*h* ∘ (λ*x*. (*t*,*x*))) **for** *t*
by (*rule continuous_map_compose* [*OF* _ *conth*]) (*simp add*: *o_def continuous_map_pairwise*)
show *?thesis*
using *h* ∗[*of 0*] ∗[*of 1*] **by** (*simp add*: *continuous_map_eq*)
qed

lemma *homotopic_with_imp_continuous*:
assumes *homotopic_with_canon P X Y f g*
shows *continuous_on X f* ∧ *continuous_on X g*
by (*meson assms continuous_map_subtopology_eu homotopic_with_imp_continuous_maps*)

lemma *homotopic_with_imp_property*:
assumes *homotopic_with P X Y f g*
shows *P f* ∧ *P g*
proof
obtain *h* **where** *h*: ⋀*x*. *h*(*0*, *x*) = *f x* ⋀*x*. *h*(*1*, *x*) = *g x* **and** *P*: ⋀*t*. *t* ∈ {*0..1*::*real*} ⟹ *P*(λ*x*. *h*(*t*,*x*))
using *assms* **by** (*force simp*: *homotopic_with_def*)
show *P f P g*
using *P* [*of 0*] *P* [*of 1*] **by** (*force simp*: *h*)+
qed

lemma *homotopic_with_equal*:
assumes *P f P g* **and** *contf*: *continuous_map X Y f* **and** *fg*: ⋀*x*. *x* ∈ *topspace X* ⟹ *f x* = *g x*
shows *homotopic_with P X Y f g*
unfolding *homotopic_with_def*
proof (*intro exI conjI allI ballI*)
let *?h* = λ(*t*::*real*,*x*). *if t* = *1 then g x else f x*
show *continuous_map* (*prod_topology* (*top_of_set* {*0..1*}) *X*) *Y ?h*
proof (*rule continuous_map_eq*)
show *continuous_map* (*prod_topology* (*top_of_set* {*0..1*}) *X*) *Y* (*f* ∘ *snd*)
by (*simp add*: *contf continuous_map_of_snd*)
qed (*auto simp*: *fg*)
show *P* (λ*x*. *?h* (*t*, *x*)) **if** *t* ∈ {*0..1*} **for** *t*
by (*cases t* = *1*) (*simp_all add*: *assms*)
qed *auto*

lemma *homotopic_with_imp_subset1*:
homotopic_with_canon P X Y f g ⟹ *f* ‘ *X* ⊆ *Y*
by (*meson continuous_map_subtopology_eu homotopic_with_imp_continuous_maps*)

lemma *homotopic_with_imp_subset2*:
  homotopic_with_canon P X Y f g $\Longrightarrow$ *g ' X* $\subseteq$ *Y*
 by (*meson continuous_map_subtopology_eu homotopic_with_imp_continuous_maps*)

lemma *homotopic_with_imp_funspace1*:
  homotopic_with_canon P X Y f g $\Longrightarrow$ *f* $\in$ *X* $\to$ *Y*
 using *homotopic_with_imp_subset1* **by** *blast*

lemma *homotopic_with_imp_funspace2*:
  homotopic_with_canon P X Y f g $\Longrightarrow$ *g* $\in$ *X* $\to$ *Y*
 using *homotopic_with_imp_subset2* **by** *blast*

lemma *homotopic_with_subset_left*:
  $[\![$*homotopic_with_canon P X Y f g*; *Z* $\subseteq$ *X*$]\!]$ $\Longrightarrow$ *homotopic_with_canon P Z Y f g*
 unfolding *homotopic_with_def* **by** (*auto elim*!: *continuous_on_subset ex_forward*)

lemma *homotopic_with_subset_right*:
  $[\![$*homotopic_with_canon P X Y f g*; *Y* $\subseteq$ *Z*$]\!]$ $\Longrightarrow$ *homotopic_with_canon P X Z f g*
 unfolding *homotopic_with_def* **by** (*auto elim*!: *continuous_on_subset ex_forward*)

5.4.2 Homotopy with P is an equivalence relation

(on continuous functions mapping X into Y that satisfy P, though this only affects reflexivity)

lemma *homotopic_with_refl* [*simp*]: *homotopic_with P X Y f f* $\longleftrightarrow$ *continuous_map X Y f* $\wedge$ *P f*
 by (*metis homotopic_with_equal homotopic_with_imp_continuous_maps homotopic_with_imp_property*)

lemma *homotopic_with_symD*:
  assumes *homotopic_with P X Y f g*
  shows *homotopic_with P X Y g f*
proof −
 let *?I01* = *subtopology euclideanreal* {*0..1*}
 let *?j* = λ*y*. (*1* − *fst y*, *snd y*)
 have *1*: *continuous_map* (*prod_topology ?I01 X*) (*prod_topology euclideanreal X*) *?j*
  by (*intro continuous_intros*; *simp add*: *continuous_map_subtopology_fst prod_topology_subtopology*)
 have ∗: *continuous_map* (*prod_topology ?I01 X*) (*prod_topology ?I01 X*) *?j*
 proof −
  have *continuous_map* (*prod_topology ?I01 X*) (*subtopology* (*prod_topology euclideanreal X*) ({*0..1*} × *topspace X*)) *?j*
   by (*simp add*: *continuous_map_into_subtopology* [*OF 1*] *image_subset_iff flip*: *image_subset_iff_funcset*)
  then show *?thesis*
   by (*simp add*: *prod_topology_subtopology*(*1*))
 qed

```
  show ?thesis
    using assms
    apply (clarsimp simp: homotopic_with_def)
    subgoal for h
      by (rule_tac x=h ∘ (λy. (1 − fst y, snd y)) in exI) (simp add: continuous_map_compose [OF *])
    done
qed

lemma homotopic_with_sym:
  homotopic_with P X Y f g ⟷ homotopic_with P X Y g f
  by (metis homotopic_with_symD)

proposition homotopic_with_trans:
    assumes homotopic_with P X Y f g  homotopic_with P X Y g h
    shows homotopic_with P X Y f h
proof −
  let ?X01 = prod_topology (subtopology euclideanreal {0..1}) X
  obtain k1 k2
    where contk1: continuous_map ?X01 Y k1 and contk2: continuous_map ?X01 Y k2
      and k12: ∀x. k1 (1, x) = g x ∀x. k2 (0, x) = g x
      ∀x. k1 (0, x) = f x ∀x. k2 (1, x) = h x
      and P:  ∀t∈{0..1}. P (λx. k1 (t, x)) ∀t∈{0..1}. P (λx. k2 (t, x))
    using assms by (auto simp: homotopic_with_def)
  define k where k ≡ λy. if fst y ≤ 1/2
                         then (k1 ∘ (λx. (2 *R fst x, snd x))) y
                         else (k2 ∘ (λx. (2 *R fst x −1, snd x))) y
  have keq: k1 (2 ∗ u, v) = k2 (2 ∗ u −1, v) if u = 1/2  for u v
    by (simp add: k12 that)
  show ?thesis
    unfolding homotopic_with_def
  proof (intro exI conjI)
    show continuous_map ?X01 Y k
      unfolding k_def
    proof (rule continuous_map_cases_le)
      show fst: continuous_map ?X01 euclideanreal fst
        using continuous_map_fst continuous_map_in_subtopology by blast
      show continuous_map ?X01 euclideanreal (λx. 1/2)
        by simp
      show continuous_map (subtopology ?X01 {y ∈ topspace ?X01. fst y ≤ 1/2}) Y
           (k1 ∘ (λx. (2 *R fst x, snd x)))
       apply (intro fst continuous_map_compose [OF _ contk1] continuous_intros continuous_map_into_subtopology continuous_map_from_subtopology | simp)+
        by (force simp: prod_topology_subtopology)
      show continuous_map (subtopology ?X01 {y ∈ topspace ?X01. 1/2 ≤ fst y}) Y
           (k2 ∘ (λx. (2 *R fst x −1, snd x)))
```

apply (*intro fst continuous_map_compose* [*OF* _ *contk2*] *continuous_intros continuous_map_into_subtopology continuous_map_from_subtopology* | *simp*)+
by (*force simp*: *prod_topology_subtopology*)
show (*k1* ∘ (λ*x*. (*2* ∗$_R$ *fst x*, *snd x*))) *y* = (*k2* ∘ (λ*x*. (*2* ∗$_R$ *fst x* −*1*, *snd x*))) *y*
if *y* ∈ *topspace ?X01* **and** *fst y* = *1/2* **for** *y*
using *that* **by** (*simp add*: *keq*)
qed
show ∀ *x*. *k* (*0*, *x*) = *f x*
by (*simp add*: *k12 k_def*)
show ∀ *x*. *k* (*1*, *x*) = *h x*
by (*simp add*: *k12 k_def*)
show ∀ *t*∈{*0..1*}. *P* (λ*x*. *k* (*t*, *x*))
proof
fix *t* **show** *t*∈{*0..1*} ⟹ *P* (λ*x*. *k* (*t*, *x*))
by (*cases t* ≤ *1/2*) (*auto simp*: *k_def P*)
qed
qed
qed

lemma *homotopic_with_id2*:
(⋀*x*. *x* ∈ *topspace X* ⟹ *g* (*f x*) = *x*) ⟹ *homotopic_with* (λ*x*. *True*) *X X* (*g* ∘ *f*) *id*
by (*metis comp_apply continuous_map_id eq_id_iff homotopic_with_equal homotopic_with_symD*)

5.4.3 Continuity lemmas

lemma *homotopic_with_compose_continuous_map_left*:
⟦*homotopic_with p X1 X2 f g*; *continuous_map X2 X3 h*; ⋀*j*. *p j* ⟹ *q*(*h* ∘ *j*)⟧
⟹ *homotopic_with q X1 X3* (*h* ∘ *f*) (*h* ∘ *g*)
unfolding *homotopic_with_def*
apply *clarify*
subgoal for *k*
by (*rule_tac x*=*h* ∘ *k* **in** *exI*) (*rule conjI continuous_map_compose* | *simp add*: *o_def*)+
done

lemma *homotopic_with_compose_continuous_map_right*:
assumes *hom*: *homotopic_with p X2 X3 f g* **and** *conth*: *continuous_map X1 X2 h*
and *q*: ⋀*j*. *p j* ⟹ *q*(*j* ∘ *h*)
shows *homotopic_with q X1 X3* (*f* ∘ *h*) (*g* ∘ *h*)
proof −
obtain *k*
where *contk*: *continuous_map* (*prod_topology* (*subtopology euclideanreal* {*0..1*}) *X2*) *X3 k*
and *k*: ∀ *x*. *k* (*0*, *x*) = *f x* ∀ *x*. *k* (*1*, *x*) = *g x* **and** *p*: ⋀*t*. *t*∈{*0..1*} ⟹ *p* (λ*x*. *k* (*t*, *x*))

using *hom* **unfolding** *homotopic_with_def* **by** *blast*
have *hsnd*: *continuous_map* (*prod_topology* (*subtopology euclideanreal* {*0..1*}) *X1*) *X2* (*h* ∘ *snd*)
by (*rule continuous_map_compose* [*OF continuous_map_snd conth*])
let *?h* = *k* ∘ (λ(*t*,*x*). (*t*,*h x*))
show *?thesis*
unfolding *homotopic_with_def*
proof (*intro exI conjI allI ballI*)
have *continuous_map* (*prod_topology* (*top_of_set* {*0..1*}) *X1*) (*prod_topology* (*top_of_set* {*0..1*::*real*}) *X2*) (λ(*t*, *x*). (*t*, *h x*))
by (*metis* (*mono_tags*, *lifting*) *case_prod_beta'* *comp_def* *continuous_map_eq* *continuous_map_fst* *continuous_map_pairedI* *hsnd*)
then show *continuous_map* (*prod_topology* (*subtopology euclideanreal* {*0..1*}) *X1*) *X3* *?h*
by (*intro conjI continuous_map_compose* [*OF _ contk*])
show *q* (λ*x*. *?h* (*t*, *x*)) **if** *t* ∈ {*0..1*} **for** *t*
using *q* [*OF p* [*OF that*]] **by** (*simp add*: *o_def*)
qed (*auto simp*: *k*)
qed

corollary *homotopic_compose*:
assumes *homotopic_with* (λ*x*. *True*) *X Y f f'* *homotopic_with* (λ*x*. *True*) *Y Z g g'*
shows *homotopic_with* (λ*x*. *True*) *X Z* (*g* ∘ *f*) (*g'* ∘ *f'*)
by (*metis assms homotopic_with_compose_continuous_map_left homotopic_with_compose_continuous_map_ri*
homotopic_with_imp_continuous_maps homotopic_with_trans)

proposition *homotopic_with_compose_continuous_right*:
⟦*homotopic_with_canon* (λ*f*. *p* (*f* ∘ *h*)) *X Y f g*; *continuous_on W h*; *h* ∈ *W* → *X*⟧
⟹ *homotopic_with_canon p W Y* (*f* ∘ *h*) (*g* ∘ *h*)
by (*simp add*: *homotopic_with_compose_continuous_map_right image_subset_iff_funcset*)

proposition *homotopic_with_compose_continuous_left*:
⟦*homotopic_with_canon* (λ*f*. *p* (*h* ∘ *f*)) *X Y f g*; *continuous_on Y h*; *h* ∈ *Y* → *Z*⟧
⟹ *homotopic_with_canon p X Z* (*h* ∘ *f*) (*h* ∘ *g*)
by (*simp add*: *homotopic_with_compose_continuous_map_left image_subset_iff_funcset*)

lemma *homotopic_from_subtopology*:
homotopic_with P X X' f g ⟹ *homotopic_with P* (*subtopology X S*) *X' f g*
by (*metis continuous_map_id_subt homotopic_with_compose_continuous_map_right o_id*)

lemma *homotopic_on_emptyI*:
assumes *P f P g*
shows *homotopic_with P trivial_topology X f g*
by (*metis assms continuous_map_on_empty empty_iff homotopic_with_equal topspace_discrete_topology*)

lemma *homotopic_on_empty*:
(*homotopic_with P trivial_topology X f g* ⟷ *P f* ∧ *P g*)
using *homotopic_on_emptyI homotopic_with_imp_property* **by** *metis*

lemma *homotopic_with_canon_on_empty*: *homotopic_with_canon* (λ*x*. *True*) {} *t f g*
by (*auto intro*: *homotopic_with_equal*)

lemma *homotopic_constant_maps*:
homotopic_with (λ*x*. *True*) *X X'* (λ*x*. *a*) (λ*x*. *b*) ⟷
X = *trivial_topology* ∨ *path_component_of X' a b* (**is** *?lhs* = *?rhs*)
proof (*cases X* = *trivial_topology*)
case *False*
then obtain *c* **where** *c*: *c* ∈ *topspace X*
by *fastforce*
have ∃ *g*. *continuous_map* (*top_of_set* {*0..1*::*real*}) *X' g* ∧ *g 0* = *a* ∧ *g 1* = *b*
if *x* ∈ *topspace X* **and** *hom*: *homotopic_with* (λ*x*. *True*) *X X'* (λ*x*. *a*) (λ*x*. *b*)
for *x*
proof −
obtain *h* :: *real* × *'a* ⇒ *'b*
where *conth*: *continuous_map* (*prod_topology* (*top_of_set* {*0..1*}) *X*) *X' h*
and *h*: ⋀*x*. *h* (*0*, *x*) = *a* ⋀*x*. *h* (*1*, *x*) = *b*
using *hom* **by** (*auto simp*: *homotopic_with_def*)
have *cont*: *continuous_map* (*top_of_set* {*0..1*}) *X'* (*h* ∘ (λ*t*. (*t*, *c*)))
by (*rule continuous_map_compose* [*OF* _ *conth*] *continuous_intros* | *simp add*: *c*)+
then show *?thesis*
by (*force simp*: *h*)
qed
moreover have *homotopic_with* (λ*x*. *True*) *X X'* (λ*x*. *g 0*) (λ*x*. *g 1*)
if *x* ∈ *topspace X a* = *g 0 b* = *g 1 continuous_map* (*top_of_set* {*0..1*}) *X' g*
for *x* **and** *g* :: *real* ⇒ *'b*
unfolding *homotopic_with_def*
by (*force intro*!: *continuous_map_compose continuous_intros c that*)
ultimately show *?thesis*
using *False*
by (*metis c path_component_of_set pathin_def*)
qed (*simp add*: *homotopic_on_empty*)

proposition *homotopic_with_eq*:
assumes *h*: *homotopic_with P X Y f g*
and *f'*: ⋀*x*. *x* ∈ *topspace X* ⟹ *f' x* = *f x*
and *g'*: ⋀*x*. *x* ∈ *topspace X* ⟹ *g' x* = *g x*
and *P*: (⋀*h k*. (⋀*x*. *x* ∈ *topspace X* ⟹ *h x* = *k x*) ⟹ *P h* ⟷ *P k*)
shows *homotopic_with P X Y f' g'*
by (*smt* (*verit*, *ccfv_SIG*) *assms homotopic_with*)

lemma *homotopic_with_prod_topology*:

assumes *homotopic_with p X1 Y1 f f'* **and** *homotopic_with q X2 Y2 g g'*
and *r*: $\bigwedge i\ j.\ [\![p\ i;\ q\ j]\!] \Longrightarrow r(\lambda(x,y).\ (i\ x,\ j\ y))$
shows *homotopic_with r* (*prod_topology X1 X2*) (*prod_topology Y1 Y2*)
$(\lambda z.\ (f(fst\ z),g(snd\ z)))$ $(\lambda z.\ (f'(fst\ z),\ g'(snd\ z)))$
proof −
obtain *h*
where *h*: *continuous_map* (*prod_topology* (*subtopology euclideanreal* {*0..1*}) *X1*) *Y1 h*
and *h0*: $\bigwedge x.\ h\ (0,\ x) = f\ x$
and *h1*: $\bigwedge x.\ h\ (1,\ x) = f'\ x$
and *p*: $\bigwedge t.\ [\![0 \le t;\ t \le 1]\!] \Longrightarrow p\ (\lambda x.\ h\ (t,x))$
using *assms* **unfolding** *homotopic_with_def* **by** *auto*
obtain *k*
where *k*: *continuous_map* (*prod_topology* (*subtopology euclideanreal* {*0..1*}) *X2*) *Y2 k*
and *k0*: $\bigwedge x.\ k\ (0,\ x) = g\ x$
and *k1*: $\bigwedge x.\ k\ (1,\ x) = g'\ x$
and *q*: $\bigwedge t.\ [\![0 \le t;\ t \le 1]\!] \Longrightarrow q\ (\lambda x.\ k\ (t,x))$
using *assms* **unfolding** *homotopic_with_def* **by** *auto*
let *?hk* = $\lambda(t,x,y).\ (h(t,x),\ k(t,y))$
show *?thesis*
unfolding *homotopic_with_def*
proof (*intro conjI allI exI*)
show *continuous_map* (*prod_topology* (*subtopology euclideanreal* {*0..1*}) (*prod_topology X1 X2*))
(*prod_topology Y1 Y2*) *?hk*
unfolding *continuous_map_pairwise case_prod_unfold*
by (*rule conjI continuous_map_pairedI continuous_intros continuous_map_id* [*unfolded id_def*]
continuous_map_fst_of [*unfolded o_def*] *continuous_map_snd_of* [*unfolded o_def*]
continuous_map_compose [*OF _ h, unfolded o_def*]
continuous_map_compose [*OF _ k, unfolded o_def*])+
next
fix *x*
show *?hk* (*0*, *x*) = (*f* (*fst x*), *g* (*snd x*)) *?hk* (*1*, *x*) = (*f'* (*fst x*), *g'* (*snd x*))
by (*auto simp*: *case_prod_beta h0 k0 h1 k1*)
qed (*auto simp*: *p q r*)
qed

lemma *homotopic_with_product_topology*:
assumes *ht*: $\bigwedge i.\ i \in I \Longrightarrow$ *homotopic_with* (*p i*) (*X i*) (*Y i*) (*f i*) (*g i*)
and *pq*: $\bigwedge h.\ (\bigwedge i.\ i \in I \Longrightarrow p\ i\ (h\ i)) \Longrightarrow q(\lambda x.\ (\lambda i{\in}I.\ h\ i\ (x\ i)))$
shows *homotopic_with q* (*product_topology X I*) (*product_topology Y I*)
$(\lambda z.\ (\lambda i{\in}I.\ (f\ i)\ (z\ i)))$ $(\lambda z.\ (\lambda i{\in}I.\ (g\ i)\ (z\ i)))$
proof −
obtain *h*
where *h*: $\bigwedge i.\ i \in I \Longrightarrow$ *continuous_map* (*prod_topology* (*subtopology euclidean-*

```
real {0..1}) (X i)) (Y i) (h i)
      and h0: ⋀i x. i ∈ I ⟹ h i (0, x) = f i x
      and h1: ⋀i x. i ∈ I ⟹ h i (1, x) = g i x
      and p: ⋀i t. ⟦i ∈ I; t ∈ {0..1}⟧ ⟹ p i (λx. h i (t,x))
    using ht unfolding homotopic_with_def by metis
  show ?thesis
    unfolding homotopic_with_def
  proof (intro conjI allI exI)
    let ?h = λ(t,z). λi∈I. h i (t,z i)
   have continuous_map (prod_topology (subtopology euclideanreal {0..1}) (product_topology
X I))
                        (Y i) (λx. h i (fst x, snd x i)) if i ∈ I for i
    proof −
     have §: continuous_map (prod_topology (top_of_set {0..1}) (product_topology
X I)) (X i) (λx. snd x i)
        using continuous_map_componentwise continuous_map_snd that by fast-
force
      show ?thesis
        unfolding continuous_map_pairwise case_prod_unfold
        by (intro conjI that § continuous_intros continuous_map_compose [OF _
h, unfolded o_def])
    qed
    then show continuous_map (prod_topology (subtopology euclideanreal {0..1})
(product_topology X I))
         (product_topology Y I) ?h
      by (auto simp: continuous_map_componentwise case_prod_beta)
    show ?h (0, x) = (λi∈I. f i (x i)) ?h (1, x) = (λi∈I. g i (x i)) for x
      by (auto simp: case_prod_beta h0 h1)
    show ∀ t∈{0..1}. q (λx. ?h (t, x))
      by (force intro: p pq)
  qed
qed
```

Homotopic triviality implicitly incorporates path-connectedness.

```
lemma homotopic_triviality:
  shows  (∀f g. continuous_on S f ∧ f ∈ S → T ∧
                continuous_on S g ∧ g ∈ S → T
                ⟶ homotopic_with_canon (λx. True) S T f g) ⟷
          (S = {} ∨ path_connected T) ∧
          (∀f. continuous_on S f ∧ f ∈ S → T ⟶ (∃c. homotopic_with_canon
(λx. True) S T f (λx. c)))
          (is ?lhs = ?rhs)
proof (cases S = {} ∨ T = {})
  case True then show ?thesis
    by (auto simp: homotopic_on_emptyI simp flip: image_subset_iff_funcset)
next
  case False show ?thesis
  proof
    assume LHS [rule_format]: ?lhs
```

have *pab*: *path_component T a b* **if** *a* ∈ *T b* ∈ *T* **for** *a b*
proof −
have *homotopic_with_canon* (λ*x*. *True*) *S T* (λ*x*. *a*) (λ*x*. *b*)
by (*simp add*: *LHS image_subset_iff that*)
then show *?thesis*
using *False homotopic_constant_maps* [*of top_of_set S top_of_set T a b*]
by (*metis path_component_of_canon_iff topspace_discrete_topology topspace_euclidean_subtopology*)
qed
moreover
have ∃ *c*. *homotopic_with_canon* (λ*x*. *True*) *S T f* (λ*x*. *c*) **if** *continuous_on S f f* ∈ *S* → *T* **for** *f*
using *False LHS continuous_on_const that* **by** *blast*
ultimately show *?rhs*
by (*simp add*: *path_connected_component*)
next
assume *RHS*: *?rhs*
with *False* **have** *T*: *path_connected T*
by *blast*
show *?lhs*
proof *clarify*
fix *f g*
assume *continuous_on S f f* ∈ *S* → *T continuous_on S g g* ∈ *S* → *T*
obtain *c d* **where** *c*: *homotopic_with_canon* (λ*x*. *True*) *S T f* (λ*x*. *c*) **and** *d*: *homotopic_with_canon* (λ*x*. *True*) *S T g* (λ*x*. *d*)
using *RHS* ‹*continuous_on S f*› ‹*continuous_on S g*› ‹*f* ∈ *S* → *T*› ‹*g* ∈ *S* → *T*› **by** *presburger*
with *T* **have** *path_component T c d*
by (*metis False ex_in_conv homotopic_with_imp_subset2 image_subset_iff path_connected_component*)
then have *homotopic_with_canon* (λ*x*. *True*) *S T* (λ*x*. *c*) (λ*x*. *d*)
by (*simp add*: *homotopic_constant_maps*)
with *c d* **show** *homotopic_with_canon* (λ*x*. *True*) *S T f g*
by (*meson homotopic_with_symD homotopic_with_trans*)
qed
qed
qed

5.4.4 Homotopy of paths, maintaining the same endpoints

definition *homotopic_paths* :: [*'a set*, *real* ⇒ *'a*, *real* ⇒ *'a*::*topological_space*] ⇒ *bool*
where
homotopic_paths S p q ≡
homotopic_with_canon (λ*r*. *pathstart r* = *pathstart p* ∧ *pathfinish r* = *pathfinish p*) {*0..1*} *S p q*

lemma *homotopic_paths*:
homotopic_paths S p q ⟷
(∃ *h*. *continuous_on* ({*0..1*} × {*0..1*}) *h* ∧

$h \in (\{0..1\} \times \{0..1\}) \to S \wedge$
$(\forall x \in \{0..1\}.\ h(0,x) = p\ x) \wedge$
$(\forall x \in \{0..1\}.\ h(1,x) = q\ x) \wedge$
$(\forall t \in \{0..1::real\}.\ pathstart(h \circ Pair\ t) = pathstart\ p \wedge$
$pathfinish(h \circ Pair\ t) = pathfinish\ p))$
by (*auto simp*: *homotopic_paths_def homotopic_with pathstart_def pathfinish_def*)

proposition *homotopic_paths_imp_pathstart*:
$homotopic\_paths\ S\ p\ q \Longrightarrow pathstart\ p = pathstart\ q$
by (*metis* (*mono_tags*, *lifting*) *homotopic_paths_def homotopic_with_imp_property*)

proposition *homotopic_paths_imp_pathfinish*:
$homotopic\_paths\ S\ p\ q \Longrightarrow pathfinish\ p = pathfinish\ q$
by (*metis* (*mono_tags*, *lifting*) *homotopic_paths_def homotopic_with_imp_property*)

lemma *homotopic_paths_imp_path*:
$homotopic\_paths\ S\ p\ q \Longrightarrow path\ p \wedge path\ q$
using *homotopic_paths_def homotopic_with_imp_continuous_maps path_def continuous_map_subtopology_eu* **by** *blast*

lemma *homotopic_paths_imp_subset*:
$homotopic\_paths\ S\ p\ q \Longrightarrow path\_image\ p \subseteq S \wedge path\_image\ q \subseteq S$
by (*metis* (*mono_tags*) *continuous_map_subtopology_eu homotopic_paths_def homotopic_with_imp_continuous_maps path_image_def*)

proposition *homotopic_paths_refl* [*simp*]: $homotopic\_paths\ S\ p\ p \longleftrightarrow path\ p \wedge path\_image\ p \subseteq S$
by (*simp add*: *homotopic_paths_def path_def path_image_def*)

proposition *homotopic_paths_sym*: $homotopic\_paths\ S\ p\ q \Longrightarrow homotopic\_paths\ S\ q\ p$
by (*metis* (*mono_tags*) *homotopic_paths_def homotopic_paths_imp_pathfinish homotopic_paths_imp_pathstart homotopic_with_symD*)

proposition *homotopic_paths_sym_eq*: $homotopic\_paths\ S\ p\ q \longleftrightarrow homotopic\_paths\ S\ q\ p$
by (*metis homotopic_paths_sym*)

proposition *homotopic_paths_trans* [*trans*]:
assumes $homotopic\_paths\ S\ p\ q$ $homotopic\_paths\ S\ q\ r$
shows $homotopic\_paths\ S\ p\ r$
using *assms homotopic_paths_imp_pathfinish homotopic_paths_imp_pathstart*
unfolding *homotopic_paths_def*
by (*smt* (*verit*, *ccfv_SIG*) *homotopic_with_mono homotopic_with_trans*)

proposition *homotopic_paths_eq*:
$[\![path\ p;\ path\_image\ p \subseteq S;\ \bigwedge t.\ t \in \{0..1\} \Longrightarrow p\ t = q\ t]\!] \Longrightarrow homotopic\_paths\ S\ p\ q$
by (*smt* (*verit*, *best*) *homotopic_paths homotopic_paths_refl*)

proposition *homotopic_paths_reparametrize*:
assumes *path p*
and *pips*: *path_image p* ⊆ *S*
and *contf*: *continuous_on* {*0..1*} *f*
and *f01* :*f* ∈ {*0..1*} → {*0..1*}
and [*simp*]: *f*(*0*) = *0 f*(*1*) = *1*
and *q*: ⋀*t*. *t* ∈ {*0..1*} ⟹ *q*(*t*) = *p*(*f t*)
shows *homotopic_paths S p q*
proof −
have *contp*: *continuous_on* {*0..1*} *p*
by (*metis* ‹*path p*› *path_def*)
then have *continuous_on* {*0..1*} (*p* ∘ *f*)
by (*meson assms*(*4*) *contf continuous_on_compose continuous_on_subset image_subset_iff_funcset*)
then have *path q*
by (*simp add*: *path_def*) (*metis q continuous_on_cong*)
have *piqs*: *path_image q* ⊆ *S*
by (*smt* (*verit*, *ccfv_threshold*) *Pi_iff assms*(*2*) *assms*(*4*) *assms*(*7*) *image_subset_iff path_defs*(*4*))
have *fb0*: ⋀*a b*. ⟦*0* ≤ *a*; *a* ≤ *1*; *0* ≤ *b*; *b* ≤ *1*⟧ ⟹ *0* ≤ (*1* − *a*) ∗ *f b* + *a* ∗ *b*
using *f01* **by** *force*
have *fb1*: ⟦*0* ≤ *a*; *a* ≤ *1*; *0* ≤ *b*; *b* ≤ *1*⟧ ⟹ (*1* − *a*) ∗ *f b* + *a* ∗ *b* ≤ *1* **for** *a b*
by (*intro convex_bound_le*) (*use f01* **in** *auto*)
have *homotopic_paths S q p*
proof (*rule homotopic_paths_trans*)
show *homotopic_paths S q* (*p* ∘ *f*)
using *q* **by** (*force intro*: *homotopic_paths_eq* [*OF* ‹*path q*› *piqs*])
next
show *homotopic_paths S* (*p* ∘ *f*) *p*
using *pips* [*unfolded path_image_def*]
apply (*simp add*: *homotopic_paths_def homotopic_with_def*)
apply (*rule_tac x*=*p* ∘ (λ*y*. (*1* − (*fst y*)) ∗$_R$ ((*f* ∘ *snd*) *y*) + (*fst y*) ∗$_R$ *snd y*) **in** *exI*)
apply (*rule conjI contf continuous_intros continuous_on_subset* [*OF contp*] | *simp*)+
by (*auto simp*: *fb0 fb1 pathstart_def pathfinish_def*)
qed
then show *?thesis*
by (*simp add*: *homotopic_paths_sym*)
qed

lemma *homotopic_paths_subset*: ⟦*homotopic_paths S p q*; *S* ⊆ *t*⟧ ⟹ *homotopic_paths t p q*
unfolding *homotopic_paths* **by** *fast*

A slightly ad-hoc but useful lemma in constructing homotopies.

lemma *continuous_on_homotopic_join_lemma*:
fixes *q* :: [*real*,*real*] ⇒ *'a*::*topological_space*

assumes *p*: *continuous_on* ({*0..1*} × {*0..1*}) (λ*y*. *p* (*fst y*) (*snd y*)) (**is** *continuous_on ?A ?p*)
and *q*: *continuous_on* ({*0..1*} × {*0..1*}) (λ*y*. *q* (*fst y*) (*snd y*)) (**is** *continuous_on ?A ?q*)
and *pf*: ⋀*t*. *t* ∈ {*0..1*} ⟹ *pathfinish*(*p t*) = *pathstart*(*q t*)
shows *continuous_on* ({*0..1*} × {*0..1*}) (λ*y*. (*p*(*fst y*) +++ *q*(*fst y*)) (*snd y*))
proof −
have §: (λ*t*. *p* (*fst t*) (*2* ∗ *snd t*)) = *?p* ∘ (λ*y*. (*fst y*, *2* ∗ *snd y*))
(λ*t*. *q* (*fst t*) (*2* ∗ *snd t* − *1*)) = *?q* ∘ (λ*y*. (*fst y*, *2* ∗ *snd y* − *1*))
by *force*+
show *?thesis*
unfolding *joinpaths_def*
proof (*rule continuous_on_cases_le*)
show *continuous_on* {*y* ∈ *?A*. *snd y* ≤ *1/2*} (λ*t*. *p* (*fst t*) (*2* ∗ *snd t*))
continuous_on {*y* ∈ *?A*. *1/2* ≤ *snd y*} (λ*t*. *q* (*fst t*) (*2* ∗ *snd t* − *1*))
continuous_on ?A snd
unfolding §
by (*rule continuous_intros continuous_on_subset* [*OF p*] *continuous_on_subset* [*OF q*] | *force*)+
qed (*use pf* **in** ‹*auto simp*: *mult.commute pathstart_def pathfinish_def*›)
qed

Congruence properties of homotopy w.r.t. path-combining operations.

lemma *homotopic_paths_reversepath_D*:
assumes *homotopic_paths S p q*
shows *homotopic_paths S* (*reversepath p*) (*reversepath q*)
using *assms*
apply (*simp add*: *homotopic_paths_def homotopic_with_def*, *clarify*)
apply (*rule_tac x*=*h* ∘ (λ*x*. (*fst x*, *1* − *snd x*)) **in** *exI*)
apply (*rule conjI continuous_intros*)+
apply (*auto simp*: *reversepath_def pathstart_def pathfinish_def elim*!: *continuous_on_subset*)
done

proposition *homotopic_paths_reversepath*:
homotopic_paths S (*reversepath p*) (*reversepath q*) ⟷ *homotopic_paths S p q*
using *homotopic_paths_reversepath_D* **by** *force*

proposition *homotopic_paths_join*:
⟦*homotopic_paths S p p*′; *homotopic_paths S q q*′; *pathfinish p* = *pathstart q*⟧ ⟹ *homotopic_paths S* (*p* +++ *q*) (*p*′ +++ *q*′)
apply (*clarsimp simp*: *homotopic_paths_def homotopic_with_def*)
apply (*rename_tac k1 k2*)
apply (*rule_tac x*=(λ*y*. ((*k1* ∘ *Pair* (*fst y*)) +++ (*k2* ∘ *Pair* (*fst y*))) (*snd y*)) **in** *exI*)
apply (*intro conjI continuous_intros continuous_on_homotopic_join_lemma*; *force simp*: *joinpaths_def pathstart_def pathfinish_def path_image_def*)

done

proposition *homotopic_paths_continuous_image*:
⟦*homotopic_paths S f g*; *continuous_on S h*; *h* ∈ *S* → *t*⟧ ⟹ *homotopic_paths t* (*h* ∘ *f*) (*h* ∘ *g*)
unfolding *homotopic_paths_def*
by (*simp add*: *homotopic_with_compose_continuous_map_left pathfinish_compose pathstart_compose image_subset_iff_funcset*)

5.4.5 Group properties for homotopy of paths

So taking equivalence classes under homotopy would give the fundamental group

proposition *homotopic_paths_rid*:
assumes *path p path_image p* ⊆ *S*
shows *homotopic_paths S* (*p* +++ *linepath* (*pathfinish p*) (*pathfinish p*)) *p*
proof −
have §: *continuous_on* {*0..1*} (λ*t*::*real*. *if t* ≤ *1/2 then 2* ∗*R t else 1*)
unfolding *split_01*
by (*rule continuous_on_cases continuous_intros* | *force simp*: *pathfinish_def joinpaths_def*)+
show *?thesis*
apply (*rule homotopic_paths_sym*)
using *assms* **unfolding** *pathfinish_def joinpaths_def*
by (*intro* § *continuous_on_cases continuous_intros homotopic_paths_reparametrize* [**where** *f* = λ*t*. *if t* ≤ *1/2 then 2* ∗*R t else 1*]; *force*)
qed

proposition *homotopic_paths_lid*:
⟦*path p*; *path_image p* ⊆ *S*⟧ ⟹ *homotopic_paths S* (*linepath* (*pathstart p*) (*pathstart p*) +++ *p*) *p*
using *homotopic_paths_rid* [*of reversepath p S*]
by (*metis homotopic_paths_reversepath path_image_reversepath path_reversepath pathfinish_linepath*
pathfinish_reversepath reversepath_joinpaths reversepath_linepath)

proposition *homotopic_paths_assoc*:
⟦*path p*; *path_image p* ⊆ *S*; *path q*; *path_image q* ⊆ *S*; *path r*; *path_image r* ⊆ *S*; *pathfinish p* = *pathstart q*;
pathfinish q = *pathstart r*⟧
⟹ *homotopic_paths S* (*p* +++ (*q* +++ *r*)) ((*p* +++ *q*) +++ *r*)
apply (*subst homotopic_paths_sym*)
apply (*rule homotopic_paths_reparametrize*
[**where** *f* = λ*t*. *if t* ≤ *1/2 then inverse 2* ∗*R t*
else if t ≤ *3 / 4 then t* − (*1 / 4*)
else 2 ∗*R t* − *1*])
apply (*simp_all del*: *le_divide_eq_numeral1 add*: *subset_path_image_join*)
apply (*rule continuous_on_cases_1 continuous_intros* | *auto simp*: *joinpaths_def*)+
done

proposition *homotopic_paths_rinv*:
assumes *path p path_image p* $\subseteq$ *S*
shows *homotopic_paths S* (*p* +++ *reversepath p*) (*linepath* (*pathstart p*) (*pathstart p*))
proof −
have *p*: *continuous_on* {*0..1*} *p*
using *assms* **by** (*auto simp*: *path_def*)
let *?A* = {*0..1*} × {*0..1*}
have *continuous_on ?A* (λ*x*. (*subpath 0* (*fst x*) *p* +++ *reversepath* (*subpath 0* (*fst x*) *p*)) (*snd x*))
unfolding *joinpaths_def subpath_def reversepath_def path_def add_0_right diff_0_right*
proof (*rule continuous_on_cases_le*)
show *continuous_on* {*x* ∈ *?A*. *snd x* ≤ *1/2*} (λ*t*. *p* (*fst t* ∗ (*2* ∗ *snd t*)))
continuous_on {*x* ∈ *?A*. *1/2* ≤ *snd x*} (λ*t*. *p* (*fst t* ∗ (*1* − (*2* ∗ *snd t* − *1*))))
continuous_on ?A snd
by (*intro continuous_on_compose2* [*OF p*] *continuous_intros*; *auto simp*: *mult_le_one*)+
qed (*auto simp*: *algebra_simps*)
then show *?thesis*
using *assms*
apply (*subst homotopic_paths_sym_eq*)
unfolding *homotopic_paths_def homotopic_with_def*
apply (*rule_tac x*=(λ*y*. (*subpath 0* (*fst y*) *p* +++ *reversepath*(*subpath 0* (*fst y*) *p*)) (*snd y*)) **in** *exI*)
apply (*force simp*: *mult_le_one path_defs joinpaths_def subpath_def reversepath_def*)
done
qed

proposition *homotopic_paths_linv*:
assumes *path p path_image p* $\subseteq$ *S*
shows *homotopic_paths S* (*reversepath p* +++ *p*) (*linepath* (*pathfinish p*) (*pathfinish p*))
using *homotopic_paths_rinv* [*of reversepath p S*] *assms* **by** *simp*

5.4.6 Homotopy of loops without requiring preservation of endpoints

definition *homotopic_loops* :: *'a*::*topological_space set* ⇒ (*real* ⇒ *'a*) ⇒ (*real* ⇒ *'a*) ⇒ *bool* **where**
homotopic_loops S p q ≡
homotopic_with_canon (λ*r*. *pathfinish r* = *pathstart r*) {*0..1*} *S p q*

lemma *homotopic_loops*:
homotopic_loops S p q ⟷
(∃*h*. *continuous_on* ({*0..1*::*real*} × {*0..1*}) *h* ∧
image h ({*0..1*} × {*0..1*}) ⊆ *S* ∧

$(\forall x \in \{0..1\}.\ h(0,x) = p\ x) \wedge$
$(\forall x \in \{0..1\}.\ h(1,x) = q\ x) \wedge$
$(\forall t \in \{0..1\}.\ pathfinish(h \circ Pair\ t) = pathstart(h \circ Pair\ t)))$
by (*simp add*: *homotopic_loops_def pathstart_def pathfinish_def homotopic_with*)

proposition *homotopic_loops_imp_loop*:
$homotopic\_loops\ S\ p\ q \Longrightarrow pathfinish\ p = pathstart\ p \wedge pathfinish\ q = pathstart\ q$
using *homotopic_with_imp_property homotopic_loops_def* **by** *blast*

proposition *homotopic_loops_imp_path*:
$homotopic\_loops\ S\ p\ q \Longrightarrow path\ p \wedge path\ q$
unfolding *homotopic_loops_def path_def*
using *homotopic_with_imp_continuous_maps continuous_map_subtopology_eu*
by *blast*

proposition *homotopic_loops_imp_subset*:
$homotopic\_loops\ S\ p\ q \Longrightarrow path\_image\ p \subseteq S \wedge path\_image\ q \subseteq S$
unfolding *homotopic_loops_def path_image_def*
by (*meson continuous_map_subtopology_eu homotopic_with_imp_continuous_maps*)

proposition *homotopic_loops_refl*:
$homotopic\_loops\ S\ p\ p \longleftrightarrow$
$path\ p \wedge path\_image\ p \subseteq S \wedge pathfinish\ p = pathstart\ p$
by (*simp add*: *homotopic_loops_def path_image_def path_def*)

proposition *homotopic_loops_sym*: $homotopic\_loops\ S\ p\ q \Longrightarrow homotopic\_loops\ S\ q\ p$
by (*simp add*: *homotopic_loops_def homotopic_with_sym*)

proposition *homotopic_loops_sym_eq*: $homotopic\_loops\ S\ p\ q \longleftrightarrow homotopic\_loops\ S\ q\ p$
by (*metis homotopic_loops_sym*)

proposition *homotopic_loops_trans*:
$[\![homotopic\_loops\ S\ p\ q; homotopic\_loops\ S\ q\ r]\!] \Longrightarrow homotopic\_loops\ S\ p\ r$
unfolding *homotopic_loops_def* **by** (*blast intro*: *homotopic_with_trans*)

proposition *homotopic_loops_subset*:
$[\![homotopic\_loops\ S\ p\ q; S \subseteq t]\!] \Longrightarrow homotopic\_loops\ t\ p\ q$
by (*fastforce simp*: *homotopic_loops*)

proposition *homotopic_loops_eq*:
$[\![path\ p; path\_image\ p \subseteq S; pathfinish\ p = pathstart\ p; \bigwedge t.\ t \in \{0..1\} \Longrightarrow p(t) = q(t)]\!]$
$\Longrightarrow homotopic\_loops\ S\ p\ q$
unfolding *homotopic_loops_def path_image_def path_def pathstart_def pathfinish_def*
by (*auto intro*: *homotopic_with_eq* [*OF homotopic_with_refl* [**where** $f = p$,

THEN iffD2]])

proposition *homotopic_loops_continuous_image*:
⟦*homotopic_loops S f g*; *continuous_on S h*; *h* ∈ *S* → *t*⟧ ⟹ *homotopic_loops t* (*h* ∘ *f*) (*h* ∘ *g*)
unfolding *homotopic_loops_def*
by (*simp add*: *homotopic_with_compose_continuous_map_left pathfinish_def pathstart_def image_subset_iff_funcset*)

5.4.7 Relations between the two variants of homotopy

proposition *homotopic_paths_imp_homotopic_loops*:
⟦*homotopic_paths S p q*; *pathfinish p* = *pathstart p*; *pathfinish q* = *pathstart p*⟧
⟹ *homotopic_loops S p q*
by (*auto simp*: *homotopic_with_def homotopic_paths_def homotopic_loops_def*)

proposition *homotopic_loops_imp_homotopic_paths_null*:
assumes *homotopic_loops S p* (*linepath a a*)
shows *homotopic_paths S p* (*linepath* (*pathstart p*) (*pathstart p*))
proof −
have *path p* **by** (*metis assms homotopic_loops_imp_path*)
have *ploop*: *pathfinish p* = *pathstart p* **by** (*metis assms homotopic_loops_imp_loop*)
have *pip*: *path_image p* ⊆ *S* **by** (*metis assms homotopic_loops_imp_subset*)
let *?A* = {*0..1*::*real*} × {*0..1*::*real*}
obtain *h* **where** *conth*: *continuous_on ?A h*
and *hs*: *h* ∈ *?A* → *S*
and *h0*[*simp*]: ⋀*x*. *x* ∈ {*0..1*} ⟹ *h*(*0*,*x*) = *p x*
and *h1*[*simp*]: ⋀*x*. *x* ∈ {*0..1*} ⟹ *h*(*1*,*x*) = *a*
and *ends*: ⋀*t*. *t* ∈ {*0..1*} ⟹ *pathfinish* (*h* ∘ *Pair t*) = *pathstart* (*h* ∘ *Pair t*)
using *assms* **by** (*auto simp*: *homotopic_loops homotopic_with image_subset_iff_funcset*)
have *conth0*: *path* (λ*u*. *h* (*u*, *0*))
unfolding *path_def*
proof (*rule continuous_on_compose* [*of* _ _ *h*, *unfolded o_def*])
show *continuous_on* ((λ*x*. (*x*, *0*)) ‘ {*0..1*}) *h*
by (*force intro*: *continuous_on_subset* [*OF conth*])
qed (*force intro*: *continuous_intros*)
have *pih0*: *path_image* (λ*u*. *h* (*u*, *0*)) ⊆ *S*
using *hs* **by** (*force simp*: *path_image_def*)
have *c1*: *continuous_on ?A* (λ*x*. *h* (*fst x* ∗ *snd x*, *0*))
proof (*rule continuous_on_compose* [*of* _ _ *h*, *unfolded o_def*])
show *continuous_on* ((λ*x*. (*fst x* ∗ *snd x*, *0*)) ‘ *?A*) *h*
by (*force simp*: *mult_le_one intro*: *continuous_on_subset* [*OF conth*])
qed (*force intro*: *continuous_intros*)+
have *c2*: *continuous_on ?A* (λ*x*. *h* (*fst x* − *fst x* ∗ *snd x*, *0*))
proof (*rule continuous_on_compose* [*of* _ _ *h*, *unfolded o_def*])
show *continuous_on* ((λ*x*. (*fst x* − *fst x* ∗ *snd x*, *0*)) ‘ *?A*) *h*
by (*auto simp*: *algebra_simps add_increasing2 mult_left_le intro*: *continuous_on_subset* [*OF conth*])

qed (*force intro*: *continuous_intros*)
have [*simp*]: $\bigwedge t.\ [\![0 \le t \wedge t \le 1]\!] \Longrightarrow h\ (t,\ 1) = h\ (t,\ 0)$
using *ends* **by** (*simp add*: *pathfinish_def pathstart_def*)
have *adhoc_le*: $c * 4 \le 1 + c * (d * 4)$ **if** $\neg\ d * 4 \le 3\ 0 \le c\ c \le 1$ **for** c *d*::*real*
proof −
have $c * 3 \le c * (d * 4)$ **using** *that less_eq_real_def* **by** *auto*
with ‹$c \le 1$› **show** *?thesis* **by** *fastforce*
qed
have $*$: $\bigwedge p\ x.\ [\![path\ p \wedge path(reversepath\ p);$
$path\_image\ p \subseteq S \wedge path\_image(reversepath\ p) \subseteq S;$
$pathfinish\ p = pathstart(linepath\ a\ a +\!\!+\!\!+ reversepath\ p) \wedge$
$pathstart(reversepath\ p) = a \wedge pathstart\ p = x]\!]$
$\Longrightarrow homotopic\_paths\ S\ (p +\!\!+\!\!+ linepath\ a\ a +\!\!+\!\!+ reversepath\ p)$ (*linepath x x*)
by (*metis homotopic_paths_lid homotopic_paths_join*
homotopic_paths_trans homotopic_paths_sym homotopic_paths_rinv)
have *1*: *homotopic_paths S p* (*p +++ linepath* (*pathfinish p*) (*pathfinish p*))
using ‹*path p*› *homotopic_paths_rid homotopic_paths_sym pip* **by** *blast*
moreover have *homotopic_paths S* (*p +++ linepath* (*pathfinish p*) (*pathfinish p*))
(*linepath* (*pathstart p*) (*pathstart p*) *+++ p +++ linepath* (*pathfinish p*) (*pathfinish p*))
using *homotopic_paths_lid* [*of p +++ linepath* (*pathfinish p*) (*pathfinish p*) *S*]
by (*metis 1 homotopic_paths_imp_path homotopic_paths_imp_subset homotopic_paths_sym pathstart_join*)
moreover
have *homotopic_paths S* (*linepath* (*pathstart p*) (*pathstart p*) *+++ p +++ linepath* (*pathfinish p*) (*pathfinish p*))
((λ*u*. *h* (*u*, 0)) *+++ linepath a a +++ reversepath* (λ*u*. *h* (*u*, 0)))
unfolding *homotopic_paths_def homotopic_with_def*
proof (*intro exI strip conjI*)
let *?h* = λ*y*. (*subpath 0* (*fst y*) (λ*u*. *h* (*u*, 0)) *+++* (λ*u*. *h* (*Pair* (*fst y*) *u*))
+++ subpath (*fst y*) *0* (λ*u*. *h* (*u*, 0))) (*snd y*)
have *continuous_on ?A ?h*
by (*intro continuous_on_homotopic_join_lemma*; *simp add*: *path_defs joinpaths_def subpath_def conth c1 c2*)
moreover have *?h* ∈ *?A* → *S*
using *hs*
unfolding *joinpaths_def subpath_def*
by (*force simp*: *algebra_simps mult_le_one mult_left_le intro*: *adhoc_le*)
ultimately show *continuous_map* (*prod_topology* (*top_of_set* {*0..1*}) (*top_of_set* {*0..1*}))
(*top_of_set S*) *?h*
by (*simp add*: *subpath_reversepath image_subset_iff_funcset*)
qed (*use ploop* **in** ‹*simp_all add*: *reversepath_def path_defs joinpaths_def o_def subpath_def conth c1 c2*›)
moreover have *homotopic_paths S* ((λ*u*. *h* (*u*, 0)) *+++ linepath a a +++*

reversepath (λ*u*. *h* (*u*, *0*)))
(*linepath* (*pathstart p*) (*pathstart p*))
by (*rule* ∗; *simp add*: *pih0 pathstart_def pathfinish_def conth0*; *simp add*: *reversepath_def joinpaths_def*)
ultimately show *?thesis*
by (*blast intro*: *homotopic_paths_trans*)
qed

proposition *homotopic_loops_conjugate*:
fixes *S* :: *'a*::*real_normed_vector set*
assumes *path p path q* **and** *pip*: *path_image p* ⊆ *S* **and** *piq*: *path_image q* ⊆ *S*
and *pq*: *pathfinish p* = *pathstart q* **and** *qloop*: *pathfinish q* = *pathstart q*
shows *homotopic_loops S* (*p* +++ *q* +++ *reversepath p*) *q*
proof −
have *contp*: *continuous_on* {*0..1*} *p* **using** ‹*path p*› [*unfolded path_def*] **by** *blast*
have *contq*: *continuous_on* {*0..1*} *q* **using** ‹*path q*› [*unfolded path_def*] **by** *blast*
let *?A* = {*0..1*::*real*} × {*0..1*::*real*}
have *c1*: *continuous_on ?A* (λ*x*. *p* ((*1* − *fst x*) ∗ *snd x* + *fst x*))
proof (*rule continuous_on_compose* [*of* _ _ *p*, *unfolded o_def*])
show *continuous_on* ((λ*x*. (*1* − *fst x*) ∗ *snd x* + *fst x*) ‘ *?A*) *p*
by (*auto intro*: *continuous_on_subset* [*OF contp*] *simp*: *algebra_simps add_increasing2 mult_right_le_one_le sum_le_prod1*)
qed (*force intro*: *continuous_intros*)
have *c2*: *continuous_on ?A* (λ*x*. *p* ((*fst x* − *1*) ∗ *snd x* + *1*))
proof (*rule continuous_on_compose* [*of* _ _ *p*, *unfolded o_def*])
show *continuous_on* ((λ*x*. (*fst x* − *1*) ∗ *snd x* + *1*) ‘ *?A*) *p*
by (*auto intro*: *continuous_on_subset* [*OF contp*] *simp*: *algebra_simps add_increasing2 mult_left_le_one_le*)
qed (*force intro*: *continuous_intros*)

have *ps1*: ⋀*a b*. ⟦*b* ∗ *2* ≤ *1*; *0* ≤ *b*; *0* ≤ *a*; *a* ≤ *1*⟧ ⟹ *p* ((*1* − *a*) ∗ (*2* ∗ *b*) + *a*) ∈ *S*
using *sum_le_prod1*
by (*force simp*: *algebra_simps add_increasing2 mult_left_le intro*: *pip* [*unfolded path_image_def*, *THEN subsetD*])
have *ps2*: ⋀*a b*. ⟦¬ *4* ∗ *b* ≤ *3*; *b* ≤ *1*; *0* ≤ *a*; *a* ≤ *1*⟧ ⟹ *p* ((*a* − *1*) ∗ (*4* ∗ *b* − *3*) + *1*) ∈ *S*
apply (*rule pip* [*unfolded path_image_def*, *THEN subsetD*])
apply (*rule image_eqI*, *blast*)
apply (*simp add*: *algebra_simps*)
by (*metis add_mono affine_ineq linear mult.commute mult.left_neutral mult_right_mono add.commute zero_le_numeral*)
have *qs*: ⋀*a b*. ⟦*4* ∗ *b* ≤ *3*; ¬ *b* ∗ *2* ≤ *1*⟧ ⟹ *q* (*4* ∗ *b* − *2*) ∈ *S*
using *path_image_def piq* **by** *fastforce*
have *homotopic_loops S* (*p* +++ *q* +++ *reversepath p*)
(*linepath* (*pathstart q*) (*pathstart q*) +++ *q* +++ *linepath* (*pathstart q*) (*pathstart q*))
unfolding *homotopic_loops_def homotopic_with_def*
proof (*intro exI strip conjI*)

let *?h* = (λ*y*. (*subpath* (*fst y*) *1 p* +++ *q* +++ *subpath 1* (*fst y*) *p*) (*snd y*))
have *continuous_on ?A* (λ*y*. *q* (*snd y*))
by (*force simp*: *contq intro*: *continuous_on_compose* [*of _ _ q*, *unfolded o_def*] *continuous_on_id continuous_on_snd*)
then have *continuous_on ?A ?h*
using *pq qloop*
by (*intro continuous_on_homotopic_join_lemma*) (*auto simp*: *path_defs joinpaths_def subpath_def c1 c2*)
then show *continuous_map* (*prod_topology* (*top_of_set* {*0..1*}) (*top_of_set* {*0..1*})) (*top_of_set S*) *?h*
by (*auto simp*: *joinpaths_def subpath_def ps1 ps2 qs*)
show *?h* (*1*,*x*) = (*linepath* (*pathstart q*) (*pathstart q*) +++ *q* +++ *linepath* (*pathstart q*) (*pathstart q*)) *x* **for** *x*
using *pq* **by** (*simp add*: *pathfinish_def subpath_refl*)
qed (*auto simp*: *subpath_reversepath*)
moreover have *homotopic_loops S* (*linepath* (*pathstart q*) (*pathstart q*) +++ *q* +++ *linepath* (*pathstart q*) (*pathstart q*)) *q*
proof −
have *homotopic_paths S* (*linepath* (*pathfinish q*) (*pathfinish q*) +++ *q*) *q*
using ‹*path q*› *homotopic_paths_lid qloop piq* **by** *auto*
hence *1*: ⋀*f*. *homotopic_paths S f q* ∨ ¬ *homotopic_paths S f* (*linepath* (*pathfinish q*) (*pathfinish q*) +++ *q*)
using *homotopic_paths_trans* **by** *blast*
hence *homotopic_paths S* (*linepath* (*pathfinish q*) (*pathfinish q*) +++ *q* +++ *linepath* (*pathfinish q*) (*pathfinish q*)) *q*
by (*smt* (*verit*, *best*) ‹*path q*› *homotopic_paths_imp_path homotopic_paths_imp_subset homotopic_paths_lid*
homotopic_paths_rid homotopic_paths_trans pathstart_join piq qloop)
thus *?thesis*
by (*metis* (*no_types*) *qloop homotopic_loops_sym homotopic_paths_imp_homotopic_loops homotopic_paths_imp_pathfinish homotopic_paths_sym*)
qed
ultimately show *?thesis*
by (*blast intro*: *homotopic_loops_trans*)
qed

lemma *homotopic_paths_loop_parts*:
assumes *loops*: *homotopic_loops S* (*p* +++ *reversepath q*) (*linepath a a*) **and** *path q*
shows *homotopic_paths S p q*
proof −
have *paths*: *homotopic_paths S* (*p* +++ *reversepath q*) (*linepath* (*pathstart p*) (*pathstart p*))
using *homotopic_loops_imp_homotopic_paths_null* [*OF loops*] **by** *simp*
then have *path p*
using ‹*path q*› *homotopic_loops_imp_path loops path_join path_join_path_ends path_reversepath* **by** *blast*
show *?thesis*
proof (*cases pathfinish p* = *pathfinish q*)

case *True*
obtain *pipq*: *path_image p ⊆ S path_image q ⊆ S*
by (*metis Un_subset_iff paths ‹path p› ‹path q› homotopic_loops_imp_subset homotopic_paths_imp_path loops*
path_image_join path_image_reversepath path_imp_reversepath path_join_eq)
have *homotopic_paths S p* (*p +++* (*linepath* (*pathfinish p*) (*pathfinish p*)))
using ‹*path p*› ‹*path_image p ⊆ S*› *homotopic_paths_rid homotopic_paths_sym*
by *blast*
moreover have *homotopic_paths S* (*p +++* (*linepath* (*pathfinish p*) (*pathfinish p*))) (*p +++* (*reversepath q +++ q*))
by (*simp add*: *True ‹path p› ‹path q› pipq homotopic_paths_join homotopic_paths_linv homotopic_paths_sym*)
moreover have *homotopic_paths S* (*p +++* (*reversepath q +++ q*)) ((*p +++ reversepath q*) *+++ q*)
by (*simp add*: *True ‹path p› ‹path q› homotopic_paths_assoc pipq*)
moreover have *homotopic_paths S* ((*p +++ reversepath q*) *+++ q*) (*linepath* (*pathstart p*) (*pathstart p*) *+++ q*)
by (*simp add*: ‹*path q*› *homotopic_paths_join paths pipq*)
ultimately show *?thesis*
by (*metis ‹path q› homotopic_paths_imp_path homotopic_paths_lid homotopic_paths_trans path_join_path_ends pathfinish_linepath pipq*(*2*))
next
case *False*
then show *?thesis*
using ‹*path q*› *homotopic_loops_imp_path loops path_join_path_ends* **by** *fastforce*
qed
qed

5.4.8 Homotopy of "nearby" function, paths and loops

lemma *homotopic_with_linear*:
fixes *f g* :: *_* ⇒ *'b*::*real_normed_vector*
assumes *contf*: *continuous_on S f*
and *contg*:*continuous_on S g*
and *sub*: ⋀*x*. *x* ∈ *S* ⟹ *closed_segment* (*f x*) (*g x*) ⊆ *t*
shows *homotopic_with_canon* (λ*z*. *True*) *S t f g*
unfolding *homotopic_with_def*
apply (*rule_tac x*=λ*y*. ((*1* − (*fst y*)) *∗*$_R$ *f*(*snd y*) + (*fst y*) *∗*$_R$ *g*(*snd y*)) **in** *exI*)
using *sub closed_segment_def*
by (*fastforce intro*: *continuous_intros continuous_on_subset* [*OF contf*] *continuous_on_compose2* [**where** *g*=*f*]
continuous_on_subset [*OF contg*] *continuous_on_compose2* [**where** *g*=*g*])

lemma *homotopic_paths_linear*:
fixes *g h* :: *real* ⇒ *'a*::*real_normed_vector*
assumes *path g path h pathstart h* = *pathstart g pathfinish h* = *pathfinish g*
⋀*t*. *t* ∈ {*0*..*1*} ⟹ *closed_segment* (*g t*) (*h t*) ⊆ *S*

shows *homotopic_paths S g h*
using *assms*
unfolding *path_def*
apply (*simp add*: *closed_segment_def pathstart_def pathfinish_def homotopic_paths_def homotopic_with_def*)
apply (*rule_tac x*=λ*y*. ((1 − (*fst y*)) ∗$_R$ (*g* ∘ *snd*) *y* + (*fst y*) ∗$_R$ (*h* ∘ *snd*) *y*) **in** *exI*)
apply (*intro conjI subsetI continuous_intros*; *force*)
done

lemma *homotopic_loops_linear*:
fixes *g h* :: *real* ⇒ 'a::*real_normed_vector*
assumes *path g path h pathfinish g* = *pathstart g pathfinish h* = *pathstart h*
⋀*t x*. *t* ∈ {0..1} ⟹ *closed_segment* (*g t*) (*h t*) ⊆ *S*
shows *homotopic_loops S g h*
using *assms*
unfolding *path_defs homotopic_loops_def homotopic_with_def*
apply (*rule_tac x*=λ*y*. ((1 − (*fst y*)) ∗$_R$ *g*(*snd y*) + (*fst y*) ∗$_R$ *h*(*snd y*)) **in** *exI*)
by (*force simp*: *closed_segment_def intro*!: *continuous_intros intro*: *continuous_on_compose2* [**where** *g*=*g*] *continuous_on_compose2* [**where** *g*=*h*])

lemma *homotopic_paths_nearby_explicit*:
assumes §: *path g path h pathstart h* = *pathstart g pathfinish h* = *pathfinish g*
and *no*: ⋀*t x*. ⟦*t* ∈ {0..1}; *x* ∉ *S*⟧ ⟹ *norm*(*h t* − *g t*) < *norm*(*g t* − *x*)
shows *homotopic_paths S g h*
using *homotopic_paths_linear* [*OF* §] **by** (*metis linorder_not_le no norm_minus_commute segment_bound1 subsetI*)

lemma *homotopic_loops_nearby_explicit*:
assumes §: *path g path h pathfinish g* = *pathstart g pathfinish h* = *pathstart h*
and *no*: ⋀*t x*. ⟦*t* ∈ {0..1}; *x* ∉ *S*⟧ ⟹ *norm*(*h t* − *g t*) < *norm*(*g t* − *x*)
shows *homotopic_loops S g h*
using *homotopic_loops_linear* [*OF* §] **by** (*metis linorder_not_le no norm_minus_commute segment_bound1 subsetI*)

lemma *homotopic_nearby_paths*:
fixes *g h* :: *real* ⇒ 'a::*euclidean_space*
assumes *path g open S path_image g* ⊆ *S*
shows ∃ *e*. 0 < *e* ∧
(∀ *h*. *path h* ∧
pathstart h = *pathstart g* ∧ *pathfinish h* = *pathfinish g* ∧
(∀ *t* ∈ {0..1}. *norm*(*h t* − *g t*) < *e*) ⟶ *homotopic_paths S g h*)
proof −
obtain *e* **where** *e* > 0 **and** *e*: ⋀*x y*. *x* ∈ *path_image g* ⟹ *y* ∈ − *S* ⟹ *e* ≤ *dist x y*
using *separate_compact_closed* [*of path_image g* −*S*] *assms* **by** *force*
show *?thesis*
using *e* [*unfolded dist_norm*] ‹*e* > 0›
by (*fastforce simp*: *path_image_def intro*!: *homotopic_paths_nearby_explicit*

assms exI)
qed

lemma *homotopic_nearby_loops*:
fixes *g h* :: *real* ⇒ *'a*::*euclidean_space*
assumes *path g open S path_image g* ⊆ *S pathfinish g* = *pathstart g*
shows ∃ *e*. *0* < *e* ∧
(∀ *h*. *path h* ∧ *pathfinish h* = *pathstart h* ∧
(∀ *t* ∈ {*0*..*1*}. *norm*(*h t* − *g t*) < *e*) ⟶ *homotopic_loops S g h*)
proof −
obtain *e* **where** *e* > *0* **and** *e*: ⋀*x y*. *x* ∈ *path_image g* ⟹ *y* ∈ − *S* ⟹ *e* ≤ *dist x y*
using *separate_compact_closed* [*of path_image g* −*S*] *assms* **by** *force*
show *?thesis*
using *e* [*unfolded dist_norm*] ‹*e* > *0*›
by (*fastforce simp*: *path_image_def intro*!: *homotopic_loops_nearby_explicit assms exI*)
qed

5.4.9 Homotopy and subpaths

lemma *homotopic_join_subpaths1*:
assumes *path g* **and** *pag*: *path_image g* ⊆ *S*
and *u*: *u* ∈ {*0*..*1*} **and** *v*: *v* ∈ {*0*..*1*} **and** *w*: *w* ∈ {*0*..*1*} *u* ≤ *v v* ≤ *w*
shows *homotopic_paths S* (*subpath u v g* +++ *subpath v w g*) (*subpath u w g*)
proof −
have *1*: *t* ∗ *2* ≤ *1* ⟹ *u* + *t* ∗ (*v* ∗ *2*) ≤ *v* + *t* ∗ (*u* ∗ *2*) **for** *t*
using *affine_ineq* ‹*u* ≤ *v*› **by** *fastforce*
have *2*: *t* ∗ *2* > *1* ⟹ *u* + (*2*∗*t* − *1*) ∗ *v* ≤ *v* + (*2*∗*t* − *1*) ∗ *w* **for** *t*
by (*metis add_mono_thms_linordered_semiring*(*1*) *diff_gt_0_iff_gt less_eq_real_def mult.commute mult_right_mono* ‹*u* ≤ *v*› ‹*v* ≤ *w*›)
have *t2*: ⋀*t*::*real*. *t*∗*2* = *1* ⟹ *t* = *1*/*2* **by** *auto*
have *homotopic_paths* (*path_image g*) (*subpath u v g* +++ *subpath v w g*) (*subpath u w g*)
proof (*cases w* = *u*)
case *True*
then show *?thesis*
by (*metis* ‹*path g*› *homotopic_paths_rinv path_image_subpath_subset path_subpath pathstart_subpath reversepath_subpath subpath_refl u v*)
next
case *False*
let *?f* = λ*t*. *if* *t* ≤ *1*/*2* *then inverse*((*w* − *u*)) ∗*R* (*2* ∗ (*v* − *u*)) ∗*R* *t*
else inverse((*w* − *u*)) ∗*R* ((*v* − *u*) + (*w* − *v*) ∗*R* (*2* ∗*R* *t* − *1*))
show *?thesis*
proof (*rule homotopic_paths_sym* [*OF homotopic_paths_reparametrize* [**where** *f* = *?f*]])
show *path* (*subpath u w g*)
using *assms*(*1*) *path_subpath u w*(*1*) **by** *blast*

show *path_image* (*subpath u w g*) ⊆ *path_image g*
by (*meson path_image_subpath_subset u w*(*1*))
show *continuous_on* {*0..1*} *?f*
unfolding *split_01*
by (*rule continuous_on_cases continuous_intros* | *force simp*: *pathfinish_def joinpaths_def dest*!: *t2*)+
show *?f* ∈ {*0..1*} → {*0..1*}
using *False assms*
by (*force simp*: *field_simps not_le mult_left_mono affine_ineq dest*!: *1 2*)
show (*subpath u v g* +++ *subpath v w g*) *t* = *subpath u w g* (*?f t*) **if** *t* ∈ {*0..1*} **for** *t*
using *assms*
unfolding *joinpaths_def subpath_def* **by** (*auto simp*: *divide_simps add.commute mult.commute mult.left_commute*)
qed (*use False* **in** *auto*)
qed
then show *?thesis*
by (*rule homotopic_paths_subset* [*OF _ pag*])
qed

lemma *homotopic_join_subpaths2*:
assumes *homotopic_paths S* (*subpath u v g* +++ *subpath v w g*) (*subpath u w g*)
shows *homotopic_paths S* (*subpath w v g* +++ *subpath v u g*) (*subpath w u g*)
by (*metis assms homotopic_paths_reversepath_D pathfinish_subpath pathstart_subpath reversepath_joinpaths reversepath_subpath*)

lemma *homotopic_join_subpaths3*:
assumes *hom*: *homotopic_paths S* (*subpath u v g* +++ *subpath v w g*) (*subpath u w g*)
and *path g* **and** *pag*: *path_image g* ⊆ *S*
and *u*: *u* ∈ {*0..1*} **and** *v*: *v* ∈ {*0..1*} **and** *w*: *w* ∈ {*0..1*}
shows *homotopic_paths S* (*subpath v w g* +++ *subpath w u g*) (*subpath v u g*)
proof −
obtain *wvg*: *path* (*subpath w v g*) *path_image* (*subpath w v g*) ⊆ *S*
and *wug*: *path* (*subpath w u g*) *path_image* (*subpath w u g*) ⊆ *S*
and *vug*: *path* (*subpath v u g*) *path_image* (*subpath v u g*) ⊆ *S*
by (*meson* ‹*path g*› *pag path_image_subpath_subset path_subpath subset_trans u v w*)
have *homotopic_paths S* (*subpath u w g* +++ *subpath w v g*)
((*subpath u v g* +++ *subpath v w g*) +++ *subpath w v g*)
by (*simp add*: *hom homotopic_paths_join homotopic_paths_sym wvg*)
also have *homotopic_paths S* ... (*subpath u v g* +++ *subpath v w g* +++ *subpath w v g*)
using *wvg vug* ‹*path g*›
by (*metis homotopic_paths_assoc homotopic_paths_sym path_image_subpath_commute path_subpath*
pathfinish_subpath pathstart_subpath u v w)
also have *homotopic_paths S* ... (*subpath u v g* +++ *linepath* (*pathfinish* (*subpath u v g*)) (*pathfinish* (*subpath u v g*)))

using *wvg vug ‹path g›*
by (*metis homotopic_paths_join homotopic_paths_linv homotopic_paths_refl path_image_subpath_commute*
path_subpath pathfinish_subpath pathstart_join pathstart_subpath reversepath_subpath u v)
also have *homotopic_paths S ... (subpath u v g)*
using *vug ‹path g›* **by** (*metis homotopic_paths_rid path_image_subpath_commute path_subpath u v*)
finally have *homotopic_paths S (subpath u w g +++ subpath w v g) (subpath u v g)* **.**
then show *?thesis*
using *homotopic_join_subpaths2* **by** *blast*
qed

proposition *homotopic_join_subpaths*:
⟦*path g*; *path_image g ⊆ S*; *u ∈ {0..1}*; *v ∈ {0..1}*; *w ∈ {0..1}*⟧
⟹ *homotopic_paths S (subpath u v g +++ subpath v w g) (subpath u w g)*
by (*smt* (*verit*, *del_insts*) *homotopic_join_subpaths1 homotopic_join_subpaths2 homotopic_join_subpaths3*)

Relating homotopy of trivial loops to path-connectedness.

lemma *path_component_imp_homotopic_points*:
assumes *path_component S a b*
shows *homotopic_loops S (linepath a a) (linepath b b)*
proof −
obtain *g* :: *real* ⇒ *'a* **where** *g*: *continuous_on {0..1} g g ∈ {0..1} → S g 0 = a g 1 = b*
using *assms* **by** (*auto simp*: *path_defs*)
then have *continuous_on* ({*0..1*} × {*0..1*}) (*g ∘ fst*)
by (*fastforce intro*!: *continuous_intros*)+
with *g* **show** *?thesis*
by (*auto simp*: *homotopic_loops_def homotopic_with_def path_defs Pi_iff*)
qed

lemma *homotopic_loops_imp_path_component_value*:
⟦*homotopic_loops S p q*; *0 ≤ t*; *t ≤ 1*⟧ ⟹ *path_component S* (*p t*) (*q t*)
apply (*clarsimp simp*: *homotopic_loops_def homotopic_with_def path_defs*)
apply (*rule_tac x=h ∘* (*λu.* (*u*, *t*)) **in** *exI*)
apply (*fastforce elim*!: *continuous_on_subset intro*!: *continuous_intros*)
done

lemma *homotopic_points_eq_path_component*:
homotopic_loops S (*linepath a a*) (*linepath b b*) ⟷ *path_component S a b*
using *homotopic_loops_imp_path_component_value path_component_imp_homotopic_points*
by *fastforce*

lemma *path_connected_eq_homotopic_points*:
path_connected S ⟷
(∀ *a b*. *a ∈ S* ∧ *b ∈ S* ⟶ *homotopic_loops S* (*linepath a a*) (*linepath b b*))

by (*auto simp*: *path_connected_def path_component_def homotopic_points_eq_path_component*)

5.4.10 Simply connected sets

defined as "all loops are homotopic (as loops)

definition *simply_connected* **where**
simply_connected S ≡
∀ *p q*. *path p* ∧ *pathfinish p* = *pathstart p* ∧ *path_image p* ⊆ *S* ∧
path q ∧ *pathfinish q* = *pathstart q* ∧ *path_image q* ⊆ *S*
⟶ *homotopic_loops S p q*

lemma *simply_connected_empty* [*iff*]: *simply_connected* {}
by (*simp add*: *simply_connected_def*)

lemma *simply_connected_imp_path_connected*:
fixes *S* :: _::*real_normed_vector set*
shows *simply_connected S* ⟹ *path_connected S*
by (*simp add*: *simply_connected_def path_connected_eq_homotopic_points*)

lemma *simply_connected_imp_connected*:
fixes *S* :: _::*real_normed_vector set*
shows *simply_connected S* ⟹ *connected S*
by (*simp add*: *path_connected_imp_connected simply_connected_imp_path_connected*)

lemma *simply_connected_eq_contractible_loop_any*:
fixes *S* :: _::*real_normed_vector set*
shows *simply_connected S* ⟷
(∀ *p a*. *path p* ∧ *path_image p* ⊆ *S* ∧ *pathfinish p* = *pathstart p* ∧ *a* ∈ *S*
⟶ *homotopic_loops S p* (*linepath a a*))
(**is** *?lhs* = *?rhs*)
proof
assume *?rhs* **then show** *?lhs*
unfolding *simply_connected_def*
by (*metis pathfinish_in_path_image subsetD homotopic_loops_trans homotopic_loops_sym*)
qed (*force simp*: *simply_connected_def*)

lemma *simply_connected_eq_contractible_loop_some*:
fixes *S* :: _::*real_normed_vector set*
shows *simply_connected S* ⟷
path_connected S ∧
(∀ *p*. *path p* ∧ *path_image p* ⊆ *S* ∧ *pathfinish p* = *pathstart p*
⟶ (∃ *a*. *a* ∈ *S* ∧ *homotopic_loops S p* (*linepath a a*)))
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
using *simply_connected_eq_contractible_loop_any* **by** (*blast intro*: *simply_connected_imp_path_connected*)
next

assume *?rhs*
then show *?lhs*
by (*meson homotopic_loops_trans path_connected_eq_homotopic_points simply_connected_eq_contractible_loop_any*)
qed

lemma *simply_connected_eq_contractible_loop_all*:
fixes *S* :: _::*real_normed_vector set*
shows *simply_connected S* ⟷
S = {} ∨
(∃ *a* ∈ *S*. ∀ *p*. *path p* ∧ *path_image p* ⊆ *S* ∧ *pathfinish p* = *pathstart p*
⟶ *homotopic_loops S p* (*linepath a a*))
by (*meson ex_in_conv homotopic_loops_sym homotopic_loops_trans simply_connected_def simply_connected_eq_contractible_loop_any*)

lemma *simply_connected_eq_contractible_path*:
fixes *S* :: _::*real_normed_vector set*
shows *simply_connected S* ⟷
path_connected S ∧
(∀ *p*. *path p* ∧ *path_image p* ⊆ *S* ∧ *pathfinish p* = *pathstart p*
⟶ *homotopic_paths S p* (*linepath* (*pathstart p*) (*pathstart p*)))
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
unfolding *simply_connected_imp_path_connected*
by (*metis simply_connected_eq_contractible_loop_some homotopic_loops_imp_homotopic_paths_n*
next
assume *?rhs*
then show *?lhs*
using *homotopic_paths_imp_homotopic_loops simply_connected_eq_contractible_loop_some*
by *fastforce*
qed

lemma *simply_connected_eq_homotopic_paths*:
fixes *S* :: _::*real_normed_vector set*
shows *simply_connected S* ⟷
path_connected S ∧
(∀ *p q*. *path p* ∧ *path_image p* ⊆ *S* ∧
path q ∧ *path_image q* ⊆ *S* ∧
pathstart q = *pathstart p* ∧ *pathfinish q* = *pathfinish p*
⟶ *homotopic_paths S p q*)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then have *pc*: *path_connected S*
and ∗: ⋀*p*. ⟦*path p*; *path_image p* ⊆ *S*;
pathfinish p = *pathstart p*⟧
⟹ *homotopic_paths S p* (*linepath* (*pathstart p*) (*pathstart p*))

```
    by (auto simp: simply_connected_eq_contractible_path)
  have homotopic_paths S p q
       if path p path_image p ⊆ S path q
          path_image q ⊆ S pathstart q = pathstart p
          pathfinish q = pathfinish p for p q
  proof −
    have homotopic_paths S p (p +++ reversepath q +++ q)
      using that
       by (smt (verit, best) homotopic_paths_join homotopic_paths_linv homo-
topic_paths_rid homotopic_paths_sym
          homotopic_paths_trans pathstart_linepath)
    also have homotopic_paths S ... ((p +++ reversepath q) +++ q)
      by (simp add: that homotopic_paths_assoc)
    also have homotopic_paths S ... (linepath (pathstart q) (pathstart q) +++ q)
      using ∗ [of p +++ reversepath q] that
     by (simp add: homotopic_paths_assoc homotopic_paths_join path_image_join)
    also have homotopic_paths S ... q
      using that homotopic_paths_lid by blast
    finally show ?thesis .
  qed
  then show ?rhs
    by (blast intro: pc ∗)
next
  assume ?rhs
  then show ?lhs
    by (force simp: simply_connected_eq_contractible_path)
qed

proposition simply_connected_Times:
  fixes S :: 'a::real_normed_vector set and T :: 'b::real_normed_vector set
  assumes S: simply_connected S and T: simply_connected T
    shows simply_connected(S × T)
proof −
  have homotopic_loops (S × T) p (linepath (a, b) (a, b))
       if path p path_image p ⊆ S × T p 1 = p 0 a ∈ S b ∈ T
       for p a b
  proof −
    have path (fst ∘ p)
      by (simp add: continuous_on_fst Path_Connected.path_continuous_image
[OF ‹path p›])
    moreover have path_image (fst ∘ p) ⊆ S
      using that by (force simp: path_image_def)
    ultimately have p1: homotopic_loops S (fst ∘ p) (linepath a a)
      using S that
       by (simp add: simply_connected_eq_contractible_loop_any pathfinish_def
pathstart_def)
    have path (snd ∘ p)
      by (simp add: continuous_on_snd Path_Connected.path_continuous_image
[OF ‹path p›])
```

moreover have *path_image* (*snd* ∘ *p*) ⊆ *T*
using *that* **by** (*force simp*: *path_image_def*)
ultimately have *p2*: *homotopic_loops T* (*snd* ∘ *p*) (*linepath b b*)
using *T that*
by (*simp add*: *simply_connected_eq_contractible_loop_any pathfinish_def pathstart_def*)
show *?thesis*
using *p1 p2* **unfolding** *homotopic_loops*
apply *clarify*
subgoal for *h k*
by (*rule_tac x*=λ*z*. (*h z*, *k z*) **in** *exI*) (*force intro*: *continuous_intros simp*: *path_defs*)
done
qed
with *assms* **show** *?thesis*
by (*simp add*: *simply_connected_eq_contractible_loop_any pathfinish_def pathstart_def*)
qed

5.4.11 Contractible sets

definition *contractible* **where**
contractible S ≡ ∃ *a*. *homotopic_with_canon* (λ*x*. *True*) *S S id* (λ*x*. *a*)

proposition *contractible_imp_simply_connected*:
fixes *S* :: _::*real_normed_vector set*
assumes *contractible S* **shows** *simply_connected S*
proof (*cases S* = {})
case *True* **then show** *?thesis* **by** *force*
next
case *False*
obtain *a* **where** *a*: *homotopic_with_canon* (λ*x*. *True*) *S S id* (λ*x*. *a*)
using *assms* **by** (*force simp*: *contractible_def*)
then have *a* ∈ *S*
using *False homotopic_with_imp_funspace2* **by** *fastforce*
have ∀ *p*. *path p* ∧
path_image p ⊆ *S* ∧ *pathfinish p* = *pathstart p* ⟶
homotopic_loops S p (*linepath a a*)
using *a* **apply** (*clarsimp simp*: *homotopic_loops_def homotopic_with_def path_defs*)
apply (*rule_tac x*=(*h* ∘ (λ*y*. (*fst y*, (*p* ∘ *snd*) *y*))) **in** *exI*)
apply (*intro conjI continuous_on_compose continuous_intros*; *force elim*: *continuous_on_subset*)
done
with ‹*a* ∈ *S*› **show** *?thesis*
by (*auto simp*: *simply_connected_eq_contractible_loop_all False*)
qed

corollary *contractible_imp_connected*:

fixes S :: _::*real_normed_vector set*
shows *contractible* $S \Longrightarrow$ *connected* S
by (*simp add*: *contractible_imp_simply_connected simply_connected_imp_connected*)

lemma *contractible_imp_path_connected*:
fixes S :: _::*real_normed_vector set*
shows *contractible* $S \Longrightarrow$ *path_connected* S
by (*simp add*: *contractible_imp_simply_connected simply_connected_imp_path_connected*)

lemma *nullhomotopic_through_contractible*:
fixes S :: _::*topological_space set*
assumes f: *continuous_on* S f $f \in S \to T$
and g: *continuous_on* T g $g \in T \to U$
and T: *contractible* T
obtains c **where** *homotopic_with_canon* $(\lambda h.\ True)$ S U $(g \circ f)$ $(\lambda x.\ c)$
proof −
obtain b **where** b: *homotopic_with_canon* $(\lambda x.\ True)$ T T *id* $(\lambda x.\ b)$
using *assms* **by** (*force simp*: *contractible_def*)
have *homotopic_with_canon* $(\lambda f.\ True)$ T U $(g \circ id)$ $(g \circ (\lambda x.\ b))$
by (*metis b continuous_map_subtopology_eu g homotopic_with_compose_continuous_map_left image_subset_iff_funcset*)
then have *homotopic_with_canon* $(\lambda f.\ True)$ S U $(g \circ id \circ f)$ $(g \circ (\lambda x.\ b) \circ f)$
by (*simp add*: *f homotopic_with_compose_continuous_map_right image_subset_iff_funcset*)
then show *?thesis*
by (*simp add*: *comp_def that*)
qed

lemma *nullhomotopic_into_contractible*:
assumes f: *continuous_on* S f $f \in S \to T$
and T: *contractible* T
obtains c **where** *homotopic_with_canon* $(\lambda h.\ True)$ S T f $(\lambda x.\ c)$
by (*rule nullhomotopic_through_contractible* [*OF f*, *of id T*]) (*use assms* **in** *auto*)

lemma *nullhomotopic_from_contractible*:
assumes f: *continuous_on* S f $f \in S \to T$
and S: *contractible* S
obtains c **where** *homotopic_with_canon* $(\lambda h.\ True)$ S T f $(\lambda x.\ c)$
by (*auto simp*: *comp_def intro*: *nullhomotopic_through_contractible* [*OF continuous_on_id _ f S*])

lemma *homotopic_through_contractible*:
fixes S :: _::*real_normed_vector set*
assumes *continuous_on* S *f1* *f1* $\in S \to T$
continuous_on T *g1* *g1* $\in T \to U$
continuous_on S *f2* *f2* $\in S \to T$
continuous_on T *g2* *g2* $\in T \to U$
contractible T *path_connected* U
shows *homotopic_with_canon* $(\lambda h.\ True)$ S U $(g1 \circ f1)$ $(g2 \circ f2)$
proof −

obtain *c1* **where** *c1*: *homotopic_with_canon* (λ*h*. *True*) *S U* (*g1* ∘ *f1*) (λ*x*. *c1*)
by (*rule nullhomotopic_through_contractible* [*of S f1 T g1 U*]) (*use assms* **in** *auto*)
obtain *c2* **where** *c2*: *homotopic_with_canon* (λ*h*. *True*) *S U* (*g2* ∘ *f2*) (λ*x*. *c2*)
by (*rule nullhomotopic_through_contractible* [*of S f2 T g2 U*]) (*use assms* **in** *auto*)
have *S* = {} ∨ (∃ *t*. *path_connected t* ∧ *t* ⊆ *U* ∧ *c2* ∈ *t* ∧ *c1* ∈ *t*)
proof (*cases S* = {})
case *True* **then show** *?thesis* **by** *force*
next
case *False*
with *c1 c2* **have** *c1* ∈ *U c2* ∈ *U*
using *homotopic_with_imp_continuous_maps*
by (*metis PiE equals0I homotopic_with_imp_funspace2*)+
with ‹*path_connected U*› **show** *?thesis* **by** *blast*
qed
then have *homotopic_with_canon* (λ*h*. *True*) *S U* (λ*x*. *c2*) (λ*x*. *c1*)
by (*auto simp*: *path_component homotopic_constant_maps*)
then show *?thesis*
using *c1 c2 homotopic_with_symD homotopic_with_trans* **by** *blast*
qed

lemma *homotopic_into_contractible*:
fixes *S* :: *'a*::*real_normed_vector set* **and** *T*:: *'b*::*real_normed_vector set*
assumes *f*: *continuous_on S f f* ∈ *S* → *T*
and *g*: *continuous_on S g g* ∈ *S* → *T*
and *T*: *contractible T*
shows *homotopic_with_canon* (λ*h*. *True*) *S T f g*
using *homotopic_through_contractible* [*of S f T id T g id*]
by (*simp add*: *assms contractible_imp_path_connected*)

lemma *homotopic_from_contractible*:
fixes *S* :: *'a*::*real_normed_vector set* **and** *T*:: *'b*::*real_normed_vector set*
assumes *f*: *continuous_on S f f* ∈ *S* → *T*
and *g*: *continuous_on S g g* ∈ *S* → *T*
and *contractible S path_connected T*
shows *homotopic_with_canon* (λ*h*. *True*) *S T f g*
using *homotopic_through_contractible* [*of S id S f T id g*]
by (*simp add*: *assms contractible_imp_path_connected*)

5.4.12 Starlike sets

definition *starlike S* ⟷ (∃ *a*∈*S*. ∀ *x*∈*S*. *closed_segment a x* ⊆ *S*)

lemma *starlike_UNIV* [*simp*]: *starlike UNIV*
by (*simp add*: *starlike_def*)

lemma *convex_imp_starlike*:
convex S ⟹ *S* ≠ {} ⟹ *starlike S*

unfolding *convex_contains_segment starlike_def* **by** *auto*

lemma *starlike_convex_tweak_boundary_points*:
fixes S :: *'a::euclidean_space set*
assumes *convex* S $S \neq \{\}$ **and** ST: *rel_interior* $S \subseteq T$ **and** TS: $T \subseteq$ *closure* S
shows *starlike* T
proof −
have *rel_interior* $S \neq \{\}$
by (*simp add*: *assms rel_interior_eq_empty*)
with ST **obtain** a **where** a: $a \in$ *rel_interior* S **and** $a \in T$ **by** *blast*
have $\bigwedge x.\ x \in T \Longrightarrow$ *open_segment* $a\ x \subseteq$ *rel_interior* S
by (*rule rel_interior_closure_convex_segment* [*OF* ‹*convex* S› a]) (*use assms* **in** *auto*)
then have $\forall\, x{\in}T.\ a \in T \wedge$ *open_segment* $a\ x \subseteq T$
using ST **by** (*blast intro*: a ‹$a \in T$› *rel_interior_closure_convex_segment* [*OF* ‹*convex* S› a])
then show *?thesis*
unfolding *starlike_def* **using** *bexI* [*OF* _ ‹$a \in T$›]
by (*simp add*: *closed_segment_eq_open*)
qed

lemma *starlike_imp_contractible_gen*:
fixes S :: *'a::real_normed_vector set*
assumes S: *starlike* S
and P: $\bigwedge a\ T.\ [\![a \in S;\ 0 \le T;\ T \le 1]\!] \Longrightarrow P(\lambda x.\ (1 - T) *_R x + T *_R a)$
obtains a **where** *homotopic_with_canon* $P\ S\ S\ (\lambda x.\ x)\ (\lambda x.\ a)$
proof −
obtain a **where** $a \in S$ **and** a: $\bigwedge x.\ x \in S \Longrightarrow$ *closed_segment* $a\ x \subseteq S$
using S **by** (*auto simp*: *starlike_def*)
have $\bigwedge t\ b.\ 0 \le t \wedge t \le 1 \Longrightarrow$
$\exists\, u.\ (1 - t) *_R b + t *_R a = (1 - u) *_R a + u *_R b \wedge 0 \le u \wedge u \le 1$
by (*metis add_diff_cancel_right' diff_ge_0_iff_ge le_add_diff_inverse pth_c*(*1*))
then have $(\lambda y.\ (1 - fst\ y) *_R snd\ y + fst\ y *_R a)$ ‘ $(\{0..1\} \times S) \subseteq S$
using a [*unfolded closed_segment_def*] **by** *force*
then have *homotopic_with_canon* $P\ S\ S\ (\lambda x.\ x)\ (\lambda x.\ a)$
using ‹$a \in S$›
unfolding *homotopic_with_def*
apply (*rule_tac* $x{=}\lambda y.\ (1 - (fst\ y)) *_R snd\ y + (fst\ y) *_R a$ **in** *exI*)
apply (*force simp*: P *intro*: *continuous_intros*)
done
then show *?thesis*
using *that* **by** *blast*
qed

lemma *starlike_imp_contractible*:
fixes S :: *'a::real_normed_vector set*
shows *starlike* $S \Longrightarrow$ *contractible* S
using *starlike_imp_contractible_gen contractible_def* **by** (*fastforce simp*: *id_def*)

lemma *contractible_UNIV* [*simp*]: *contractible* (*UNIV* :: *'a*::*real_normed_vector set*)
by (*simp add*: *starlike_imp_contractible*)

lemma *starlike_imp_simply_connected*:
fixes *S* :: *'a*::*real_normed_vector set*
shows *starlike S* ⟹ *simply_connected S*
by (*simp add*: *contractible_imp_simply_connected starlike_imp_contractible*)

lemma *convex_imp_simply_connected*:
fixes *S* :: *'a*::*real_normed_vector set*
shows *convex S* ⟹ *simply_connected S*
using *convex_imp_starlike starlike_imp_simply_connected* **by** *blast*

lemma *starlike_imp_path_connected*:
fixes *S* :: *'a*::*real_normed_vector set*
shows *starlike S* ⟹ *path_connected S*
by (*simp add*: *simply_connected_imp_path_connected starlike_imp_simply_connected*)

lemma *starlike_imp_connected*:
fixes *S* :: *'a*::*real_normed_vector set*
shows *starlike S* ⟹ *connected S*
by (*simp add*: *path_connected_imp_connected starlike_imp_path_connected*)

lemma *is_interval_simply_connected_1*:
fixes *S* :: *real set*
shows *is_interval S* ⟷ *simply_connected S*
by (*meson convex_imp_simply_connected is_interval_connected_1 is_interval_convex_1 simply_connected_imp_connected*)

lemma *contractible_empty* [*simp*]: *contractible* {}
by (*simp add*: *contractible_def homotopic_on_emptyI*)

lemma *contractible_convex_tweak_boundary_points*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *convex S* **and** *TS*: *rel_interior S* ⊆ *T T* ⊆ *closure S*
shows *contractible T*
by (*metis assms closure_eq_empty contractible_empty empty_subsetI starlike_convex_tweak_boundary_points starlike_imp_contractible subset_antisym*)

lemma *convex_imp_contractible*:
fixes *S* :: *'a*::*real_normed_vector set*
shows *convex S* ⟹ *contractible S*
using *contractible_empty convex_imp_starlike starlike_imp_contractible* **by** *blast*

lemma *contractible_sing* [*simp*]:
fixes *a* :: *'a*::*real_normed_vector*
shows *contractible* {*a*}
by (*rule convex_imp_contractible* [*OF convex_singleton*])

lemma *is_interval_contractible_1*:
fixes S :: *real set*
shows *is_interval* $S \longleftrightarrow$ *contractible* S
using *contractible_imp_simply_connected convex_imp_contractible is_interval_convex_1*
is_interval_simply_connected_1 **by** *auto*

lemma *contractible_Times*:
fixes S :: *'a::euclidean_space set* **and** T :: *'b::euclidean_space set*
assumes S: *contractible* S **and** T: *contractible* T
shows *contractible* $(S \times T)$
proof −
obtain *a h* **where** *conth*: *continuous_on* $(\{0..1\} \times S)$ h
and *hsub*: $h \in (\{0..1\} \times S) \to S$
and [*simp*]: $\bigwedge x.\ x \in S \Longrightarrow h\ (0,\ x) = x$
and [*simp*]: $\bigwedge x.\ x \in S \Longrightarrow h\ (1{::}real,\ x) = a$
using S **by** (*force simp*: *contractible_def homotopic_with*)
obtain *b k* **where** *contk*: *continuous_on* $(\{0..1\} \times T)$ k
and *ksub*: $k \in (\{0..1\} \times T) \to T$
and [*simp*]: $\bigwedge x.\ x \in T \Longrightarrow k\ (0,\ x) = x$
and [*simp*]: $\bigwedge x.\ x \in T \Longrightarrow k\ (1{::}real,\ x) = b$
using T **by** (*force simp*: *contractible_def homotopic_with*)
show *?thesis*
apply (*simp add*: *contractible_def homotopic_with*)
apply (*rule exI* [**where** $x{=}a$])
apply (*rule exI* [**where** $x{=}b$])
apply (*rule exI* [**where** $x = \lambda z.\ (h\ (fst\ z,\ fst(snd\ z)),\ k\ (fst\ z,\ snd(snd\ z)))$])
using *hsub ksub*
apply (*fastforce intro*!: *continuous_intros continuous_on_compose2* [*OF conth*] *continuous_on_compose2* [*OF contk*])
done
qed

5.4.13 Local versions of topological properties in general

definition *locally* :: $('a{::}topological\_space\ set \Rightarrow bool) \Rightarrow 'a\ set \Rightarrow bool$
where
locally $P\ S \equiv$
$\forall w\ x.\ openin\ (top\_of\_set\ S)\ w \wedge x \in w$
$\longrightarrow (\exists U\ V.\ openin\ (top\_of\_set\ S)\ U \wedge P\ V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq w)$

lemma *locallyI*:
assumes $\bigwedge w\ x.\ [\![openin\ (top\_of\_set\ S)\ w;\ x \in w]\!] \Longrightarrow \exists U\ V.\ openin\ (top\_of\_set\ S)\ U \wedge P\ V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq w$
shows *locally* $P\ S$
using *assms* **by** (*force simp*: *locally_def*)

lemma *locallyE*:
assumes *locally P S openin* (*top_of_set S*) *w x* ∈ *w*
obtains *U V* **where** *openin* (*top_of_set S*) *U P V x* ∈ *U U* ⊆ *V V* ⊆ *w*
using *assms* **unfolding** *locally_def* **by** *meson*

lemma *locally_mono*:
assumes *locally P S* ⋀*T*. *P T* ⟹ *Q T*
shows *locally Q S*
by (*metis assms locally_def*)

lemma *locally_open_subset*:
assumes *locally P S openin* (*top_of_set S*) *t*
shows *locally P t*
by (*smt* (*verit*, *ccfv_SIG*) *assms order.trans locally_def openin_imp_subset openin_subset_trans openin_trans*)

lemma *locally_diff_closed*:
⟦*locally P S*; *closedin* (*top_of_set S*) *t*⟧ ⟹ *locally P* (*S* − *t*)
using *locally_open_subset closedin_def* **by** *fastforce*

lemma *locally_empty* [*iff*]: *locally P* {}
by (*simp add*: *locally_def openin_subtopology*)

lemma *locally_singleton* [*iff*]:
fixes *a* :: *'a*::*metric_space*
shows *locally P* {*a*} ⟷ *P* {*a*}
proof −
have ∀ *x*::*real*. ¬ *0* < *x* ⟹ *P* {*a*}
using *zero_less_one* **by** *blast*
then show *?thesis*
unfolding *locally_def*
by (*auto simp*: *openin_euclidean_subtopology_iff subset_singleton_iff conj_disj_distribR*)
qed

lemma *locally_iff*:
locally P S ⟷
(∀ *T x*. *open T* ∧ *x* ∈ *S* ∩ *T* ⟶ (∃ *U*. *open U* ∧ (∃ *V*. *P V* ∧ *x* ∈ *S* ∩ *U* ∧ *S* ∩ *U* ⊆ *V* ∧ *V* ⊆ *S* ∩ *T*)))
by (*smt* (*verit*) *locally_def openin_open*)

lemma *locally_Int*:
assumes *S*: *locally P S* **and** *T*: *locally P T*
and *P*: ⋀*S T*. *P S* ∧ *P T* ⟹ *P*(*S* ∩ *T*)
shows *locally P* (*S* ∩ *T*)
unfolding *locally_iff*
proof *clarify*
fix *A x*
assume *open A x* ∈ *A x* ∈ *S x* ∈ *T*
then obtain *U1 V1 U2 V2*

where *open U1 P V1 x* ∈ *S* ∩ *U1 S* ∩ *U1* ⊆ *V1* ∧ *V1* ⊆ *S* ∩ *A*
open U2 P V2 x ∈ *T* ∩ *U2 T* ∩ *U2* ⊆ *V2* ∧ *V2* ⊆ *T* ∩ *A*
using *S T* **unfolding** *locally_iff* **by** (*meson IntI*)
then have *S* ∩ *T* ∩ (*U1* ∩ *U2*) ⊆ *V1* ∩ *V2 V1* ∩ *V2* ⊆ *S* ∩ *T* ∩ *A x* ∈ *S* ∩ *T* ∩ (*U1* ∩ *U2*)
by *blast*+
moreover have *P* (*V1* ∩ *V2*)
by (*simp add*: *P* ‹*P V1*› ‹*P V2*›)
ultimately show ∃ *U*. *open U* ∧ (∃ *V*. *P V* ∧ *x* ∈ *S* ∩ *T* ∩ *U* ∧ *S* ∩ *T* ∩ *U* ⊆ *V* ∧ *V* ⊆ *S* ∩ *T* ∩ *A*)
using ‹*open U1*› ‹*open U2*› **by** *blast*
qed

lemma *locally_Times*:
fixes *S* :: ('*a*::*metric_space*) *set* **and** *T* :: ('*b*::*metric_space*) *set*
assumes *PS*: *locally P S* **and** *QT*: *locally Q T* **and** *R*: ⋀*S T*. *P S* ∧ *Q T* ⟹ *R*(*S* × *T*)
shows *locally R* (*S* × *T*)
unfolding *locally_def*
proof (*clarify*)
fix *W x y*
assume *W*: *openin* (*top_of_set* (*S* × *T*)) *W* **and** *xy*: (*x*, *y*) ∈ *W*
then obtain *U V* **where** *openin* (*top_of_set S*) *U x* ∈ *U*
openin (*top_of_set T*) *V y* ∈ *V U* × *V* ⊆ *W*
using *Times_in_interior_subtopology* **by** *metis*
then obtain *U1 U2 V1 V2*
where *opeS*: *openin* (*top_of_set S*) *U1* ∧ *P U2* ∧ *x* ∈ *U1* ∧ *U1* ⊆ *U2* ∧ *U2* ⊆ *U*
and *opeT*: *openin* (*top_of_set T*) *V1* ∧ *Q V2* ∧ *y* ∈ *V1* ∧ *V1* ⊆ *V2* ∧ *V2* ⊆ *V*
by (*meson PS QT locallyE*)
then have *openin* (*top_of_set* (*S* × *T*)) (*U1* × *V1*)
by (*simp add*: *openin_Times*)
moreover have *R* (*U2* × *V2*)
by (*simp add*: *R opeS opeT*)
moreover have *U1* × *V1* ⊆ *U2* × *V2* ∧ *U2* × *V2* ⊆ *W*
using *opeS opeT* ‹*U* × *V* ⊆ *W*› **by** *auto*
ultimately show ∃ *U V*. *openin* (*top_of_set* (*S* × *T*)) *U* ∧ *R V* ∧ (*x*,*y*) ∈ *U* ∧ *U* ⊆ *V* ∧ *V* ⊆ *W*
using *opeS opeT* **by** *auto*
qed

proposition *homeomorphism_locally_imp*:
fixes *S* :: '*a*::*metric_space set* **and** *T* :: '*b*::*t2_space set*
assumes *S*: *locally P S* **and** *hom*: *homeomorphism S T f g*
and *Q*: ⋀*S S*'. ⟦*P S*; *homeomorphism S S*' *f g*⟧ ⟹ *Q S*'
shows *locally Q T*

proof (*clarsimp simp*: *locally_def*)
fix *W y*
assume $y \in W$ **and** *openin* (*top_of_set T*) *W*
then obtain *A* **where** *T*: *open A* $W = T \cap A$
by (*force simp*: *openin_open*)
then have $W \subseteq T$ **by** *auto*
have *f*: $\bigwedge x.\ x \in S \Longrightarrow g(f\ x) = x$ $f\ `\ S = T$ *continuous_on S f*
and *g*: $\bigwedge y.\ y \in T \Longrightarrow f(g\ y) = y$ $g\ `\ T = S$ *continuous_on T g*
using *hom* **by** (*auto simp*: *homeomorphism_def*)
have *gw*: $g\ `\ W = S \cap f\ -`\ W$
using ‹$W \subseteq T$› *g* **by** *force*
have *openin* (*top_of_set S*) ($g\ `\ W$)
using ‹*openin* (*top_of_set T*) *W*› *continuous_on_open f gw* **by** *auto*
then obtain *U V*
where *osu*: *openin* (*top_of_set S*) *U* **and** *uv*: *P V g y* $\in U$ $U \subseteq V$ $V \subseteq g\ `\ W$
by (*metis S* ‹$y \in W$› *image_eqI locallyE*)
have $V \subseteq S$ **using** *uv* **by** (*simp add*: *gw*)
have *fv*: $f\ `\ V = T \cap \{x.\ g\ x \in V\}$
using ‹$f\ `\ S = T$› *f* ‹$V \subseteq S$› **by** *auto*
have *contvf*: *continuous_on V f*
using ‹$V \subseteq S$› *continuous_on_subset f*(*3*) **by** *blast*
have *openin* (*top_of_set* ($g\ `\ T$)) *U*
using ‹$g\ `\ T = S$› **by** (*simp add*: *osu*)
then have *openin* (*top_of_set T*) ($T \cap g\ -`\ U$)
using ‹*continuous_on T g*› *continuous_on_open* [*THEN iffD1*] **by** *blast*
moreover have $\exists V.\ Q\ V \wedge y \in (T \cap g\ -`\ U) \wedge (T \cap g\ -`\ U) \subseteq V \wedge V \subseteq W$
proof (*intro exI conjI*)
show $f\ `\ V \subseteq W$
using *uv* **using** *Int_lower2 gw image_subsetI mem_Collect_eq subset_iff* **by** *auto*
then have *contvg*: *continuous_on* ($f\ `\ V$) *g*
using ‹$W \subseteq T$› *continuous_on_subset* [*OF g*(*3*)] **by** *blast*
have $V \subseteq g\ `\ f\ `\ V$
by (*metis* ‹$V \subseteq S$› *hom homeomorphism_def homeomorphism_of_subsets order_refl*)
then have *homv*: *homeomorphism V* ($f\ `\ V$) *f g*
using ‹$V \subseteq S$› *f* **by** (*auto simp*: *homeomorphism_def contvf contvg*)
show *Q* ($f\ `\ V$)
using *Q homv* ‹*P V*› **by** *blast*
show $y \in T \cap g\ -`\ U$
using *T*(*2*) ‹$y \in W$› ‹$g\ y \in U$› **by** *blast*
show $T \cap g\ -`\ U \subseteq f\ `\ V$
using *g*(*1*) *image_iff uv*(*3*) **by** *fastforce*
qed
ultimately show $\exists U.$ *openin* (*top_of_set T*) $U \wedge (\exists v.\ Q\ v \wedge y \in U \wedge U \subseteq v \wedge v \subseteq W)$
by *meson*
qed

lemma *homeomorphism_locally*:
fixes *f*:: *'a*::*metric_space* ⇒ *'b*::*metric_space*
assumes *homeomorphism S T f g*
and ⋀*S T*. *homeomorphism S T f g* ⟹ (*P S* ⟷ *Q T*)
shows *locally P S* ⟷ *locally Q T*
by (*smt* (*verit*) *assms homeomorphism_locally_imp homeomorphism_symD*)

lemma *homeomorphic_locally*:
fixes *S*:: *'a*::*metric_space set* **and** *T*:: *'b*::*metric_space set*
assumes *hom*: *S homeomorphic T*
and *iff*: ⋀*X Y*. *X homeomorphic Y* ⟹ (*P X* ⟷ *Q Y*)
shows *locally P S* ⟷ *locally Q T*
by (*smt* (*verit*, *ccfv_SIG*) *hom homeomorphic_def homeomorphism_locally homeomorphism_locally_imp iff*)

lemma *homeomorphic_local_compactness*:
fixes *S*:: *'a*::*metric_space set* **and** *T*:: *'b*::*metric_space set*
shows *S homeomorphic T* ⟹ *locally compact S* ⟷ *locally compact T*
by (*simp add*: *homeomorphic_compactness homeomorphic_locally*)

lemma *locally_translation*:
fixes *P* :: *'a* :: *real_normed_vector set* ⇒ *bool*
shows (⋀*S*. *P* ((+) *a* ‘ *S*) = *P S*) ⟹ *locally P* ((+) *a* ‘ *S*) = *locally P S*
using *homeomorphism_locally* [*OF homeomorphism_translation*]
by (*metis* (*full_types*) *homeomorphism_image2*)

lemma *locally_injective_linear_image*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *f*: *linear f inj f* **and** *iff*: ⋀*S*. *P* (*f* ‘ *S*) ⟷ *Q S*
shows *locally P* (*f* ‘ *S*) ⟷ *locally Q S*
by (*smt* (*verit*) *f homeomorphism_image2 homeomorphism_locally iff linear_homeomorphism_image*)

lemma *locally_open_map_image*:
fixes *f* :: *'a*::*real_normed_vector* ⇒ *'b*::*real_normed_vector*
assumes *P*: *locally P S*
and *f*: *continuous_on S f*
and *oo*: ⋀*T*. *openin* (*top_of_set S*) *T* ⟹ *openin* (*top_of_set* (*f* ‘ *S*)) (*f* ‘ *T*)
and *Q*: ⋀*T*. ⟦*T* ⊆ *S*; *P T*⟧ ⟹ *Q*(*f* ‘ *T*)
shows *locally Q* (*f* ‘ *S*)
proof (*clarsimp simp*: *locally_def*)
fix *W y*
assume *oiw*: *openin* (*top_of_set* (*f* ‘ *S*)) *W* **and** *y* ∈ *W*
then have *W* ⊆ *f* ‘ *S* **by** (*simp add*: *openin_euclidean_subtopology_iff*)
have *oivf*: *openin* (*top_of_set S*) (*S* ∩ *f* −‘ *W*)
by (*rule continuous_on_open* [*THEN iffD1*, *rule_format*, *OF f oiw*])
then obtain *x* **where** *x* ∈ *S f x* = *y*
using ‹*W* ⊆ *f* ‘ *S*› ‹*y* ∈ *W*› **by** *blast*
then obtain *U V*

where *openin* (*top_of_set S*) *U P V x* ∈ *U U* ⊆ *V V* ⊆ *S* ∩ *f* −‘ *W*
by (*metis IntI P* ‹*y* ∈ *W*› *locallyE oivf vimageI*)
then have *openin* (*top_of_set* (*f* ‘ *S*)) (*f* ‘ *U*)
by (*simp add*: *oo*)
then show ∃ *X*. *openin* (*top_of_set* (*f* ‘ *S*)) *X* ∧ (∃ *Y*. *Q Y* ∧ *y* ∈ *X* ∧ *X* ⊆ *Y* ∧ *Y* ⊆ *W*)
using *Q* ‹*P V*› ‹*U* ⊆ *V*› ‹*V* ⊆ *S* ∩ *f* −‘ *W*› ‹*f x* = *y*› ‹*x* ∈ *U*› **by** *blast*
qed

5.4.14 An induction principle for connected sets

proposition *connected_induction*:
assumes *connected S*
and *opD*: ⋀*T a*. ⟦*openin* (*top_of_set S*) *T*; *a* ∈ *T*⟧ ⟹ ∃ *z*. *z* ∈ *T* ∧ *P z*
and *opI*: ⋀*a*. *a* ∈ *S*
⟹ ∃ *T*. *openin* (*top_of_set S*) *T* ∧ *a* ∈ *T* ∧
(∀ *x* ∈ *T*. ∀ *y* ∈ *T*. *P x* ∧ *P y* ∧ *Q x* ⟶ *Q y*)
and *etc*: *a* ∈ *S b* ∈ *S P a P b Q a*
shows *Q b*
proof −
let *?A* = {*b*. ∃ *T*. *openin* (*top_of_set S*) *T* ∧ *b* ∈ *T* ∧ (∀ *x*∈*T*. *P x* ⟶ *Q x*)}
let *?B* = {*b*. ∃ *T*. *openin* (*top_of_set S*) *T* ∧ *b* ∈ *T* ∧ (∀ *x*∈*T*. *P x* ⟶ ¬ *Q x*)}
have *?A* ∩ *?B* = {}
by (*clarsimp simp*: *set_eq_iff*) (*metis* (*no_types*, *opaque_lifting*) *Int_iff opD openin_Int*)
moreover have *S* ⊆ *?A* ∪ *?B*
by *clarsimp* (*meson opI*)
moreover have *openin* (*top_of_set S*) *?A*
by (*subst openin_subopen*, *blast*)
moreover have *openin* (*top_of_set S*) *?B*
by (*subst openin_subopen*, *blast*)
ultimately have *?A* = {} ∨ *?B* = {}
by (*metis* (*no_types*, *lifting*) ‹*connected S*› *connected_openin*)
then show *?thesis*
by *clarsimp* (*meson opI etc*)
qed

lemma *connected_equivalence_relation_gen*:
assumes *connected S*
and *etc*: *a* ∈ *S b* ∈ *S P a P b*
and *trans*: ⋀*x y z*. ⟦*R x y*; *R y z*⟧ ⟹ *R x z*
and *opD*: ⋀*T a*. ⟦*openin* (*top_of_set S*) *T*; *a* ∈ *T*⟧ ⟹ ∃ *z*. *z* ∈ *T* ∧ *P z*
and *opI*: ⋀*a*. *a* ∈ *S*
⟹ ∃ *T*. *openin* (*top_of_set S*) *T* ∧ *a* ∈ *T* ∧
(∀ *x* ∈ *T*. ∀ *y* ∈ *T*. *P x* ∧ *P y* ⟶ *R x y*)
shows *R a b*
proof −
have ⋀*a b c*. ⟦*a* ∈ *S*; *P a*; *b* ∈ *S*; *c* ∈ *S*; *P b*; *P c*; *R a b*⟧ ⟹ *R a c*
apply (*rule connected_induction* [*OF* ‹*connected S*› *opD*], *simp_all*)

```
    by (meson trans opI)
  then show ?thesis by (metis etc opI)
qed

lemma connected_induction_simple:
  assumes connected S
      and etc: a ∈ S b ∈ S P a
      and opI: ⋀a. a ∈ S
            ⟹ ∃ T. openin (top_of_set S) T ∧ a ∈ T ∧
                  (∀ x ∈ T. ∀ y ∈ T. P x ⟶ P y)
    shows P b
  by (rule connected_induction [OF ‹connected S› _, where P = λx. True])
     (use opI etc in auto)

lemma connected_equivalence_relation:
  assumes connected S
      and etc: a ∈ S b ∈ S
      and sym: ⋀x y. ⟦R x y; x ∈ S; y ∈ S⟧ ⟹ R y x
      and trans: ⋀x y z. ⟦R x y; R y z; x ∈ S; y ∈ S; z ∈ S⟧ ⟹ R x z
      and opI: ⋀a. a ∈ S ⟹ ∃ T. openin (top_of_set S) T ∧ a ∈ T ∧ (∀ x ∈ T.
R a x)
    shows R a b
proof −
  have ⋀a b c. ⟦a ∈ S; b ∈ S; c ∈ S; R a b⟧ ⟹ R a c
     by (smt (verit, ccfv_threshold) connected_induction_simple [OF ‹connected
S›]
           assms openin_imp_subset subset_eq)
  then show ?thesis by (metis etc opI)
qed

lemma locally_constant_imp_constant:
  assumes connected S
      and opI: ⋀a. a ∈ S
            ⟹ ∃ T. openin (top_of_set S) T ∧ a ∈ T ∧ (∀ x ∈ T. f x = f a)
    shows f constant_on S
proof −
  have ⋀x y. x ∈ S ⟹ y ∈ S ⟹ f x = f y
    apply (rule connected_equivalence_relation [OF ‹connected S›], simp_all)
    by (metis opI)
  then show ?thesis
    by (metis constant_on_def)
qed

lemma locally_constant:
  assumes connected S
  shows locally (λU. f constant_on U) S ⟷ f constant_on S (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
```

by (*smt* (*verit*, *del_insts*) *assms constant_on_def locally_constant_imp_constant locally_def openin_subtopology_self subset_iff*)
next
assume *?rhs* **then show** *?lhs*
by (*metis constant_on_subset locallyI openin_imp_subset order_refl*)
qed

5.4.15 Basic properties of local compactness

proposition *locally_compact*:
fixes $S :: 'a :: metric\_space\ set$
shows
locally compact $S \longleftrightarrow$
$(\forall x \in S.\ \exists u\ v.\ x \in u \wedge u \subseteq v \wedge v \subseteq S \wedge$
openin (*top_of_set* S) $u \wedge$ *compact* v)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (*meson locallyE openin_subtopology_self*)
next
assume r [*rule_format*]: *?rhs*
have $*$: $\exists u\ v.$
openin (*top_of_set* S) $u \wedge$
compact $v \wedge x \in u \wedge u \subseteq v \wedge v \subseteq S \cap T$
if *open* T $x \in S$ $x \in T$ **for** x T
proof −
obtain U V **where** *uv*: $x \in U$ $U \subseteq V$ $V \subseteq S$ *compact* V *openin* (*top_of_set* S) U
using r [*OF* ‹$x \in S$›] **by** *auto*
obtain e **where** $e{>}0$ **and** e: *cball* x $e \subseteq T$
using *open_contains_cball* ‹*open* T› ‹$x \in T$› **by** *blast*
show *?thesis*
apply (*rule_tac* $x{=}(S \cap ball\ x\ e) \cap U$ **in** *exI*)
apply (*rule_tac* $x{=}cball\ x\ e \cap V$ **in** *exI*)
using *that* ‹$e > 0$› e *uv*
apply *auto*
done
qed
show *?lhs*
by (*rule locallyI*) (*metis* $*$ *Int_iff openin_open*)
qed

lemma *locally_compactE*:
fixes $S :: 'a :: metric\_space\ set$
assumes *locally compact* S
obtains $u\ v$ **where** $\bigwedge x.\ x \in S \Longrightarrow x \in u\ x \wedge u\ x \subseteq v\ x \wedge v\ x \subseteq S \wedge$
openin (*top_of_set* S) $(u\ x) \wedge$ *compact* $(v\ x)$
using *assms* **unfolding** *locally_compact* **by** *metis*

lemma *locally_compact_alt*:
fixes $S :: 'a :: heine\_borel\ set$
shows *locally compact* $S \longleftrightarrow$
$(\forall x \in S.\ \exists U.\ x \in U \wedge$
openin (*top_of_set* S) $U \wedge$ *compact*(*closure* U) $\wedge$ *closure* $U \subseteq S$)
by (*smt* (*verit*, *ccfv_threshold*) *bounded_subset closure_closed closure_mono closure_subset*
compact_closure compact_imp_closed order.trans locally_compact)

lemma *locally_compact_Int_cball*:
fixes $S :: 'a :: heine\_borel\ set$
shows *locally compact* $S \longleftrightarrow (\forall x \in S.\ \exists e.\ 0 < e \wedge closed(cball\ x\ e \cap S))$
(**is** *?lhs* = *?rhs*)
proof
assume *L*: *?lhs*
then have $\bigwedge x\ U\ V\ e.\ [\![U \subseteq V;\ V \subseteq S;\ compact\ V;\ 0 < e;\ cball\ x\ e \cap S \subseteq U]\!] \Longrightarrow closed\ (cball\ x\ e \cap S)$
by (*metis compact_Int compact_cball compact_imp_closed inf.absorb_iff2 inf.assoc inf.orderE*)
with *L* **show** *?rhs*
by (*meson locally_compactE openin_contains_cball*)
next
assume *R*: *?rhs*
show *?lhs* **unfolding** *locally_compact*
proof
fix x
assume $x \in S$
then obtain e **where** $e{>}0$ **and** *compact* (*cball* x $e \cap S$)
by (*metis Int_commute compact_Int_closed compact_cball inf.right_idem R*)
moreover have $\forall y{\in}ball\ x\ e \cap S.\ \exists \varepsilon{>}0.\ cball\ y\ \varepsilon \cap S \subseteq ball\ x\ e$
by (*meson Elementary_Metric_Spaces.open_ball IntD1 le_infI1 open_contains_cball_eq*)
moreover have *openin* (*top_of_set* S) (*ball* x $e \cap S$)
by (*simp add*: *inf_commute openin_open_Int*)
ultimately show $\exists U\ V.\ x \in U \wedge U \subseteq V \wedge V \subseteq S \wedge$ *openin* (*top_of_set* S) $U \wedge$ *compact* V
by (*metis Int_iff ‹0 < e› ‹x ∈ S› ball_subset_cball centre_in_ball inf_commute inf_le1 inf_mono order_refl*)
qed
qed

lemma *locally_compact_compact*:
fixes $S :: 'a :: heine\_borel\ set$
shows *locally compact* $S \longleftrightarrow$
$(\forall K.\ K \subseteq S \wedge compact\ K \longrightarrow (\exists U\ V.\ K \subseteq U \wedge U \subseteq V \wedge V \subseteq S \wedge$
openin (*top_of_set* S) $U \wedge$ *compact* V))
(**is** *?lhs* = *?rhs*)

proof
assume *?lhs*
then obtain u v **where**
uv: $\bigwedge x.\ x \in S \Longrightarrow x \in u\ x \wedge u\ x \subseteq v\ x \wedge v\ x \subseteq S \wedge$
openin (*top_of_set S*) ($u\ x$) $\wedge$ *compact* ($v\ x$)
by (*metis locally_compactE*)
have $*$: $\exists U\ V.\ K \subseteq U \wedge U \subseteq V \wedge V \subseteq S \wedge$ *openin* (*top_of_set S*) $U \wedge$ *compact* V
if $K \subseteq S$ *compact* K **for** K
proof −
have $\bigwedge C.\ (\forall\, c{\in}C.$ *openin* (*top_of_set K*) $c) \wedge K \subseteq \bigcup C \Longrightarrow$
$\exists\, D{\subseteq}C.$ *finite* $D \wedge K \subseteq \bigcup D$
using *that* **by** (*simp add*: *compact_eq_openin_cover*)
moreover have $\forall\, c \in (\lambda x.\ K \cap u\ x)$ ‘ $K.$ *openin* (*top_of_set K*) c
using *that* **by** *clarify* (*metis subsetD inf.absorb_iff2 openin_subset openin_subtopology_Int_subset topspace_euclidean_subtopology uv*)
moreover have $K \subseteq \bigcup((\lambda x.\ K \cap u\ x)$ ‘ $K)$
using *that* **by** *clarsimp* (*meson subsetCE uv*)
ultimately obtain D **where** $D \subseteq (\lambda x.\ K \cap u\ x)$ ‘ K *finite* D $K \subseteq \bigcup D$
by *metis*
then obtain T **where** T: $T \subseteq K$ *finite* T $K \subseteq \bigcup((\lambda x.\ K \cap u\ x)$ ‘ $T)$
by (*metis finite_subset_image*)
have *Tuv*: $\bigcup(u$ ‘ $T) \subseteq \bigcup(v$ ‘ $T)$
using T *that* **by** (*force dest*!: *uv*)
moreover
have *openin* (*top_of_set S*) ($\bigcup$ (u ‘ T))
using T *that uv* **by** *fastforce*
moreover
obtain *compact* ($\bigcup$ (v ‘ T)) $\bigcup$ (v ‘ $T) \subseteq S$
by (*metis T UN_subset_iff* ‹$K \subseteq S$› *compact_UN subset_iff uv*)
ultimately show *?thesis*
using T **by** *auto*
qed
show *?rhs*
by (*blast intro*: $*$)
next
assume *?rhs*
then show *?lhs*
apply (*clarsimp simp*: *locally_compact*)
apply (*drule_tac* x={x} **in** *spec*, *simp*)
done
qed

lemma *open_imp_locally_compact*:
fixes S :: *'a* :: *heine_borel set*
assumes *open S*
shows *locally compact S*
proof −
have $*$: $\exists U\ V.\ x \in U \wedge U \subseteq V \wedge V \subseteq S \wedge$ *openin* (*top_of_set S*) $U \wedge$ *compact*

```
V
        if x ∈ S for x
  proof −
    obtain e where e>0 and e: cball x e ⊆ S
      using open_contains_cball assms ‹x ∈ S› by blast
    have ope: openin (top_of_set S) (ball x e)
      by (meson e open_ball ball_subset_cball dual_order.trans open_subset)
    show ?thesis
      by (meson ‹0 < e› ball_subset_cball centre_in_ball compact_cball e ope)
  qed
  show ?thesis
    unfolding locally_compact by (blast intro: ∗)
qed

lemma closed_imp_locally_compact:
  fixes S :: 'a :: heine_borel set
  assumes closed S
    shows locally compact S
proof −
  have ∗: ∃ U V. x ∈ U ∧ U ⊆ V ∧ V ⊆ S ∧ openin (top_of_set S) U ∧ compact
V
        if x ∈ S for x
    apply (rule_tac x = S ∩ ball x 1 in exI, rule_tac x = S ∩ cball x 1 in exI)
    using ‹x ∈ S› assms by auto
  show ?thesis
    unfolding locally_compact by (blast intro: ∗)
qed

lemma locally_compact_UNIV: locally compact (UNIV :: 'a :: heine_borel set)
  by (simp add: closed_imp_locally_compact)

lemma locally_compact_Int:
  fixes S :: 'a :: t2_space set
  shows ⟦locally compact S; locally compact T⟧ ⟹ locally compact (S ∩ T)
  by (simp add: compact_Int locally_Int)

lemma locally_compact_closedin:
  fixes S :: 'a :: heine_borel set
  shows ⟦closedin (top_of_set S) T; locally compact S⟧
        ⟹ locally compact T
  unfolding closedin_closed
  using closed_imp_locally_compact locally_compact_Int by blast

lemma locally_compact_delete:
    fixes S :: 'a :: t1_space set
    shows locally compact S ⟹ locally compact (S − {a})
  by (auto simp: openin_delete locally_open_subset)

lemma locally_closed:
```

```
  fixes S :: 'a :: heine_borel set
  shows locally closed S ⟷ locally compact S
      (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    unfolding locally_def
    apply (elim all_forward imp_forward asm_rl exE)
    apply (rename_tac U V)
    apply (rule_tac x = U ∩ ball x 1 in exI)
    apply (rule_tac x = V ∩ cball x 1 in exI)
    apply (force intro: openin_trans)
    done
next
  assume ?rhs then show ?lhs
    using compact_eq_bounded_closed locally_mono by blast
qed

lemma locally_compact_openin_Un:
  fixes S :: 'a::euclidean_space set
  assumes LCS: locally compact S and LCT: locally compact T
      and opS: openin (top_of_set (S ∪ T)) S
      and opT: openin (top_of_set (S ∪ T)) T
    shows locally compact (S ∪ T)
proof −
  have ∃e>0. closed (cball x e ∩ (S ∪ T)) if x ∈ S for x
  proof −
    obtain e1 where e1 > 0 and e1: closed (cball x e1 ∩ S)
      using LCS ‹x ∈ S› unfolding locally_compact_Int_cball by blast
    moreover obtain e2 where e2 > 0 and e2: cball x e2 ∩ (S ∪ T) ⊆ S
      by (meson ‹x ∈ S› opS openin_contains_cball)
    then have cball x e2 ∩ (S ∪ T) = cball x e2 ∩ S
      by force
    ultimately have closed (cball x (min e1 e2) ∩ (S ∪ T))
     by (metis (no_types, lifting) cball_min_Int closed_Int closed_cball inf_assoc
inf_commute)
    then show ?thesis
      by (metis ‹0 < e1› ‹0 < e2› min_def)
  qed
  moreover have ∃e>0. closed (cball x e ∩ (S ∪ T)) if x ∈ T for x
  proof −
    obtain e1 where e1 > 0 and e1: closed (cball x e1 ∩ T)
      using LCT ‹x ∈ T› unfolding locally_compact_Int_cball by blast
    moreover obtain e2 where e2 > 0 and e2: cball x e2 ∩ (S ∪ T) ⊆ T
      by (meson ‹x ∈ T› opT openin_contains_cball)
    then have cball x e2 ∩ (S ∪ T) = cball x e2 ∩ T
      by force
    moreover have closed (cball x e1 ∩ (cball x e2 ∩ T))
      by (metis closed_Int closed_cball e1 inf_left_commute)
```

```
    ultimately show ?thesis
      by (rule_tac x=min e1 e2 in exI) (simp add: ‹0 < e2› cball_min_Int
inf_assoc)
  qed
  ultimately show ?thesis
    by (force simp: locally_compact_Int_cball)
qed

lemma locally_compact_closedin_Un:
  fixes S :: 'a::euclidean_space set
  assumes LCS: locally compact S and LCT:locally compact T
      and clS: closedin (top_of_set (S ∪ T)) S
      and clT: closedin (top_of_set (S ∪ T)) T
    shows locally compact (S ∪ T)
proof −
  have ∃e>0. closed (cball x e ∩ (S ∪ T)) if x ∈ S x ∈ T for x
  proof −
    obtain e1 where e1 > 0 and e1: closed (cball x e1 ∩ S)
      using LCS ‹x ∈ S› unfolding locally_compact_Int_cball by blast
    moreover
    obtain e2 where e2 > 0 and e2: closed (cball x e2 ∩ T)
      using LCT ‹x ∈ T› unfolding locally_compact_Int_cball by blast
    moreover have closed (cball x (min e1 e2) ∩ (S ∪ T))
      by (smt (verit) Int_Un_distrib2 Int_commute cball_min_Int closed_Int
closed_Un closed_cball e1 e2 inf_left_commute)
    ultimately show ?thesis
      by (rule_tac x=min e1 e2 in exI) linarith
  qed
  moreover
  have ∃e>0. closed (cball x e ∩ (S ∪ T)) if x: x ∈ S x ∉ T for x
  proof −
    obtain e1 where e1 > 0 and e1: closed (cball x e1 ∩ S)
      using LCS ‹x ∈ S› unfolding locally_compact_Int_cball by blast
    moreover
    obtain e2 where e2>0 and cball x e2 ∩ (S ∪ T) ⊆ S − T
      using clT x by (fastforce simp: openin_contains_cball closedin_def)
    then have closed (cball x e2 ∩ T)
    proof −
      have {} = T − (T − cball x e2)
        using Diff_subset Int_Diff ‹cball x e2 ∩ (S ∪ T) ⊆ S − T› by auto
      then show ?thesis
        by (simp add: Diff_Diff_Int inf_commute)
    qed
    with e1 have closed ((cball x e1 ∩ cball x e2) ∩ (S ∪ T))
      apply (simp add: inf_commute inf_sup_distrib2)
      by (metis closed_Int closed_Un closed_cball inf_assoc inf_left_commute)
    then have closed (cball x (min e1 e2) ∩ (S ∪ T))
      by (simp add: cball_min_Int inf_commute)
    ultimately show ?thesis
```

```
    using ‹0 < e2› by (rule_tac x=min e1 e2 in exI) linarith
  qed
  moreover
  have ∃e>0. closed (cball x e ∩ (S ∪ T)) if x: x ∉ S x ∈ T for x
  proof −
    obtain e1 where e1 > 0 and e1: closed (cball x e1 ∩ T)
      using LCT ‹x ∈ T› unfolding locally_compact_Int_cball by blast
    moreover
    obtain e2 where e2>0 and cball x e2 ∩ (S ∪ T) ⊆ S ∪ T − S
      using clS x by (fastforce simp: openin_contains_cball closedin_def)
    then have closed (cball x e2 ∩ S)
       by (metis Diff_disjoint Int_empty_right closed_empty inf.left_commute
inf.orderE inf_sup_absorb)
    with e1 have closed ((cball x e1 ∩ cball x e2) ∩ (S ∪ T))
      apply (simp add: inf_commute inf_sup_distrib2)
      by (metis closed_Int closed_Un closed_cball inf_assoc inf_left_commute)
    then have closed (cball x (min e1 e2) ∩ (S ∪ T))
      by (auto simp: cball_min_Int)
    ultimately show ?thesis
      using ‹0 < e2› by (rule_tac x=min e1 e2 in exI) linarith
  qed
  ultimately show ?thesis
    by (auto simp: locally_compact_Int_cball)
qed

lemma locally_compact_Times:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  shows ⟦locally compact S; locally compact T⟧ ⟹ locally compact (S × T)
  by (auto simp: compact_Times locally_Times)

lemma locally_compact_compact_subopen:
  fixes S :: 'a :: heine_borel set
  shows
   locally compact S ⟷
    (∀K T. K ⊆ S ∧ compact K ∧ open T ∧ K ⊆ T
          ⟶ (∃U V. K ⊆ U ∧ U ⊆ V ∧ U ⊆ T ∧ V ⊆ S ∧
                   openin (top_of_set S) U ∧ compact V))
   (is ?lhs = ?rhs)
proof
  assume L: ?lhs
  show ?rhs
  proof clarify
    fix K :: 'a set and T :: 'a set
    assume K ⊆ S and compact K and open T and K ⊆ T
    obtain U V where K ⊆ U U ⊆ V V ⊆ S compact V
                and ope: openin (top_of_set S) U
      using L unfolding locally_compact_compact by (meson ‹K ⊆ S› ‹compact
K›)
    show ∃U V. K ⊆ U ∧ U ⊆ V ∧ U ⊆ T ∧ V ⊆ S ∧
```

```
            openin (top_of_set S) U ∧ compact V
    proof (intro exI conjI)
      show K ⊆ U ∩ T
        by (simp add: ‹K ⊆ T› ‹K ⊆ U›)
      show U ∩ T ⊆ closure(U ∩ T)
        by (rule closure_subset)
      show closure (U ∩ T) ⊆ S
         by (metis ‹U ⊆ V› ‹V ⊆ S› ‹compact V› closure_closed closure_mono
compact_imp_closed inf.cobounded1 subset_trans)
      show openin (top_of_set S) (U ∩ T)
        by (simp add: ‹open T› ope openin_Int_open)
      show compact (closure (U ∩ T))
          by (meson Int_lower1 ‹U ⊆ V› ‹compact V› bounded_subset com-
pact_closure compact_eq_bounded_closed)
    qed auto
  qed
next
  assume ?rhs then show ?lhs
    unfolding locally_compact_compact
     by (metis open_openin openin_topspace subtopology_superset top.extremum
topspace_euclidean_subtopology)
qed
```

5.4.16 Sura-Bura's results about compact components of sets

```
proposition Sura_Bura_compact:
  fixes S :: 'a::euclidean_space set
  assumes compact S and C: C ∈ components S
  shows C = ⋂{T. C ⊆ T ∧ openin (top_of_set S) T ∧
                    closedin (top_of_set S) T}
        (is C = ⋂ ?T)
proof
  obtain x where x: C = connected_component_set S x and x ∈ S
    using C by (auto simp: components_def)
  have C ⊆ S
    by (simp add: C in_components_subset)
  have ⋂ ?T ⊆ connected_component_set S x
  proof (rule connected_component_maximal)
    have x ∈ C
      by (simp add: ‹x ∈ S› x)
    then show x ∈ ⋂ ?T
      by blast
    have clo: closed (⋂ ?T)
    by (simp add: ‹compact S› closed_Inter closedin_compact_eq compact_imp_closed)
    have False
      if K1: closedin (top_of_set (⋂ ?T)) K1 and
         K2: closedin (top_of_set (⋂ ?T)) K2 and
         K12_Int: K1 ∩ K2 = {} and K12_Un: K1 ∪ K2 = ⋂ ?T and K1 ≠ {}
K2 ≠ {}
```

```
      for K1 K2
    proof −
      have closed K1 closed K2
        using closedin_closed_trans clo K1 K2 by blast+
      then obtain V1 V2 where open V1 open V2 K1 ⊆ V1 K2 ⊆ V2 and V12:
V1 ∩ V2 = {}
        using separation_normal ‹K1 ∩ K2 = {}› by metis
      have SV12_ne: (S − (V1 ∪ V2)) ∩ (⋂ ?𝒯) ≠ {}
      proof (rule compact_imp_fip)
        show compact (S − (V1 ∪ V2))
          by (simp add: ‹open V1› ‹open V2› ‹compact S› compact_diff open_Un)
        show clo𝒯: closed T if T ∈ ?𝒯 for T
          using that ‹compact S›
          by (force intro: closedin_closed_trans simp add: compact_imp_closed)
        show (S − (V1 ∪ V2)) ∩ ⋂ℱ ≠ {} if finite ℱ and ℱ: ℱ ⊆ ?𝒯 for ℱ
        proof
          assume djo: (S − (V1 ∪ V2)) ∩ ⋂ℱ = {}
          obtain D where opeD: openin (top_of_set S) D
                     and cloD: closedin (top_of_set S) D
                     and C ⊆ D and DV12: D ⊆ V1 ∪ V2
          proof (cases ℱ = {})
            case True
            with ‹C ⊆ S› djo that show ?thesis
              by force
          next
            case False show ?thesis
            proof
              show ope: openin (top_of_set S) (⋂ℱ)
                using openin_Inter ‹finite ℱ› False ℱ by blast
              then show closedin (top_of_set S) (⋂ℱ)
                by (meson clo𝒯 ℱ closed_Inter closed_subset openin_imp_subset
subset_eq)
              show C ⊆ ⋂ℱ
                using ℱ by auto
              show ⋂ℱ ⊆ V1 ∪ V2
                using ope djo openin_imp_subset by fastforce
            qed
          qed
          have connected C
            by (simp add: x)
          have closed D
            using ‹compact S› cloD closedin_closed_trans compact_imp_closed by
blast
          have cloV1: closedin (top_of_set D) (D ∩ closure V1)
            and cloV2: closedin (top_of_set D) (D ∩ closure V2)
            by (simp_all add: closedin_closed_Int)
          moreover have D ∩ closure V1 = D ∩ V1 D ∩ closure V2 = D ∩ V2
            using ‹D ⊆ V1 ∪ V2› ‹open V1› ‹open V2› V12
          by (auto simp: closure_subset [THEN subsetD] closure_iff_nhds_not_empty,
```

```
blast+)
        ultimately have cloDV1: closedin (top_of_set D) (D ∩ V1)
                    and cloDV2:  closedin (top_of_set D) (D ∩ V2)
          by metis+
        then obtain U1 U2 where closed U1 closed U2
            and D1: D ∩ V1 = D ∩ U1 and D2: D ∩ V2 = D ∩ U2
          by (auto simp: closedin_closed)
        have D ∩ U1 ∩ C ≠ {}
        proof
          assume D ∩ U1 ∩ C = {}
          then have ∗: C ⊆ D ∩ V2
            using D1 DV12 ‹C ⊆ D› by auto
          have 1: openin (top_of_set S) (D ∩ V2)
            by (simp add: ‹open V2› opeD openin_Int_open)
          have 2: closedin (top_of_set S) (D ∩ V2)
            using cloD cloDV2 closedin_trans by blast
          have ⋂ ?T ⊆ D ∩ V2
            by (rule Inter_lower) (use ∗ 1 2 in simp)
          then show False
            using K1 V12 ‹K1 ≠ {}› ‹K1 ⊆ V1› closedin_imp_subset by blast
        qed
        moreover have D ∩ U2 ∩ C ≠ {}
        proof
          assume D ∩ U2 ∩ C = {}
          then have ∗: C ⊆ D ∩ V1
            using D2 DV12 ‹C ⊆ D› by auto
          have 1: openin (top_of_set S) (D ∩ V1)
            by (simp add: ‹open V1› opeD openin_Int_open)
          have 2: closedin (top_of_set S) (D ∩ V1)
            using cloD cloDV1 closedin_trans by blast
          have ⋂ ?T ⊆ D ∩ V1
            by (rule Inter_lower) (use ∗ 1 2 in simp)
          then show False
            using K2 V12 ‹K2 ≠ {}› ‹K2 ⊆ V2› closedin_imp_subset by blast
        qed
        ultimately show False
         using ‹connected C› [unfolded connected_closed, simplified, rule_format,
of concl: D ∩ U1 D ∩ U2]
          using ‹C ⊆ D› D1 D2 V12 DV12 ‹closed U1› ‹closed U2› ‹closed D›
          by blast
      qed
    qed
    show False
      by (metis (full_types) DiffE UnE Un_upper2 SV12_ne ‹K1 ⊆ V1› ‹K2 ⊆
V2› disjoint_iff_not_equal subsetCE sup_ge1 K12_Un)
  qed
  then show connected (⋂ ?T)
    by (auto simp: connected_closedin_eq)
  show ⋂ ?T ⊆ S
```

```
    by (fastforce simp: C in_components_subset)
  qed
  with x show ⋂ ?𝒯 ⊆ C by simp
qed auto


corollary Sura_Bura_clopen_subset:
  fixes S :: 'a::euclidean_space set
  assumes S: locally compact S and C: C ∈ components S and compact C
      and U: open U C ⊆ U
  obtains K where openin (top_of_set S) K compact K C ⊆ K K ⊆ U
proof (rule ccontr)
  assume ¬ thesis
  with that have neg: ∄K. openin (top_of_set S) K ∧ compact K ∧ C ⊆ K ∧ K
⊆ U
    by metis
  obtain V K where C ⊆ V V ⊆ U V ⊆ K K ⊆ S compact K
            and openSV: openin (top_of_set S) V
   using S U ‹compact C› by (meson C in_components_subset locally_compact_compact_subopen)
  let ?𝒯 = {T. C ⊆ T ∧ openin (top_of_set K) T ∧ compact T ∧ T ⊆ K}
  have CK: C ∈ components K
    by (meson C ‹C ⊆ V› ‹K ⊆ S› ‹V ⊆ K› components_intermediate_subset
subset_trans)
  with ‹compact K›
  have C = ⋂{T. C ⊆ T ∧ openin (top_of_set K) T ∧ closedin (top_of_set K)
T}
    by (simp add: Sura_Bura_compact)
  then have Ceq: C = ⋂ ?𝒯
    by (simp add: closedin_compact_eq ‹compact K›)
  obtain W where open W and W: V = S ∩ W
    using openSV by (auto simp: openin_open)
  have −(U ∩ W) ∩ ⋂ ?𝒯 ≠ {}
  proof (rule closed_imp_fip_compact)
    show − (U ∩ W) ∩ ⋂ℱ ≠ {}
      if finite ℱ and ℱ: ℱ ⊆ ?𝒯 for ℱ
    proof (cases ℱ = {})
      case True
      have False if U = UNIV W = UNIV
      proof −
        have V = S
          by (simp add: W ‹W = UNIV›)
        with neg show False
          using ‹C ⊆ V› ‹K ⊆ S› ‹V ⊆ K› ‹V ⊆ U› ‹compact K› by auto
      qed
      with True show ?thesis
        by auto
    next
      case False
      show ?thesis
```

```
      proof
        assume − (U ∩ W) ∩ ⋂ℱ = {}
        then have FUW: ⋂ℱ ⊆ U ∩ W
          by blast
        have C ⊆ ⋂ℱ
          using ℱ by auto
        moreover have compact (⋂ℱ)
          by (metis (no_types, lifting) compact_Inter False mem_Collect_eq sub-
setCE ℱ)
        moreover have ⋂ℱ ⊆ K
          using False that(2) by fastforce
        moreover have opeKF: openin (top_of_set K) (⋂ℱ)
          using False ℱ ‹finite ℱ› by blast
        then have opeVF: openin (top_of_set V) (⋂ℱ)
         using W ‹K ⊆ S› ‹V ⊆ K› opeKF ‹⋂ℱ ⊆ K› FUW openin_subset_trans
by fastforce
        then have openin (top_of_set S) (⋂ℱ)
          by (metis opeSV openin_trans)
        moreover have ⋂ℱ ⊆ U
          by (meson ‹V ⊆ U› opeVF dual_order.trans openin_imp_subset)
        ultimately show False
          using neg by blast
      qed
    qed
  qed (use ‹open W› ‹open U› in auto)
  with W Ceq ‹C ⊆ V› ‹C ⊆ U› show False
    by auto
qed


corollary Sura_Bura_clopen_subset_alt:
  fixes S :: 'a::euclidean_space set
  assumes S: locally compact S and C: C ∈ components S and compact C
      and opeSU: openin (top_of_set S) U and C ⊆ U
  obtains K where openin (top_of_set S) K compact K C ⊆ K K ⊆ U
proof −
  obtain V where open V U = S ∩ V
    using opeSU by (auto simp: openin_open)
  with ‹C ⊆ U› have C ⊆ V
    by auto
  then show ?thesis
    using Sura_Bura_clopen_subset [OF S C ‹compact C› ‹open V›]
    by (metis ‹U = S ∩ V› inf.bounded_iff openin_imp_subset that)
qed

corollary Sura_Bura:
  fixes S :: 'a::euclidean_space set
  assumes locally compact S C ∈ components S compact C
  shows C = ⋂ {K. C ⊆ K ∧ compact K ∧ openin (top_of_set S) K}
```

(**is** $C = ?rhs$)
proof
show $?rhs \subseteq C$
proof (*clarsimp*, *rule ccontr*)
fix x
assume $*$: $\forall X.\ C \subseteq X \wedge compact\ X \wedge openin\ (top\_of\_set\ S)\ X \longrightarrow x \in X$
and $x \notin C$
obtain $U\ V$ **where** *open U open V* $\{x\} \subseteq U\ C \subseteq V\ U \cap V = \{\}$
using *separation_normal* [*of* $\{x\}$ C]
by (*metis Int_empty_left* ‹$x \notin C$› ‹*compact C*› *closed_empty closed_insert compact_imp_closed insert_disjoint*(*1*))
have $x \notin V$
using ‹$U \cap V = \{\}$› ‹$\{x\} \subseteq U$› **by** *blast*
then show *False*
by (*meson* $*$ *Sura_Bura_clopen_subset* ‹$C \subseteq V$› ‹*open V*› *assms*(*1*) *assms*(*2*) *assms*(*3*) *subsetCE*)
qed
qed *blast*

5.4.17 Special cases of local connectedness and path connectedness

lemma *locally_connected_1*:
assumes
$\bigwedge V\ x.\ [\![openin\ (top\_of\_set\ S)\ V;\ x \in V]\!] \Longrightarrow \exists U.\ openin\ (top\_of\_set\ S)\ U \wedge connected\ U \wedge x \in U \wedge U \subseteq V$
shows *locally connected S*
by (*metis assms locally_def*)

lemma *locally_connected_2*:
assumes *locally connected S*
openin (*top_of_set S*) *t*
$x \in t$
shows *openin* (*top_of_set S*) (*connected_component_set t x*)
proof −
{ fix $y :: 'a$
let *?SS* = *top_of_set S*
assume *1*: *openin ?SS t*
$\forall w\ x.\ openin\ ?SS\ w \wedge x \in w \longrightarrow (\exists u.\ openin\ ?SS\ u \wedge (\exists v.\ connected\ v \wedge x \in u \wedge u \subseteq v \wedge v \subseteq w))$
and *connected_component t x y*
then have $y \in t$ **and** *y*: $y \in connected\_component\_set\ t\ x$
using *connected_component_subset* **by** *blast*+
obtain F **where**
$\forall x\ y.\ (\exists w.\ openin\ ?SS\ w \wedge (\exists u.\ connected\ u \wedge x \in w \wedge w \subseteq u \wedge u \subseteq y)) = (openin\ ?SS\ (F\ x\ y) \wedge (\exists u.\ connected\ u \wedge x \in F\ x\ y \wedge F\ x\ y \subseteq u \wedge u \subseteq y))$
by *moura*
then obtain G **where**
$\forall a\ A.\ (\exists U.\ openin\ ?SS\ U \wedge (\exists V.\ connected\ V \wedge a \in U \wedge U \subseteq V \wedge V \subseteq$

$A)) = (\mathit{openin}\ ?SS\ (F\ a\ A) \wedge \mathit{connected}\ (G\ a\ A) \wedge a \in F\ a\ A \wedge F\ a\ A \subseteq G\ a\ A$
$\wedge\ G\ a\ A \subseteq A)$
by *moura*
then have $*$: $\mathit{openin}\ ?SS\ (F\ y\ t) \wedge \mathit{connected}\ (G\ y\ t) \wedge y \in F\ y\ t \wedge F\ y\ t \subseteq$
$G\ y\ t \wedge G\ y\ t \subseteq t$
using *1* ‹$y \in t$› **by** *presburger*
have $G\ y\ t \subseteq \mathit{connected\_component\_set}\ t\ y$
by (*metis* (*no_types*) $*$ *connected_component_eq_self connected_component_mono*
contra_subsetD)
then have $\exists A.\ \mathit{openin}\ ?SS\ A \wedge y \in A \wedge A \subseteq \mathit{connected\_component\_set}\ t\ x$
by (*metis* (*no_types*) $*$ *connected_component_eq dual_order.trans y*)
}
then show *?thesis*
using *assms openin_subopen* **by** (*force simp*: *locally_def*)
qed

lemma *locally_connected_3*:
assumes $\bigwedge t\ x.\ [\![\mathit{openin}\ (\mathit{top\_of\_set}\ S)\ t;\ x \in t]\!]$
$\Longrightarrow \mathit{openin}\ (\mathit{top\_of\_set}\ S)$
$(\mathit{connected\_component\_set}\ t\ x)$
$\mathit{openin}\ (\mathit{top\_of\_set}\ S)\ v\ x \in v$
shows $\exists u.\ \mathit{openin}\ (\mathit{top\_of\_set}\ S)\ u \wedge \mathit{connected}\ u \wedge x \in u \wedge u \subseteq v$
using *assms connected_component_subset* **by** *fastforce*

lemma *locally_connected*:
locally connected $S \longleftrightarrow$
$(\forall v\ x.\ \mathit{openin}\ (\mathit{top\_of\_set}\ S)\ v \wedge x \in v$
$\longrightarrow (\exists u.\ \mathit{openin}\ (\mathit{top\_of\_set}\ S)\ u \wedge \mathit{connected}\ u \wedge x \in u \wedge u \subseteq v))$
by (*metis locally_connected_1 locally_connected_2 locally_connected_3*)

lemma *locally_connected_open_connected_component*:
locally connected $S \longleftrightarrow$
$(\forall t\ x.\ \mathit{openin}\ (\mathit{top\_of\_set}\ S)\ t \wedge x \in t$
$\longrightarrow \mathit{openin}\ (\mathit{top\_of\_set}\ S)\ (\mathit{connected\_component\_set}\ t\ x))$
by (*metis locally_connected_1 locally_connected_2 locally_connected_3*)

lemma *locally_path_connected_1*:
assumes
$\bigwedge v\ x.\ [\![\mathit{openin}\ (\mathit{top\_of\_set}\ S)\ v;\ x \in v]\!]$
$\Longrightarrow \exists u.\ \mathit{openin}\ (\mathit{top\_of\_set}\ S)\ u \wedge \mathit{path\_connected}\ u \wedge x \in u \wedge u \subseteq v$
shows *locally path_connected S*
by (*force simp*: *locally_def dest*: *assms*)

lemma *locally_path_connected_2*:
assumes *locally path_connected S*
$\mathit{openin}\ (\mathit{top\_of\_set}\ S)\ t$
$x \in t$
shows $\mathit{openin}\ (\mathit{top\_of\_set}\ S)\ (\mathit{path\_component\_set}\ t\ x)$
proof −

```
  { fix y :: 'a
    let ?SS = top_of_set S
    assume 1: openin ?SS t
          ∀ w x. openin ?SS w ∧ x ∈ w ⟶ (∃ u. openin ?SS u ∧ (∃ v. path_connected
v ∧ x ∈ u ∧ u ⊆ v ∧ v ⊆ w))
    and path_component t x y
    then have y ∈ t and y: y ∈ path_component_set t x
      using path_component_mem(2) by blast+
    obtain F where
      ∀ x y. (∃ w. openin ?SS w ∧ (∃ u. path_connected u ∧ x ∈ w ∧ w ⊆ u ∧ u ⊆
y)) = (openin ?SS (F x y) ∧ (∃ u. path_connected u ∧ x ∈ F x y ∧ F x y ⊆ u ∧
u ⊆ y))
      by moura
    then obtain G where
      ∀ a A. (∃ U. openin ?SS U ∧ (∃ V. path_connected V ∧ a ∈ U ∧ U ⊆ V ∧
V ⊆ A)) = (openin ?SS (F a A) ∧ path_connected (G a A) ∧ a ∈ F a A ∧ F a A
⊆ G a A ∧ G a A ⊆ A)
      by moura
    then have ∗: openin ?SS (F y t) ∧ path_connected (G y t) ∧ y ∈ F y t ∧ F y
t ⊆ G y t ∧ G y t ⊆ t
      using 1 ‹y ∈ t› by presburger
    have G y t ⊆ path_component_set t y
      using ∗ path_component_maximal rev_subsetD by blast
    then have ∃ A. openin ?SS A ∧ y ∈ A ∧ A ⊆ path_component_set t x
    by (metis ∗ ‹G y t ⊆ path_component_set t y› dual_order.trans path_component_eq
y)
  }
  then show ?thesis
    using assms openin_subopen by (force simp: locally_def)
qed

lemma locally_path_connected_3:
  assumes ⋀t x. ⟦openin (top_of_set S) t; x ∈ t⟧
            ⟹ openin (top_of_set S) (path_component_set t x)
          openin (top_of_set S) v x ∈ v
   shows  ∃ u. openin (top_of_set S) u ∧ path_connected u ∧ x ∈ u ∧ u ⊆ v
proof −
  have path_component v x x
    by (meson assms(3) path_component_refl)
  then show ?thesis
   by (metis assms mem_Collect_eq path_component_subset path_connected_path_component)
qed

proposition locally_path_connected:
  locally path_connected S ⟷
   (∀ V x. openin (top_of_set S) V ∧ x ∈ V
         ⟶ (∃ U. openin (top_of_set S) U ∧ path_connected U ∧ x ∈ U ∧ U ⊆
V))
 by (metis locally_path_connected_1 locally_path_connected_2 locally_path_connected_3)
```

proposition *locally_path_connected_open_path_component*:
locally path_connected S ⟷
(∀ *t x*. *openin* (*top_of_set S*) *t* ∧ *x* ∈ *t*
⟶ *openin* (*top_of_set S*) (*path_component_set t x*))
by (*metis locally_path_connected_1 locally_path_connected_2 locally_path_connected_3*)

lemma *locally_connected_open_component*:
locally connected S ⟷
(∀ *t c*. *openin* (*top_of_set S*) *t* ∧ *c* ∈ *components t*
⟶ *openin* (*top_of_set S*) *c*)
by (*metis components_iff locally_connected_open_connected_component*)

proposition *locally_connected_im_kleinen*:
locally connected S ⟷
(∀ *v x*. *openin* (*top_of_set S*) *v* ∧ *x* ∈ *v*
⟶ (∃ *u*. *openin* (*top_of_set S*) *u* ∧
x ∈ *u* ∧ *u* ⊆ *v* ∧
(∀ *y*. *y* ∈ *u* ⟶ (∃ *c*. *connected c* ∧ *c* ⊆ *v* ∧ *x* ∈ *c* ∧ *y* ∈ *c*))))
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (*fastforce simp*: *locally_connected*)
next
assume *?rhs*
have ∗: ∃ *T*. *openin* (*top_of_set S*) *T* ∧ *x* ∈ *T* ∧ *T* ⊆ *c*
if *openin* (*top_of_set S*) *t* **and** *c*: *c* ∈ *components t* **and** *x* ∈ *c* **for** *t c x*
proof −
from *that* ‹*?rhs*› [*rule_format*, *of t x*]
obtain *u* **where** *u*:
openin (*top_of_set S*) *u* ∧ *x* ∈ *u* ∧ *u* ⊆ *t* ∧
(∀ *y*. *y* ∈ *u* ⟶ (∃ *c*. *connected c* ∧ *c* ⊆ *t* ∧ *x* ∈ *c* ∧ *y* ∈ *c*))
using *in_components_subset* **by** *auto*
obtain *F* :: *'a set* ⇒ *'a set* ⇒ *'a* **where**
∀ *x y*. (∃ *z*. *z* ∈ *x* ∧ *y* = *connected_component_set x z*) = (*F x y* ∈ *x* ∧ *y* =
connected_component_set x (*F x y*))
by *moura*
then have *F*: *F t c* ∈ *t* ∧ *c* = *connected_component_set t* (*F t c*)
by (*meson components_iff c*)
obtain *G* :: *'a set* ⇒ *'a set* ⇒ *'a* **where**
G: ∀ *x y*. (∃ *z*. *z* ∈ *y* ∧ *z* ∉ *x*) = (*G x y* ∈ *y* ∧ *G x y* ∉ *x*)
by *moura*
have *G c u* ∉ *u* ∨ *G c u* ∈ *c*
using *F* **by** (*metis* (*full_types*) *u connected_componentI connected_component_eq*
mem_Collect_eq that(*3*))
then show *?thesis*
using *G u* **by** *auto*
qed

```
  show ?lhs
    unfolding locally_connected_open_component by (meson * openin_subopen)
qed

proposition locally_path_connected_im_kleinen:
  locally path_connected S ⟷
   (∀ v x. openin (top_of_set S) v ∧ x ∈ v
       ⟶ (∃ u. openin (top_of_set S) u ∧
                x ∈ u ∧ u ⊆ v ∧
                (∀ y. y ∈ u ⟶ (∃ p. path p ∧ path_image p ⊆ v ∧
                                    pathstart p = x ∧ pathfinish p = y))))
   (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    apply (simp add: locally_path_connected path_connected_def)
    apply (erule all_forward ex_forward imp_forward conjE | simp)+
    by (meson dual_order.trans)
next
  assume ?rhs
  have *: ∃ T. openin (top_of_set S) T ∧
               x ∈ T ∧ T ⊆ path_component_set u z
      if openin (top_of_set S) u and z ∈ u and c: path_component u z x for u z
x
  proof −
    have x ∈ u
      by (meson c path_component_mem(2))
    with that ‹?rhs› [rule_format, of u x]
    obtain U where U:
      openin (top_of_set S) U ∧ x ∈ U ∧ U ⊆ u ∧
        (∀ y. y ∈ U ⟶ (∃ p. path p ∧ path_image p ⊆ u ∧ pathstart p = x ∧
pathfinish p = y))
       by blast
    show ?thesis
       by (metis U c mem_Collect_eq path_component_def path_component_eq
subsetI)
  qed
  show ?lhs
    unfolding locally_path_connected_open_path_component
    using * openin_subopen by fastforce
qed

lemma locally_path_connected_imp_locally_connected:
  locally path_connected S ⟹ locally connected S
using locally_mono path_connected_imp_connected by blast

lemma locally_connected_components:
  ⟦locally connected S; c ∈ components S⟧ ⟹ locally connected c
by (meson locally_connected_open_component locally_open_subset openin_subtopology_self)
```

lemma *locally_path_connected_components*:
  ⟦*locally path_connected S*; *c* ∈ *components S*⟧ ⟹ *locally path_connected c*
by (*meson locally_connected_open_component locally_open_subset locally_path_connected_imp_locally_connecte*
openin_subtopology_self)

lemma *locally_path_connected_connected_component*:
  locally path_connected S ⟹ *locally path_connected* (*connected_component_set*
S x)
by (*metis components_iff connected_component_eq_empty locally_empty locally_path_connected_components*)

lemma *open_imp_locally_path_connected*:
  fixes *S* :: *'a* :: *real_normed_vector set*
  assumes *open S*
  shows *locally path_connected S*
proof (*rule locally_mono*)
  show *locally convex S*
    using *assms* **unfolding** *locally_def*
   by (*meson open_ball centre_in_ball convex_ball openE open_subset openin_imp_subset*
openin_open_trans subset_trans)
  show ⋀*T*::*'a set*. *convex T* ⟹ *path_connected T*
    using *convex_imp_path_connected* **by** *blast*
qed

lemma *open_imp_locally_connected*:
  fixes *S* :: *'a* :: *real_normed_vector set*
  shows *open S* ⟹ *locally connected S*
by (*simp add*: *locally_path_connected_imp_locally_connected open_imp_locally_path_connected*)

lemma *locally_path_connected_UNIV*: *locally path_connected* (*UNIV*::*'a* :: *real_normed_vector*
set)
  by (*simp add*: *open_imp_locally_path_connected*)

lemma *locally_connected_UNIV*: *locally connected* (*UNIV*::*'a* :: *real_normed_vector*
set)
  by (*simp add*: *open_imp_locally_connected*)

lemma *openin_connected_component_locally_connected*:
    locally connected S
    ⟹ *openin* (*top_of_set S*) (*connected_component_set S x*)
  by (*metis connected_component_eq_empty locally_connected_2 openin_empty*
openin_subtopology_self)

lemma *openin_components_locally_connected*:
    ⟦*locally connected S*; *c* ∈ *components S*⟧ ⟹ *openin* (*top_of_set S*) *c*
  using *locally_connected_open_component openin_subtopology_self* **by** *blast*

lemma *openin_path_component_locally_path_connected*:
  locally path_connected S

$\Longrightarrow$ *openin* (*top_of_set S*) (*path_component_set S x*)
by (*metis* (*no_types*) *empty_iff locally_path_connected_2 openin_subopen openin_subtopology_self path_component_eq_empty*)

lemma *closedin_path_component_locally_path_connected*:
assumes *locally path_connected S*
shows *closedin* (*top_of_set S*) (*path_component_set S x*)
proof −
have *openin* (*top_of_set S*) ($\bigcup$ ({*path_component_set S y* |*y*. *y* ∈ *S*} − {*path_component_set S x*}))
using *locally_path_connected_2 assms* **by** *fastforce*
then show *?thesis*
by (*simp add*: *closedin_def path_component_subset complement_path_component_Union*)
qed

lemma *convex_imp_locally_path_connected*:
fixes *S* :: *'a*:: *real_normed_vector set*
assumes *convex S*
shows *locally path_connected S*
proof (*clarsimp simp*: *locally_path_connected*)
fix *V x*
assume *openin* (*top_of_set S*) *V* **and** *x* ∈ *V*
then obtain *T e* **where** *V* = *S* ∩ *T x* ∈ *S* 0 < *e ball x e* ⊆ *T*
by (*metis Int_iff openE openin_open*)
then have *openin* (*top_of_set S*) (*S* ∩ *ball x e*) *path_connected* (*S* ∩ *ball x e*)
by (*simp_all add*: *assms convex_Int convex_imp_path_connected openin_open_Int*)
then show ∃ *U*. *openin* (*top_of_set S*) *U* ∧ *path_connected U* ∧ *x* ∈ *U* ∧ *U* ⊆ *V*
using ‹*0* < *e*› ‹*V* = *S* ∩ *T*› ‹*ball x e* ⊆ *T*› ‹*x* ∈ *S*› **by** *auto*
qed

lemma *convex_imp_locally_connected*:
fixes *S* :: *'a*:: *real_normed_vector set*
shows *convex S* $\Longrightarrow$ *locally connected S*
by (*simp add*: *locally_path_connected_imp_locally_connected convex_imp_locally_path_connected*)

5.4.18 Relations between components and path components

lemma *path_component_eq_connected_component*:
assumes *locally path_connected S*
shows (*path_component S x* = *connected_component S x*)
proof (*cases x* ∈ *S*)
case *True*
have *openin* (*top_of_set* (*connected_component_set S x*)) (*path_component_set S x*)
proof (*rule openin_subset_trans*)
show *openin* (*top_of_set S*) (*path_component_set S x*)
by (*simp add*: *True assms locally_path_connected_2*)
show *connected_component_set S x* ⊆ *S*

by (*simp add*: *connected_component_subset*)
qed (*simp add*: *path_component_subset_connected_component*)
moreover have *closedin* (*top_of_set* (*connected_component_set S x*)) (*path_component_set S x*)
proof (*rule closedin_subset_trans* [*of S*])
show *closedin* (*top_of_set S*) (*path_component_set S x*)
by (*simp add*: *assms closedin_path_component_locally_path_connected*)
show *connected_component_set S x* ⊆ *S*
by (*simp add*: *connected_component_subset*)
qed (*simp add*: *path_component_subset_connected_component*)
ultimately have ∗: *path_component_set S x* = *connected_component_set S x*
by (*metis connected_connected_component connected_clopen True path_component_eq_empty*)
then show *?thesis*
by *blast*
next
case *False* **then show** *?thesis*
by (*metis Collect_empty_eq_bot connected_component_eq_empty path_component_eq_empty*)
qed

lemma *path_component_eq_connected_component_set*:
locally path_connected S ⟹ (*path_component_set S x* = *connected_component_set S x*)
by (*simp add*: *path_component_eq_connected_component*)

lemma *locally_path_connected_path_component*:
locally path_connected S ⟹ *locally path_connected* (*path_component_set S x*)
using *locally_path_connected_connected_component path_component_eq_connected_component*
by *fastforce*

lemma *open_path_connected_component*:
fixes *S* :: *'a* :: *real_normed_vector set*
shows *open S* ⟹ *path_component S x* = *connected_component S x*
by (*simp add*: *path_component_eq_connected_component open_imp_locally_path_connected*)

lemma *open_path_connected_component_set*:
fixes *S* :: *'a* :: *real_normed_vector set*
shows *open S* ⟹ *path_component_set S x* = *connected_component_set S x*
by (*simp add*: *open_path_connected_component*)

proposition *locally_connected_quotient_image*:
assumes *lcS*: *locally connected S*
and *oo*: ⋀*T*. *T* ⊆ *f* ‘ *S*
⟹ *openin* (*top_of_set S*) (*S* ∩ *f* −‘ *T*) ⟷
openin (*top_of_set* (*f* ‘ *S*)) *T*
shows *locally connected* (*f* ‘ *S*)
proof (*clarsimp simp*: *locally_connected_open_component*)
fix *U C*
assume *opefSU*: *openin* (*top_of_set* (*f* ‘ *S*)) *U* **and** *C* ∈ *components U*

```
  then have C ⊆ U U ⊆ f ‘ S
    by (meson in_components_subset openin_imp_subset)+
  then have openin (top_of_set (f ‘ S)) C ⟷
            openin (top_of_set S) (S ∩ f −‘ C)
    by (auto simp: oo)
  moreover have openin (top_of_set S) (S ∩ f −‘ C)
  proof (subst openin_subopen, clarify)
    fix x
    assume x ∈ S f x ∈ C
    show ∃ T. openin (top_of_set S) T ∧ x ∈ T ∧ T ⊆ (S ∩ f −‘ C)
    proof (intro conjI exI)
      show openin (top_of_set S) (connected_component_set (S ∩ f −‘ U) x)
      proof (rule ccontr)
       assume ∗∗: ¬ openin (top_of_set S) (connected_component_set (S ∩ f −‘
U) x)
        then have x ∉ (S ∩ f −‘ U)
          using ‹U ⊆ f ‘ S› opefSU lcS locally_connected_2 oo by blast
        with ∗∗ show False
       by (metis (no_types) connected_component_eq_empty empty_iff openin_subopen)
      qed
    next
      show x ∈ connected_component_set (S ∩ f −‘ U) x
        using ‹C ⊆ U› ‹f x ∈ C› ‹x ∈ S› by auto
    next
      have contf: continuous_on S f
        by (simp add: continuous_on_open oo openin_imp_subset)
      then have continuous_on (connected_component_set (S ∩ f −‘ U) x) f
      by (meson connected_component_subset continuous_on_subset inf.boundedE)
      then have connected (f ‘ connected_component_set (S ∩ f −‘ U) x)
      by (rule connected_continuous_image [OF _ connected_connected_component])
      moreover have f ‘ connected_component_set (S ∩ f −‘ U) x ⊆ U
        using connected_component_in by blast
      moreover have C ∩ f ‘ connected_component_set (S ∩ f −‘ U) x ≠ {}
        using ‹C ⊆ U› ‹f x ∈ C› ‹x ∈ S› by fastforce
      ultimately have fC: f ‘ (connected_component_set (S ∩ f −‘ U) x) ⊆ C
        by (rule components_maximal [OF ‹C ∈ components U›])
      have cUC: connected_component_set (S ∩ f −‘ U) x ⊆ (S ∩ f −‘ C)
        using connected_component_subset fC by blast
     have connected_component_set (S ∩ f −‘ U) x ⊆ connected_component_set
(S ∩ f −‘ C) x
      proof −
        { assume x ∈ connected_component_set (S ∩ f −‘ U) x
          then have ?thesis
            using cUC connected_component_idemp connected_component_mono
by blast }
        then show ?thesis
          using connected_component_eq_empty by auto
      qed
      also have ... ⊆ (S ∩ f −‘ C)
```

```
      by (rule connected_component_subset)
    finally show connected_component_set (S ∩ f -` U) x ⊆ (S ∩ f -` C) .
  qed
 qed
 ultimately show openin (top_of_set (f ` S)) C
  by metis
qed
```

The proof resembles that above but is not identical!

```
proposition locally_path_connected_quotient_image:
 assumes lcS: locally path_connected S
    and oo: ⋀T. T ⊆ f ` S
            ⟹ openin (top_of_set S) (S ∩ f -` T) ⟷ openin (top_of_set (f
` S)) T
  shows locally path_connected (f ` S)
proof (clarsimp simp: locally_path_connected_open_path_component)
 fix U y
 assume opefSU: openin (top_of_set (f ` S)) U and y ∈ U
 then have path_component_set U y ⊆ U U ⊆ f ` S
  by (meson path_component_subset openin_imp_subset)+
 then have openin (top_of_set (f ` S)) (path_component_set U y) ⟷
           openin (top_of_set S) (S ∩ f -` path_component_set U y)
 proof -
  have path_component_set U y ⊆ f ` S
   using ‹U ⊆ f ` S› ‹path_component_set U y ⊆ U› by blast
  then show ?thesis
   using oo by blast
 qed
 moreover have openin (top_of_set S) (S ∩ f -` path_component_set U y)
 proof (subst openin_subopen, clarify)
  fix x
  assume x ∈ S and Uyfx: path_component U y (f x)
  then have f x ∈ U
   using path_component_mem by blast
  show ∃T. openin (top_of_set S) T ∧ x ∈ T ∧ T ⊆ (S ∩ f -` path_component_set
U y)
  proof (intro conjI exI)
   show openin (top_of_set S) (path_component_set (S ∩ f -` U) x)
   proof (rule ccontr)
    assume **: ¬ openin (top_of_set S) (path_component_set (S ∩ f -` U)
x)
    then have x ∉ (S ∩ f -` U)
     by (metis (no_types, lifting) ‹U ⊆ f ` S› opefSU lcS oo locally_path_connected_open_path_component)
    then show False
     using ** ‹path_component_set U y ⊆ U› ‹x ∈ S› ‹path_component U y
(f x)› by blast
   qed
  next
   show x ∈ path_component_set (S ∩ f -` U) x
```

```
      by (simp add: ‹f x ∈ U› ‹x ∈ S› path_component_refl)
    next
     have contf: continuous_on S f
      by (simp add: continuous_on_open oo openin_imp_subset)
     then have continuous_on (path_component_set (S ∩ f -‘ U) x) f
      by (meson Int_lower1 continuous_on_subset path_component_subset)
     then have path_connected (f ‘ path_component_set (S ∩ f -‘ U) x)
      by (simp add: path_connected_continuous_image)
     moreover have f ‘ path_component_set (S ∩ f -‘ U) x ⊆ U
      using path_component_mem by fastforce
     moreover have f x ∈ f ‘ path_component_set (S ∩ f -‘ U) x
      by (force simp: ‹x ∈ S› ‹f x ∈ U› path_component_refl_eq)
    ultimately have f ‘ (path_component_set (S ∩ f -‘ U) x) ⊆ path_component_set
U (f x)
      by (meson path_component_maximal)
     also have  ... ⊆ path_component_set U y
       by (simp add: Uyfx path_component_maximal path_component_subset
path_component_sym)
    finally have fC: f ‘ (path_component_set (S ∩ f -‘ U) x) ⊆ path_component_set
U y .
    have cUC: path_component_set (S ∩ f -‘ U) x ⊆ (S ∩ f -‘ path_component_set
U y)
      using path_component_subset fC by blast
    have path_component_set (S ∩ f -‘ U) x ⊆ path_component_set (S ∩ f -‘
path_component_set U y) x
     proof -
      have ⋀a. path_component_set (path_component_set (S ∩ f -‘ U) x) a ⊆
path_component_set (S ∩ f -‘ path_component_set U y) a
       using cUC path_component_mono by blast
      then show ?thesis
       using path_component_path_component by blast
     qed
     also have ... ⊆ (S ∩ f -‘ path_component_set U y)
      by (rule path_component_subset)
    finally show path_component_set (S ∩ f -‘ U) x ⊆ (S ∩ f -‘ path_component_set
U y) .
   qed
  qed
  ultimately show openin (top_of_set (f ‘ S)) (path_component_set U y)
   by metis
qed
```

5.4.19 Components, continuity, openin, closedin

```
lemma continuous_on_components_gen:
 fixes f :: 'a::topological_space ⇒ 'b::topological_space
  assumes ⋀C. C ∈ components S ⟹
              openin (top_of_set S) C ∧ continuous_on C f
    shows continuous_on S f
```

proof (*clarsimp simp*: *continuous_openin_preimage_eq*)
fix $t :: 'b\ set$
assume *open t*
have $*$: $S \cap f -` t = (\bigcup c \in components\ S.\ c \cap f -` t)$
by *auto*
show *openin* (*top_of_set S*) ($S \cap f -` t$)
unfolding $*$ **using** ‹*open t*› *assms continuous_openin_preimage_gen openin_trans openin_Union* **by** *blast*
qed

lemma *continuous_on_components*:
fixes $f :: 'a::topological\_space \Rightarrow 'b::topological\_space$
assumes *locally connected S* $\bigwedge C.\ C \in components\ S \Longrightarrow continuous\_on\ C\ f$
shows *continuous_on S f*
proof (*rule continuous_on_components_gen*)
fix *C*
assume $C \in components\ S$
then show *openin* (*top_of_set S*) $C \wedge continuous\_on\ C\ f$
by (*simp add*: *assms openin_components_locally_connected*)
qed

lemma *continuous_on_components_eq*:
locally connected S
$\Longrightarrow$ (*continuous_on S f* $\longleftrightarrow$ ($\forall c \in components\ S.\ continuous\_on\ c\ f$))
by (*meson continuous_on_components continuous_on_subset in_components_subset*)

lemma *continuous_on_components_open*:
fixes $S :: 'a::real\_normed\_vector\ set$
assumes *open S*
$\bigwedge c.\ c \in components\ S \Longrightarrow continuous\_on\ c\ f$
shows *continuous_on S f*
using *continuous_on_components open_imp_locally_connected assms* **by** *blast*

lemma *continuous_on_components_open_eq*:
fixes $S :: 'a::real\_normed\_vector\ set$
shows *open S* $\Longrightarrow$ (*continuous_on S f* $\longleftrightarrow$ ($\forall c \in components\ S.\ continuous\_on\ c\ f$))
using *continuous_on_subset in_components_subset*
by (*blast intro*: *continuous_on_components_open*)

lemma *closedin_union_complement_components*:
assumes *U*: *locally connected U*
and *S*: *closedin* (*top_of_set U*) *S*
and *cuS*: $c \subseteq components(U - S)$
shows *closedin* (*top_of_set U*) ($S \cup \bigcup c$)
proof −
have *di*: $(\bigwedge S\ T.\ S \in c \wedge T \in c' \Longrightarrow disjnt\ S\ T) \Longrightarrow disjnt\ (\bigcup\ c)\ (\bigcup\ c')$ **for** c'
by (*simp add*: *disjnt_def*) *blast*
have $S \subseteq U$

```
    using S closedin_imp_subset by blast
  moreover have U - S = ⋃c ∪ ⋃(components (U - S) - c)
    by (metis Diff_partition Union_components Union_Un_distrib assms(3))
  moreover have disjnt (⋃c) (⋃(components (U - S) - c))
    apply (rule di)
    by (metis di DiffD1 DiffD2 assms(3) components_nonoverlap disjnt_def sub-
setCE)
  ultimately have eq: S ∪ ⋃c = U - (⋃(components(U - S) - c))
    by (auto simp: disjnt_def)
  have *: openin (top_of_set U) (⋃(components (U - S) - c))
  proof (rule openin_Union [OF openin_trans [of U - S]])
    show openin (top_of_set (U - S)) T if T ∈ components (U - S) - c for T
    using that by (simp add: U S locally_diff_closed openin_components_locally_connected)
    show openin (top_of_set U) (U - S) if T ∈ components (U - S) - c for T
      using that by (simp add: openin_diff S)
  qed
  have closedin (top_of_set U) (U - ⋃ (components (U - S) - c))
    by (metis closedin_diff closedin_topspace topspace_euclidean_subtopology *)
  then have openin (top_of_set U) (U - (U - ⋃(components (U - S) - c)))
    by (simp add: openin_diff)
  then show ?thesis
    by (force simp: eq closedin_def)
qed

lemma closed_union_complement_components:
  fixes S :: 'a::real_normed_vector set
  assumes S: closed S and c: c ⊆ components(- S)
    shows closed(S ∪ ⋃ c)
proof -
  have closedin (top_of_set UNIV) (S ∪ ⋃c)
   by (metis Compl_eq_Diff_UNIV S c closed_closedin closedin_union_complement_components
locally_connected_UNIV subtopology_UNIV)
  then show ?thesis by simp
qed

lemma closedin_Un_complement_component:
  fixes S :: 'a::real_normed_vector set
  assumes u: locally connected u
      and S: closedin (top_of_set u) S
      and c:  c ∈ components(u - S)
    shows closedin (top_of_set u) (S ∪ c)
proof -
  have closedin (top_of_set u) (S ∪ ⋃{c})
    using c by (blast intro: closedin_union_complement_components [OF u S])
  then show ?thesis
    by simp
qed

lemma closed_Un_complement_component:
```

fixes S :: $'a$::*real_normed_vector set*
assumes S: *closed* S **and** c: $c \in$ *components*$(-S)$
shows *closed* $(S \cup c)$
by (*metis Compl_eq_Diff_UNIV S c closed_closedin closedin_Un_complement_component locally_connected_UNIV subtopology_UNIV*)

5.4.20 Existence of isometry between subspaces of same dimension

lemma *isometry_subset_subspace*:
fixes S :: $'a$::*euclidean_space set*
and T :: $'b$::*euclidean_space set*
assumes S: *subspace* S
and T: *subspace* T
and d: *dim* $S \leq$ *dim* T
obtains f **where** *linear* f $f \in S \to T$ $\bigwedge x.\ x \in S \Longrightarrow$ *norm*$(f\ x) =$ *norm* x
proof −
obtain B **where** $B \subseteq S$ **and** *Borth*: *pairwise orthogonal* B
and *B1*: $\bigwedge x.\ x \in B \Longrightarrow$ *norm* $x = 1$
and *independent* B *finite* B *card* $B =$ *dim* S *span* $B = S$
by (*metis orthonormal_basis_subspace* [*OF S*] *independent_finite*)
obtain C **where** $C \subseteq T$ **and** *Corth*: *pairwise orthogonal* C
and *C1*:$\bigwedge x.\ x \in C \Longrightarrow$ *norm* $x = 1$
and *independent* C *finite* C *card* $C =$ *dim* T *span* $C = T$
by (*metis orthonormal_basis_subspace* [*OF T*] *independent_finite*)
obtain *fb* **where** *fb* ‘ $B \subseteq C$ *inj_on fb* B
by (*metis* ‹*card* $B =$ *dim* S› ‹*card* $C =$ *dim* T› ‹*finite* B› ‹*finite* C› *card_le_inj*
d)
then have *pairwise_orth_fb*: *pairwise* $(\lambda v\ j.\ orthogonal\ (fb\ v)\ (fb\ j))\ B$
using *Corth* **unfolding** *pairwise_def inj_on_def*
by (*blast intro*: *orthogonal_clauses*)
obtain f **where** *linear* f **and** *ffb*: $\bigwedge x.\ x \in B \Longrightarrow f\ x = fb\ x$
using *linear_independent_extend* ‹*independent* B› **by** *fastforce*
have *span* $(f$ ‘ $B) \subseteq$ *span* C
by (*metis* ‹*fb* ‘ $B \subseteq C$› *ffb image_cong span_mono*)
then have f ‘ $S \subseteq T$
unfolding ‹*span* $B = S$› ‹*span* $C = T$› *span_linear_image*[*OF* ‹*linear* f›] **.**
have [*simp*]: $\bigwedge x.\ x \in B \Longrightarrow$ *norm* $(fb\ x) =$ *norm* x
using *B1 C1* ‹*fb* ‘ $B \subseteq C$› **by** *auto*
have *norm* $(f\ x) =$ *norm* x **if** $x \in S$ **for** x
proof −
interpret *linear* f **by** *fact*
obtain a **where** x: $x = (\sum v \in B.\ a\ v *_R v)$
using ‹*finite* B› ‹*span* $B = S$› ‹$x \in S$› *span_finite* **by** *fastforce*
have *norm* $(f\ x)\hat{}2 =$ *norm* $(\sum v{\in}B.\ a\ v *_R fb\ v)\hat{}2$ **by** (*simp add*: *sum scale*
ffb x)
also have ... $= (\sum v{\in}B.$ *norm* $((a\ v *_R fb\ v))\hat{}2)$
proof (*rule norm_sum_Pythagorean* [*OF* ‹*finite* B›])
show *pairwise* $(\lambda v\ j.\ orthogonal\ (a\ v *_R fb\ v)\ (a\ j *_R fb\ j))\ B$

```
      by (rule pairwise_ortho_scaleR [OF pairwise_orth_fb])
    qed
    also have ... = norm x ^2
      by (simp add: x pairwise_ortho_scaleR Borth norm_sum_Pythagorean [OF
‹finite B›])
    finally show ?thesis
      by (simp add: norm_eq_sqrt_inner)
  qed
  then show ?thesis
    by (meson ‹f ‘ S ⊆ T› ‹linear f› image_subset_iff_funcset that)
qed

proposition isometries_subspaces:
  fixes S :: 'a::euclidean_space set
    and T :: 'b::euclidean_space set
  assumes S: subspace S
      and T: subspace T
      and d: dim S = dim T
  obtains f g where linear f linear g f ‘ S = T g ‘ T = S
                    ⋀x. x ∈ S ⟹ norm(f x) = norm x
                    ⋀x. x ∈ T ⟹ norm(g x) = norm x
                    ⋀x. x ∈ S ⟹ g(f x) = x
                    ⋀x. x ∈ T ⟹ f(g x) = x
proof −
  obtain B where B ⊆ S and Borth: pairwise orthogonal B
           and B1: ⋀x. x ∈ B ⟹ norm x = 1
           and independent B finite B card B = dim S span B = S
    by (metis orthonormal_basis_subspace [OF S] independent_finite)
  obtain C where C ⊆ T and Corth: pairwise orthogonal C
           and C1:⋀x. x ∈ C ⟹ norm x = 1
           and independent C finite C card C = dim T span C = T
    by (metis orthonormal_basis_subspace [OF T] independent_finite)
  obtain fb where bij_betw fb B C
    by (metis ‹finite B› ‹finite C› bij_betw_iff_card ‹card B = dim S› ‹card C =
dim T› d)
  then have pairwise_orth_fb: pairwise (λv j. orthogonal (fb v) (fb j)) B
    using Corth unfolding pairwise_def inj_on_def bij_betw_def
    by (blast intro: orthogonal_clauses)
  obtain f where linear f and ffb: ⋀x. x ∈ B ⟹ f x = fb x
    using linear_independent_extend ‹independent B› by fastforce
  interpret f: linear f by fact
  define gb where gb ≡ inv_into B fb
  then have pairwise_orth_gb: pairwise (λv j. orthogonal (gb v) (gb j)) C
    using Borth ‹bij_betw fb B C› unfolding pairwise_def bij_betw_def by force
  obtain g where linear g and ggb: ⋀x. x ∈ C ⟹ g x = gb x
    using linear_independent_extend ‹independent C› by fastforce
  interpret g: linear g by fact
  have span (f ‘ B) ⊆ span C
    by (metis ‹bij_betw fb B C› bij_betw_imp_surj_on eq_iff ffb image_cong)
```

```
  then have f ‘ S ⊆ T
    unfolding ‹span B = S› ‹span C = T› span_linear_image[OF ‹linear f›] .
  have [simp]: ⋀x. x ∈ B ⟹ norm (fb x) = norm x
    using B1 C1 ‹bij_betw fb B C› bij_betw_imp_surj_on by fastforce
  have f [simp]: norm (f x) = norm x g (f x) = x if x ∈ S for x
  proof −
    obtain a where x: x = (∑ v ∈ B. a v *R v)
      using ‹finite B› ‹span B = S› ‹x ∈ S› span_finite by fastforce
    have f x = (∑ v ∈ B. f (a v *R v))
      using linear_sum [OF ‹linear f›] x by auto
    also have ... = (∑ v ∈ B. a v *R f v)
      by (simp add: f.sum f.scale)
    also have ... = (∑ v ∈ B. a v *R fb v)
      by (simp add: ffb cong: sum.cong)
    finally have *: f x = (∑ v∈B. a v *R fb v) .
    then have (norm (f x))^2 = (norm (∑ v∈B. a v *R fb v))^2 by simp
    also have ... = (∑ v∈B. norm ((a v *R fb v))^2)
    proof (rule norm_sum_Pythagorean [OF ‹finite B›])
      show pairwise (λv j. orthogonal (a v *R fb v) (a j *R fb j)) B
        by (rule pairwise_ortho_scaleR [OF pairwise_orth_fb])
    qed
    also have ... = (norm x)^2
      by (simp add: x pairwise_ortho_scaleR Borth norm_sum_Pythagorean [OF
‹finite B›])
    finally show norm (f x) = norm x
      by (simp add: norm_eq_sqrt_inner)
    have g (f x) = g (∑ v∈B. a v *R fb v) by (simp add: *)
    also have ... = (∑ v∈B. g (a v *R fb v))
      by (simp add: g.sum g.scale)
    also have ... = (∑ v∈B. a v *R g (fb v))
      by (simp add: g.scale)
    also have ... = (∑ v∈B. a v *R v)
    proof (rule sum.cong [OF refl])
      show a x *R g (fb x) = a x *R x if x ∈ B for x
        using that ‹bij_betw fb B C› bij_betwE bij_betw_inv_into_left gb_def ggb
by fastforce
    qed
    also have ... = x
      using x by blast
    finally show g (f x) = x .
  qed
  have [simp]: ⋀x. x ∈ C ⟹ norm (gb x) = norm x
   by (metis B1 C1 ‹bij_betw fb B C› bij_betw_imp_surj_on gb_def inv_into_into)
  have g [simp]: f (g x) = x if x ∈ T for x
  proof −
    obtain a where x: x = (∑ v ∈ C. a v *R v)
      using ‹finite C› ‹span C = T› ‹x ∈ T› span_finite by fastforce
    have g x = (∑ v ∈ C. g (a v *R v))
      by (simp add: x g.sum)
```

```
    also have ... = (∑ v ∈ C. a v *R g v)
      by (simp add: g.scale)
    also have ... = (∑ v ∈ C. a v *R gb v)
      by (simp add: ggb cong: sum.cong)
    finally have f (g x) = f (∑ v∈C. a v *R gb v) by simp
    also have ... = (∑ v∈C. f (a v *R gb v))
      by (simp add: f.scale f.sum)
    also have ... = (∑ v∈C. a v *R f (gb v))
      by (simp add: f.scale f.sum)
    also have ... = (∑ v∈C. a v *R v)
      using ‹bij_betw fb B C›
     by (simp add: bij_betw_def gb_def bij_betw_inv_into_right ffb inv_into_into)
    also have ... = x
      using x by blast
    finally show f (g x) = x .
  qed
  have gim: g ‘ T = S
    by (metis (full_types) S T ‹f ‘ S ⊆ T› d dim_eq_span dim_image_le f(2)
g.linear_axioms
        image_iff linear_subspace_image span_eq_iff subset_iff)
  have fim: f ‘ S = T
    using ‹g ‘ T = S› image_iff by fastforce
  have [simp]: norm (g x) = norm x if x ∈ T for x
    using fim that by auto
  show ?thesis
    by (rule that [OF ‹linear f› ‹linear g›]) (simp_all add: fim gim)
qed

corollary isometry_subspaces:
  fixes S :: 'a::euclidean_space set
    and T :: 'b::euclidean_space set
  assumes S: subspace S
      and T: subspace T
      and d: dim S = dim T
  obtains f where linear f f ‘ S = T ⋀x. x ∈ S ⟹ norm(f x) = norm x
using isometries_subspaces [OF assms]
by metis

corollary isomorphisms_UNIV_UNIV:
  assumes DIM('M) = DIM('N)
  obtains f::'M::euclidean_space ⇒'N::euclidean_space and g
  where linear f linear g
                ⋀x. norm(f x) = norm x ⋀y. norm(g y) = norm y
                ⋀x. g (f x) = x ⋀y. f(g y) = y
  using assms by (auto intro: isometries_subspaces [of UNIV::'M set UNIV::'N
set])

lemma homeomorphic_subspaces:
  fixes S :: 'a::euclidean_space set
```

```
    and T :: 'b::euclidean_space set
  assumes S: subspace S
      and T: subspace T
      and d: dim S = dim T
    shows S homeomorphic T
proof −
  obtain f g where linear f linear g f ‘ S = T g ‘ T = S
                   ⋀x. x ∈ S ⟹ g(f x) = x ⋀x. x ∈ T ⟹ f(g x) = x
    by (blast intro: isometries_subspaces [OF assms])
  then show ?thesis
    unfolding homeomorphic_def homeomorphism_def
    apply (rule_tac x=f in exI, rule_tac x=g in exI)
    apply (auto simp: linear_continuous_on linear_conv_bounded_linear)
    done
qed

lemma homeomorphic_affine_sets:
  assumes affine S affine T aff_dim S = aff_dim T
    shows S homeomorphic T
proof (cases S = {} ∨ T = {})
  case True with assms aff_dim_empty homeomorphic_empty show ?thesis
    by metis
next
  case False
  then obtain a b where ab: a ∈ S b ∈ T by auto
  then have ss: subspace ((+) (− a) ‘ S) subspace ((+) (− b) ‘ T)
    using affine_diffs_subspace assms by blast+
  have dd: dim ((+) (− a) ‘ S) = dim ((+) (− b) ‘ T)
    using assms ab by (simp add: aff_dim_eq_dim [OF hull_inc] image_def)
  have S homeomorphic ((+) (− a) ‘ S)
    by (fact homeomorphic_translation)
  also have ... homeomorphic ((+) (− b) ‘ T)
    by (rule homeomorphic_subspaces [OF ss dd])
  also have ... homeomorphic T
   using homeomorphic_translation [of T − b] by (simp add: homeomorphic_sym
[of T])
  finally show ?thesis .
qed
```

5.4.21 Retracts, in a general sense, preserve (co)homotopic triviality)

```
locale Retracts =
  fixes S h t k
  assumes conth: continuous_on S h
      and imh: h ‘ S = t
      and contk: continuous_on t k
      and imk: k ∈ t → S
      and idhk: ⋀y. y ∈ t ⟹ h(k y) = y
```

begin

lemma *homotopically_trivial_retraction_gen*:
assumes P: $\bigwedge f.\ [\![\mathit{continuous\_on}\ U\ f;\ f \in U \to t;\ Q\ f]\!] \Longrightarrow P(k \circ f)$
and Q: $\bigwedge f.\ [\![\mathit{continuous\_on}\ U\ f;\ f \in U \to S;\ P\ f]\!] \Longrightarrow Q(h \circ f)$
and *Qeq*: $\bigwedge h\ k.\ (\bigwedge x.\ x \in U \Longrightarrow h\ x = k\ x) \Longrightarrow Q\ h = Q\ k$
and *hom*: $\bigwedge f\ g.\ [\![\mathit{continuous\_on}\ U\ f;\ f \in U \to S;\ P\ f;$
$\mathit{continuous\_on}\ U\ g;\ g \in U \to S;\ P\ g]\!]$
$\Longrightarrow \mathit{homotopic\_with\_canon}\ P\ U\ S\ f\ g$
and *contf*: $\mathit{continuous\_on}\ U\ f$ **and** *imf*: $f \in U \to t$ **and** *Qf*: $Q\ f$
and *contg*: $\mathit{continuous\_on}\ U\ g$ **and** *img*: $g \in U \to t$ **and** *Qg*: $Q\ g$
shows $\mathit{homotopic\_with\_canon}\ Q\ U\ t\ f\ g$
proof −
have $\mathit{continuous\_on}\ U\ (k \circ f)$
by (*meson contf continuous_on_compose continuous_on_subset contk funcset_image imf*)
moreover have $(k \circ f)\ `\ U \subseteq S$
using *imf imk* **by** *fastforce*
moreover have $P\ (k \circ f)$
by (*simp add*: *P Qf contf imf*)
moreover have $\mathit{continuous\_on}\ U\ (k \circ g)$
by (*meson contg continuous_on_compose continuous_on_subset contk funcset_image img*)
moreover have $(k \circ g)\ `\ U \subseteq S$
using *img imk* **by** *fastforce*
moreover have $P\ (k \circ g)$
by (*simp add*: *P Qg contg img*)
ultimately have $\mathit{homotopic\_with\_canon}\ P\ U\ S\ (k \circ f)\ (k \circ g)$
by (*simp add*: *hom image_subset_iff*)
then have $\mathit{homotopic\_with\_canon}\ Q\ U\ t\ (h \circ (k \circ f))\ (h \circ (k \circ g))$
apply (*rule homotopic_with_compose_continuous_left* [*OF homotopic_with_mono*])
using *Q conth imh* **by** *force*+
then show *?thesis*
proof (*rule homotopic_with_eq*; *simp*)
show $\bigwedge h\ k.\ (\bigwedge x.\ x \in U \Longrightarrow h\ x = k\ x) \Longrightarrow Q\ h = Q\ k$
using *Qeq topspace_euclidean_subtopology* **by** *blast*
show $\bigwedge x.\ x \in U \Longrightarrow f\ x = h\ (k\ (f\ x))\ \bigwedge x.\ x \in U \Longrightarrow g\ x = h\ (k\ (g\ x))$
using *idhk imf img* **by** *fastforce*+
qed
qed

lemma *homotopically_trivial_retraction_null_gen*:
assumes P: $\bigwedge f.\ [\![\mathit{continuous\_on}\ U\ f;\ f \in U \to t;\ Q\ f]\!] \Longrightarrow P(k \circ f)$
and Q: $\bigwedge f.\ [\![\mathit{continuous\_on}\ U\ f;\ f \in U \to S;\ P\ f]\!] \Longrightarrow Q(h \circ f)$
and *Qeq*: $\bigwedge h\ k.\ (\bigwedge x.\ x \in U \Longrightarrow h\ x = k\ x) \Longrightarrow Q\ h = Q\ k$
and *hom*: $\bigwedge f.\ [\![\mathit{continuous\_on}\ U\ f;\ f \in U \to S;\ P\ f]\!]$
$\Longrightarrow \exists c.\ \mathit{homotopic\_with\_canon}\ P\ U\ S\ f\ (\lambda x.\ c)$
and *contf*: $\mathit{continuous\_on}\ U\ f$ **and** *imf*:$f \in U \to t$ **and** *Qf*: $Q\ f$

obtains c **where** *homotopic_with_canon* Q U t f $(\lambda x.\ c)$
proof −
have *feq*: $\bigwedge x.\ x \in U \Longrightarrow (h \circ (k \circ f))\ x = f\ x$ **using** *idhk imf* **by** *auto*
have *continuous_on* U $(k \circ f)$
by (*meson contf continuous_on_compose continuous_on_subset contk funcset_image imf*)
moreover have $(k \circ f) \in U \to S$
using *imf imk* **by** *fastforce*
moreover have P $(k \circ f)$
by (*simp add*: P *Qf contf imf*)
ultimately obtain c **where** *homotopic_with_canon* P U S $(k \circ f)$ $(\lambda x.\ c)$
by (*metis hom*)
then have *homotopic_with_canon* Q U t $(h \circ (k \circ f))$ $(h \circ (\lambda x.\ c))$
apply (*rule homotopic_with_compose_continuous_left* [*OF homotopic_with_mono*])
using Q *conth imh* **by** *force+*
then have *homotopic_with_canon* Q U t f $(\lambda x.\ h\ c)$
proof (*rule homotopic_with_eq*)
show $\bigwedge x.\ x \in$ *topspace* (*top_of_set* U) $\Longrightarrow f\ x = (h \circ (k \circ f))\ x$
using *feq* **by** *auto*
show $\bigwedge h\ k.\ (\bigwedge x.\ x \in$ *topspace* (*top_of_set* U) $\Longrightarrow h\ x = k\ x) \Longrightarrow Q\ h = Q\ k$
using *Qeq topspace_euclidean_subtopology* **by** *blast*
qed *auto*
then show *?thesis*
using *that* **by** *blast*
qed

lemma *cohomotopically_trivial_retraction_gen*:
assumes P: $\bigwedge f.\ [\![$*continuous_on* $t\ f$; $f \in t \to U$; $Q\ f]\!] \Longrightarrow P(f \circ h)$
and Q: $\bigwedge f.\ [\![$*continuous_on* $S\ f$; $f \in S \to U$; $P\ f]\!] \Longrightarrow Q(f \circ k)$
and *Qeq*: $\bigwedge h\ k.\ (\bigwedge x.\ x \in t \Longrightarrow h\ x = k\ x) \Longrightarrow Q\ h = Q\ k$
and *hom*: $\bigwedge f\ g.\ [\![$*continuous_on* $S\ f$; $f \in S \to U$; $P\ f$;
continuous_on $S\ g$; $g \in S \to U$; $P\ g]\!]$
$\Longrightarrow$ *homotopic_with_canon* $P\ S\ U\ f\ g$
and *contf*: *continuous_on* $t\ f$ **and** *imf*: $f \in t \to U$ **and** *Qf*: $Q\ f$
and *contg*: *continuous_on* $t\ g$ **and** *img*: $g \in t \to U$ **and** *Qg*: $Q\ g$
shows *homotopic_with_canon* $Q\ t\ U\ f\ g$
proof −
have *feq*: $\bigwedge x.\ x \in t \Longrightarrow (f \circ h \circ k)\ x = f\ x$ **using** *idhk imf* **by** *auto*
have *geq*: $\bigwedge x.\ x \in t \Longrightarrow (g \circ h \circ k)\ x = g\ x$ **using** *idhk img* **by** *auto*
have *continuous_on* S $(f \circ h)$
using *contf conth continuous_on_compose imh* **by** *blast*
moreover have $(f \circ h) \in S \to U$
using *imf imh* **by** *fastforce*
moreover have P $(f \circ h)$
by (*simp add*: P *Qf contf imf*)
moreover have *continuous_on* S $(g \circ h)$
using *contg continuous_on_compose continuous_on_subset conth imh* **by** *blast*
moreover have $(g \circ h) \in S \to U$
using *img imh* **by** *fastforce*

```
  moreover have P (g ∘ h)
    by (simp add: P Qg contg img)
  ultimately have homotopic_with_canon P S U (f ∘ h) (g ∘ h)
    by (simp add: hom)
  then have homotopic_with_canon Q t U (f ∘ h ∘ k) (g ∘ h ∘ k)
   apply (rule homotopic_with_compose_continuous_right [OF homotopic_with_mono])
    using Q contk imk by force+
  then show ?thesis
  proof (rule homotopic_with_eq)
    show f x = (f ∘ h ∘ k) x g x = (g ∘ h ∘ k) x
      if x ∈ topspace (top_of_set t) for x
      using feq geq that by force+
  qed (use Qeq topspace_euclidean_subtopology in blast)
qed

lemma cohomotopically_trivial_retraction_null_gen:
  assumes P: ⋀f. ⟦continuous_on t f; f ∈ t → U; Q f⟧ ⟹ P(f ∘ h)
      and Q: ⋀f. ⟦continuous_on S f; f ∈ S → U; P f⟧ ⟹ Q(f ∘ k)
      and Qeq: ⋀h k. (⋀x. x ∈ t ⟹ h x = k x) ⟹ Q h = Q k
      and hom: ⋀f g. ⟦continuous_on S f; f ∈ S → U; P f⟧
                     ⟹ ∃c. homotopic_with_canon P S U f (λx. c)
      and contf: continuous_on t f and imf: f ∈ t → U and Qf: Q f
  obtains c where homotopic_with_canon Q t U f (λx. c)
proof −
  have feq: ⋀x. x ∈ t ⟹ (f ∘ h ∘ k) x = f x using idhk imf by auto
  have continuous_on S (f ∘ h)
    using contf conth continuous_on_compose imh by blast
  moreover have (f ∘ h) ∈ S → U
    using imf imh by fastforce
  moreover have P (f ∘ h)
    by (simp add: P Qf contf imf)
  ultimately obtain c where homotopic_with_canon P S U (f ∘ h) (λx. c)
    by (metis hom)
  then have §: homotopic_with_canon Q t U (f ∘ h ∘ k) ((λx. c) ∘ k)
  proof (rule homotopic_with_compose_continuous_right [OF homotopic_with_mono])
    show ⋀h. ⟦continuous_map (top_of_set S) (top_of_set U) h; P h⟧ ⟹ Q (h
∘ k)
      using Q by auto
  qed (use contk imk in force)+
  moreover have homotopic_with_canon Q t U f (λx. c)
    using homotopic_with_eq [OF §] feq Qeq by fastforce
  ultimately show ?thesis
    using that by blast
qed

end

lemma simply_connected_retraction_gen:
  shows ⟦simply_connected S; continuous_on S h; h ‘ S = T;
```

continuous_on T k; $k \in T \to S$; $\bigwedge y.\ y \in T \Longrightarrow h(k\ y) = y$⟧
⟹ *simply_connected T*
apply (*simp add*: *simply_connected_def path_def path_image_def homotopic_loops_def*, *clarify*)
apply (*rule Retracts.homotopically_trivial_retraction_gen*
[*of S h _ k _ λp. pathfinish p = pathstart p λp. pathfinish p = pathstart p*])
apply (*simp_all add*: *Retracts_def pathfinish_def pathstart_def image_subset_iff_funcset*)
done

lemma *homeomorphic_simply_connected*:
⟦*S homeomorphic T*; *simply_connected S*⟧ ⟹ *simply_connected T*
by (*auto simp*: *homeomorphic_def homeomorphism_def intro*: *simply_connected_retraction_gen*)

lemma *homeomorphic_simply_connected_eq*:
S homeomorphic T ⟹ (*simply_connected S* ⟷ *simply_connected T*)
by (*metis homeomorphic_simply_connected homeomorphic_sym*)

5.4.22 Homotopy equivalence

5.4.23 Homotopy equivalence of topological spaces.

definition *homotopy_equivalent_space*
(**infix** *homotopy'_equivalent'_space 50*)
where *X homotopy_equivalent_space Y* ≡
(∃*f g*. *continuous_map X Y f* ∧
continuous_map Y X g ∧
homotopic_with (λ*x*. *True*) *X X* (*g* ∘ *f*) *id* ∧
homotopic_with (λ*x*. *True*) *Y Y* (*f* ∘ *g*) *id*)

lemma *homeomorphic_imp_homotopy_equivalent_space*:
X homeomorphic_space Y ⟹ *X homotopy_equivalent_space Y*
unfolding *homeomorphic_space_def homotopy_equivalent_space_def*
apply (*erule ex_forward*)+
by (*simp add*: *homotopic_with_equal homotopic_with_sym homeomorphic_maps_def*)

lemma *homotopy_equivalent_space_refl*:
X homotopy_equivalent_space X
by (*simp add*: *homeomorphic_imp_homotopy_equivalent_space homeomorphic_space_refl*)

lemma *homotopy_equivalent_space_sym*:
X homotopy_equivalent_space Y ⟷ *Y homotopy_equivalent_space X*
by (*meson homotopy_equivalent_space_def*)

lemma *homotopy_eqv_trans* [*trans*]:
assumes *1*: *X homotopy_equivalent_space Y* **and** *2*: *Y homotopy_equivalent_space U*
shows *X homotopy_equivalent_space U*
proof −
obtain *f1 g1* **where** *f1*: *continuous_map X Y f1*
and *g1*: *continuous_map Y X g1*

and *hom1*: *homotopic_with* (λ*x*. *True*) *X X* (*g1* ∘ *f1*) *id*
homotopic_with (λ*x*. *True*) *Y Y* (*f1* ∘ *g1*) *id*
using *1* **by** (*auto simp*: *homotopy_equivalent_space_def*)
obtain *f2 g2* **where** *f2*: *continuous_map Y U f2*
and *g2*: *continuous_map U Y g2*
and *hom2*: *homotopic_with* (λ*x*. *True*) *Y Y* (*g2* ∘ *f2*) *id*
homotopic_with (λ*x*. *True*) *U U* (*f2* ∘ *g2*) *id*
using *2* **by** (*auto simp*: *homotopy_equivalent_space_def*)
have *homotopic_with* (λ*f*. *True*) *X Y* (*g2* ∘ *f2* ∘ *f1*) (*id* ∘ *f1*)
using *f1 hom2*(*1*) *homotopic_with_compose_continuous_map_right* **by** *metis*
then have *homotopic_with* (λ*f*. *True*) *X Y* (*g2* ∘ (*f2* ∘ *f1*)) (*id* ∘ *f1*)
by (*simp add*: *o_assoc*)
then have *homotopic_with* (λ*x*. *True*) *X X*
(*g1* ∘ (*g2* ∘ (*f2* ∘ *f1*))) (*g1* ∘ (*id* ∘ *f1*))
by (*simp add*: *g1 homotopic_with_compose_continuous_map_left*)
moreover have *homotopic_with* (λ*x*. *True*) *X X* (*g1* ∘ *id* ∘ *f1*) *id*
using *hom1* **by** *simp*
ultimately have *SS*: *homotopic_with* (λ*x*. *True*) *X X* (*g1* ∘ *g2* ∘ (*f2* ∘ *f1*)) *id*
by (*metis comp_assoc homotopic_with_trans id_comp*)
have *homotopic_with* (λ*f*. *True*) *U Y* (*f1* ∘ *g1* ∘ *g2*) (*id* ∘ *g2*)
using *g2 hom1*(*2*) *homotopic_with_compose_continuous_map_right* **by** *fastforce*
then have *homotopic_with* (λ*f*. *True*) *U Y* (*f1* ∘ (*g1* ∘ *g2*)) (*id* ∘ *g2*)
by (*simp add*: *o_assoc*)
then have *homotopic_with* (λ*x*. *True*) *U U*
(*f2* ∘ (*f1* ∘ (*g1* ∘ *g2*))) (*f2* ∘ (*id* ∘ *g2*))
by (*simp add*: *f2 homotopic_with_compose_continuous_map_left*)
moreover have *homotopic_with* (λ*x*. *True*) *U U* (*f2* ∘ *id* ∘ *g2*) *id*
using *hom2* **by** *simp*
ultimately have *UU*: *homotopic_with* (λ*x*. *True*) *U U* (*f2* ∘ *f1* ∘ (*g1* ∘ *g2*)) *id*
by (*simp add*: *fun.map_comp hom2*(*2*) *homotopic_with_trans*)
show *?thesis*
unfolding *homotopy_equivalent_space_def*
by (*blast intro*: *f1 f2 g1 g2 continuous_map_compose SS UU*)
qed

lemma *deformation_retraction_imp_homotopy_equivalent_space*:
⟦*homotopic_with* (λ*x*. *True*) *X X* (*S* ∘ *r*) *id*; *retraction_maps X Y r S*⟧
⟹ *X homotopy_equivalent_space Y*
unfolding *homotopy_equivalent_space_def retraction_maps_def*
using *homotopic_with_id2* **by** *fastforce*

lemma *deformation_retract_imp_homotopy_equivalent_space*:
⟦*homotopic_with* (λ*x*. *True*) *X X r id*; *retraction_maps X Y r id*⟧
⟹ *X homotopy_equivalent_space Y*
using *deformation_retraction_imp_homotopy_equivalent_space* **by** *force*

lemma *deformation_retract_of_space*:
S ⊆ *topspace X* ∧

(∃ *r*. *homotopic_with* (λ*x*. *True*) *X X id r* ∧ *retraction_maps X* (*subtopology X S*) *r id*) ⟷
S retract_of_space X ∧ (∃ *f*. *homotopic_with* (λ*x*. *True*) *X X id f* ∧ *f* ‘ (*topspace X*) ⊆ *S*)
proof (*cases S* ⊆ *topspace X*)
case *True*
moreover have (∃ *r*. *homotopic_with* (λ*x*. *True*) *X X id r* ∧ *retraction_maps X* (*subtopology X S*) *r id*)
⟷ (*S retract_of_space X* ∧ (∃ *f*. *homotopic_with* (λ*x*. *True*) *X X id f* ∧ *f* ‘ *topspace X* ⊆ *S*))
unfolding *retract_of_space_def*
proof *safe*
fix *f r*
assume *f*: *homotopic_with* (λ*x*. *True*) *X X id f*
and *fS*: *f* ‘ *topspace X* ⊆ *S*
and *r*: *continuous_map X* (*subtopology X S*) *r*
and *req*: ∀ *x*∈*S*. *r x* = *x*
show ∃ *r*. *homotopic_with* (λ*x*. *True*) *X X id r* ∧ *retraction_maps X* (*subtopology X S*) *r id*
proof (*intro exI conjI*)
have *homotopic_with* (λ*x*. *True*) *X X f r*
proof (*rule homotopic_with_eq*)
show *homotopic_with* (λ*x*. *True*) *X X* (*r* ∘ *f*) (*r* ∘ *id*)
by (*metis continuous_map_into_fulltopology f homotopic_with_compose_continuous_map_left homotopic_with_symD r*)
show *f x* = (*r* ∘ *f*) *x* **if** *x* ∈ *topspace X* **for** *x*
using *that fS req* **by** *auto*
qed *auto*
then show *homotopic_with* (λ*x*. *True*) *X X id r*
by (*rule homotopic_with_trans* [*OF f*])
next
show *retraction_maps X* (*subtopology X S*) *r id*
by (*simp add*: *r req retraction_maps_def*)
qed
qed (*use True* **in** ‹*auto simp*: *retraction_maps_def topspace_subtopology_subset continuous_map_in_subtopology*›)
ultimately show *?thesis* **by** *simp*
qed (*auto simp*: *retract_of_space_def retraction_maps_def*)

5.4.24 Contractible spaces

The definition (which agrees with "contractible" on subsets of Euclidean space) is a little cryptic because we don’t in fact assume that the constant "a" is in the space. This forces the convention that the empty space / set is contractible, avoiding some special cases.

definition *contractible_space* **where**
contractible_space X ≡ ∃ *a*. *homotopic_with* (λ*x*. *True*) *X X id* (λ*x*. *a*)

lemma *contractible_space_top_of_set* [*simp*]:*contractible_space* (*top_of_set S*)
⟷ *contractible S*
by (*auto simp*: *contractible_space_def contractible_def*)

lemma *contractible_space_empty* [*simp*]:
contractible_space trivial_topology
unfolding *contractible_space_def homotopic_with_def*
apply (*rule_tac x*=*undefined* **in** *exI*)
apply (*rule_tac x*=λ(*t*,*x*). *if t* = *0 then x else undefined* **in** *exI*)
apply (*auto simp*: *continuous_map_on_empty*)
done

lemma *contractible_space_singleton* [*simp*]:
contractible_space (*discrete_topology*{*a*})
unfolding *contractible_space_def homotopic_with_def*
apply (*rule_tac x*=*a* **in** *exI*)
apply (*rule_tac x*=λ(*t*,*x*). *if t* = *0 then x else a* **in** *exI*)
apply (*auto intro*: *continuous_map_eq* [**where** *f* = λ*z*. *a*])
done

lemma *contractible_space_subset_singleton*:
topspace X ⊆ {*a*} ⟹ *contractible_space X*
by (*metis contractible_space_empty contractible_space_singleton null_topspace_iff_trivial subset_singletonD subtopology_eq_discrete_topology_sing*)

lemma *contractible_space_subtopology_singleton* [*simp*]:
contractible_space (*subtopology X* {*a*})
by (*meson contractible_space_subset_singleton insert_subset path_connectedin_singleton path_connectedin_subtopology subsetI*)

lemma *contractible_space*:
contractible_space X ⟷
X = *trivial_topology* ∨
(∃ *a* ∈ *topspace X*. *homotopic_with* (λ*x*. *True*) *X X id* (λ*x*. *a*))
proof (*cases X* = *trivial_topology*)
case *False*
then show *?thesis*
using *homotopic_with_imp_continuous_maps* **by** (*fastforce simp*: *contractible_space_def*)
qed (*simp add*: *contractible_space_empty*)

lemma *contractible_imp_path_connected_space*:
assumes *contractible_space X* **shows** *path_connected_space X*
proof (*cases X* = *trivial_topology*)
case *False*
have ∗: *path_connected_space X*
if *a* ∈ *topspace X* **and** *conth*: *continuous_map* (*prod_topology* (*top_of_set* {*0..1*}) *X*) *X h*
and *h*: ∀ *x*. *h* (*0*, *x*) = *x* ∀ *x*. *h* (*1*, *x*) = *a*
for *a* **and** *h* :: *real* × *'a* ⇒ *'a*

proof −
have *path_component_of X b a* **if** *b* ∈ *topspace X* **for** *b*
unfolding *path_component_of_def*
proof (*intro exI conjI*)
let *?g* = *h* ∘ (λ*x*. (*x*,*b*))
show *pathin X ?g*
unfolding *pathin_def*
proof (*rule continuous_map_compose* [*OF* _ *conth*])
show *continuous_map* (*top_of_set* {*0..1*}) (*prod_topology* (*top_of_set* {*0..1*}) *X*) (λ*x*. (*x*, *b*))
using *that* **by** (*auto intro*!: *continuous_intros*)
qed
qed (*use h* **in** *auto*)
then show *?thesis*
by (*metis path_component_of_equiv path_connected_space_iff_path_component*)
qed
show *?thesis*
using *assms False* **by** (*auto simp*: *contractible_space homotopic_with_def* ∗)
qed (*simp add*: *path_connected_space_topspace_empty*)

lemma *contractible_imp_connected_space*:
contractible_space X ⟹ *connected_space X*
by (*simp add*: *contractible_imp_path_connected_space path_connected_imp_connected_space*)

lemma *contractible_space_alt*:
contractible_space X ⟷ (∀ *a* ∈ *topspace X*. *homotopic_with* (λ*x*. *True*) *X X id* (λ*x*. *a*)) (**is** *?lhs* = *?rhs*)
proof
assume *X*: *?lhs*
then obtain *a* **where** *a*: *homotopic_with* (λ*x*. *True*) *X X id* (λ*x*. *a*)
by (*auto simp*: *contractible_space_def*)
show *?rhs*
proof
show *homotopic_with* (λ*x*. *True*) *X X id* (λ*x*. *b*) **if** *b* ∈ *topspace X* **for** *b*
proof (*rule homotopic_with_trans* [*OF a*])
show *homotopic_with* (λ*x*. *True*) *X X* (λ*x*. *a*) (λ*x*. *b*)
using *homotopic_constant_maps path_connected_space_imp_path_component_of*
by (*metis X a contractible_imp_path_connected_space homotopic_with_sym homotopic_with_trans path_component_of_equiv that*)
qed
qed
next
assume *R*: *?rhs*
then show *?lhs*
using *contractible_space_def* **by** *fastforce*
qed

lemma *compose_const* [*simp*]: *f* ∘ (λ*x*. *a*) = (λ*x*. *f a*) (λ*x*. *a*) ∘ *g* = (λ*x*. *a*)

by (*simp_all add*: *o_def*)

lemma *nullhomotopic_through_contractible_space*:
assumes *f*: *continuous_map X Y f* **and** *g*: *continuous_map Y Z g* **and** *Y*: *contractible_space Y*
obtains *c* **where** *homotopic_with* (λ*h*. *True*) *X Z* (*g* ∘ *f*) (λ*x*. *c*)
proof −
obtain *b* **where** *b*: *homotopic_with* (λ*x*. *True*) *Y Y id* (λ*x*. *b*)
using *Y* **by** (*auto simp*: *contractible_space_def*)
show *thesis*
using *homotopic_with_compose_continuous_map_right*
[*OF homotopic_with_compose_continuous_map_left* [*OF b g*] *f*]
by (*force simp*: *that*)
qed

lemma *nullhomotopic_into_contractible_space*:
assumes *f*: *continuous_map X Y f* **and** *Y*: *contractible_space Y*
obtains *c* **where** *homotopic_with* (λ*h*. *True*) *X Y f* (λ*x*. *c*)
using *nullhomotopic_through_contractible_space* [*OF f _ Y*]
by (*metis continuous_map_id id_comp*)

lemma *nullhomotopic_from_contractible_space*:
assumes *f*: *continuous_map X Y f* **and** *X*: *contractible_space X*
obtains *c* **where** *homotopic_with* (λ*h*. *True*) *X Y f* (λ*x*. *c*)
using *nullhomotopic_through_contractible_space* [*OF _ f X*]
by (*metis comp_id continuous_map_id*)

lemma *homotopy_dominated_contractibility*:
assumes *f*: *continuous_map X Y f* **and** *g*: *continuous_map Y X g*
and *hom*: *homotopic_with* (λ*x*. *True*) *Y Y* (*f* ∘ *g*) *id* **and** *X*: *contractible_space X*
shows *contractible_space Y*
proof −
obtain *c* **where** *c*: *homotopic_with* (λ*h*. *True*) *X Y f* (λ*x*. *c*)
using *nullhomotopic_from_contractible_space* [*OF f X*] **.**
have *homotopic_with* (λ*x*. *True*) *Y Y* (*f* ∘ *g*) (λ*x*. *c*)
using *homotopic_with_compose_continuous_map_right* [*OF c g*] **by** *fastforce*
then have *homotopic_with* (λ*x*. *True*) *Y Y id* (λ*x*. *c*)
using *homotopic_with_trans* [*OF _ hom*] *homotopic_with_symD* **by** *blast*
then show *?thesis*
unfolding *contractible_space_def* **..**
qed

lemma *homotopy_equivalent_space_contractibility*:
X homotopy_equivalent_space Y ⟹ (*contractible_space X* ⟷ *contractible_space Y*)
unfolding *homotopy_equivalent_space_def*
by (*blast intro*: *homotopy_dominated_contractibility*)

lemma *homeomorphic_space_contractibility*:
X homeomorphic_space Y
⟹ (*contractible_space X* ⟷ *contractible_space Y*)
by (*simp add*: *homeomorphic_imp_homotopy_equivalent_space homotopy_equivalent_space_contractibility*)

lemma *homotopic_through_contractible_space*:
continuous_map X Y f ∧
continuous_map X Y f′ ∧
continuous_map Y Z g ∧
continuous_map Y Z g′ ∧
contractible_space Y ∧ *path_connected_space Z*
⟹ *homotopic_with* (λ*h*. *True*) *X Z* (*g* ∘ *f*) (*g′* ∘ *f′*)
using *nullhomotopic_through_contractible_space* [*of X Y f Z g*]
using *nullhomotopic_through_contractible_space* [*of X Y f′ Z g′*]
by (*smt* (*verit*) *continuous_map_const homotopic_constant_maps homotopic_with_imp_continuous_maps homotopic_with_symD homotopic_with_trans path_connected_space_imp_path_component_of*)

lemma *homotopic_from_contractible_space*:
continuous_map X Y f ∧ *continuous_map X Y g* ∧
contractible_space X ∧ *path_connected_space Y*
⟹ *homotopic_with* (λ*x*. *True*) *X Y f g*
by (*metis comp_id continuous_map_id homotopic_through_contractible_space*)

lemma *homotopic_into_contractible_space*:
continuous_map X Y f ∧ *continuous_map X Y g* ∧
contractible_space Y
⟹ *homotopic_with* (λ*x*. *True*) *X Y f g*
by (*metis continuous_map_id contractible_imp_path_connected_space homotopic_through_contractible_space id_comp*)

lemma *contractible_eq_homotopy_equivalent_singleton_subtopology*:
contractible_space X ⟷
X = *trivial_topology* ∨ (∃ *a* ∈ *topspace X*. *X homotopy_equivalent_space* (*subtopology X* {*a*}))(**is** *?lhs* = *?rhs*)
proof (*cases X* = *trivial_topology*)
case *False*
show *?thesis*
proof
assume *?lhs*
then obtain *a* **where** *a*: *homotopic_with* (λ*x*. *True*) *X X id* (λ*x*. *a*)
by (*auto simp*: *contractible_space_def*)
then have *a* ∈ *topspace X*
by (*metis False continuous_map_const homotopic_with_imp_continuous_maps*)
then have *homotopic_with* (λ*x*. *True*) (*subtopology X* {*a*}) (*subtopology X* {*a*}) *id* (λ*x*. *a*)
using *connectedin_absolute connectedin_sing contractible_space_alt contractible_space_subtopology_singleton* **by** *fastforce*
then have *X homotopy_equivalent_space subtopology X* {*a*}
unfolding *homotopy_equivalent_space_def* **using** ‹*a* ∈ *topspace X*›

by (*metis* (*full_types*) *a comp_id continuous_map_const continuous_map_id_subt empty_subsetI homotopic_with_symD*
id_comp insertI1 insert_subset topspace_subtopology_subset)
with ‹*a* ∈ *topspace X*› **show** *?rhs*
by *blast*
next
assume *?rhs*
then show *?lhs*
by (*meson False contractible_space_subtopology_singleton homotopy_equivalent_space_contractibili*
qed
qed (*simp add*: *contractible_space_empty*)

lemma *contractible_space_retraction_map_image*:
assumes *retraction_map X Y f* **and** *X*: *contractible_space X*
shows *contractible_space Y*
proof −
obtain *g* **where** *f*: *continuous_map X Y f* **and** *g*: *continuous_map Y X g* **and** *fg*: ∀ *y* ∈ *topspace Y*. *f*(*g y*) = *y*
using *assms* **by** (*auto simp*: *retraction_map_def retraction_maps_def*)
obtain *a* **where** *a*: *homotopic_with* (λ*x*. *True*) *X X id* (λ*x*. *a*)
using *X* **by** (*auto simp*: *contractible_space_def*)
have *homotopic_with* (λ*x*. *True*) *Y Y id* (λ*x*. *f a*)
proof (*rule homotopic_with_eq*)
show *homotopic_with* (λ*x*. *True*) *Y Y* (*f* ∘ *id* ∘ *g*) (*f* ∘ (λ*x*. *a*) ∘ *g*)
using *f g a homotopic_with_compose_continuous_map_left homotopic_with_compose_continuous_*
by *metis*
qed (*use fg* **in** *auto*)
then show *?thesis*
unfolding *contractible_space_def* **by** *blast*
qed

lemma *contractible_space_prod_topology*:
contractible_space(*prod_topology X Y*) ⟷
X = *trivial_topology* ∨ *Y* = *trivial_topology* ∨ *contractible_space X* ∧ *contractible_space Y*
proof (*cases X* = *trivial_topology* ∨ *Y* = *trivial_topology*)
case *True*
then have (*prod_topology X Y*) = *trivial_topology*
by *simp*
then show *?thesis*
by (*auto simp*: *contractible_space_empty*)
next
case *False*
have *contractible_space*(*prod_topology X Y*) ⟷ *contractible_space X* ∧ *contractible_space Y*
proof *safe*
assume *XY*: *contractible_space* (*prod_topology X Y*)
with *False* **have** *retraction_map* (*prod_topology X Y*) *X fst*
by (*auto simp*: *contractible_space False retraction_map_fst*)

```
    then show contractible_space X
      by (rule contractible_space_retraction_map_image [OF _ XY])
    have retraction_map (prod_topology X Y) Y snd
     using False XY by (auto simp: contractible_space False retraction_map_snd)
    then show contractible_space Y
      by (rule contractible_space_retraction_map_image [OF _ XY])
  next
    assume contractible_space X and contractible_space Y
    with False obtain a b
      where a ∈ topspace X and a: homotopic_with (λx. True) X X id (λx. a)
        and b ∈ topspace Y and b: homotopic_with (λx. True) Y Y id (λx. b)
      by (auto simp: contractible_space)
    with False show contractible_space (prod_topology X Y)
      apply (simp add: contractible_space)
      apply (rule_tac x=a in bexI)
       apply (rule_tac x=b in bexI)
      using homotopic_with_prod_topology [OF a b]
      apply (metis (no_types, lifting) case_prod_Pair case_prod_beta' eq_id_iff)
       apply auto
      done
  qed
  with False show ?thesis
    by auto
qed


lemma contractible_space_product_topology:
  contractible_space(product_topology X I) ⟷
    (product_topology X I) = trivial_topology ∨ (∀ i ∈ I. contractible_space(X i))
proof (cases (product_topology X I) = trivial_topology)
  case False
  have 1: contractible_space (X i)
    if XI: contractible_space (product_topology X I) and i ∈ I
    for i
  proof (rule contractible_space_retraction_map_image [OF _ XI])
    show retraction_map (product_topology X I) (X i) (λx. x i)
      using False by (simp add: retraction_map_product_projection ‹i ∈ I›)
  qed
  have 2: contractible_space (product_topology X I)
    if x ∈ topspace (product_topology X I) and cs: ∀ i∈I. contractible_space (X i)
    for x :: 'a ⇒ 'b
  proof −
   obtain f where f: ⋀i. i∈I ⟹ homotopic_with (λx. True) (X i) (X i) id (λx.
f i)
      using cs unfolding contractible_space_def by metis
    have homotopic_with (λx. True)
                  (product_topology X I) (product_topology X I) id (λx. restrict
f I)
      by (rule homotopic_with_eq [OF homotopic_with_product_topology [OF f]])
```

```
(auto)
    then show ?thesis
      by (auto simp: contractible_space_def)
  qed
  show ?thesis
    using False 1 2 by (meson equals0I subtopology_eq_discrete_topology_empty)
qed auto


lemma contractible_space_subtopology_euclideanreal [simp]:
  contractible_space(subtopology euclideanreal S) ⟷ is_interval S
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then have path_connectedin (subtopology euclideanreal S) S
   using contractible_imp_path_connected_space path_connectedin_topspace path_connectedin_absolut
    by (simp add: contractible_imp_path_connected)
  then show ?rhs
    by (simp add: is_interval_path_connected_1)
next
  assume ?rhs
  then have convex S
    by (simp add: is_interval_convex_1)
  show ?lhs
  proof (cases S = {})
    case False
    then obtain z where z ∈ S
      by blast
    show ?thesis
      unfolding contractible_space_def homotopic_with_def
    proof (intro exI conjI allI)
      note § = convexD [OF ‹convex S›, simplified]
      show continuous_map (prod_topology (top_of_set {0..1}) (top_of_set S))
(top_of_set S)
                          (λ(t,x). (1 − t) ∗ x + t ∗ z)
        using ‹z ∈ S›
        by (auto simp: case_prod_unfold intro!: continuous_intros §)
    qed auto
  qed (simp add: contractible_space_empty)
qed


corollary contractible_space_euclideanreal: contractible_space euclideanreal
proof −
  have contractible_space (subtopology euclideanreal UNIV)
    using contractible_space_subtopology_euclideanreal by blast
  then show ?thesis
    by simp
qed
```

abbreviation *homotopy_eqv* :: *'a::topological_space set* ⇒ *'b::topological_space set* ⇒ *bool*
(**infix** *homotopy'_eqv 50*)
where *S homotopy_eqv T* ≡ *top_of_set S homotopy_equivalent_space top_of_set T*

lemma *homeomorphic_imp_homotopy_eqv*: *S homeomorphic T* ⟹ *S homotopy_eqv T*
unfolding *homeomorphic_def homeomorphism_def homotopy_equivalent_space_def*
by (*metis continuous_map_subtopology_eu homotopic_with_id2 openin_imp_subset openin_subtopology_self topspace_euclidean_subtopology*)

lemma *homotopy_eqv_inj_linear_image*:
fixes *f* :: *'a::euclidean_space* ⇒ *'b::euclidean_space*
assumes *linear f inj f*
shows (*f* ‘ *S*) *homotopy_eqv S*
by (*metis assms homeomorphic_sym homeomorphic_imp_homotopy_eqv linear_homeomorphic_image*)

lemma *homotopy_eqv_translation*:
fixes *S* :: *'a::real_normed_vector set*
shows (+) *a* ‘ *S homotopy_eqv S*
using *homeomorphic_imp_homotopy_eqv homeomorphic_translation homeomorphic_sym* **by** *blast*

lemma *homotopy_eqv_homotopic_triviality_imp*:
fixes *S* :: *'a::real_normed_vector set*
and *T* :: *'b::real_normed_vector set*
and *U* :: *'c::real_normed_vector set*
assumes *S homotopy_eqv T*
and *f*: *continuous_on U f f* ∈ *U* → *T*
and *g*: *continuous_on U g g* ∈ *U* → *T*
and *homUS*: ⋀*f g*. ⟦*continuous_on U f*; *f* ∈ *U* → *S*;
continuous_on U g; *g* ∈ *U* → *S*⟧
⟹ *homotopic_with_canon* (λ*x*. *True*) *U S f g*
shows *homotopic_with_canon* (λ*x*. *True*) *U T f g*
proof −
obtain *h k* **where** *h*: *continuous_on S h h* ∈ *S* → *T*
and *k*: *continuous_on T k k* ∈ *T* → *S*
and *hom*: *homotopic_with_canon* (λ*x*. *True*) *S S* (*k* ∘ *h*) *id*
homotopic_with_canon (λ*x*. *True*) *T T* (*h* ∘ *k*) *id*
using *assms* **by** (*force simp*: *homotopy_equivalent_space_def image_subset_iff_funcset*)
have *homotopic_with_canon* (λ*f*. *True*) *U S* (*k* ∘ *f*) (*k* ∘ *g*)
proof (*rule homUS*)
show *continuous_on U* (*k* ∘ *f*) *continuous_on U* (*k* ∘ *g*)
using *continuous_on_compose continuous_on_subset f g k* **by** (*metis funcset_image*)+

qed (*use f g k* **in** ‹(*force simp*: *o_def*)+›)
then have *homotopic_with_canon* (λ*x*. *True*) *U T* (*h* ∘ (*k* ∘ *f*)) (*h* ∘ (*k* ∘ *g*))
by (*simp add*: *h homotopic_with_compose_continuous_map_left image_subset_iff_funcset*)
moreover have *homotopic_with_canon* (λ*x*. *True*) *U T* (*h* ∘ *k* ∘ *f*) (*id* ∘ *f*)
by (*rule homotopic_with_compose_continuous_right* [**where** *X*=*T* **and** *Y*=*T*];
simp add: *hom f*)
moreover have *homotopic_with_canon* (λ*x*. *True*) *U T* (*h* ∘ *k* ∘ *g*) (*id* ∘ *g*)
by (*rule homotopic_with_compose_continuous_right* [**where** *X*=*T* **and** *Y*=*T*];
simp add: *hom g*)
ultimately show *homotopic_with_canon* (λ*x*. *True*) *U T f g*
unfolding *o_assoc*
by (*metis homotopic_with_trans homotopic_with_sym id_comp*)
qed

lemma *homotopy_eqv_homotopic_triviality*:
fixes *S* :: *'a*::*real_normed_vector set*
and *T* :: *'b*::*real_normed_vector set*
and *U* :: *'c*::*real_normed_vector set*
assumes *S homotopy_eqv T*
shows (∀*f g*. *continuous_on U f* ∧ *f* ∈ *U* → *S* ∧
continuous_on U g ∧ *g* ∈ *U* → *S*
⟶ *homotopic_with_canon* (λ*x*. *True*) *U S f g*) ⟷
(∀*f g*. *continuous_on U f* ∧ *f* ∈ *U* → *T* ∧
continuous_on U g ∧ *g* ∈ *U* → *T*
⟶ *homotopic_with_canon* (λ*x*. *True*) *U T f g*)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (*metis assms homotopy_eqv_homotopic_triviality_imp*)
next
assume *?rhs*
moreover
have *T homotopy_eqv S*
using *assms homotopy_equivalent_space_sym* **by** *blast*
ultimately show *?lhs*
by (*blast intro*: *homotopy_eqv_homotopic_triviality_imp*)
qed

lemma *homotopy_eqv_cohomotopic_triviality_null_imp*:
fixes *S* :: *'a*::*real_normed_vector set*
and *T* :: *'b*::*real_normed_vector set*
and *U* :: *'c*::*real_normed_vector set*
assumes *S homotopy_eqv T*
and *f*: *continuous_on T f f* ∈ *T* → *U*
and *homSU*: ⋀*f*. ⟦*continuous_on S f*; *f* ∈ *S* → *U*⟧
⟹ ∃ *c*. *homotopic_with_canon* (λ*x*. *True*) *S U f* (λ*x*. *c*)
obtains *c* **where** *homotopic_with_canon* (λ*x*. *True*) *T U f* (λ*x*. *c*)

proof −
obtain *h k* **where** *h*: *continuous_on S h h* ∈ *S* → *T*
and *k*: *continuous_on T k k* ∈ *T* → *S*
and *hom*: *homotopic_with_canon* (λ*x*. *True*) *S S* (*k* ∘ *h*) *id*
homotopic_with_canon (λ*x*. *True*) *T T* (*h* ∘ *k*) *id*
using *assms* **by** (*force simp*: *homotopy_equivalent_space_def image_subset_iff_funcset*)
obtain *c* **where** *homotopic_with_canon* (λ*x*. *True*) *S U* (*f* ∘ *h*) (λ*x*. *c*)
proof (*rule exE* [*OF homSU*])
show *continuous_on S* (*f* ∘ *h*)
by (*metis continuous_on_compose continuous_on_subset f h funcset_image*)
qed (*use f h* **in** *force*)
then have *homotopic_with_canon* (λ*x*. *True*) *T U* ((*f* ∘ *h*) ∘ *k*) ((λ*x*. *c*) ∘ *k*)
by (*rule homotopic_with_compose_continuous_right* [**where** *X*=*S*]) (*use k* **in** *auto*)
moreover have *homotopic_with_canon* (λ*x*. *True*) *T U* (*f* ∘ *id*) (*f* ∘ (*h* ∘ *k*))
by (*rule homotopic_with_compose_continuous_left* [**where** *Y*=*T*])
(*use f* **in** ‹*auto simp*: *hom homotopic_with_symD*›)
ultimately show *?thesis*
using *that homotopic_with_trans* **by** (*fastforce simp*: *o_def*)
qed

lemma *homotopy_eqv_cohomotopic_triviality_null*:
fixes *S* :: *'a*::*real_normed_vector set*
and *T* :: *'b*::*real_normed_vector set*
and *U* :: *'c*::*real_normed_vector set*
assumes *S homotopy_eqv T*
shows (∀*f*. *continuous_on S f* ∧ *f* ∈ *S* → *U*
⟶ (∃*c*. *homotopic_with_canon* (λ*x*. *True*) *S U f* (λ*x*. *c*))) ⟷
(∀*f*. *continuous_on T f* ∧ *f* ∈ *T* → *U*
⟶ (∃*c*. *homotopic_with_canon* (λ*x*. *True*) *T U f* (λ*x*. *c*)))
by (*rule iffI*; *metis assms homotopy_eqv_cohomotopic_triviality_null_imp homotopy_equivalent_space_sym*)

Similar to the proof above

lemma *homotopy_eqv_homotopic_triviality_null_imp*:
fixes *S* :: *'a*::*real_normed_vector set*
and *T* :: *'b*::*real_normed_vector set*
and *U* :: *'c*::*real_normed_vector set*
assumes *S homotopy_eqv T*
and *f*: *continuous_on U f f* ∈ *U* → *T*
and *homSU*: ⋀*f*. ⟦*continuous_on U f*; *f* ∈ *U* → *S*⟧
⟹ ∃*c*. *homotopic_with_canon* (λ*x*. *True*) *U S f* (λ*x*. *c*)
shows ∃*c*. *homotopic_with_canon* (λ*x*. *True*) *U T f* (λ*x*. *c*)
proof −
obtain *h k* **where** *h*: *continuous_on S h h* ∈ *S* → *T*
and *k*: *continuous_on T k k* ∈ *T* → *S*
and *hom*: *homotopic_with_canon* (λ*x*. *True*) *S S* (*k* ∘ *h*) *id*
homotopic_with_canon (λ*x*. *True*) *T T* (*h* ∘ *k*) *id*
using *assms* **by** (*force simp*: *homotopy_equivalent_space_def image_subset_iff_funcset*)

obtain *c*::*'a* **where** *homotopic_with_canon* (λ*x*. *True*) *U S* (*k* ∘ *f*) (λ*x*. *c*)
proof (*rule exE* [*OF homSU* [*of k* ∘ *f*]])
show *continuous_on U* (*k* ∘ *f*)
using *continuous_on_compose continuous_on_subset f k* **by** (*metis funcset_image*)
qed (*use f k* **in** *force*)
then have *homotopic_with_canon* (λ*x*. *True*) *U T* (*h* ∘ (*k* ∘ *f*)) (*h* ∘ (λ*x*. *c*))
by (*rule homotopic_with_compose_continuous_left* [**where** *Y*=*S*]) (*use h* **in** *auto*)
moreover have *homotopic_with_canon* (λ*x*. *True*) *U T* (*id* ∘ *f*) ((*h* ∘ *k*) ∘ *f*)
by (*rule homotopic_with_compose_continuous_right* [**where** *X*=*T*])
(*use f* **in** ‹*auto simp*: *hom homotopic_with_symD*›)
ultimately show *?thesis*
using *homotopic_with_trans* **by** (*fastforce simp*: *o_def*)
qed

lemma *homotopy_eqv_homotopic_triviality_null*:
fixes *S* :: *'a*::*real_normed_vector set*
and *T* :: *'b*::*real_normed_vector set*
and *U* :: *'c*::*real_normed_vector set*
assumes *S homotopy_eqv T*
shows (∀*f*. *continuous_on U f* ∧ *f* ∈ *U* → *S*
⟶ (∃*c*. *homotopic_with_canon* (λ*x*. *True*) *U S f* (λ*x*. *c*))) ⟷
(∀*f*. *continuous_on U f* ∧ *f* ∈ *U* → *T*
⟶ (∃*c*. *homotopic_with_canon* (λ*x*. *True*) *U T f* (λ*x*. *c*)))
by (*rule iffI*; *metis assms homotopy_eqv_homotopic_triviality_null_imp homotopy_equivalent_space_sym*)

lemma *homotopy_eqv_contractible_sets*:
fixes *S* :: *'a*::*real_normed_vector set*
and *T* :: *'b*::*real_normed_vector set*
assumes *contractible S contractible T S* = {} ⟷ *T* = {}
shows *S homotopy_eqv T*
proof (*cases S* = {})
case *True* **with** *assms* **show** *?thesis*
using *homeomorphic_imp_homotopy_eqv* **by** *fastforce*
next
case *False*
with *assms* **obtain** *a b* **where** *a* ∈ *S b* ∈ *T*
by *auto*
then show *?thesis*
unfolding *homotopy_equivalent_space_def*
apply (*rule_tac x*=λ*x*. *b* **in** *exI*, *rule_tac x*=λ*x*. *a* **in** *exI*)
apply (*intro assms conjI continuous_on_id' homotopic_into_contractible*; *force*)
done
qed

lemma *homotopy_eqv_empty1* [*simp*]:

fixes S :: *'a::real_normed_vector set*
shows S *homotopy_eqv* ({}::*'b::real_normed_vector set*) $\longleftrightarrow$ $S = \{\}$ (**is** *?lhs* = *?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*metis continuous_map_subtopology_eu empty_iff equalityI homotopy_equivalent_space_def image_subset_iff subsetI*)
qed (*use homeomorphic_imp_homotopy_eqv* **in** *force*)

lemma *homotopy_eqv_empty2* [*simp*]:
fixes S :: *'a::real_normed_vector set*
shows ({}::*'b::real_normed_vector set*) *homotopy_eqv* $S \longleftrightarrow S = \{\}$
using *homotopy_equivalent_space_sym homotopy_eqv_empty1* **by** *blast*

lemma *homotopy_eqv_contractibility*:
fixes S :: *'a::real_normed_vector set* **and** T :: *'b::real_normed_vector set*
shows S *homotopy_eqv* $T \Longrightarrow$ (*contractible* $S \longleftrightarrow$ *contractible* T)
by (*meson contractible_space_top_of_set homotopy_equivalent_space_contractibility*)

lemma *homotopy_eqv_sing*:
fixes S :: *'a::real_normed_vector set* **and** a :: *'b::real_normed_vector*
shows S *homotopy_eqv* $\{a\} \longleftrightarrow S \neq \{\} \wedge$ *contractible* S
by (*metis contractible_sing empty_not_insert homotopy_eqv_contractibility homotopy_eqv_contractible_sets homotopy_eqv_empty2*)

lemma *homeomorphic_contractible_eq*:
fixes S :: *'a::real_normed_vector set* **and** T :: *'b::real_normed_vector set*
shows S *homeomorphic* $T \Longrightarrow$ (*contractible* $S \longleftrightarrow$ *contractible* T)
by (*simp add*: *homeomorphic_imp_homotopy_eqv homotopy_eqv_contractibility*)

lemma *homeomorphic_contractible*:
fixes S :: *'a::real_normed_vector set* **and** T :: *'b::real_normed_vector set*
shows $[\![$*contractible* S; S *homeomorphic* $T]\!] \Longrightarrow$ *contractible* T
by (*metis homeomorphic_contractible_eq*)

5.4.25 Misc other results

lemma *bounded_connected_Compl_real*:
fixes S :: *real set*
assumes *bounded* S **and** *conn*: *connected*$(-S)$
shows $S = \{\}$
proof −
obtain a b **where** $S \subseteq$ *box* a b
by (*meson assms bounded_subset_box_symmetric*)
then have $a \notin S$ $b \notin S$
by *auto*
then have $\forall x.\ a \leq x \wedge x \leq b \longrightarrow x \in -S$
by (*meson Compl_iff conn connected_iff_interval*)
then show *?thesis*

using ‹$S \subseteq$ *box a b*› **by** *auto*
qed

corollary *bounded_path_connected_Compl_real*:
fixes S :: *real set*
assumes *bounded S path_connected*($-$ S) **shows** $S = \{\}$
by (*simp add*: *assms bounded_connected_Compl_real path_connected_imp_connected*)

lemma *bounded_connected_Compl_1*:
fixes S :: *'a*::{*euclidean_space*} *set*
assumes *bounded S* **and** *conn*: *connected*($-$ S) **and** *1*: *DIM*(*'a*) = *1*
shows $S = \{\}$
proof −
have *DIM*(*'a*) = *DIM*(*real*)
by (*simp add*: *1*)
then obtain *f*::*'a* ⇒ *real* **and** *g*
where *linear f* ⋀*x*. *norm*(*f x*) = *norm x* **and** *fg*: ⋀*x*. *g*(*f x*) = *x* ⋀*y*. *f*(*g y*) = *y*
by (*rule isomorphisms_UNIV_UNIV*) *blast*
with ‹*bounded S*› **have** *bounded* (*f* ‘ *S*)
using *bounded_linear_image linear_linear* **by** *blast*
have *bij f* **by** (*metis fg bijI'*)
have *connected* (*f* ‘ (−*S*))
using *connected_linear_image assms* ‹*linear f*› **by** *blast*
moreover have *f* ‘ (−*S*) = − (*f* ‘ *S*)
by (*simp add*: ‹*bij f*› *bij_image_Compl_eq*)
finally have *connected* (− (*f* ‘ *S*))
by *simp*
then have *f* ‘ *S* = {}
using ‹*bounded* (*f* ‘ *S*)› *bounded_connected_Compl_real* **by** *blast*
then show *?thesis*
by *blast*
qed

lemma *connected_card_eq_iff_nontrivial*:
fixes S :: *'a*::*metric_space set*
shows *connected S* ⟹ *uncountable S* ⟷ ¬(∃ *a*. *S* ⊆ {*a*})
by (*metis connected_uncountable finite.emptyI finite.insertI rev_finite_subset singleton_iff subsetI uncountable_infinite*)

lemma *connected_finite_iff_sing*:
fixes S :: *'a*::*metric_space set*
assumes *connected S*
shows *finite S* ⟷ *S* = {} ∨ (∃ *a*. *S* = {*a*})
using *assms connected_uncountable countable_finite* **by** *blast*

5.4.26 Some simple positive connection theorems

proposition *path_connected_convex_diff_countable*:
fixes U :: *'a*::*euclidean_space set*

```
  assumes convex U ¬ collinear U countable S
    shows path_connected(U − S)
proof (clarsimp simp: path_connected_def)
  fix a b
  assume a ∈ U a ∉ S b ∈ U b ∉ S
  let ?m = midpoint a b
  show ∃ g. path g ∧ path_image g ⊆ U − S ∧ pathstart g = a ∧ pathfinish g = b
  proof (cases a = b)
    case True
    then show ?thesis
      by (metis DiffI ‹a ∈ U› ‹a ∉ S› path_component_def path_component_refl)
  next
    case False
    then have a ≠ ?m b ≠ ?m
      using midpoint_eq_endpoint by fastforce+
    have ?m ∈ U
      using ‹a ∈ U› ‹b ∈ U› ‹convex U› convex_contains_segment by force
    obtain c where c ∈ U and nc_abc: ¬ collinear {a,b,c}
        by (metis False ‹a ∈ U› ‹b ∈ U› ‹¬ collinear U› collinear_triples in-
sert_absorb)
    have ncoll_mca: ¬ collinear {?m,c,a}
      by (metis (full_types) ‹a ≠ ?m› collinear_3_trans collinear_midpoint in-
sert_commute nc_abc)
    have ncoll_mcb: ¬ collinear {?m,c,b}
      by (metis (full_types) ‹b ≠ ?m› collinear_3_trans collinear_midpoint in-
sert_commute nc_abc)
    have c ≠ ?m
      by (metis collinear_midpoint insert_commute nc_abc)
    then have closed_segment ?m c ⊆ U
      by (simp add: ‹c ∈ U› ‹?m ∈ U› ‹convex U› closed_segment_subset)
    then obtain z where z: z ∈ closed_segment ?m c
                    and disjS: (closed_segment a z ∪ closed_segment z b) ∩ S = {}
    proof −
    have False if closed_segment ?m c ⊆ {z. (closed_segment a z ∪ closed_segment
z b) ∩ S ≠ {}}
      proof −
        have closb: closed_segment ?m c ⊆
                {z ∈ closed_segment ?m c. closed_segment a z ∩ S ≠ {}} ∪ {z ∈
closed_segment ?m c. closed_segment z b ∩ S ≠ {}}
          using that by blast
        have ∗: countable {z ∈ closed_segment ?m c. closed_segment z u ∩ S ≠ {}}
          if u ∈ U u ∉ S and ncoll: ¬ collinear {?m, c, u} for u
        proof −
        have ∗∗: False if x1: x1 ∈ closed_segment ?m c and x2: x2 ∈ closed_segment
?m c
                        and x1 ≠ x2 x1 ≠ u
                        and w: w ∈ closed_segment x1 u w ∈ closed_segment x2 u
                        and w ∈ S for x1 x2 w
          proof −
```

```
          have x1 ∈ affine hull {?m,c} x2 ∈ affine hull {?m,c}
            using segment_as_ball x1 x2 by auto
          then have coll_x1: collinear {x1, ?m, c} and coll_x2: collinear {?m,
c, x2}
        by (simp_all add: affine_hull_3_imp_collinear) (metis affine_hull_3_imp_collinear
insert_commute)
          have ¬ collinear {x1, u, x2}
          proof
            assume collinear {x1, u, x2}
            then have collinear {?m, c, u}
              by (metis (full_types) ‹c ≠ ?m› coll_x1 coll_x2 collinear_3_trans
insert_commute ncoll ‹x1 ≠ x2›)
            with ncoll show False ..
          qed
          then have closed_segment x1 u ∩ closed_segment u x2 = {u}
            by (blast intro!: Int_closed_segment)
          then have w = u
            using closed_segment_commute w by auto
          show ?thesis
            using ‹u ∉ S› ‹w = u› that(7) by auto
        qed
        then have disj: disjoint ((⋃z∈closed_segment ?m c. {closed_segment z
u ∩ S}))
          by (fastforce simp: pairwise_def disjnt_def)
        have cou: countable ((⋃z ∈ closed_segment ?m c. {closed_segment z u ∩
S}) − {{}})
          apply (rule pairwise_disjnt_countable_Union [OF _ pairwise_subset
[OF disj]])
          apply (rule countable_subset [OF _ ‹countable S›], auto)
          done
          define f where f ≡ λX. (THE z. z ∈ closed_segment ?m c ∧ X =
closed_segment z u ∩ S)
        show ?thesis
          proof (rule countable_subset [OF _ countable_image [OF cou, where
f=f]], clarify)
          fix x
          assume x: x ∈ closed_segment ?m c closed_segment x u ∩ S ≠ {}
          show x ∈ f ‘ ((⋃z∈closed_segment ?m c. {closed_segment z u ∩ S}) −
{{}})
          proof (rule_tac x=closed_segment x u ∩ S in image_eqI)
            show x = f (closed_segment x u ∩ S)
              unfolding f_def
              by (rule the_equality [symmetric]) (use x in ‹auto dest: ∗∗›)
          qed (use x in auto)
        qed
      qed
      have uncountable (closed_segment ?m c)
        by (metis ‹c ≠ ?m› uncountable_closed_segment)
      then show False
```

```
        using closb * [OF ‹a ∈ U› ‹a ∉ S› ncoll_mca] * [OF ‹b ∈ U› ‹b ∉ S›
ncoll_mcb]
        by (simp add: closed_segment_commute countable_subset)
    qed
    then show ?thesis
      by (force intro: that)
  qed
  show ?thesis
  proof (intro exI conjI)
    have path_image (linepath a z +++ linepath z b) ⊆ U
      by (metis ‹a ∈ U› ‹b ∈ U› ‹closed_segment ?m c ⊆ U› z ‹convex U›
closed_segment_subset contra_subsetD path_image_linepath subset_path_image_join)
    with disjS show path_image (linepath a z +++ linepath z b) ⊆ U - S
      by (force simp: path_image_join)
  qed auto
 qed
qed


corollary connected_convex_diff_countable:
 fixes U :: 'a::euclidean_space set
 assumes convex U ¬ collinear U countable S
 shows connected(U - S)
 by (simp add: assms path_connected_convex_diff_countable path_connected_imp_connected)

lemma path_connected_punctured_convex:
 assumes convex S and aff: aff_dim S ≠ 1
   shows path_connected(S - {a})
proof -
 consider aff_dim S = -1 | aff_dim S = 0 | aff_dim S ≥ 2
   using assms aff_dim_geq [of S] by linarith
 then show ?thesis
 proof cases
   assume aff_dim S = -1
   then show ?thesis
     by (metis aff_dim_empty empty_Diff path_connected_empty)
 next
   assume aff_dim S = 0
   then show ?thesis
    by (metis aff_dim_eq_0 Diff_cancel Diff_empty Diff_insert0 convex_empty
convex_imp_path_connected path_connected_singleton singletonD)
 next
   assume ge2: aff_dim S ≥ 2
   then have ¬ collinear S
   proof (clarsimp simp: collinear_affine_hull)
     fix u v
     assume S ⊆ affine hull {u, v}
     then have aff_dim S ≤ aff_dim {u, v}
       by (metis (no_types) aff_dim_affine_hull aff_dim_subset)
```

```
      with ge2 show False
      by (metis (no_types) aff_dim_2 antisym aff not_numeral_le_zero one_le_numeral
order_trans)
    qed
    moreover have countable {a}
      by simp
    ultimately show ?thesis
      by (metis path_connected_convex_diff_countable [OF ‹convex S›])
  qed
qed

lemma connected_punctured_convex:
  shows ⟦convex S; aff_dim S ≠ 1⟧ ⟹ connected(S − {a})
  using path_connected_imp_connected path_connected_punctured_convex by blast

lemma path_connected_complement_countable:
  fixes S :: 'a::euclidean_space set
  assumes 2 ≤ DIM('a) countable S
  shows path_connected(− S)
proof −
  have ¬ collinear (UNIV::'a set)
    using assms by (auto simp: collinear_aff_dim [of UNIV :: 'a set])
  then have path_connected(UNIV − S)
    by (simp add: ‹countable S› path_connected_convex_diff_countable)
  then show ?thesis
    by (simp add: Compl_eq_Diff_UNIV)
qed

proposition path_connected_openin_diff_countable:
  fixes S :: 'a::euclidean_space set
  assumes connected S and ope: openin (top_of_set (affine hull S)) S
      and ¬ collinear S countable T
    shows path_connected(S − T)
proof (clarsimp simp: path_connected_component)
  fix x y
  assume xy: x ∈ S x ∉ T y ∈ S y ∉ T
  show path_component (S − T) x y
  proof (rule connected_equivalence_relation_gen [OF ‹connected S›, where P
= λx. x ∉ T])
    show ∃z. z ∈ U ∧ z ∉ T if opeU: openin (top_of_set S) U and x ∈ U for
U x
    proof −
      have openin (top_of_set (affine hull S)) U
        using opeU ope openin_trans by blast
      with ‹x ∈ U› obtain r where Usub: U ⊆ affine hull S and r > 0
                            and subU: ball x r ∩ affine hull S ⊆ U
        by (auto simp: openin_contains_ball)
      with ‹x ∈ U› have x: x ∈ ball x r ∩ affine hull S
        by auto
```

```
    have ¬ S ⊆ {x}
      using ‹¬ collinear S› collinear_subset by blast
    then obtain x' where x' ≠ x x' ∈ S
      by blast
    obtain y where y: y ≠ x y ∈ ball x r ∩ affine hull S
    proof
      show x + (r / 2 / norm(x' − x)) *R (x' − x) ≠ x
        using ‹x' ≠ x› ‹r > 0› by auto
      show x + (r / 2 / norm (x' − x)) *R (x' − x) ∈ ball x r ∩ affine hull S
        using ‹x' ≠ x› ‹r > 0› ‹x' ∈ S› x
        by (simp add: dist_norm mem_affine_3_minus hull_inc)
    qed
    have convex (ball x r ∩ affine hull S)
      by (simp add: affine_imp_convex convex_Int)
    with x y subU have uncountable U
      by (meson countable_subset uncountable_convex)
    then have ¬ U ⊆ T
      using ‹countable T› countable_subset by blast
    then show ?thesis by blast
  qed
  show ∃U. openin (top_of_set S) U ∧ x ∈ U ∧
          (∀x∈U. ∀y∈U. x ∉ T ∧ y ∉ T ⟶ path_component (S − T) x y)
       if x ∈ S for x
  proof −
    obtain r where Ssub: S ⊆ affine hull S and r > 0
              and subS: ball x r ∩ affine hull S ⊆ S
      using ope ‹x ∈ S› by (auto simp: openin_contains_ball)
    then have conv: convex (ball x r ∩ affine hull S)
      by (simp add: affine_imp_convex convex_Int)
    have ¬ aff_dim (affine hull S) ≤ 1
      using ‹¬ collinear S› collinear_aff_dim by auto
    then have ¬ aff_dim (ball x r ∩ affine hull S) ≤ 1
     by (metis (no_types, opaque_lifting) aff_dim_convex_Int_open IntI open_ball
‹0 < r› aff_dim_affine_hull affine_affine_hull affine_imp_convex centre_in_ball
empty_iff hull_subset inf_commute subsetCE that)
    then have ¬ collinear (ball x r ∩ affine hull S)
      by (simp add: collinear_aff_dim)
    then have *: path_connected ((ball x r ∩ affine hull S) − T)
     by (rule path_connected_convex_diff_countable [OF conv _ ‹countable T›])
    have ST: ball x r ∩ affine hull S − T ⊆ S − T
      using subS by auto
    show ?thesis
    proof (intro exI conjI)
      show x ∈ ball x r ∩ affine hull S
        using ‹x ∈ S› ‹r > 0› by (simp add: hull_inc)
      have openin (top_of_set (affine hull S)) (ball x r ∩ affine hull S)
        by (subst inf.commute) (simp add: openin_Int_open)
      then show openin (top_of_set S) (ball x r ∩ affine hull S)
        by (rule openin_subset_trans [OF _ subS Ssub])
```

qed (*use* ∗ *path_component_trans* **in** ‹*auto simp*: *path_connected_component path_component_of_subset* [*OF ST*]›)
qed
qed (*use xy path_component_trans* **in** *auto*)
qed

corollary *connected_openin_diff_countable*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *connected S* **and** *ope*: *openin* (*top_of_set* (*affine hull S*)) *S*
and ¬ *collinear S countable T*
shows *connected*(*S* − *T*)
by (*metis path_connected_imp_connected path_connected_openin_diff_countable* [*OF assms*])

corollary *path_connected_open_diff_countable*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *2* ≤ *DIM*(*'a*) *open S connected S countable T*
shows *path_connected*(*S* − *T*)
proof (*cases S* = {})
case *True*
then show *?thesis*
by (*simp*)
next
case *False*
show *?thesis*
proof (*rule path_connected_openin_diff_countable*)
show *openin* (*top_of_set* (*affine hull S*)) *S*
by (*simp add*: *assms hull_subset open_subset*)
show ¬ *collinear S*
using *assms False* **by** (*simp add*: *collinear_aff_dim aff_dim_open*)
qed (*simp_all add*: *assms*)
qed

corollary *connected_open_diff_countable*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *2* ≤ *DIM*(*'a*) *open S connected S countable T*
shows *connected*(*S* − *T*)
by (*simp add*: *assms path_connected_imp_connected path_connected_open_diff_countable*)

5.4.27 Self-homeomorphisms shuffling points about

The theorem *homeomorphism_moving_points_exists*

lemma *homeomorphism_moving_point_1*:
fixes *a* :: *'a*::*euclidean_space*
assumes *affine T a* ∈ *T* **and** *u*: *u* ∈ *ball a r* ∩ *T*
obtains *f g* **where** *homeomorphism* (*cball a r* ∩ *T*) (*cball a r* ∩ *T*) *f g*
f a = *u* ⋀*x*. *x* ∈ *sphere a r* ⟹ *f x* = *x*
proof −
have *nou*: *norm* (*u* − *a*) < *r* **and** *u* ∈ *T*

```
    using u by (auto simp: dist_norm norm_minus_commute)
  then have 0 < r
    by (metis DiffD1 Diff_Diff_Int ball_eq_empty centre_in_ball not_le u)
  define f where f ≡ λx. (1 − norm(x − a) / r) *R (u − a) + x
  have *: False if eq: x + (norm y / r) *R u = y + (norm x / r) *R u
                and nou: norm u < r and yx: norm y < norm x for x y and u::'a
  proof −
    have x = y + (norm x / r − (norm y / r)) *R u
      using eq by (simp add: algebra_simps)
    then have norm x = norm (y + ((norm x − norm y) / r) *R u)
      by (metis diff_divide_distrib)
    also have ... ≤ norm y + norm(((norm x − norm y) / r) *R u)
      using norm_triangle_ineq by blast
    also have ... = norm y + (norm x − norm y) * (norm u / r)
      using yx ‹r > 0›
      by (simp add: field_split_simps)
    also have ... < norm y + (norm x − norm y) * 1
    proof (subst add_less_cancel_left)
      show (norm x − norm y) * (norm u / r) < (norm x − norm y) * 1
      proof (rule mult_strict_left_mono)
        show norm u / r < 1
          using ‹0 < r› divide_less_eq_1_pos nou by blast
      qed (simp add: yx)
    qed
    also have ... = norm x
      by simp
    finally show False by simp
  qed
  have inj f
    unfolding f_def
  proof (clarsimp simp: inj_on_def)
    fix x y
    assume (1 − norm (x − a) / r) *R (u − a) + x =
           (1 − norm (y − a) / r) *R (u − a) + y
    then have eq: (x − a) + (norm (y − a) / r) *R (u − a) = (y − a) + (norm
(x − a) / r) *R (u − a)
      by (auto simp: algebra_simps)
    show x=y
    proof (cases norm (x − a) = norm (y − a))
      case True
      then show ?thesis
        using eq by auto
    next
      case False
      then consider norm (x − a) < norm (y − a) | norm (x − a) > norm (y −
a)
        by linarith
      then have False
      proof cases
```

```
      case 1 show False
        using * [OF _ nou 1] eq by simp
    next
      case 2 with * [OF eq nou] show False
        by auto
    qed
    then show x=y ..
  qed
 qed
 then have inj_onf: inj_on f (cball a r ∩ T)
  using inj_on_Int by fastforce
 have contf: continuous_on (cball a r ∩ T) f
  unfolding f_def using ‹0 < r› by (intro continuous_intros) blast
 have fim: f ‘ (cball a r ∩ T) = cball a r ∩ T
 proof
  have *: norm (y + (1 − norm y / r) *R u) ≤ r if norm y ≤ r norm u < r
for y u::'a
  proof −
   have norm (y + (1 − norm y / r) *R u) ≤ norm y + norm((1 − norm y /
r) *R u)
    using norm_triangle_ineq by blast
   also have ... = norm y + abs(1 − norm y / r) * norm u
    by simp
   also have ... ≤ r
   proof −
    have (r − norm u) * (r − norm y) ≥ 0
     using that by auto
    then have r * norm u + r * norm y ≤ r * r + norm u * norm y
     by (simp add: algebra_simps)
    then show ?thesis
    using that ‹0 < r› by (simp add: abs_if field_simps)
   qed
   finally show ?thesis .
  qed
  have f ‘ (cball a r) ⊆ cball a r
   using * nou
   apply (clarsimp simp: dist_norm norm_minus_commute f_def)
   by (metis diff_add_eq diff_diff_add diff_diff_eq2 norm_minus_commute)
  moreover have f ‘ T ⊆ T
   unfolding f_def using ‹affine T› ‹a ∈ T› ‹u ∈ T›
   by (force simp: add.commute mem_affine_3_minus)
  ultimately show f ‘ (cball a r ∩ T) ⊆ cball a r ∩ T
   by blast
 next
  show cball a r ∩ T ⊆ f ‘ (cball a r ∩ T)
  proof (clarsimp simp: dist_norm norm_minus_commute)
   fix x
   assume x: norm (x − a) ≤ r and x ∈ T
   have ∃ v ∈ {0..1}. ((1 − v) * r − norm ((x − a) − v *R (u − a))) • 1 = 0
```

```
      by (rule ivt_decreasing_component_on_1) (auto simp: x continuous_intros)
     then obtain v where 0 ≤ v v ≤ 1
       and v: (1 − v) ∗ r = norm ((x − a) − v ∗R (u − a))
       by auto
     then have n: norm (a − (x − v ∗R (u − a))) = r − r ∗ v
       by (simp add: field_simps norm_minus_commute)
     show x ∈ f ‘ (cball a r ∩ T)
     proof (rule image_eqI)
       show x = f (x − v ∗R (u − a))
         using ‹r > 0› v by (simp add: f_def) (simp add: field_simps)
       have x − v ∗R (u − a) ∈ cball a r
         using ‹r > 0›‹0 ≤ v›
         by (simp add: dist_norm n)
       moreover have x − v ∗R (u − a) ∈ T
         by (simp add: f_def ‹u ∈ T› ‹x ∈ T› assms mem_affine_3_minus2)
       ultimately show x − v ∗R (u − a) ∈ cball a r ∩ T
         by blast
     qed
   qed
 qed
 have compact (cball a r ∩ T)
   by (simp add: affine_closed compact_Int_closed ‹affine T›)
 then obtain g where homeomorphism (cball a r ∩ T) (cball a r ∩ T) f g
   by (metis homeomorphism_compact [OF _ contf fim inj_onf])
 then show thesis
   apply (rule_tac f=f in that)
   using ‹r > 0› by (simp_all add: f_def dist_norm norm_minus_commute)
qed

corollary homeomorphism_moving_point_2:
 fixes a :: 'a::euclidean_space
 assumes affine T a ∈ T and u: u ∈ ball a r ∩ T and v: v ∈ ball a r ∩ T
 obtains f g where homeomorphism (cball a r ∩ T) (cball a r ∩ T) f g
                   f u = v ⋀x. ⟦x ∈ sphere a r; x ∈ T⟧ ⟹ f x = x
proof −
 have 0 < r
   by (metis DiffD1 Diff_Diff_Int ball_eq_empty centre_in_ball not_le u)
 obtain f1 g1 where hom1: homeomorphism (cball a r ∩ T) (cball a r ∩ T) f1
g1
                and f1 a = u and f1: ⋀x. x ∈ sphere a r ⟹ f1 x = x
   using homeomorphism_moving_point_1 [OF ‹affine T› ‹a ∈ T› u] by blast
 obtain f2 g2 where hom2: homeomorphism (cball a r ∩ T) (cball a r ∩ T) f2
g2
                and f2 a = v and f2: ⋀x. x ∈ sphere a r ⟹ f2 x = x
   using homeomorphism_moving_point_1 [OF ‹affine T› ‹a ∈ T› v] by blast
 show ?thesis
 proof
   show homeomorphism (cball a r ∩ T) (cball a r ∩ T) (f2 ∘ g1) (f1 ∘ g2)
     by (metis homeomorphism_compose homeomorphism_symD hom1 hom2)
```

```
    have g1 u = a
      using ‹0 < r› ‹f1 a = u› assms hom1 homeomorphism_apply1 by fastforce
    then show (f2 ∘ g1) u = v
      by (simp add: ‹f2 a = v›)
    show ⋀x. ⟦x ∈ sphere a r; x ∈ T⟧ ⟹ (f2 ∘ g1) x = x
      using f1 f2 hom1 homeomorphism_apply1 by fastforce
  qed
qed


corollary homeomorphism_moving_point_3:
  fixes a :: 'a::euclidean_space
  assumes affine T a ∈ T and ST: ball a r ∩ T ⊆ S S ⊆ T
      and u: u ∈ ball a r ∩ T and v: v ∈ ball a r ∩ T
  obtains f g where homeomorphism S S f g
                    f u = v {x. ¬ (f x = x ∧ g x = x)} ⊆ ball a r ∩ T
proof −
  obtain f g where hom: homeomorphism (cball a r ∩ T) (cball a r ∩ T) f g
              and f u = v and fid: ⋀x. ⟦x ∈ sphere a r; x ∈ T⟧ ⟹ f x = x
    using homeomorphism_moving_point_2 [OF ‹affine T› ‹a ∈ T› u v] by blast
  have gid: ⋀x. ⟦x ∈ sphere a r; x ∈ T⟧ ⟹ g x = x
    using fid hom homeomorphism_apply1 by fastforce
  define ff where ff ≡ λx. if x ∈ ball a r ∩ T then f x else x
  define gg where gg ≡ λx. if x ∈ ball a r ∩ T then g x else x
  show ?thesis
  proof
    show homeomorphism S S ff gg
    proof (rule homeomorphismI)
      have continuous_on ((cball a r ∩ T) ∪ (T − ball a r)) ff
        unfolding ff_def
        using homeomorphism_cont1 [OF hom]
        by (intro continuous_on_cases) (auto simp: affine_closed ‹affine T› fid)
      then show continuous_on S ff
        by (rule continuous_on_subset) (use ST in auto)
      have continuous_on ((cball a r ∩ T) ∪ (T − ball a r)) gg
        unfolding gg_def
        using homeomorphism_cont2 [OF hom]
        by (intro continuous_on_cases) (auto simp: affine_closed ‹affine T› gid)
      then show continuous_on S gg
        by (rule continuous_on_subset) (use ST in auto)
      show ff ‘ S ⊆ S
      proof (clarsimp simp: ff_def)
        fix x
        assume x ∈ S and x: dist a x < r and x ∈ T
        then have f x ∈ cball a r ∩ T
          using homeomorphism_image1 [OF hom] by force
        then show f x ∈ S
          using ST(1) ‹x ∈ T› gid hom homeomorphism_def x by fastforce
      qed
```

```
      show gg ‘ S ⊆ S
      proof (clarsimp simp: gg_def)
        fix x
        assume x ∈ S and x: dist a x < r and x ∈ T
        then have g x ∈ cball a r ∩ T
          using homeomorphism_image2 [OF hom] by force
        then have g x ∈ ball a r
          using homeomorphism_apply2 [OF hom]
                by (metis Diff_Diff_Int Diff_iff ‹x ∈ T› cball_def fid le_less
mem_Collect_eq mem_ball mem_sphere x)
        then show g x ∈ S
          using ST(1) ‹g x ∈ cball a r ∩ T› by force
        qed
      show ⋀x. x ∈ S ⟹ gg (ff x) = x
        unfolding ff_def gg_def
          using homeomorphism_apply1 [OF hom] homeomorphism_image1 [OF
hom]
         by simp (metis Int_iff homeomorphism_apply1 [OF hom] fid image_eqI
less_eq_real_def mem_cball mem_sphere)
      show ⋀x. x ∈ S ⟹ ff (gg x) = x
        unfolding ff_def gg_def
          using homeomorphism_apply2 [OF hom] homeomorphism_image2 [OF
hom]
       by simp (metis Int_iff fid image_eqI less_eq_real_def mem_cball mem_sphere)
     qed
     show ff u = v
       using u by (auto simp: ff_def ‹f u = v›)
     show {x. ¬ (ff x = x ∧ gg x = x)} ⊆ ball a r ∩ T
       by (auto simp: ff_def gg_def)
   qed
qed


proposition homeomorphism_moving_point:
  fixes a :: 'a::euclidean_space
  assumes ope: openin (top_of_set (affine hull S)) S
      and S ⊆ T
      and TS: T ⊆ affine hull S
      and S: connected S a ∈ S b ∈ S
  obtains f g where homeomorphism T T f g f a = b
                    {x. ¬ (f x = x ∧ g x = x)} ⊆ S
                    bounded {x. ¬ (f x = x ∧ g x = x)}
proof −
  have 1: ∃h k. homeomorphism T T h k ∧ h (f d) = d ∧
             {x. ¬ (h x = x ∧ k x = x)} ⊆ S ∧ bounded {x. ¬ (h x = x ∧ k x = x)}
       if d ∈ S f d ∈ S and homfg: homeomorphism T T f g
       and S: {x. ¬ (f x = x ∧ g x = x)} ⊆ S
       and bo: bounded {x. ¬ (f x = x ∧ g x = x)} for d f g
  proof (intro exI conjI)
```

```
    show homgf: homeomorphism T T g f
      by (metis homeomorphism_symD homfg)
    then show g (f d) = d
      by (meson ‹S ⊆ T› homeomorphism_def subsetD ‹d ∈ S›)
    show {x. ¬ (g x = x ∧ f x = x)} ⊆ S
      using S by blast
    show bounded {x. ¬ (g x = x ∧ f x = x)}
      using bo by (simp add: conj_commute)
  qed
  have 2: ∃f g. homeomorphism T T f g ∧ f x = f2 (f1 x) ∧
                {x. ¬ (f x = x ∧ g x = x)} ⊆ S ∧ bounded {x. ¬ (f x = x ∧ g x =
x)}
            if x ∈ S f1 x ∈ S f2 (f1 x) ∈ S
              and hom: homeomorphism T T f1 g1 homeomorphism T T f2 g2
              and sub: {x. ¬ (f1 x = x ∧ g1 x = x)} ⊆ S  {x. ¬ (f2 x = x ∧ g2 x
= x)} ⊆ S
               and bo: bounded {x. ¬ (f1 x = x ∧ g1 x = x)}  bounded {x. ¬ (f2 x
= x ∧ g2 x = x)}
            for x f1 f2 g1 g2
  proof (intro exI conjI)
    show homgf: homeomorphism T T (f2 ∘ f1) (g1 ∘ g2)
      by (metis homeomorphism_compose hom)
    then show (f2 ∘ f1) x = f2 (f1 x)
      by force
    show {x. ¬ ((f2 ∘ f1) x = x ∧ (g1 ∘ g2) x = x)} ⊆ S
      using sub by force
    have bounded ({x. ¬(f1 x = x ∧ g1 x = x)} ∪ {x. ¬(f2 x = x ∧ g2 x = x)})
      using bo by simp
    then show bounded {x. ¬ ((f2 ∘ f1) x = x ∧ (g1 ∘ g2) x = x)}
      by (rule bounded_subset) auto
  qed
  have 3: ∃U. openin (top_of_set S) U ∧
              d ∈ U ∧
              (∀x∈U.
                  ∃f g. homeomorphism T T f g ∧ f d = x ∧
                        {x. ¬ (f x = x ∧ g x = x)} ⊆ S ∧
                        bounded {x. ¬ (f x = x ∧ g x = x)})
          if d ∈ S for d
  proof −
    obtain r where r > 0 and r: ball d r ∩ affine hull S ⊆ S
      by (metis ‹d ∈ S› ope openin_contains_ball)
    have ∗: ∃f g. homeomorphism T T f g ∧ f d = e ∧
                  {x. ¬ (f x = x ∧ g x = x)} ⊆ S ∧
                  bounded {x. ¬ (f x = x ∧ g x = x)} if e ∈ S e ∈ ball d r for e
      apply (rule homeomorphism_moving_point_3 [of affine hull S d r T d e])
      using r ‹S ⊆ T› TS that
          apply (auto simp: ‹d ∈ S› ‹0 < r› hull_inc)
      using bounded_subset by blast
    show ?thesis
```

```
    by (rule_tac x=S ∩ ball d r in exI) (fastforce simp: openin_open_Int ‹0 <
r› that intro: ∗)
  qed
  have ∃f g. homeomorphism T T f g ∧ f a = b ∧
            {x. ¬ (f x = x ∧ g x = x)} ⊆ S ∧ bounded {x. ¬ (f x = x ∧ g x = x)}
    by (rule connected_equivalence_relation [OF S]; blast intro: 1 2 3)
  then show ?thesis
    using that by auto
qed


lemma homeomorphism_moving_points_exists_gen:
  assumes K: finite K ⋀i. i ∈ K ⟹ x i ∈ S ∧ y i ∈ S
           pairwise (λi j. (x i ≠ x j) ∧ (y i ≠ y j)) K
      and 2 ≤ aff_dim S
      and ope: openin (top_of_set (affine hull S)) S
      and S ⊆ T T ⊆ affine hull S connected S
  shows ∃f g. homeomorphism T T f g ∧ (∀ i ∈ K. f(x i) = y i) ∧
            {x. ¬ (f x = x ∧ g x = x)} ⊆ S ∧ bounded {x. ¬ (f x = x ∧ g x = x)}
  using assms
proof (induction K)
  case empty
  then show ?case
    by (force simp: homeomorphism_ident)
next
  case (insert i K)
  then have xney: ⋀j. ⟦j ∈ K; j ≠ i⟧ ⟹ x i ≠ x j ∧ y i ≠ y j
      and pw: pairwise (λi j. x i ≠ x j ∧ y i ≠ y j) K
      and x i ∈ S y i ∈ S
      and xyS: ⋀i. i ∈ K ⟹ x i ∈ S ∧ y i ∈ S
    by (simp_all add: pairwise_insert)
  obtain f g where homfg: homeomorphism T T f g and feq: ⋀i. i ∈ K ⟹ f(x
i) = y i
              and fg_sub: {x. ¬ (f x = x ∧ g x = x)} ⊆ S
              and bo_fg: bounded {x. ¬ (f x = x ∧ g x = x)}
    using insert.IH [OF xyS pw] insert.prems by (blast intro: that)
  then have ∃f g. homeomorphism T T f g ∧ (∀ i ∈ K. f(x i) = y i) ∧
                {x. ¬ (f x = x ∧ g x = x)} ⊆ S ∧ bounded {x. ¬ (f x = x ∧ g x =
x)}
    using insert by blast
  have aff_eq: affine hull (S − y ‘ K) = affine hull S
  proof (rule affine_hull_Diff [OF ope])
    show finite (y ‘ K)
      by (simp add: insert.hyps(1))
    show y ‘ K ⊂ S
      using ‹y i ∈ S› insert.hyps(2) xney xyS by fastforce
  qed
  have f_in_S: f x ∈ S if x ∈ S for x
    using homfg fg_sub homeomorphism_apply1 ‹S ⊆ T›
```

```
  proof −
    have (f (f x) ≠ f x ∨ g (f x) ≠ f x) ∨ f x ∈ S
      by (metis ‹S ⊆ T› homfg subsetD homeomorphism_apply1 that)
    then show ?thesis
      using fg_sub by force
  qed
  obtain h k where homhk: homeomorphism T T h k and heq: h (f (x i)) = y i
              and hk_sub: {x. ¬ (h x = x ∧ k x = x)} ⊆ S − y ‘ K
              and bo_hk:  bounded {x. ¬ (h x = x ∧ k x = x)}
  proof (rule homeomorphism_moving_point [of S − y‘K T f(x i) y i])
    show openin (top_of_set (affine hull (S − y ‘ K))) (S − y ‘ K)
        by (simp add: aff_eq openin_diff finite_imp_closedin image_subset_iff
hull_inc insert xyS)
    show S − y ‘ K ⊆ T
      using ‹S ⊆ T› by auto
    show T ⊆ affine hull (S − y ‘ K)
      using insert by (simp add: aff_eq)
    show connected (S − y ‘ K)
    proof (rule connected_openin_diff_countable [OF ‹connected S› ope])
      show ¬ collinear S
        using collinear_aff_dim ‹2 ≤ aff_dim S› by force
      show countable (y ‘ K)
        using countable_finite insert.hyps(1) by blast
    qed
    have ⋀k. ⟦f (x i) = y k; k ∈ K⟧ ⟹ False
        by (metis feq homfg ‹x i ∈ S› homeomorphism_def ‹S ⊆ T› ‹i ∉ K›
subsetCE xney xyS)
    then show f (x i) ∈ S − y ‘ K
      by (auto simp: f_in_S ‹x i ∈ S›)
    show y i ∈ S − y ‘ K
      using insert.hyps xney by (auto simp: ‹y i ∈ S›)
  qed blast
  show ?case
  proof (intro exI conjI)
    show homeomorphism T T (h ∘ f) (g ∘ k)
      using homfg homhk homeomorphism_compose by blast
    show ∀ i ∈ insert i K. (h ∘ f) (x i) = y i
      using feq hk_sub by (auto simp: heq)
    show {x. ¬ ((h ∘ f) x = x ∧ (g ∘ k) x = x)} ⊆ S
      using fg_sub hk_sub by force
    have bounded ({x. ¬(f x = x ∧ g x = x)} ∪ {x. ¬(h x = x ∧ k x = x)})
      using bo_fg bo_hk bounded_Un by blast
    then show bounded {x. ¬ ((h ∘ f) x = x ∧ (g ∘ k) x = x)}
      by (rule bounded_subset) auto
  qed
qed

proposition homeomorphism_moving_points_exists:
  fixes S :: 'a::euclidean_space set
```

assumes *2*: $2 \le DIM('a)$ *open S connected S* $S \subseteq T$ *finite K*
and *KS*: $\bigwedge i.\ i \in K \Longrightarrow x\ i \in S \wedge y\ i \in S$
and *pw*: *pairwise* $(\lambda i\ j.\ (x\ i \neq x\ j) \wedge (y\ i \neq y\ j))\ K$
and *S*: $S \subseteq T\ T \subseteq$ *affine hull S connected S*
obtains *f g* **where** *homeomorphism T T f g* $\bigwedge i.\ i \in K \Longrightarrow f(x\ i) = y\ i$ $\{x.\ \neg\ (f\ x = x \wedge g\ x = x)\} \subseteq S$ *bounded* $\{x.\ (\neg\ (f\ x = x \wedge g\ x = x))\}$
proof (*cases* $S = \{\}$)
case *True*
then show *?thesis*
using *KS homeomorphism_ident that* **by** *fastforce*
next
case *False*
then have *affS*: *affine hull S* = *UNIV*
by (*simp add*: *affine_hull_open* ‹*open S*›)
then have *ope*: *openin* (*top_of_set* (*affine hull S*)) *S*
using ‹*open S*› *open_openin* **by** *auto*
have $2 \le DIM('a)$ **by** (*rule 2*)
also have ... = *aff_dim* (*UNIV* :: *'a set*)
by *simp*
also have ... $\le$ *aff_dim S*
by (*metis aff_dim_UNIV aff_dim_affine_hull aff_dim_le_DIM affS*)
finally have $2 \le$ *aff_dim S*
by *linarith*
then show *?thesis*
using *homeomorphism_moving_points_exists_gen* [*OF* ‹*finite K*› *KS pw* _ *ope S*] *that* **by** *fastforce*
qed

The theorem *homeomorphism_grouping_points_exists*

lemma *homeomorphism_grouping_point_1*:
fixes *a*::*real* **and** *c*::*real*
assumes $a < b\ c < d$
obtains *f g* **where** *homeomorphism* (*cbox a b*) (*cbox c d*) *f g f a* = *c f b* = *d*
proof −
define *f* **where** $f \equiv \lambda x.\ ((d - c)\ /\ (b - a)) * x + (c - a * ((d - c)\ /\ (b - a)))$
have $\exists g.$ *homeomorphism* (*cbox a b*) (*cbox c d*) *f g*
proof (*rule homeomorphism_compact*)
show *continuous_on* (*cbox a b*) *f*
unfolding *f_def* **by** (*intro continuous_intros*)
have *f* ‘ $\{a..b\} = \{c..d\}$
unfolding *f_def image_affinity_atLeastAtMost*
using *assms sum_sqs_eq* **by** (*auto simp*: *field_split_simps*)
then show *f* ‘ *cbox a b* = *cbox c d*
by *auto*
show *inj_on f* (*cbox a b*)
unfolding *f_def inj_on_def* **using** *assms* **by** *auto*

```
  qed auto
  then obtain g where homeomorphism (cbox a b) (cbox c d) f g ..
  then show ?thesis
  proof
    show f a = c
      by (simp add: f_def)
    show f b = d
      using assms sum_sqs_eq [of a b] by (auto simp: f_def field_split_simps)
  qed
qed

lemma homeomorphism_grouping_point_2:
  fixes a::real and w::real
  assumes hom_ab: homeomorphism (cbox a b) (cbox u v) f1 g1
      and hom_bc: homeomorphism (cbox b c) (cbox v w) f2 g2
      and b ∈ cbox a c v ∈ cbox u w
      and eq: f1 a = u f1 b = v f2 b = v f2 c = w
 obtains f g where homeomorphism (cbox a c) (cbox u w) f g f a = u f c = w
                   ⋀x. x ∈ cbox a b ⟹ f x = f1 x ⋀x. x ∈ cbox b c ⟹ f x = f2 x
proof −
  have le: a ≤ b b ≤ c u ≤ v v ≤ w
    using assms by simp_all
  then have ac: cbox a c = cbox a b ∪ cbox b c and uw: cbox u w = cbox u v ∪
cbox v w
    by auto
  define f where f ≡ λx. if x ≤ b then f1 x else f2 x
  have ∃ g. homeomorphism (cbox a c) (cbox u w) f g
  proof (rule homeomorphism_compact)
    have cf1: continuous_on (cbox a b) f1
      using hom_ab homeomorphism_cont1 by blast
    have cf2: continuous_on (cbox b c) f2
      using hom_bc homeomorphism_cont1 by blast
    show continuous_on (cbox a c) f
      unfolding f_def using le eq
     by (force intro: continuous_on_cases_le [OF continuous_on_subset [OF cf1]
continuous_on_subset [OF cf2]])
    have f ‘ cbox a b = f1 ‘ cbox a b f ‘ cbox b c = f2 ‘ cbox b c
      unfolding f_def using eq by force+
    then show f ‘ cbox a c = cbox u w
     unfolding ac uw image_Un by (metis hom_ab hom_bc homeomorphism_def)
    have neq12: f1 x ≠ f2 y if x: a ≤ x x ≤ b and y: b < y y ≤ c for x y
    proof −
      have f1 x ∈ cbox u v
        by (metis hom_ab homeomorphism_def image_eqI mem_box_real(2) x)
      moreover have f2 y ∈ cbox v w
          by (metis (full_types) hom_bc homeomorphism_def image_subset_iff
mem_box_real(2) not_le not_less_iff_gr_or_eq order_refl y)
      moreover have f2 y ≠ f2 b
        by (metis cancel_comm_monoid_add_class.diff_cancel diff_gt_0_iff_gt
```

hom_bc homeomorphism_def le(2) less_imp_le less_numeral_extra(3) mem_box_real(2) order_refl y)
ultimately show *?thesis*
using *le eq* **by** *simp*
qed
have *inj_on f1* (*cbox a b*)
by (*metis* (*full_types*) *hom_ab homeomorphism_def inj_onI*)
moreover have *inj_on f2* (*cbox b c*)
by (*metis* (*full_types*) *hom_bc homeomorphism_def inj_onI*)
ultimately show *inj_on f* (*cbox a c*)
apply (*simp* (*no_asm*) *add*: *inj_on_def*)
apply (*simp add*: *f_def inj_on_eq_iff*)
using *neq12* **by** *force*
qed *auto*
then obtain *g* **where** *homeomorphism* (*cbox a c*) (*cbox u w*) *f g* **..**
then show *?thesis*
using *eq f_def le that* **by** *force*
qed

lemma *homeomorphism_grouping_point_3*:
fixes *a*::*real*
assumes *cbox_sub*: *cbox c d* ⊆ *box a b cbox u v* ⊆ *box a b*
and *box_ne*: *box c d* ≠ {} *box u v* ≠ {}
obtains *f g* **where** *homeomorphism* (*cbox a b*) (*cbox a b*) *f g f a* = *a f b* = *b*
⋀*x*. *x* ∈ *cbox c d* ⟹ *f x* ∈ *cbox u v*
proof −
have *less*: *a* < *c a* < *u d* < *b v* < *b c* < *d u* < *v cbox c d* ≠ {}
using *assms*
by (*simp_all add*: *cbox_sub subset_eq*)
obtain *f1 g1* **where** *1*: *homeomorphism* (*cbox a c*) (*cbox a u*) *f1 g1*
and *f1_eq*: *f1 a* = *a f1 c* = *u*
using *homeomorphism_grouping_point_1* [*OF* ‹*a* < *c*› ‹*a* < *u*›] **.**
obtain *f2 g2* **where** *2*: *homeomorphism* (*cbox c d*) (*cbox u v*) *f2 g2*
and *f2_eq*: *f2 c* = *u f2 d* = *v*
using *homeomorphism_grouping_point_1* [*OF* ‹*c* < *d*› ‹*u* < *v*›] **.**
obtain *f3 g3* **where** *3*: *homeomorphism* (*cbox d b*) (*cbox v b*) *f3 g3*
and *f3_eq*: *f3 d* = *v f3 b* = *b*
using *homeomorphism_grouping_point_1* [*OF* ‹*d* < *b*› ‹*v* < *b*›] **.**
obtain *f4 g4* **where** *4*: *homeomorphism* (*cbox a d*) (*cbox a v*) *f4 g4* **and** *f4 a* = *a f4 d* = *v*
and *f4_eq*: ⋀*x*. *x* ∈ *cbox a c* ⟹ *f4 x* = *f1 x* ⋀*x*. *x* ∈ *cbox c d* ⟹ *f4 x* = *f2 x*
using *homeomorphism_grouping_point_2* [*OF 1 2*] *less* **by** (*auto simp*: *f1_eq f2_eq*)
obtain *f g* **where** *fg*: *homeomorphism* (*cbox a b*) (*cbox a b*) *f g f a* = *a f b* = *b*
and *f_eq*: ⋀*x*. *x* ∈ *cbox a d* ⟹ *f x* = *f4 x* ⋀*x*. *x* ∈ *cbox d b* ⟹ *f x* = *f3 x*
using *homeomorphism_grouping_point_2* [*OF 4 3*] *less* **by** (*auto simp*: *f4_eq f3_eq f2_eq f1_eq*)

```
  show ?thesis
  proof (rule that [OF fg])
    show f x ∈ cbox u v if x ∈ cbox c d for x
      using that f4_eq f_eq homeomorphism_image1 [OF 2]
     by (metis atLeastAtMost_iff box_real(2) image_eqI less(1) less_eq_real_def
order_trans)
  qed
qed


lemma homeomorphism_grouping_point_4:
  fixes T :: real set
  assumes open U open S connected S U ≠ {} finite K K ⊆ S U ⊆ S S ⊆ T
  obtains f g where homeomorphism T T f g
                    ⋀x. x ∈ K ⟹ f x ∈ U {x. (¬ (f x = x ∧ g x = x))} ⊆ S
                    bounded {x. (¬ (f x = x ∧ g x = x))}
proof −
  obtain c d where box c d ≠ {} cbox c d ⊆ U
  proof −
    obtain u where u ∈ U
      using ‹U ≠ {}› by blast
    then obtain e where e > 0 cball u e ⊆ U
      using ‹open U› open_contains_cball by blast
    then show ?thesis
      by (rule_tac c=u and d=u+e in that) (auto simp: dist_norm subset_iff)
  qed
  have compact K
    by (simp add: ‹finite K› finite_imp_compact)
  obtain a b where box a b ≠ {} K ⊆ cbox a b cbox a b ⊆ S
  proof (cases K = {})
    case True then show ?thesis
      using ‹box c d ≠ {}› ‹cbox c d ⊆ U› ‹U ⊆ S› that by blast
  next
    case False
    then obtain a b where a ∈ K b ∈ K
          and a: ⋀x. x ∈ K ⟹ a ≤ x and b: ⋀x. x ∈ K ⟹ x ≤ b
      using compact_attains_inf compact_attains_sup by (metis ‹compact K›)+
    obtain e where e > 0 cball b e ⊆ S
      using ‹open S› open_contains_cball
      by (metis ‹b ∈ K› ‹K ⊆ S› subsetD)
    show ?thesis
    proof
      show box a (b + e) ≠ {}
        using ‹0 < e› ‹b ∈ K› a by force
      show K ⊆ cbox a (b + e)
        using ‹0 < e› a b by fastforce
      have a ∈ S
        using ‹a ∈ K› assms(6) by blast
      have b + e ∈ S
```

```
      using ‹0 < e› ‹cball b e ⊆ S› by (force simp: dist_norm)
    show cbox a (b + e) ⊆ S
      using ‹a ∈ S› ‹b + e ∈ S› ‹connected S› connected_contains_Icc by auto
  qed
 qed
 obtain w z where cbox w z ⊆ S and sub_wz: cbox a b ∪ cbox c d ⊆ box w z
 proof −
  have a ∈ S b ∈ S
    using ‹box a b ≠ {}› ‹cbox a b ⊆ S› by auto
  moreover have c ∈ S d ∈ S
    using ‹box c d ≠ {}› ‹cbox c d ⊆ U› ‹U ⊆ S› by force+
  ultimately have min a c ∈ S max b d ∈ S
    by linarith+
  then obtain e1 e2 where e1 > 0 cball (min a c) e1 ⊆ S e2 > 0 cball (max
b d) e2 ⊆ S
    using ‹open S› open_contains_cball by metis
  then have ∗: min a c − e1 ∈ S max b d + e2 ∈ S
    by (auto simp: dist_norm)
  show ?thesis
  proof
    show cbox (min a c − e1) (max b d+ e2) ⊆ S
      using ∗ ‹connected S› connected_contains_Icc by auto
    show cbox a b ∪ cbox c d ⊆ box (min a c − e1) (max b d + e2)
      using ‹0 < e1› ‹0 < e2› by auto
  qed
 qed
 then
 obtain f g where hom: homeomorphism (cbox w z) (cbox w z) f g
            and f w = w f z = z
            and fin: ⋀x. x ∈ cbox a b ⟹ f x ∈ cbox c d
  using homeomorphism_grouping_point_3 [of a b w z c d]
  using ‹box a b ≠ {}› ‹box c d ≠ {}› by blast
 have contfg: continuous_on (cbox w z) f continuous_on (cbox w z) g
  using hom homeomorphism_def by blast+
 define f′ where f′ ≡ λx. if x ∈ cbox w z then f x else x
 define g′ where g′ ≡ λx. if x ∈ cbox w z then g x else x
 show ?thesis
 proof
  have T: cbox w z ∪ (T − box w z) = T
    using ‹cbox w z ⊆ S› ‹S ⊆ T› by auto
  show homeomorphism T T f′ g′
  proof
    have clo: closedin (top_of_set (cbox w z ∪ (T − box w z))) (T − box w z)
     by (metis Diff_Diff_Int Diff_subset T closedin_def open_box openin_open_Int
topspace_euclidean_subtopology)
    have ⋀x. ⟦w ≤ x ∧ x ≤ z; w < x ⟶ ¬ x < z⟧ ⟹ f x = x
      using ‹f w = w› ‹f z = z› by auto
    moreover have ⋀x. ⟦w ≤ x ∧ x ≤ z; w < x ⟶ ¬ x < z⟧ ⟹ g x = x
      using ‹f w = w› ‹f z = z› hom homeomorphism_apply1 by fastforce
```

ultimately
have *continuous_on* (*cbox w z* ∪ (*T* − *box w z*)) *f′ continuous_on* (*cbox w z* ∪ (*T* − *box w z*)) *g′*
unfolding *f′_def g′_def*
by (*intro continuous_on_cases_local contfg continuous_on_id clo*; *auto simp*: *closed_subset*)+
then show *continuous_on T f′ continuous_on T g′*
by (*simp_all only*: *T*)
show *f′* ‘ *T* ⊆ *T*
unfolding *f′_def*
by *clarsimp* (*metis* ‹*cbox w z* ⊆ *S*› ‹*S* ⊆ *T*› *subsetD hom homeomorphism_def imageI mem_box_real*(*2*))
show *g′* ‘ *T* ⊆ *T*
unfolding *g′_def*
by *clarsimp* (*metis* ‹*cbox w z* ⊆ *S*› ‹*S* ⊆ *T*› *subsetD hom homeomorphism_def imageI mem_box_real*(*2*))
show ⋀*x*. *x* ∈ *T* ⟹ *g′* (*f′ x*) = *x*
unfolding *f′_def g′_def*
using *homeomorphism_apply1* [*OF hom*] *homeomorphism_image1* [*OF hom*] **by** *fastforce*
show ⋀*y*. *y* ∈ *T* ⟹ *f′* (*g′ y*) = *y*
unfolding *f′_def g′_def*
using *homeomorphism_apply2* [*OF hom*] *homeomorphism_image2* [*OF hom*] **by** *fastforce*
qed
show ⋀*x*. *x* ∈ *K* ⟹ *f′ x* ∈ *U*
using *fin sub_wz* ‹*K* ⊆ *cbox a b*› ‹*cbox c d* ⊆ *U*› **by** (*force simp*: *f′_def*)
show {*x*. ¬ (*f′ x* = *x* ∧ *g′ x* = *x*)} ⊆ *S*
using ‹*cbox w z* ⊆ *S*› **by** (*auto simp*: *f′_def g′_def*)
show *bounded* {*x*. ¬ (*f′ x* = *x* ∧ *g′ x* = *x*)}
proof (*rule bounded_subset* [*of cbox w z*])
show *bounded* (*cbox w z*)
using *bounded_cbox* **by** *blast*
show {*x*. ¬ (*f′ x* = *x* ∧ *g′ x* = *x*)} ⊆ *cbox w z*
by (*auto simp*: *f′_def g′_def*)
qed
qed
qed

proposition *homeomorphism_grouping_points_exists*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *open U open S connected S U* ≠ {} *finite K K* ⊆ *S U* ⊆ *S S* ⊆ *T*
obtains *f g* **where** *homeomorphism T T f g* {*x*. (¬ (*f x* = *x* ∧ *g x* = *x*))} ⊆ *S*
bounded {*x*. (¬ (*f x* = *x* ∧ *g x* = *x*))} ⋀*x*. *x* ∈ *K* ⟹ *f x* ∈ *U*
proof (*cases* 2 ≤ *DIM*(*'a*))
case *True*
have *TS*: *T* ⊆ *affine hull S*
using *affine_hull_open assms* **by** *blast*
have *infinite U*

using ‹*open U*› ‹$U \neq \{\}$› *finite_imp_not_open* **by** *blast*
then obtain *P* **where** $P \subseteq U$ *finite P card K = card P*
using *infinite_arbitrarily_large* **by** *metis*
then obtain γ **where** γ: *bij_betw* γ *K P*
using ‹*finite K*› *finite_same_card_bij* **by** *blast*
obtain *f g* **where** *homeomorphism T T f g* $\bigwedge i.\ i \in K \Longrightarrow f\ (id\ i) = \gamma\ i$ $\{x.\ \neg (f\ x = x \wedge g\ x = x)\} \subseteq S$ *bounded* $\{x.\ \neg (f\ x = x \wedge g\ x = x)\}$
proof (*rule homeomorphism_moving_points_exists* [*OF True* ‹*open S*› ‹*connected S*› ‹$S \subseteq T$› ‹*finite K*›])
show $\bigwedge i.\ i \in K \Longrightarrow id\ i \in S \wedge \gamma\ i \in S$
using ‹$P \subseteq U$› ‹*bij_betw* γ *K P*› ‹$K \subseteq S$› ‹$U \subseteq S$› *bij_betwE* **by** *blast*
show *pairwise* $(\lambda i\ j.\ id\ i \neq id\ j \wedge \gamma\ i \neq \gamma\ j)$ *K*
using γ **by** (*auto simp*: *pairwise_def bij_betw_def inj_on_def*)
qed (*use affine_hull_open assms that* **in** *auto*)
then show *?thesis*
using γ ‹$P \subseteq U$› *bij_betwE* **by** (*fastforce simp*: *intro*!: *that*)
next
case *False*
with *DIM_positive* **have** $DIM('a) = 1$
by (*simp add*: *dual_order.antisym*)
then obtain *h*::*'a* ⇒*real* **and** *j*
where *linear h linear j*
and *noh*: $\bigwedge x.\ norm(h\ x) = norm\ x$ **and** *noj*: $\bigwedge y.\ norm(j\ y) = norm\ y$
and *hj*: $\bigwedge x.\ j(h\ x) = x$ $\bigwedge y.\ h(j\ y) = y$
and *ranh*: *surj h*
using *isomorphisms_UNIV_UNIV*
by (*metis* (*mono_tags*, *opaque_lifting*) *DIM_real UNIV_eq_I range_eqI*)
obtain *f g* **where** *hom*: *homeomorphism* (*h* ‘ *T*) (*h* ‘ *T*) *f g*
and *f*: $\bigwedge x.\ x \in h$ ‘ $K \Longrightarrow f\ x \in h$ ‘ U
and *sub*: $\{x.\ \neg (f\ x = x \wedge g\ x = x)\} \subseteq h$ ‘ S
and *bou*: *bounded* $\{x.\ \neg (f\ x = x \wedge g\ x = x)\}$
apply (*rule homeomorphism_grouping_point_4* [*of h* ‘ *U h* ‘ *S h* ‘ *K h* ‘ *T*])
by (*simp_all add*: *assms image_mono* ‹*linear h*› *open_surjective_linear_image connected_linear_image ranh*)
have *jf*: $j\ (f\ (h\ x)) = x \longleftrightarrow f\ (h\ x) = h\ x$ **for** *x*
by (*metis hj*)
have *jg*: $j\ (g\ (h\ x)) = x \longleftrightarrow g\ (h\ x) = h\ x$ **for** *x*
by (*metis hj*)
have *cont_hj*: *continuous_on X h continuous_on Y j* **for** *X Y*
by (*simp_all add*: ‹*linear h*› ‹*linear j*› *linear_linear linear_continuous_on*)
show *?thesis*
proof
show *homeomorphism T T* $(j \circ f \circ h)$ $(j \circ g \circ h)$
proof
show *continuous_on T* $(j \circ f \circ h)$ *continuous_on T* $(j \circ g \circ h)$
using *hom homeomorphism_def*
by (*blast intro*: *continuous_on_compose cont_hj*)+
show $(j \circ f \circ h)$ ‘ $T \subseteq T$ $(j \circ g \circ h)$ ‘ $T \subseteq T$
by *auto* (*metis* (*mono_tags*, *opaque_lifting*) *hj*(*1*) *hom homeomorphism_def*

```
imageE imageI)+
      show ⋀x. x ∈ T ⟹ (j ∘ g ∘ h) ((j ∘ f ∘ h) x) = x
        using hj hom homeomorphism_apply1 by fastforce
      show ⋀y. y ∈ T ⟹ (j ∘ f ∘ h) ((j ∘ g ∘ h) y) = y
        using hj hom homeomorphism_apply2 by fastforce
    qed
    show {x. ¬ ((j ∘ f ∘ h) x = x ∧ (j ∘ g ∘ h) x = x)} ⊆ S
    proof (clarsimp simp: jf jg hj)
      show f (h x) = h x ⟶ g (h x) ≠ h x ⟹ x ∈ S for x
        using sub [THEN subsetD, of h x] hj by simp (metis imageE)
    qed
    have bounded (j ` {x. (¬ (f x = x ∧ g x = x))})
    by (rule bounded_linear_image [OF bou]) (use ‹linear j› linear_conv_bounded_linear
in auto)
    moreover
    have ∗: {x. ¬((j ∘ f ∘ h) x = x ∧ (j ∘ g ∘ h) x = x)} = j ` {x. (¬ (f x = x ∧
g x = x))}
      using hj by (auto simp: jf jg image_iff, metis+)
    ultimately show bounded {x. ¬ ((j ∘ f ∘ h) x = x ∧ (j ∘ g ∘ h) x = x)}
      by metis
    show ⋀x. x ∈ K ⟹ (j ∘ f ∘ h) x ∈ U
      using f hj by fastforce
  qed
qed


proposition homeomorphism_grouping_points_exists_gen:
  fixes S :: 'a::euclidean_space set
  assumes openU: openin (top_of_set S) U
      and openS: openin (top_of_set (affine hull S)) S
      and U ≠ {} finite K K ⊆ S and S: S ⊆ T T ⊆ affine hull S connected S
  obtains f g where homeomorphism T T f g {x. (¬ (f x = x ∧ g x = x))} ⊆ S
                    bounded {x. (¬ (f x = x ∧ g x = x))} ⋀x. x ∈ K ⟹ f x ∈ U
proof (cases 2 ≤ aff_dim S)
  case True
  have openU': openin (top_of_set (affine hull S)) U
    using openS openU openin_trans by blast
  obtain u where u ∈ U u ∈ S
    using ‹U ≠ {}› openU openin_imp_subset by fastforce+
  have infinite U
  proof (rule infinite_openin [OF openU ‹u ∈ U›])
    show u islimpt S
      using True ‹u ∈ S› assms(8) connected_imp_perfect_aff_dim by fastforce
  qed
  then obtain P where P ⊆ U finite P card K = card P
    using infinite_arbitrarily_large by metis
  then obtain γ where γ: bij_betw γ K P
    using ‹finite K› finite_same_card_bij by blast
  have ∃f g. homeomorphism T T f g ∧ (∀i ∈ K. f(id i) = γ i) ∧
```

$\{x.\ \neg\ (f\ x = x \wedge g\ x = x)\} \subseteq S \wedge$ *bounded* $\{x.\ \neg\ (f\ x = x \wedge g\ x = x)\}$
proof (*rule homeomorphism_moving_points_exists_gen* [*OF* ‹*finite K*› _ _ *True opeS S*])
show $\bigwedge i.\ i \in K \Longrightarrow id\ i \in S \wedge \gamma\ i \in S$
by (*metis id_apply opeU openin_contains_cball subsetCE* ‹$P \subseteq U$› ‹*bij_betw* γ *K P*› ‹$K \subseteq S$› *bij_betwE*)
show *pairwise* $(\lambda i\ j.\ id\ i \neq id\ j \wedge \gamma\ i \neq \gamma\ j)$ *K*
using γ **by** (*auto simp*: *pairwise_def bij_betw_def inj_on_def*)
qed
then show *?thesis*
using γ ‹$P \subseteq U$› *bij_betwE* **by** (*fastforce simp*: *intro*!: *that*)
next
case *False*
with *aff_dim_geq* [*of S*] **consider** *aff_dim S* $= -1$ | *aff_dim S* $= 0$ | *aff_dim S* $= 1$ **by** *linarith*
then show *?thesis*
proof *cases*
assume *aff_dim S* $= -1$
then have $S = \{\}$
using *aff_dim_empty* **by** *blast*
then have *False*
using ‹$U \neq \{\}$› ‹$K \subseteq S$› *openin_imp_subset* [*OF opeU*] **by** *blast*
then show *?thesis* **..**
next
assume *aff_dim S* $= 0$
then obtain *a* **where** $S = \{a\}$
using *aff_dim_eq_0* **by** *blast*
then have $K \subseteq U$
using ‹$U \neq \{\}$› ‹$K \subseteq S$› *openin_imp_subset* [*OF opeU*] **by** *blast*
show *?thesis*
using ‹$K \subseteq U$› **by** (*intro that* [*of id id*]) (*auto intro*: *homeomorphismI*)
next
assume *aff_dim S* $= 1$
then have *affine hull S homeomorphic* (*UNIV* :: *real set*)
by (*auto simp*: *homeomorphic_affine_sets*)
then obtain $h :: 'a \Rightarrow real$ **and** *j* **where** *homhj*: *homeomorphism* (*affine hull S*) *UNIV h j*
using *homeomorphic_def* **by** *blast*
then have *h*: $\bigwedge x.\ x \in$ *affine hull S* $\Longrightarrow j(h(x)) = x$ **and** *j*: $\bigwedge y.\ j\ y \in$ *affine hull S* $\wedge\ h(j\ y) = y$
by (*auto simp*: *homeomorphism_def*)
have *connh*: *connected* (*h* ‘ *S*)
by (*meson Topological_Spaces.connected_continuous_image* ‹*connected S*› *homeomorphism_cont1 homeomorphism_of_subsets homhj hull_subset top_greatest*)
have *hUS*: *h* ‘ *U* $\subseteq$ *h* ‘ *S*
by (*meson homeomorphism_imp_open_map homeomorphism_of_subsets homhj hull_subset opeS opeU open_UNIV openin_open_eq*)
have *opn*: *openin* (*top_of_set* (*affine hull S*)) *U* $\Longrightarrow$ *open* (*h* ‘ *U*) **for** *U*
using *homeomorphism_imp_open_map* [*OF homhj*] **by** *simp*

```
    have open (h ‘ U) open (h ‘ S)
      by (auto intro: opeS opeU openin_trans opn)
    then obtain f g where hom: homeomorphism (h ‘ T) (h ‘ T) f g
                and f: ⋀x. x ∈ h ‘ K ⟹ f x ∈ h ‘ U
                and sub: {x. ¬ (f x = x ∧ g x = x)} ⊆ h ‘ S
                and bou: bounded {x. ¬ (f x = x ∧ g x = x)}
      apply (rule homeomorphism_grouping_points_exists [of h ‘ U h ‘ S h ‘ K h
‘ T])
      using assms by (auto simp: connh hUS)
    have jf: ⋀x. x ∈ affine hull S ⟹ j (f (h x)) = x ⟷ f (h x) = h x
      by (metis h j)
    have jg: ⋀x. x ∈ affine hull S ⟹ j (g (h x)) = x ⟷ g (h x) = h x
      by (metis h j)
    have cont_hj: continuous_on T h  continuous_on Y j for Y
    proof (rule continuous_on_subset [OF _ ‹T ⊆ affine hull S›])
      show continuous_on (affine hull S) h
        using homeomorphism_def homhj by blast
    qed (meson continuous_on_subset homeomorphism_def homhj top_greatest)
    define f' where f' ≡ λx. if x ∈ affine hull S then (j ∘ f ∘ h) x else x
    define g' where g' ≡ λx. if x ∈ affine hull S then (j ∘ g ∘ h) x else x
    show ?thesis
    proof
      show homeomorphism T T f' g'
      proof
        have continuous_on T (j ∘ f ∘ h)
            using hom homeomorphism_def by (intro continuous_on_compose
cont_hj) blast
        then show continuous_on T f'
          apply (rule continuous_on_eq)
          using ‹T ⊆ affine hull S› f'_def by auto
        have continuous_on T (j ∘ g ∘ h)
            using hom homeomorphism_def by (intro continuous_on_compose
cont_hj) blast
        then show continuous_on T g'
          apply (rule continuous_on_eq)
          using ‹T ⊆ affine hull S› g'_def by auto
        show f' ‘ T ⊆ T
        proof (clarsimp simp: f'_def)
          fix x assume x ∈ T
          then have f (h x) ∈ h ‘ T
            by (metis (no_types) hom homeomorphism_def image_subset_iff sub-
set_refl)
          then show j (f (h x)) ∈ T
            using ‹T ⊆ affine hull S› h by auto
        qed
        show g' ‘ T ⊆ T
        proof (clarsimp simp: g'_def)
          fix x assume x ∈ T
          then have g (h x) ∈ h ‘ T
```

```
          by (metis (no_types) hom homeomorphism_def image_subset_iff sub-
set_refl)
        then show j (g (h x)) ∈ T
          using ‹T ⊆ affine hull S› h by auto
      qed
      show ⋀x. x ∈ T ⟹ g' (f' x) = x
       using h j hom homeomorphism_apply1 by (fastforce simp: f'_def g'_def)
      show ⋀y. y ∈ T ⟹ f' (g' y) = y
       using h j hom homeomorphism_apply2 by (fastforce simp: f'_def g'_def)
    qed
  next
    have §: ⋀x y. ⟦x ∈ affine hull S; h x = h y; y ∈ S⟧ ⟹ x ∈ S
      by (metis h hull_inc)
    show {x. ¬ (f' x = x ∧ g' x = x)} ⊆ S
      using sub by (simp add: f'_def g'_def jf jg) (force elim: §)
  next
    have compact (j ‘ closure {x. ¬ (f x = x ∧ g x = x)})
      using bou by (auto simp: compact_continuous_image cont_hj)
    then have bounded (j ‘ {x. ¬ (f x = x ∧ g x = x)})
      by (rule bounded_closure_image [OF compact_imp_bounded])
    moreover
    have ∗: {x ∈ affine hull S. j (f (h x)) ≠ x ∨ j (g (h x)) ≠ x} = j ‘ {x. (¬ (f
x = x ∧ g x = x))}
      using h j by (auto simp: image_iff; metis)
    ultimately have bounded {x ∈ affine hull S. j (f (h x)) ≠ x ∨ j (g (h x)) ≠
x}
      by metis
    then show bounded {x. ¬ (f' x = x ∧ g' x = x)}
      by (simp add: f'_def g'_def Collect_mono bounded_subset)
  next
    show f' x ∈ U if x ∈ K for x
    proof −
      have U ⊆ S
        using opeU openin_imp_subset by blast
      then have j (f (h x)) ∈ U
        using f h hull_subset that by fastforce
      then show f' x ∈ U
        using ‹K ⊆ S› S f'_def that by auto
    qed
  qed
 qed
qed
```

5.4.28 Nullhomotopic mappings

A mapping out of a sphere is nullhomotopic iff it extends to the ball. This even works out in the degenerate cases when the radius is ≤ 0, and we also don't need to explicitly assume continuity since it's already implicit in both sides of the equivalence.

lemma *nullhomotopic_from_lemma*:
assumes *contg*: *continuous_on* (*cball a r* − {*a*}) *g*
and *fa*: ⋀*e*. *0* < *e*
⟹ ∃ *d*. *0* < *d* ∧ (∀ *x*. *x* ≠ *a* ∧ *norm*(*x* − *a*) < *d* ⟶ *norm*(*g x* − *f a*) < *e*)
and *r*: ⋀*x*. *x* ∈ *cball a r* ∧ *x* ≠ *a* ⟹ *f x* = *g x*
shows *continuous_on* (*cball a r*) *f*
proof (*clarsimp simp*: *continuous_on_eq_continuous_within Ball_def*)
fix *x*
assume *x*: *dist a x* ≤ *r*
show *continuous* (*at x within cball a r*) *f*
proof (*cases x*=*a*)
case *True*
then show *?thesis*
by (*metis continuous_within_eps_delta fa dist_norm dist_self r*)
next
case *False*
show *?thesis*
proof (*rule continuous_transform_within* [**where** *f*=*g* **and** *d* = *norm*(*x*−*a*)])
have ∃ *d*>*0*. ∀ *x*′∈*cball a r*.
dist x′ *x* < *d* ⟶ *dist* (*g x*′) (*g x*) < *e* **if** *e*>*0* **for** *e*
proof −
obtain *d* **where** *d* > *0*
and *d*: ⋀*x*′. ⟦*dist x*′ *a* ≤ *r*; *x*′ ≠ *a*; *dist x*′ *x* < *d*⟧ ⟹
dist (*g x*′) (*g x*) < *e*
using *contg False x* ‹*e*>*0*›
unfolding *continuous_on_iff* **by** (*fastforce simp*: *dist_commute intro*: *that*)
show *?thesis*
using ‹*d* > *0*› ‹*x* ≠ *a*›
by (*rule_tac x*=*min d* (*norm*(*x* − *a*)) **in** *exI*)
(*auto simp*: *dist_commute dist_norm* [*symmetric*] *intro*!: *d*)
qed
then show *continuous* (*at x within cball a r*) *g*
using *contg False* **by** (*auto simp*: *continuous_within_eps_delta*)
show *0* < *norm* (*x* − *a*)
using *False* **by** *force*
show *x* ∈ *cball a r*
by (*simp add*: *x*)
show ⋀*x*′. ⟦*x*′ ∈ *cball a r*; *dist x*′ *x* < *norm* (*x* − *a*)⟧
⟹ *g x*′ = *f x*′
by (*metis dist_commute dist_norm less_le r*)
qed
qed
qed

proposition *nullhomotopic_from_sphere_extension*:
fixes *f* :: ′*M*::*euclidean_space* ⇒ ′*a*::*real_normed_vector*
shows (∃ *c*. *homotopic_with_canon* (λ*x*. *True*) (*sphere a r*) *S f* (λ*x*. *c*)) ⟷

```
        (∃g. continuous_on (cball a r) g ∧ g ‘ (cball a r) ⊆ S ∧
            (∀x ∈ sphere a r. g x = f x))
       (is ?lhs = ?rhs)
proof (cases r 0::real rule: linorder_cases)
  case less
  then show ?thesis
    by (simp add: homotopic_on_emptyI)
next
  case equal
  show ?thesis
  proof
    assume L: ?lhs
    with equal have [simp]: f a ∈ S
      using homotopic_with_imp_subset1 by fastforce
    obtain h:: real × 'M ⇒ 'a
      where h: continuous_on ({0..1} × {a}) h h ‘ ({0..1} × {a}) ⊆ S h (0, a)
= f a
      using L equal by (auto simp: homotopic_with)
    then have continuous_on (cball a r) (λx. h (0, a)) (λx. h (0, a)) ‘ cball a r
⊆ S
      by (auto simp: equal)
    then show ?rhs
      using h(3) local.equal by force
  next
    assume ?rhs
    then show ?lhs
      using equal continuous_on_const by (force simp: homotopic_with)
  qed
next
  case greater
  let ?P = continuous_on {x. norm(x − a) = r} f ∧ f ‘ {x. norm(x − a) = r}
⊆ S
  have ?P if ?lhs using that
  proof
    fix c
    assume c: homotopic_with_canon (λx. True) (sphere a r) S f (λx. c)
    then have contf: continuous_on (sphere a r) f
      by (metis homotopic_with_imp_continuous)
    moreover have fim: f ‘ sphere a r ⊆ S
    by (meson continuous_map_subtopology_eu c homotopic_with_imp_continuous_maps)
    show ?P
     using contf fim by (auto simp: sphere_def dist_norm norm_minus_commute)
  qed
  moreover have ?P if ?rhs using that
  proof
    fix g
    assume g: continuous_on (cball a r) g ∧ g ‘ cball a r ⊆ S ∧ (∀xa∈sphere a r.
g xa = f xa)
    then have f ‘ {x. norm (x − a) = r} ⊆ S
```

```
      using sphere_cball [of a r] unfolding image_subset_iff sphere_def
      by (metis dist_commute dist_norm mem_Collect_eq subset_eq)
    with g show ?P
      by (auto simp: dist_norm norm_minus_commute elim!: continuous_on_eq
[OF continuous_on_subset])
  qed
  moreover have ?thesis if ?P
  proof
    assume ?lhs
    then obtain c where homotopic_with_canon (λx. True) (sphere a r) S (λx.
c) f
      using homotopic_with_sym by blast
    then obtain h where conth: continuous_on ({0..1::real} × sphere a r) h
                    and him: h ` ({0..1} × sphere a r) ⊆ S
                    and h: ⋀x. h(0, x) = c ⋀x. h(1, x) = f x
      by (auto simp: homotopic_with_def)
    obtain b1::'M where b1 ∈ Basis
      using SOME_Basis by auto
    have c ∈ h ` ({0..1} × sphere a r)
    proof
      show c = h (0, a + r *R b1)
        by (simp add: h)
      show (0, a + r *R b1) ∈ {0..1::real} × sphere a r
        using greater ‹b1 ∈ Basis› by (auto simp: dist_norm)
    qed
    then have c ∈ S
      using him by blast
    have uconth: uniformly_continuous_on ({0..1::real} × (sphere a r)) h
      by (force intro: compact_Times conth compact_uniformly_continuous)
    let ?g = λx. h (norm (x − a)/r,
                    a + (if x = a then r *R b1 else (r / norm(x − a)) *R (x − a)))
    let ?g' = λx. h (norm (x − a)/r, a + (r / norm(x − a)) *R (x − a))
    show ?rhs
    proof (intro exI conjI)
      have continuous_on (cball a r − {a}) ?g'
        using greater
      by (force simp: dist_norm norm_minus_commute intro: continuous_on_compose2
[OF conth] continuous_intros)
      then show continuous_on (cball a r) ?g
      proof (rule nullhomotopic_from_lemma)
        show ∃ d>0. ∀ x. x ≠ a ∧ norm (x − a) < d ⟶ norm (?g' x − ?g a) < e
if 0 < e for e
        proof −
          obtain d where 0 < d
            and d: ⋀x x'. ⟦x ∈ {0..1} × sphere a r; x' ∈ {0..1} × sphere a r; norm
( x' − x) < d⟧
                          ⟹ norm (h x' − h x) < e
            using uniformly_continuous_onE [OF uconth ‹0 < e›] by (auto simp:
dist_norm)
```

```
        have ∗: norm (h (norm (x − a) / r,
                    a + (r / norm (x − a)) ∗R (x − a)) − h (0, a + r ∗R b1)) <
e (is  norm (?ha − ?hb) < e)
                  if x ≠ a norm (x − a) < r norm (x − a) < d ∗ r for x
        proof −
          have norm (?ha − ?hb) = norm (?ha − h (0, a + (r / norm (x − a))
∗R (x − a)))
            by (simp add: h)
          also have ... < e
            using greater ‹0 < d› ‹b1 ∈ Basis› that
            by (intro d) (simp_all add: dist_norm, simp add: field_simps)
          finally show ?thesis .
        qed
        show ?thesis
          using greater ‹0 < d›
          by (rule_tac x = min r (d ∗ r) in exI) (auto simp: ∗)
      qed
      show ⋀x. x ∈ cball a r ∧ x ≠ a ⟹ ?g x = ?g′ x
        by auto
    qed
  next
    show ?g ‘ cball a r ⊆ S
      using greater him ‹c ∈ S›
      by (force simp: h dist_norm norm_minus_commute)
  next
    show ∀ x∈sphere a r. ?g x = f x
      using greater by (auto simp: h dist_norm norm_minus_commute)
  qed
 next
  assume ?rhs
  then obtain g where contg: continuous_on (cball a r) g
                  and gim: g ‘ cball a r ⊆ S
                  and gf: ∀ x ∈ sphere a r. g x = f x
    by auto
  let ?h = λy. g (a + (fst y) ∗R (snd y − a))
  have continuous_on ({0..1} × sphere a r) ?h
  proof (rule continuous_on_compose2 [OF contg])
    show continuous_on ({0..1} × sphere a r) (λx. a + fst x ∗R (snd x − a))
      by (intro continuous_intros)
    qed (auto simp: dist_norm norm_minus_commute mult_left_le_one_le)
  moreover
  have ?h ‘ ({0..1} × sphere a r) ⊆ S
    by (auto simp: dist_norm norm_minus_commute mult_left_le_one_le gim
[THEN subsetD])
  moreover
  have ∀ x∈sphere a r. ?h (0, x) = g a ∀ x∈sphere a r. ?h (1, x) = f x
    by (auto simp: dist_norm norm_minus_commute mult_left_le_one_le gf)
  ultimately have homotopic_with_canon (λx. True) (sphere a r) S (λx. g a) f
    by (auto simp: homotopic_with)
```

then show *?lhs*
using *homotopic_with_symD* **by** *blast*
qed
ultimately
show *?thesis* **by** *meson*
qed

end

5.5 Euclidean space and n-spheres, as subtopologies of n-dimensional space

theory *Abstract_Euclidean_Space*
imports *Homotopy Locally*
begin

5.5.1 Euclidean spaces as abstract topologies

definition *Euclidean_space* :: *nat* ⇒ (*nat* ⇒ *real*) *topology*
where *Euclidean_space n* ≡ *subtopology* (*powertop_real UNIV*) {*x*. ∀ *i*≥*n*. *x i* = *0*}

lemma *topspace_Euclidean_space*:
topspace(*Euclidean_space n*) = {*x*. ∀ *i*≥*n*. *x i* = *0*}
by (*simp add*: *Euclidean_space_def*)

lemma *nontrivial_Euclidean_space*: *Euclidean_space n* ≠ *trivial_topology*
using *topspace_Euclidean_space* [*of n*] **by** *force*

lemma *subset_Euclidean_space* [*simp*]:
topspace(*Euclidean_space m*) ⊆ *topspace*(*Euclidean_space n*) ⟷ *m* ≤ *n*
apply (*simp add*: *topspace_Euclidean_space subset_iff*, *safe*)
apply (*drule_tac x*=(λ*i*. *if i* < *m then 1 else 0*) **in** *spec*)
apply *auto*
using *not_less* **by** *fastforce*

lemma *topspace_Euclidean_space_alt*:
topspace(*Euclidean_space n*) = (⋂ *i* ∈ {*n*..}. {*x*. *x* ∈ *topspace*(*powertop_real UNIV*) ∧ *x i* ∈ {*0*}})
by (*auto simp*: *topspace_Euclidean_space*)

lemma *closedin_Euclidean_space*:
closedin (*powertop_real UNIV*) (*topspace*(*Euclidean_space n*))
proof −
have *closedin* (*powertop_real UNIV*) {*x*. *x i* = *0*} **if** *n* ≤ *i* **for** *i*
proof −
have *closedin* (*powertop_real UNIV*) {*x* ∈ *topspace* (*powertop_real UNIV*). *x i* ∈ {*0*}}

proof (*rule closedin_continuous_map_preimage*)
show *continuous_map* (*powertop_real UNIV*) *euclideanreal* (λ*x*. *x i*)
by (*metis UNIV_I continuous_map_product_coordinates*)
show *closedin euclideanreal* {*0*}
by *simp*
qed
then show *?thesis*
by *auto*
qed
then show *?thesis*
unfolding *topspace_Euclidean_space_alt*
by *force*
qed

lemma *closedin_Euclidean_imp_closed*: *closedin* (*Euclidean_space m*) *S* ⟹ *closed S*
by (*metis Euclidean_space_def closed_closedin closedin_Euclidean_space closedin_closed_subtopology euclidean_product_topology topspace_Euclidean_space*)

lemma *closedin_Euclidean_space_iff*:
closedin (*Euclidean_space m*) *S* ⟷ *closed S* ∧ *S* ⊆ *topspace* (*Euclidean_space m*)
(**is** *?lhs* ⟷ *?rhs*)
proof
show *?lhs* ⟹ *?rhs*
using *closedin_closed_subtopology topspace_Euclidean_space*
by (*fastforce simp*: *topspace_Euclidean_space_alt closedin_Euclidean_imp_closed*)
show *?rhs* ⟹ *?lhs*
apply (*simp add*: *closedin_subtopology Euclidean_space_def*)
by (*metis* (*no_types*) *closed_closedin euclidean_product_topology inf.orderE*)
qed

lemma *continuous_map_componentwise_Euclidean_space*:
continuous_map X (*Euclidean_space n*) (λ*x i*. *if i* < *n then f x i else 0*) ⟷
(∀ *i* < *n*. *continuous_map X euclideanreal* (λ*x*. *f x i*))
proof −
have ∗: *continuous_map X euclideanreal* (λ*x*. *if k* < *n then f x k else 0*)
if ⋀*i*. *i*<*n* ⟹ *continuous_map X euclideanreal* (λ*x*. *f x i*) **for** *k*
by (*intro continuous_intros that*)
show *?thesis*
unfolding *Euclidean_space_def continuous_map_in_subtopology*
by (*fastforce simp*: *continuous_map_componentwise_UNIV* ∗ *elim*: *continuous_map_eq*)
qed

lemma *continuous_map_Euclidean_space_add* [*continuous_intros*]:
⟦*continuous_map X* (*Euclidean_space n*) *f*; *continuous_map X* (*Euclidean_space n*) *g*⟧
⟹ *continuous_map X* (*Euclidean_space n*) (λ*x i*. *f x i* + *g x i*)

unfolding *Euclidean_space_def continuous_map_in_subtopology*
by (*fastforce simp add*: *continuous_map_componentwise_UNIV continuous_map_add*)

lemma *continuous_map_Euclidean_space_diff* [*continuous_intros*]:
⟦*continuous_map X* (*Euclidean_space n*) *f*; *continuous_map X* (*Euclidean_space n*) *g*⟧
⟹ *continuous_map X* (*Euclidean_space n*) (λ*x i*. *f x i* − *g x i*)
unfolding *Euclidean_space_def continuous_map_in_subtopology*
by (*fastforce simp add*: *continuous_map_componentwise_UNIV continuous_map_diff*)

lemma *continuous_map_Euclidean_space_iff*:
continuous_map (*Euclidean_space m*) *euclidean g*
= *continuous_on* (*topspace* (*Euclidean_space m*)) *g*
proof
assume *continuous_map* (*Euclidean_space m*) *euclidean g*
then have *continuous_map* (*top_of_set* {*f*. ∀ *n*≥*m*. *f n* = *0*}) *euclidean g*
by (*simp add*: *Euclidean_space_def euclidean_product_topology*)
then show *continuous_on* (*topspace* (*Euclidean_space m*)) *g*
by (*metis continuous_map_subtopology_eu subtopology_topspace topspace_Euclidean_space*)
next
assume *continuous_on* (*topspace* (*Euclidean_space m*)) *g*
then have *continuous_map* (*top_of_set* {*f*. ∀ *n*≥*m*. *f n* = *0*}) *euclidean g*
by (*metis* (*lifting*) *continuous_map_into_fulltopology continuous_map_subtopology_eu order_refl topspace_Euclidean_space*)
then show *continuous_map* (*Euclidean_space m*) *euclidean g*
by (*simp add*: *Euclidean_space_def euclidean_product_topology*)
qed

lemma *cm_Euclidean_space_iff_continuous_on*:
continuous_map (*subtopology* (*Euclidean_space m*) *S*) (*Euclidean_space n*) *f*
⟷ *continuous_on* (*topspace* (*subtopology* (*Euclidean_space m*) *S*)) *f* ∧
f ∈ (*topspace* (*subtopology* (*Euclidean_space m*) *S*)) → *topspace* (*Euclidean_space n*)
(**is** *?P* ⟷ *?Q* ∧ *?R*)
proof −
have *?Q* **if** *?P*
proof −
have ⋀*n*. *Euclidean_space n* = *top_of_set* {*f*. ∀ *m*≥*n*. *f m* = *0*}
by (*simp add*: *Euclidean_space_def euclidean_product_topology*)
with *that* **show** *?thesis*
by (*simp add*: *subtopology_subtopology*)
qed
moreover
have *?R* **if** *?P*
using *that* **by** (*simp add*: *image_subset_iff continuous_map_def*)
moreover
have *?P* **if** *?Q ?R*
proof −
have *continuous_map* (*top_of_set* (*topspace* (*subtopology* (*subtopology* (*powertop_real*

UNIV) {*f*. ∀ *n*≥*m*. *f n* = *0*}) *S*))) (*top_of_set* (*topspace* (*subtopology* (*powertop_real UNIV*) {*f*. ∀ *na*≥*n*. *f na* = *0*}))) *f*
using *Euclidean_space_def that* **by** *auto*
then show *?thesis*
by (*simp add*: *Euclidean_space_def euclidean_product_topology subtopology_subtopology*)
qed
ultimately show *?thesis*
by *auto*
qed

lemma *homeomorphic_Euclidean_space_product_topology*:
Euclidean_space n homeomorphic_space product_topology (λ*i*. *euclideanreal*) {..<*n*}
proof −
have *cm*: *continuous_map* (*product_topology* (λ*i*. *euclideanreal*) {..<*n*}) *euclideanreal* (λ*x*. *if k* < *n then x k else 0*) **for** *k*
by (*auto intro*: *continuous_map_if continuous_map_product_projection*)
show *?thesis*
unfolding *homeomorphic_space_def homeomorphic_maps_def*
apply (*rule_tac x*=λ*f*. *restrict f* {..<*n*} **in** *exI*)
apply (*rule_tac x*=λ*f i*. *if i* < *n then f i else 0* **in** *exI*)
apply (*simp add*: *Euclidean_space_def continuous_map_in_subtopology*)
apply (*intro conjI continuous_map_from_subtopology*)
apply (*force simp*: *continuous_map_componentwise cm intro*: *continuous_map_product_projection*)+
done
qed

lemma *contractible_Euclidean_space* [*simp*]: *contractible_space* (*Euclidean_space n*)
using *homeomorphic_Euclidean_space_product_topology contractible_space_euclideanreal contractible_space_product_topology homeomorphic_space_contractibility* **by** *blast*

lemma *path_connected_Euclidean_space*: *path_connected_space* (*Euclidean_space n*)
by (*simp add*: *contractible_imp_path_connected_space*)

lemma *connected_Euclidean_space*: *connected_space* (*Euclidean_space n*)
by (*simp add*: *contractible_imp_connected_space*)

lemma *locally_path_connected_Euclidean_space*:
locally_path_connected_space (*Euclidean_space n*)
apply (*simp add*: *homeomorphic_locally_path_connected_space* [*OF homeomorphic_Euclidean_space_product_topology* [*of n*]] *locally_path_connected_space_product_topology*)
using *locally_path_connected_space_euclideanreal* **by** *auto*

lemma *compact_Euclidean_space* [*simp*]:
compact_space (*Euclidean_space n*) ⟷ *n* = *0*
using *homeomorphic_compact_space* [*OF homeomorphic_Euclidean_space_product_topology*]

by (*auto simp*: *product_topology_trivial_iff compact_space_product_topology*)

5.5.2 n-dimensional spheres

definition *nsphere* **where**
nsphere n ≡ *subtopology* (*Euclidean_space* (*Suc n*)) { *x*. (∑ *i*≤*n*. *x i* ^ *2*) = *1* }

lemma *nsphere*:
nsphere n = *subtopology* (*powertop_real UNIV*)
{*x*. (∑ *i*≤*n*. *x i* ^ *2*) = *1* ∧ (∀ *i*>*n*. *x i* = *0*)}
by (*simp add*: *nsphere_def Euclidean_space_def subtopology_subtopology Suc_le_eq Collect_conj_eq Int_commute*)

lemma *continuous_map_nsphere_projection*: *continuous_map* (*nsphere n*) *euclideanreal* (λ*x*. *x k*)
unfolding *nsphere*
by (*blast intro*: *continuous_map_from_subtopology* [*OF continuous_map_product_projection*])

lemma *in_topspace_nsphere*: (λ*n*. *if n* = *0 then 1 else 0*) ∈ *topspace* (*nsphere n*)
by (*simp add*: *nsphere_def topspace_Euclidean_space power2_eq_square if_distrib* [**where** *f* = λ*x*. *x* ∗ _] *cong*: *if_cong*)

lemma *nonempty_nsphere* [*simp*]: (*nsphere n*) ≠ *trivial_topology*
by (*metis discrete_topology_unique empty_iff in_topspace_nsphere*)

lemma *subtopology_nsphere_equator*:
subtopology (*nsphere* (*Suc n*)) {*x*. *x*(*Suc n*) = *0*} = *nsphere n*
proof −
have ({*x*. (∑ *i*≤*n*. (*x i*)²) + (*x* (*Suc n*))² = *1* ∧ (∀ *i*>*Suc n*. *x i* = *0*)} ∩ {*x*. *x* (*Suc n*) = *0*})
= {*x*. (∑ *i*≤*n*. (*x i*)²) = *1* ∧ (∀ *i*>*n*. *x i* = (*0*::*real*))}
using *Suc_lessI* [*of n*] **by** (*fastforce simp*: *set_eq_iff*)
then show *?thesis*
by (*simp add*: *nsphere subtopology_subtopology*)
qed

lemma *topspace_nsphere_minus1*:
assumes *x*: *x* ∈ *topspace* (*nsphere n*) **and** *x n* = *0*
shows *x* ∈ *topspace* (*nsphere* (*n* − *Suc 0*))
proof (*cases n* = *0*)
case *True*
then show *?thesis*
using *x* **by** *auto*
next
case *False*

```
  have subt_eq: nsphere (n - Suc 0) = subtopology (nsphere n) {x. x n = 0}
    by (metis False Suc_pred le_zero_eq not_le subtopology_nsphere_equator)
  with x show ?thesis
    by (simp add: assms)
qed

lemma continuous_map_nsphere_reflection:
  continuous_map (nsphere n) (nsphere n) (λx i. if i = k then -x i else x i)
proof -
  have cm: continuous_map (powertop_real UNIV) euclideanreal (λx. if j = k
then - x j else x j) for j
  proof (cases j=k)
    case True
    then show ?thesis
      by simp (metis UNIV_I continuous_map_product_projection)
  next
    case False
    then show ?thesis
      by (auto intro: continuous_map_product_projection)
  qed
  have eq: (if i = k then x k * x k else x i * x i) = x i * x i for i and x :: nat ⇒
real
    by simp
  show ?thesis
   apply (simp add: nsphere continuous_map_in_subtopology continuous_map_componentwise_UNIV
                    continuous_map_from_subtopology cm)
   apply (intro conjI allI impI continuous_intros continuous_map_from_subtopology
continuous_map_product_projection)
       apply (auto simp: power2_eq_square if_distrib [where f = λx. x * _] eq
cong: if_cong)
    done
qed


proposition contractible_space_upper_hemisphere:
  assumes k ≤ n
  shows contractible_space(subtopology (nsphere n) {x. x k ≥ 0})
proof -
  define p:: nat ⇒ real where p ≡ λi. if i = k then 1 else 0
  have p ∈ topspace(nsphere n)
    using assms
    by (simp add: nsphere p_def power2_eq_square if_distrib [where f = λx. x *
_] cong: if_cong)
  let ?g = λx i. x i / sqrt(∑ j≤n. x j ^ 2)
  let ?h = λ(t,q) i. (1 - t) * q i + t * p i
  let ?Y = subtopology (Euclidean_space (Suc n)) {x. 0 ≤ x k ∧ (∃ i≤n. x i ≠
0)}
  have continuous_map (prod_topology (top_of_set {0..1}) (subtopology (nsphere
n) {x. 0 ≤ x k}))
```

```
                (subtopology (nsphere n) {x. 0 ≤ x k}) (?g ∘ ?h)
  proof (rule continuous_map_compose)
    have ∗: ⟦0 ≤ b k; (∑ i≤n. (b i)²) = 1; ∀ i>n. b i = 0; 0 ≤ a; a ≤ 1⟧
          ⟹ ∃ i. (i = k ⟶ (1 − a) ∗ b k + a ≠ 0) ∧
                (i ≠ k ⟶ i ≤ n ∧ a ≠ 1 ∧ b i ≠ 0) for a::real and b
      apply (cases a ≠ 1 ∧ b k = 0; simp)
       apply (metis (no_types, lifting) atMost_iff sum.neutral zero_power2)
       by (metis add.commute add_le_same_cancel2 diff_ge_0_iff_ge diff_zero
less_eq_real_def mult_eq_0_iff mult_nonneg_nonneg not_le numeral_One zero_neq_numeral)
    show continuous_map (prod_topology (top_of_set {0..1}) (subtopology (nsphere
n) {x. 0 ≤ x k})) ?Y ?h
      using assms
      apply (auto simp: ∗ nsphere continuous_map_componentwise_UNIV
             prod_topology_subtopology subtopology_subtopology case_prod_unfold
            continuous_map_in_subtopology Euclidean_space_def p_def if_distrib
[where f = λx. _ ∗ x] cong: if_cong)
     apply (intro continuous_map_prod_snd continuous_intros continuous_map_from_subtopology)
       apply auto
      done
  next
    have 1: ⋀x i. ⟦ i ≤ n; x i ≠ 0⟧ ⟹ (∑ i≤n. (x i / sqrt (∑ j≤n. (x j)²))²) = 1
      by (force simp: sum_nonneg sum_nonneg_eq_0_iff field_split_simps simp
flip: sum_divide_distrib)
    have cm: continuous_map ?Y (nsphere n) (λx i. x i / sqrt (∑ j≤n. (x j)²))
      unfolding Euclidean_space_def nsphere subtopology_subtopology continu-
ous_map_in_subtopology
    proof (intro continuous_intros conjI)
      show continuous_map
             (subtopology (powertop_real UNIV) ({x. ∀ i≥Suc n. x i = 0} ∩ {x. 0
≤ x k ∧ (∃ i≤n. x i ≠ 0)}))
             (powertop_real UNIV) (λx i. x i / sqrt (∑ j≤n. (x j)²))
       unfolding continuous_map_componentwise
      by (intro continuous_intros conjI ballI) (auto simp: sum_nonneg_eq_0_iff)
    qed (auto simp: 1)
    show continuous_map ?Y (subtopology (nsphere n) {x. 0 ≤ x k}) (λx i. x i /
sqrt (∑ j≤n. (x j)²))
      by (force simp: cm sum_nonneg continuous_map_in_subtopology if_distrib
[where f = λx. _ ∗ x] cong: if_cong)
  qed
  moreover have (?g ∘ ?h) (0, x) = x
    if x ∈ topspace (subtopology (nsphere n) {x. 0 ≤ x k}) for x
    using that
    by (simp add: assms nsphere)
  moreover
  have (?g ∘ ?h) (1, x) = p
    if x ∈ topspace (subtopology (nsphere n) {x. 0 ≤ x k}) for x
    by (force simp: assms p_def power2_eq_square if_distrib [where f = λx. x ∗
_] cong: if_cong)
  ultimately
```

```
  show ?thesis
    apply (simp add: contractible_space_def homotopic_with)
    apply (rule_tac x=p in exI)
    apply (rule_tac x=?g ∘ ?h in exI, force)
    done
qed


corollary contractible_space_lower_hemisphere:
  assumes k ≤ n
  shows contractible_space(subtopology (nsphere n) {x. x k ≤ 0})
proof −
  have contractible_space (subtopology (nsphere n) {x. 0 ≤ x k}) = ?thesis
  proof (rule homeomorphic_space_contractibility)
    show subtopology (nsphere n) {x. 0 ≤ x k} homeomorphic_space subtopology
(nsphere n) {x. x k ≤ 0}
      unfolding homeomorphic_space_def homeomorphic_maps_def
      apply (rule_tac x=λx i. if i = k then −(x i) else x i in exI)+
    apply (auto simp: continuous_map_in_subtopology continuous_map_from_subtopology
                continuous_map_nsphere_reflection)
      done
  qed
  then show ?thesis
    using contractible_space_upper_hemisphere [OF assms] by metis
qed


proposition nullhomotopic_nonsurjective_sphere_map:
  assumes f: continuous_map (nsphere p) (nsphere p) f
    and fim: f ‘ (topspace(nsphere p)) ≠ topspace(nsphere p)
  obtains a where homotopic_with (λx. True) (nsphere p) (nsphere p) f (λx. a)
proof −
  obtain a where a: a ∈ topspace(nsphere p) a ∉ f ‘ (topspace(nsphere p))
    using fim continuous_map_image_subset_topspace f by blast
  then have a1: (∑ i≤p. (a i)^2) = 1 and a0: ⋀i. i > p ⟹ a i = 0
    by (simp_all add: nsphere)
  have f1: (∑ j≤p. (f x j)^2) = 1 if x ∈ topspace (nsphere p) for x
  proof −
    have f x ∈ topspace (nsphere p)
      using continuous_map_image_subset_topspace f that by blast
    then show ?thesis
      by (simp add: nsphere)
  qed
  show thesis
  proof
    let ?g = λx i. x i / sqrt(∑ j≤p. x j ^ 2)
    let ?h = λ(t,x) i. (1 − t) ∗ f x i − t ∗ a i
    let ?Y = subtopology (Euclidean_space(Suc p)) (− {λi. 0})
    let ?T01 = top_of_set {0..1::real}
```

have *1*: *continuous_map* (*prod_topology ?T01* (*nsphere p*)) (*nsphere p*) (*?g* ∘ *?h*)
proof (*rule continuous_map_compose*)
have *continuous_map* (*prod_topology ?T01* (*nsphere p*)) *euclideanreal* ((λ*x*. *f x k*) ∘ *snd*) **for** *k*
unfolding *nsphere*
apply (*simp add*: *continuous_map_of_snd flip*: *null_topspace_iff_trivial*)
apply (*rule continuous_map_compose* [*of* _ *nsphere p f*, *unfolded o_def*])
using *f* **apply** (*simp add*: *nsphere*)
by (*simp add*: *continuous_map_nsphere_projection*)
then have *continuous_map* (*prod_topology ?T01* (*nsphere p*)) *euclideanreal* (λ*r*. *?h r k*)
for *k*
unfolding *case_prod_unfold o_def*
by (*intro continuous_map_into_fulltopology* [*OF continuous_map_fst*] *continuous_intros*) *auto*
moreover have *?h* ‘ ({*0..1*} × *topspace* (*nsphere p*)) ⊆ {*x*. ∀ *i*≥*Suc p*. *x i* = *0*}
using *continuous_map_image_subset_topspace* [*OF f*]
by (*auto simp*: *nsphere image_subset_iff a0*)
moreover have (λ*i*. *0*) ∉ *?h* ‘ ({*0..1*} × *topspace* (*nsphere p*))
proof *clarify*
fix *t b*
assume *eq*: (λ*i*. *0*) = (λ*i*. (*1* − *t*) ∗ *f b i* − *t* ∗ *a i*) **and** *t* ∈ {*0..1*} **and** *b*: *b* ∈ *topspace* (*nsphere p*)
have (*1* − *t*)^2 = (∑ *i*≤*p*. ((*1* − *t*) ∗ *f b i*)^2)
using *f1* [*OF b*] **by** (*simp add*: *power_mult_distrib flip*: *sum_distrib_left*)
also have ... = (∑ *i*≤*p*. (*t* ∗ *a i*)^2)
using *eq* **by** (*simp add*: *fun_eq_iff*)
also have ... = *t*^2
using *a1* **by** (*simp add*: *power_mult_distrib flip*: *sum_distrib_left*)
finally have *1* − *t* = *t*
by (*simp add*: *power2_eq_iff*)
then have ∗: *t* = *1*/*2*
by *simp*
have *fba*: *f b* ≠ *a*
using *a*(*2*) *b* **by** *blast*
then show *False*
using *eq* **unfolding** ∗ **by** (*simp add*: *fun_eq_iff*)
qed
ultimately show *continuous_map* (*prod_topology ?T01* (*nsphere p*)) *?Y ?h*
by (*simp add*: *Euclidean_space_def continuous_map_in_subtopology continuous_map_componentwise_UNIV*)
next
have ∗: ⟦∀ *i*≥*Suc p*. *x i* = *0*; *x* ≠ (λ*i*. *0*)⟧ ⟹ (∑ *j*≤*p*. (*x j*)^2) ≠ *0* **for** *x* :: *nat* ⇒ *real*
by (*force simp*: *fun_eq_iff not_less_eq_eq sum_nonneg_eq_0_iff*)
show *continuous_map ?Y* (*nsphere p*) *?g*
apply (*simp add*: *Euclidean_space_def continuous_map_in_subtopology*

continuous_map_componentwise_UNIV
nsphere continuous_map_componentwise subtopology_subtopology)
apply (*intro conjI allI continuous_intros continuous_map_from_subtopology*
[*OF continuous_map_product_projection*])
apply (*simp_all add*: ∗)
apply (*force simp*: *sum_nonneg fun_eq_iff not_less_eq_eq sum_nonneg_eq_0_iff*
power_divide simp flip: *sum_divide_distrib*)
done
qed
have *2*: (*?g* ∘ *?h*) (*0*, *x*) = *f x* **if** *x* ∈ *topspace* (*nsphere p*) **for** *x*
using *that f1* **by** *simp*
have *3*: (*?g* ∘ *?h*) (*1*, *x*) = (λ*i*. − *a i*) **for** *x*
using *a* **by** (*force simp*: *field_split_simps nsphere*)
then show *homotopic_with* (λ*x*. *True*) (*nsphere p*) (*nsphere p*) *f* (λ*x*. (λ*i*. −
a i))
by (*force simp*: *homotopic_with intro*: *1 2 3*)
qed
qed

lemma *Hausdorff_Euclidean_space*:
Hausdorff_space (*Euclidean_space n*)
unfolding *Euclidean_space_def*
by (*rule Hausdorff_space_subtopology*) (*metis Hausdorff_space_euclidean Haus-*
dorff_space_product_topology)

end

5.6 Various Forms of Topological Spaces

theory *Abstract_Topological_Spaces*
imports *Lindelof_Spaces Locally Abstract_Euclidean_Space Sum_Topology FSigma*
begin

5.6.1 Connected topological spaces

lemma *connected_space_eq_frontier_eq_empty*:
connected_space X ⟷ (∀ *S*. *S* ⊆ *topspace X* ∧ *X frontier_of S* = {} ⟶ *S* =
{} ∨ *S* = *topspace X*)
by (*meson clopenin_eq_frontier_of connected_space_clopen_in*)

lemma *connected_space_frontier_eq_empty*:
connected_space X ∧ *S* ⊆ *topspace X*
⟹ (*X frontier_of S* = {} ⟷ *S* = {} ∨ *S* = *topspace X*)
by (*meson connected_space_eq_frontier_eq_empty frontier_of_empty frontier_of_topspace*)

lemma *connectedin_eq_subset_separated_union*:
connectedin X C ⟷
C ⊆ *topspace X* ∧ (∀ *S T*. *separatedin X S T* ∧ *C* ⊆ *S* ∪ *T* ⟶ *C* ⊆ *S* ∨
C ⊆ *T*) (**is** *?lhs=?rhs*)

proof
assume *?lhs* **then show** *?rhs*
using *connectedin_subset_topspace connectedin_subset_separated_union* **by** *blast*
next
assume *?rhs*
then show *?lhs*
by (*metis closure_of_subset connectedin_separation dual_order.eq_iff inf.orderE separatedin_def sup.boundedE*)
qed

lemma *connectedin_clopen_cases*:
⟦*connectedin X C*; *closedin X T*; *openin X T*⟧ ⟹ $C \subseteq T \vee$ *disjnt C T*
by (*metis Diff_eq_empty_iff Int_empty_right clopenin_eq_frontier_of connectedin_Int_frontier_of disjnt_def*)

lemma *connected_space_quotient_map_image*:
⟦*quotient_map X X' q*; *connected_space X*⟧ ⟹ *connected_space X'*
by (*metis connectedin_continuous_map_image connectedin_topspace quotient_imp_continuous_map quotient_imp_surjective_map*)

lemma *connected_space_retraction_map_image*:
⟦*retraction_map X X' r*; *connected_space X*⟧ ⟹ *connected_space X'*
using *connected_space_quotient_map_image retraction_imp_quotient_map* **by** *blast*

lemma *connectedin_imp_perfect_gen*:
assumes *X*: *t1_space X* **and** *S*: *connectedin X S* **and** *nontriv*: $\nexists a.\ S = \{a\}$
shows $S \subseteq X$ *derived_set_of S*
unfolding *derived_set_of_def*
proof (*intro subsetI CollectI conjI strip*)
show *XS*: $x \in$ *topspace X* **if** $x \in S$ **for** *x*
using *that S connectedin* **by** *fastforce*
show $\exists y.\ y \neq x \wedge y \in S \wedge y \in T$
if $x \in S$ **and** $x \in T \wedge$ *openin X T* **for** *x T*
proof −
have *opeXx*: *openin X* (*topspace X* − $\{x\}$)
by (*meson X openin_topspace t1_space_openin_delete_alt*)
moreover
have $S \subseteq T \cup$ (*topspace X* − $\{x\}$)
using *XS that*(*2*) **by** *auto*
moreover have (*topspace X* − $\{x\}$) ∩ $S \neq \{\}$
by (*metis Diff_triv S connectedin double_diff empty_subsetI inf_commute insert_subsetI nontriv that*(*1*))
ultimately show *?thesis*
using *that connectedinD* [*OF S*, *of T topspace X* − $\{x\}$]
by *blast*
qed
qed

lemma *connectedin_imp_perfect*:
⟦*Hausdorff_space X*; *connectedin X S*; ∄*a*. *S* = {*a*}⟧ ⟹ *S* ⊆ *X derived_set_of S*
by (*simp add*: *Hausdorff_imp_t1_space connectedin_imp_perfect_gen*)

5.6.2 The notion of "separated between" (complement of "connected between)"

definition *separated_between*
where *separated_between X S T* ≡
∃ *U V*. *openin X U* ∧ *openin X V* ∧ *U* ∪ *V* = *topspace X* ∧ *disjnt U V* ∧ *S* ⊆ *U* ∧ *T* ⊆ *V*

lemma *separated_between_alt*:
separated_between X S T ⟷
(∃ *U V*. *closedin X U* ∧ *closedin X V* ∧ *U* ∪ *V* = *topspace X* ∧ *disjnt U V* ∧ *S* ⊆ *U* ∧ *T* ⊆ *V*)
unfolding *separated_between_def*
by (*metis separatedin_open_sets separation_closedin_Un_gen subtopology_topspace*
separatedin_closed_sets separation_openin_Un_gen)

lemma *separated_between*:
separated_between X S T ⟷
(∃ *U*. *closedin X U* ∧ *openin X U* ∧ *S* ⊆ *U* ∧ *T* ⊆ *topspace X* − *U*)
unfolding *separated_between_def closedin_def disjnt_def*
by (*smt* (*verit*, *del_insts*) *Diff_cancel Diff_disjoint Diff_partition Un_Diff Un_Diff_Int openin_subset*)

lemma *separated_between_mono*:
⟦*separated_between X S T*; *S'* ⊆ *S*; *T'* ⊆ *T*⟧ ⟹ *separated_between X S' T'*
by (*meson order.trans separated_between*)

lemma *separated_between_refl*:
separated_between X S S ⟷ *S* = {}
unfolding *separated_between_def*
by (*metis Un_empty_right disjnt_def disjnt_empty2 disjnt_subset2 disjnt_sym le_iff_inf openin_empty openin_topspace*)

lemma *separated_between_sym*:
separated_between X S T ⟷ *separated_between X T S*
by (*metis disjnt_sym separated_between_alt sup_commute*)

lemma *separated_between_imp_subset*:
separated_between X S T ⟹ *S* ⊆ *topspace X* ∧ *T* ⊆ *topspace X*
by (*metis le_supI1 le_supI2 separated_between_def*)

lemma *separated_between_empty*:

(*separated_between X {} S* ⟷ *S* ⊆ *topspace X*) ∧ (*separated_between X S {}* ⟷ *S* ⊆ *topspace X*)
by (*metis Diff_empty bot.extremum closedin_empty openin_empty separated_between separated_between_imp_subset separated_between_sym*)

lemma *separated_between_Un*:
separated_between X S (*T* ∪ *U*) ⟷ *separated_between X S T* ∧ *separated_between X S U*
by (*auto simp*: *separated_between*)

lemma *separated_between_Un'*:
separated_between X (*S* ∪ *T*) *U* ⟷ *separated_between X S U* ∧ *separated_between X T U*
by (*simp add*: *separated_between_Un separated_between_sym*)

lemma *separated_between_imp_disjoint*:
separated_between X S T ⟹ *disjnt S T*
by (*meson disjnt_iff separated_between_def subsetD*)

lemma *separated_between_imp_separatedin*:
separated_between X S T ⟹ *separatedin X S T*
by (*meson separated_between_def separatedin_mono separatedin_open_sets*)

lemma *separated_between_full*:
assumes *S* ∪ *T* = *topspace X*
shows *separated_between X S T* ⟷ *disjnt S T* ∧ *closedin X S* ∧ *openin X S* ∧ *closedin X T* ∧ *openin X T*
proof −
have *separated_between X S T* ⟶ *separatedin X S T*
by (*simp add*: *separated_between_imp_separatedin*)
then show *?thesis*
unfolding *separated_between_def*
by (*metis assms separation_closedin_Un_gen separation_openin_Un_gen subset_refl subtopology_topspace*)
qed

lemma *separated_between_eq_separatedin*:
S ∪ *T* = *topspace X* ⟹ (*separated_between X S T* ⟷ *separatedin X S T*)
by (*simp add*: *separated_between_full separatedin_full*)

lemma *separated_between_pointwise_left*:
assumes *compactin X S*
shows *separated_between X S T* ⟷
(*S* = {} ⟶ *T* ⊆ *topspace X*) ∧ (∀ *x* ∈ *S*. *separated_between X* {*x*} *T*)
(**is** *?lhs=?rhs*)
proof
assume *?lhs* **then show** *?rhs*
using *separated_between_imp_subset separated_between_mono* **by** *fastforce*

next
assume *R*: *?rhs*
then have $T \subseteq$ *topspace X*
by (*meson equals0I separated_between_imp_subset*)
show *?lhs*
proof −
obtain *U* **where** *U*: $\forall x \in S$. *openin X* (*U x*)
$\forall x \in S. \exists V$. *openin X V* $\wedge$ *U x* $\cup$ *V* = *topspace X* $\wedge$ *disjnt* (*U x*) *V* $\wedge$ $\{x\} \subseteq$ *U x* $\wedge$ *T* $\subseteq$ *V*
using *R* **unfolding** *separated_between_def* **by** *metis*
then have $S \subseteq \bigcup (U \mathbin{`} S)$
by *blast*
then obtain *K* **where** *finite K K* $\subseteq$ *S* **and** *K*: $S \subseteq (\bigcup i{\in}K.\ U\ i)$
using *assms U* **unfolding** *compactin_def* **by** (*smt* (*verit*) *finite_subset_image imageE*)
show *?thesis*
unfolding *separated_between*
proof (*intro conjI exI*)
have $\bigwedge x.\ x \in K \Longrightarrow$ *closedin X* (*U x*)
by (*smt* (*verit*) ‹*K* $\subseteq$ *S*› *Diff_cancel U*(*2*) *Un_Diff Un_Diff_Int disjnt_def openin_closedin_eq subsetD*)
then show *closedin X* ($\bigcup$ (*U* ‘ *K*))
by (*metis* (*mono_tags*, *lifting*) ‹*finite K*› *closedin_Union finite_imageI image_iff*)
show *openin X* ($\bigcup$ (*U* ‘ *K*))
using *U*(*1*) ‹*K* $\subseteq$ *S*› **by** *blast*
show *S* $\subseteq \bigcup$ (*U* ‘ *K*)
by (*simp add*: *K*)
have $\bigwedge x\ i.\ [\![x \in T;\ i \in K;\ x \in U\ i]\!] \Longrightarrow$ *False*
by (*meson U*(*2*) ‹*K* $\subseteq$ *S*› *disjnt_iff subsetD*)
then show *T* $\subseteq$ *topspace X* − $\bigcup$ (*U* ‘ *K*)
using ‹*T* $\subseteq$ *topspace X*› **by** *auto*
qed
qed
qed

lemma *separated_between_pointwise_right*:
compactin X T
$\Longrightarrow$ *separated_between X S T* $\longleftrightarrow$ (*T* = {} $\longrightarrow$ *S* $\subseteq$ *topspace X*) $\wedge$ ($\forall y \in T$. *separated_between X S* $\{y\}$)
by (*meson separated_between_pointwise_left separated_between_sym*)

lemma *separated_between_closure_of*:
S $\subseteq$ *topspace X* $\Longrightarrow$ *separated_between X* (*X closure_of S*) *T* $\longleftrightarrow$ *separated_between X S T*
by (*meson closure_of_minimal_eq separated_between_alt*)

lemma *separated_between_closure_of'*:

$T \subseteq$ *topspace* $X \Longrightarrow$ *separated_between* $X\ S$ (X *closure_of* T) $\longleftrightarrow$ *separated_between* $X\ S\ T$
by (*meson separated_between_closure_of separated_between_sym*)

lemma *separated_between_closure_of_eq*:
separated_between $X\ S\ T \longleftrightarrow S \subseteq$ *topspace* $X \wedge$ *separated_between* X (X *closure_of* S) T
by (*metis separated_between_closure_of separated_between_imp_subset*)

lemma *separated_between_closure_of_eq'*:
separated_between $X\ S\ T \longleftrightarrow T \subseteq$ *topspace* $X \wedge$ *separated_between* $X\ S$ (X *closure_of* T)
by (*metis separated_between_closure_of' separated_between_imp_subset*)

lemma *separated_between_frontier_of_eq'*:
separated_between $X\ S\ T \longleftrightarrow$
$T \subseteq$ *topspace* $X \wedge$ *disjnt* $S\ T \wedge$ *separated_between* $X\ S$ (X *frontier_of* T) (**is** *?lhs=?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*metis interior_of_union_frontier_of separated_between_Un separated_between_closure_of_eq'*
separated_between_imp_disjoint)
next
assume *R*: *?rhs*
then obtain *U* **where** *U*: *closedin* $X\ U$ *openin* $X\ U\ S \subseteq U\ X$ *closure_of* $T -$ X *interior_of* $T \subseteq$ *topspace* $X - U$
by (*metis frontier_of_def separated_between*)
show *?lhs*
proof (*rule separated_between_mono* [*of* _ $S\ X$ *closure_of* T])
have *separated_between* $X\ S\ T$
unfolding *separated_between*
proof (*intro conjI exI*)
show $S \subseteq U - T\ T \subseteq$ *topspace* $X - (U - T)$
using $R\ U(3)$ **by** (*force simp*: *disjnt_iff*)+
have $T \subseteq X$ *closure_of* T
by (*simp add*: R *closure_of_subset*)
then have $*$: $U - T = U - X$ *interior_of* T
using $U(4)$ *interior_of_subset* **by** *fastforce*
then show *closedin* X ($U - T$)
by (*simp add*: $U(1)$ *closedin_diff*)
have $U \cap X$ *frontier_of* $T = \{\}$
using $U(4)$ *frontier_of_def* **by** *fastforce*
then show *openin* X ($U - T$)
by (*metis* $*$ *Diff_Un* $U(2)$ *Un_Diff_Int Un_Int_eq*(1) *closedin_closure_of interior_of_union_frontier_of openin_diff sup_bot_right*)
qed
then show *separated_between* $X\ S$ (X *closure_of* T)
by (*simp add*: R *separated_between_closure_of'*)

qed (*auto simp*: *R closure_of_subset*)
qed

lemma *separated_between_frontier_of_eq*:
separated_between X S T ⟷ *S* ⊆ *topspace X* ∧ *disjnt S T* ∧ *separated_between X* (*X frontier_of S*) *T*
by (*metis disjnt_sym separated_between_frontier_of_eq' separated_between_sym*)

lemma *separated_between_frontier_of*:
⟦*S* ⊆ *topspace X*; *disjnt S T*⟧
⟹ (*separated_between X* (*X frontier_of S*) *T* ⟷ *separated_between X S T*)
using *separated_between_frontier_of_eq* **by** *blast*

lemma *separated_between_frontier_of'*:
⟦*T* ⊆ *topspace X*; *disjnt S T*⟧
⟹ (*separated_between X S* (*X frontier_of T*) ⟷ *separated_between X S T*)
using *separated_between_frontier_of_eq'* **by** *auto*

lemma *connected_space_separated_between*:
connected_space X ⟷ (∀ *S T*. *separated_between X S T* ⟶ *S* = {} ∨ *T* = {}) (**is** *?lhs=?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*metis Diff_cancel connected_space_clopen_in separated_between subset_empty*)
next
assume *?rhs* **then show** *?lhs*
by (*meson connected_space_eq_not_separated separated_between_eq_separatedin*)
qed

lemma *connected_space_imp_separated_between_trivial*:
connected_space X
⟹ (*separated_between X S T* ⟷ *S* = {} ∧ *T* ⊆ *topspace X* ∨ *S* ⊆ *topspace X* ∧ *T* = {})
by (*metis connected_space_separated_between separated_between_empty*)

5.6.3 Connected components

lemma *connected_component_of_subtopology_eq*:
connected_component_of (*subtopology X U*) *a* = *connected_component_of X a*
⟷
connected_component_of_set X a ⊆ *U*
by (*force simp*: *connected_component_of_set connectedin_subtopology connected_component_of_def fun_eq_iff subset_iff*)

lemma *connected_components_of_subtopology*:
assumes *C* ∈ *connected_components_of X C* ⊆ *U*
shows *C* ∈ *connected_components_of* (*subtopology X U*)
proof −
obtain *a* **where** *a*: *connected_component_of_set X a* ⊆ *U* **and** *a* ∈ *topspace X*

```
          and Ceq: C = connected_component_of_set X a
    using assms by (force simp: connected_components_of_def)
  then have a ∈ U
    by (simp add: connected_component_of_refl in_mono)
  then have connected_component_of_set X a = connected_component_of_set
(subtopology X U) a
    by (metis a connected_component_of_subtopology_eq)
  then show ?thesis
   by (simp add: Ceq ‹a ∈ U› ‹a ∈ topspace X› connected_component_in_connected_components_of)
qed

thm connected_space_iff_components_eq

lemma open_in_finite_connected_components:
  assumes finite(connected_components_of X) C ∈ connected_components_of X
  shows openin X C
proof −
  have closedin X (topspace X − C)
     by (metis DiffD1 assms closedin_Union closedin_connected_components_of
complement_connected_components_of_Union finite_Diff)
  then show ?thesis
    by (simp add: assms connected_components_of_subset openin_closedin)
qed
thm connected_component_of_eq_overlap

lemma connected_components_of_disjoint:
  assumes C ∈ connected_components_of X C′ ∈ connected_components_of X
    shows (disjnt C C′ ⟷ (C ≠ C′))
proof −
  have C ≠ {}
    using nonempty_connected_components_of assms by blast
  with assms show ?thesis
   by (metis disjnt_self_iff_empty pairwiseD pairwise_disjoint_connected_components_of)
qed

lemma connected_components_of_overlap:
   ⟦C ∈ connected_components_of X; C′ ∈ connected_components_of X⟧ ⟹ C
∩ C′ ≠ {} ⟷ C = C′
  by (meson connected_components_of_disjoint disjnt_def)

lemma pairwise_separated_connected_components_of:
   pairwise (separatedin X) (connected_components_of X)
 by (simp add: closedin_connected_components_of connected_components_of_disjoint
pairwiseI separatedin_closed_sets)

lemma finite_connected_components_of_finite:
   finite(topspace X) ⟹ finite(connected_components_of X)
  by (simp add: Union_connected_components_of finite_UnionD)
```

lemma *connected_component_of_unique*:
  ⟦*x* ∈ *C*; *connectedin X C*; ⋀*C'*. *x* ∈ *C'* ∧ *connectedin X C'* ⟹ *C'* ⊆ *C*⟧
    ⟹ *connected_component_of_set X x* = *C*
 by (*meson connected_component_of_maximal connectedin_connected_component_of subsetD subset_antisym*)

lemma *closedin_connected_component_of_subtopology*:
  ⟦*C* ∈ *connected_components_of* (*subtopology X s*); *X closure_of C* ⊆ *s*⟧ ⟹ *closedin X C*
 by (*metis closedin_Int_closure_of closedin_connected_components_of closure_of_eq inf.absorb_iff2*)

lemma *connected_component_of_discrete_topology*:
  connected_component_of_set (*discrete_topology U*) *x* = (*if x* ∈ *U then* {*x*} *else* {})
 by (*simp add*: *locally_path_connected_space_discrete_topology flip*: *path_component_eq_connected_component_*

lemma *connected_components_of_discrete_topology*:
  connected_components_of (*discrete_topology U*) = (λ*x*. {*x*}) ` *U*
 by (*simp add*: *connected_component_of_discrete_topology connected_components_of_def*)

lemma *connected_component_of_continuous_image*:
  ⟦*continuous_map X Y f*; *connected_component_of X x y*⟧
    ⟹ *connected_component_of Y* (*f x*) (*f y*)
 by (*meson connected_component_of_def connectedin_continuous_map_image image_eqI*)

lemma *homeomorphic_map_connected_component_of*:
 assumes *homeomorphic_map X Y f* **and** *x*: *x* ∈ *topspace X*
 shows *connected_component_of_set Y* (*f x*) = *f* ` (*connected_component_of_set X x*)
proof −
 obtain *g* **where** *g*: *continuous_map X Y f*
   continuous_map Y X g ⋀*x*. *x* ∈ *topspace X* ⟹ *g* (*f x*) = *x*
   ⋀*y*. *y* ∈ *topspace Y* ⟹ *f* (*g y*) = *y*
   using *assms*(*1*) *homeomorphic_map_maps homeomorphic_maps_def* **by** *fastforce*
 show *?thesis*
   using *connected_component_in_topspace* [*of Y*] *x g*
       connected_component_of_continuous_image [*of X Y f*]
       connected_component_of_continuous_image [*of Y X g*]
   by *force*
qed

lemma *homeomorphic_map_connected_components_of*:
 assumes *homeomorphic_map X Y f*
 shows *connected_components_of Y* = (*image f*) ` (*connected_components_of X*)
proof −

have *topspace Y = f ‘ topspace X*
by (*metis assms homeomorphic_imp_surjective_map*)
with *homeomorphic_map_connected_component_of* [*OF assms*] **show** *?thesis*
by (*auto simp*: *connected_components_of_def image_iff*)
qed

lemma *connected_component_of_pair*:
connected_component_of_set (*prod_topology X Y*) (*x*,*y*) =
connected_component_of_set X x × *connected_component_of_set Y y*
proof (*cases x* ∈ *topspace X* ∧ *y* ∈ *topspace Y*)
case *True*
show *?thesis*
proof (*rule connected_component_of_unique*)
show (*x*, *y*) ∈ *connected_component_of_set X x* × *connected_component_of_set Y y*
using *True* **by** (*simp add*: *connected_component_of_refl*)
show *connectedin* (*prod_topology X Y*) (*connected_component_of_set X x* × *connected_component_of_set Y y*)
by (*metis connectedin_Times connectedin_connected_component_of*)
show *C* ⊆ *connected_component_of_set X x* × *connected_component_of_set Y y*
if (*x*, *y*) ∈ *C* ∧ *connectedin* (*prod_topology X Y*) *C* **for** *C*
using *that* **unfolding** *connected_component_of_def*
apply *clarsimp*
by (*metis* (*no_types*) *connectedin_continuous_map_image continuous_map_fst continuous_map_snd fst_conv imageI snd_conv*)
qed
next
case *False* **then show** *?thesis*
by (*metis Sigma_empty1 Sigma_empty2 connected_component_of_eq_empty mem_Sigma_iff topspace_prod_topology*)
qed

lemma *connected_components_of_prod_topology*:
connected_components_of (*prod_topology X Y*) =
{*C* × *D* | *C D*. *C* ∈ *connected_components_of X* ∧ *D* ∈ *connected_components_of Y*} (**is** *?lhs*=*?rhs*)
proof
show *?lhs* ⊆ *?rhs*
apply (*clarsimp simp*: *connected_components_of_def*)
by (*metis* (*no_types*) *connected_component_of_pair imageI*)
next
show *?rhs* ⊆ *?lhs*
using *connected_component_of_pair*
by (*fastforce simp*: *connected_components_of_def*)
qed

lemma *connected_component_of_product_topology*:

connected_component_of_set (*product_topology X I*) *x* =
(*if x* ∈ *extensional I then PiE I* (λ*i*. *connected_component_of_set* (*X i*) (*x i*))
else {})
(**is** *?lhs* = *If* _ *?R* _)
proof (*cases x* ∈ *topspace*(*product_topology X I*))
case *True*
have *?lhs* = (Π$_E$ *i*∈*I*. *connected_component_of_set* (*X i*) (*x i*))
if *xX*: ⋀*i*. *i*∈*I* ⟹ *x i* ∈ *topspace* (*X i*) **and** *ext*: *x* ∈ *extensional I*
proof (*rule connected_component_of_unique*)
show *x* ∈ *?R*
by (*simp add*: *PiE_iff connected_component_of_refl local.ext xX*)
show *connectedin* (*product_topology X I*) *?R*
by (*simp add*: *connectedin_PiE connectedin_connected_component_of*)
show *C* ⊆ *?R*
if *x* ∈ *C* ∧ *connectedin* (*product_topology X I*) *C* **for** *C*
proof −
have *C* ⊆ *extensional I*
using *PiE_def connectedin_subset_topspace that* **by** *fastforce*
have ⋀*y*. *y* ∈ *C* ⟹ *y* ∈ (Π *i*∈*I*. *connected_component_of_set* (*X i*) (*x i*))
apply (*simp add*: *connected_component_of_def Pi_def*)
by (*metis connectedin_continuous_map_image continuous_map_product_projection imageI that*)
then show *?thesis*
using *PiE_def* ‹*C* ⊆ *extensional I*› **by** *fastforce*
qed
qed
with *True* **show** *?thesis*
by (*simp add*: *PiE_iff*)
next
case *False*
then show *?thesis*
apply (*simp add*: *PiE_iff*)
by (*smt* (*verit*) *Collect_empty_eq False PiE_eq_empty_iff PiE_iff connected_component_of_eq_empty*)
qed

lemma *connected_components_of_product_topology*:
connected_components_of (*product_topology X I*) =
{*PiE I B* |*B*. ∀ *i* ∈ *I*. *B i* ∈ *connected_components_of*(*X i*)} (**is** *?lhs*=*?rhs*)
proof
show *?lhs* ⊆ *?rhs*
by (*auto simp*: *connected_components_of_def connected_component_of_product_topology PiE_iff*)
show *?rhs* ⊆ *?lhs*
proof
fix *F*
assume *F* ∈ *?rhs*
then obtain *B* **where** *Feq*: *F* = *Pi*$_E$ *I B* **and**
∀ *i*∈*I*. ∃ *x*∈*topspace* (*X i*). *B i* = *connected_component_of_set* (*X i*) *x*

by (*force simp*: *connected_components_of_def connected_component_of_product_topology image_iff*)
then obtain *f* **where**
f: $\bigwedge i.\ i \in I \Longrightarrow f\ i \in topspace\ (X\ i) \wedge B\ i = connected\_component\_of\_set\ (X\ i)\ (f\ i)$
by *metis*
then have $(\lambda i{\in}I.\ f\ i) \in ((\Pi_E\ i{\in}I.\ topspace\ (X\ i)) \cap extensional\ I)$
by *simp*
with *f* **show** *F* ∈ *?lhs*
unfolding *Feq connected_components_of_def connected_component_of_product_topology image_iff*
by (*smt* (*verit*, *del_insts*) *PiE_cong restrict_PiE_iff restrict_apply' restrict_extensional topspace_product_topology*)
qed
qed

5.6.4 Monotone maps (in the general topological sense)

definition *monotone_map*
where *monotone_map X Y f* ==
$f\ `\ (topspace\ X) \subseteq topspace\ Y \wedge$
$(\forall\, y \in topspace\ Y.\ connectedin\ X\ \{x \in topspace\ X.\ f\ x = y\})$

lemma *monotone_map*:
monotone_map X Y f ⟷
$f\ `\ (topspace\ X) \subseteq topspace\ Y \wedge (\forall\, y.\ connectedin\ X\ \{x \in topspace\ X.\ f\ x = y\})$
apply (*simp add*: *monotone_map_def*)
by (*metis* (*mono_tags*, *lifting*) *connectedin_empty* [*of X*] *Collect_empty_eq image_subset_iff*)

lemma *monotone_map_in_subtopology*:
monotone_map X (*subtopology Y S*) *f* ⟷ *monotone_map X Y f* ∧ *f* ‘ (*topspace X*) ⊆ *S*
by (*smt* (*verit*, *del_insts*) *le_inf_iff monotone_map topspace_subtopology*)

lemma *monotone_map_from_subtopology*:
assumes *monotone_map X Y f*
$\bigwedge x\ y.\ x \in topspace\ X \wedge y \in topspace\ X \wedge x \in S \wedge f\ x = f\ y \Longrightarrow y \in S$
shows *monotone_map* (*subtopology X S*) *Y f*
using *assms*
unfolding *monotone_map_def connectedin_subtopology*
by (*smt* (*verit*, *del_insts*) *Collect_cong Collect_empty_eq IntE IntI connectedin_empty image_subset_iff mem_Collect_eq subsetI topspace_subtopology*)

lemma *monotone_map_restriction*:
monotone_map X Y f ∧ $\{x \in topspace\ X.\ f\ x \in v\} = u$
$\Longrightarrow$ *monotone_map* (*subtopology X u*) (*subtopology Y v*) *f*
by (*smt* (*verit*, *best*) *IntI Int_Collect image_subset_iff mem_Collect_eq mono-*

tone_map monotone_map_from_subtopology topspace_subtopology)

lemma *injective_imp_monotone_map*:
assumes *f ‘ topspace X ⊆ topspace Y inj_on f* (*topspace X*)
shows *monotone_map X Y f*
unfolding *monotone_map_def*
proof (*intro conjI assms strip*)
fix *y*
assume *y ∈ topspace Y*
then have {*x ∈ topspace X*. *f x* = *y*} = {} ∨ (∃ *a ∈ topspace X*. {*x ∈ topspace X*. *f x* = *y*} = {*a*})
using *assms*(*2*) **unfolding** *inj_on_def* **by** *blast*
then show *connectedin X* {*x ∈ topspace X*. *f x* = *y*}
by (*metis* (*no_types*, *lifting*) *connectedin_empty connectedin_sing*)
qed

lemma *embedding_imp_monotone_map*:
embedding_map X Y f ⟹ *monotone_map X Y f*
by (*metis* (*no_types*) *embedding_map_def homeomorphic_eq_everything_map inf.absorb_iff2 injective_imp_monotone_map topspace_subtopology*)

lemma *section_imp_monotone_map*:
section_map X Y f ⟹ *monotone_map X Y f*
by (*simp add*: *embedding_imp_monotone_map section_imp_embedding_map*)

lemma *homeomorphic_imp_monotone_map*:
homeomorphic_map X Y f ⟹ *monotone_map X Y f*
by (*meson section_and_retraction_eq_homeomorphic_map section_imp_monotone_map*)

proposition *connected_space_monotone_quotient_map_preimage*:
assumes *f*: *monotone_map X Y f quotient_map X Y f* **and** *connected_space Y*
shows *connected_space X*
proof (*rule ccontr*)
assume ¬ *connected_space X*
then obtain *U V* **where** *openin X U openin X V U* ∩ *V* = {}
U ≠ {} *V* ≠ {} **and** *topUV*: *topspace X* ⊆ *U* ∪ *V*
by (*auto simp*: *connected_space_def*)
then have *UVsub*: *U* ⊆ *topspace X V* ⊆ *topspace X*
by (*auto simp*: *openin_subset*)
have ¬ *connected_space Y*
unfolding *connected_space_def not_not*
proof (*intro exI conjI*)
show *topspace Y* ⊆ *f‘U* ∪ *f‘V*
by (*metis f*(*2*) *image_Un quotient_imp_surjective_map subset_Un_eq topUV*)
show *f‘U* ≠ {}
by (*simp add*: ‹*U* ≠ {}›)
show (*f‘V*) ≠ {}
by (*simp add*: ‹*V* ≠ {}›)

```
    have ∗: y ∉ f ‘ V if y ∈ f ‘ U for y
    proof −
      have §: connectedin X {x ∈ topspace X. f x = y}
        using f(1) monotone_map by fastforce
      show ?thesis
        using connectedinD [OF § ‹openin X U› ‹openin X V›] UVsub topUV ‹U
∩ V = {}› that
        by (force simp: disjoint_iff)
    qed
    then show f‘U ∩ f‘V = {}
      by blast
    show openin Y (f‘U)
      using f ‹openin X U› topUV ∗ unfolding quotient_map_saturated_open
by force
    show openin Y (f‘V)
      using f ‹openin X V› topUV ∗ unfolding quotient_map_saturated_open
by force
  qed
  then show False
    by (simp add: assms)
qed

lemma connectedin_monotone_quotient_map_preimage:
  assumes monotone_map X Y f quotient_map X Y f connectedin Y C openin Y
C ∨ closedin Y C
  shows connectedin X {x ∈ topspace X. f x ∈ C}
proof −
  have connected_space (subtopology X {x ∈ topspace X. f x ∈ C})
  proof −
    have connected_space (subtopology Y C)
      using ‹connectedin Y C› connectedin_def by blast
    moreover have quotient_map (subtopology X {a ∈ topspace X. f a ∈ C})
(subtopology Y C) f
      by (simp add: assms quotient_map_restriction)
    ultimately show ?thesis
    using ‹monotone_map X Y f› connected_space_monotone_quotient_map_preimage
monotone_map_restriction by blast
  qed
  then show ?thesis
    by (simp add: connectedin_def)
qed

lemma monotone_open_map:
  assumes continuous_map X Y f open_map X Y f and fim: f ‘ (topspace X) =
topspace Y
  shows monotone_map X Y f ⟷ (∀ C. connectedin Y C ⟶ connectedin X {x
∈ topspace X. f x ∈ C})
        (is ?lhs=?rhs)
proof
```

assume *L*: *?lhs*
show *?rhs*
unfolding *connectedin_def*
proof (*intro strip conjI*)
fix *C*
assume *C*: $C \subseteq$ *topspace* $Y \wedge$ *connected_space* (*subtopology* Y C)
show *connected_space* (*subtopology* X $\{x \in$ *topspace* X. $f\,x \in C\}$)
proof (*rule connected_space_monotone_quotient_map_preimage*)
show *monotone_map* (*subtopology* X $\{x \in$ *topspace* X. $f\,x \in C\}$) (*subtopology* Y C) f
by (*simp add*: *L monotone_map_restriction*)
show *quotient_map* (*subtopology* X $\{x \in$ *topspace* X. $f\,x \in C\}$) (*subtopology* Y C) f
proof (*rule continuous_open_imp_quotient_map*)
show *continuous_map* (*subtopology* X $\{x \in$ *topspace* X. $f\,x \in C\}$) (*subtopology* Y C) f
using *assms continuous_map_from_subtopology continuous_map_in_subtopology* **by** *fastforce*
qed (*use open_map_restriction assms* **in** *fastforce*)+
qed (*simp add*: *C*)
qed *auto*
next
assume *?rhs*
then have $\forall\, y.$ *connectedin* Y $\{y\} \longrightarrow$ *connectedin* X $\{x \in$ *topspace* X. $f\,x = y\}$
by (*smt* (*verit*) *Collect_cong singletonD singletonI*)
then show *?lhs*
by (*simp add*: *fim monotone_map_def*)
qed

lemma *monotone_closed_map*:
assumes *continuous_map* X Y f *closed_map* X Y f **and** *fim*: f ‘ (*topspace* X) = *topspace* Y
shows *monotone_map* X Y $f \longleftrightarrow (\forall\, C.$ *connectedin* Y $C \longrightarrow$ *connectedin* X $\{x \in$ *topspace* X. $f\,x \in C\}$)
(**is** *?lhs=?rhs*)
proof
assume *L*: *?lhs*
show *?rhs*
unfolding *connectedin_def*
proof (*intro strip conjI*)
fix *C*
assume *C*: $C \subseteq$ *topspace* $Y \wedge$ *connected_space* (*subtopology* Y C)
show *connected_space* (*subtopology* X $\{x \in$ *topspace* X. $f\,x \in C\}$)
proof (*rule connected_space_monotone_quotient_map_preimage*)
show *monotone_map* (*subtopology* X $\{x \in$ *topspace* X. $f\,x \in C\}$) (*subtopology* Y C) f
by (*simp add*: *L monotone_map_restriction*)
show *quotient_map* (*subtopology* X $\{x \in$ *topspace* X. $f\,x \in C\}$) (*subtopology* Y C) f

proof (*rule continuous_closed_imp_quotient_map*)
show *continuous_map* (*subtopology X* {$x \in$ *topspace X*. *f x* $\in$ *C*}) (*subtopology Y C*) *f*
using *assms continuous_map_from_subtopology continuous_map_in_subtopology*
by *fastforce*
qed (*use closed_map_restriction assms* **in** *fastforce*)+
qed (*simp add*: *C*)
qed *auto*
next
assume *?rhs*
then have $\forall$ *y*. *connectedin Y* {*y*} $\longrightarrow$ *connectedin X* {$x \in$ *topspace X*. *f x* = *y*}
by (*smt* (*verit*) *Collect_cong singletonD singletonI*)
then show *?lhs*
by (*simp add*: *fim monotone_map_def*)
qed

5.6.5 Other countability properties

definition *second_countable*
where *second_countable X* $\equiv$
$\exists \mathcal{B}$. *countable* $\mathcal{B} \wedge (\forall V \in \mathcal{B}.$ *openin X V*$) \wedge$
$(\forall U\ x.$ *openin X U* $\wedge\ x \in U \longrightarrow (\exists V \in \mathcal{B}.\ x \in V \wedge V \subseteq U))$

definition *first_countable*
where *first_countable X* $\equiv$
$\forall x \in$ *topspace X*.
$\exists \mathcal{B}$. *countable* $\mathcal{B} \wedge (\forall V \in \mathcal{B}.$ *openin X V*$) \wedge$
$(\forall U.$ *openin X U* $\wedge\ x \in U \longrightarrow (\exists V \in \mathcal{B}.\ x \in V \wedge V \subseteq U))$

definition *separable_space*
where *separable_space X* $\equiv$
$\exists C.$ *countable C* $\wedge$ *C* $\subseteq$ *topspace X* $\wedge$ *X closure_of C* = *topspace X*

lemma *second_countable*:
second_countable X $\longleftrightarrow$
$(\exists \mathcal{B}.$ *countable* $\mathcal{B} \wedge$ *openin X* = *arbitrary union_of* $(\lambda x.\ x \in \mathcal{B}))$
by (*smt* (*verit*) *openin_topology_base_unique second_countable_def*)

lemma *second_countable_subtopology*:
assumes *second_countable X*
shows *second_countable* (*subtopology X S*)
proof −
obtain $\mathcal{B}$ **where** $\mathcal{B}$: *countable* $\mathcal{B}$ $\bigwedge V.\ V \in \mathcal{B} \Longrightarrow$ *openin X V*
$\bigwedge U\ x.$ *openin X U* $\wedge\ x \in U \longrightarrow (\exists V \in \mathcal{B}.\ x \in V \wedge V \subseteq U)$
using *assms* **by** (*auto simp*: *second_countable_def*)
show *?thesis*
unfolding *second_countable_def*
proof (*intro exI conjI*)
show $\forall V \in ((\cap) S)$ ‘ $\mathcal{B}$. *openin* (*subtopology X S*) *V*

```
    using openin_subtopology_Int2 ℬ by blast
  show ∀ U x. openin (subtopology X S) U ∧ x ∈ U ⟶ (∃ V∈((∩)S) ` ℬ. x ∈
V ∧ V ⊆ U)
    using ℬ unfolding openin_subtopology
    by (smt (verit, del_insts) IntI image_iff inf_commute inf_le1 subset_iff)
 qed (use ℬ in auto)
qed


lemma second_countable_discrete_topology:
  second_countable(discrete_topology U) ⟷ countable U (is ?lhs=?rhs)
proof
 assume L: ?lhs
 then
 obtain ℬ where ℬ: countable ℬ ⋀V. V ∈ ℬ ⟹ V ⊆ U
   ⋀W x. W ⊆ U ∧ x ∈ W ⟶ (∃ V ∈ ℬ. x ∈ V ∧ V ⊆ W)
   by (auto simp: second_countable_def)
 then have {x} ∈ ℬ if x ∈ U for x
   by (metis empty_subsetI insertCI insert_subset subset_antisym that)
 then show ?rhs
  by (smt (verit) countable_subset image_subsetI ‹countable ℬ› countable_image_inj_on
[OF _ inj_singleton])
next
 assume ?rhs
 then show ?lhs
   unfolding second_countable_def
   by (rule_tac x=(λx. {x}) ` U in exI) auto
qed

lemma second_countable_open_map_image:
 assumes continuous_map X Y f open_map X Y f
  and fim: f ` (topspace X) = topspace Y and second_countable X
 shows second_countable Y
proof −
 have openXYf: ⋀U. openin X U ⟶ openin Y (f ` U)
   using assms by (auto simp: open_map_def)
 obtain ℬ where ℬ: countable ℬ ⋀V. V ∈ ℬ ⟹ openin X V
   and ∗: ⋀U x. openin X U ∧ x ∈ U ⟶ (∃ V ∈ ℬ. x ∈ V ∧ V ⊆ U)
   using assms by (auto simp: second_countable_def)
 show ?thesis
   unfolding second_countable_def
 proof (intro exI conjI strip)
   fix V y
   assume V: openin Y V ∧ y ∈ V
   then obtain x where x ∈ topspace X and x: f x = y
     by (metis fim image_iff openin_subset subsetD)

   then obtain W where W∈ℬ x ∈ W W ⊆ {x ∈ topspace X. f x ∈ V}
   using ∗ [of {x ∈ topspace X. f x ∈ V} x] V assms openin_continuous_map_preimage
```

by *force*
then show $\exists W \in (\mathit{image}\ f)\ `\ \mathcal{B}.\ y \in W \wedge W \subseteq V$
using x **by** *auto*
qed (*use* $\mathcal{B}$ *openXYf* **in** *auto*)
qed

lemma *homeomorphic_space_second_countability*:
X *homeomorphic_space* $Y \Longrightarrow$ (*second_countable* $X \longleftrightarrow$ *second_countable* Y)
by (*meson homeomorphic_eq_everything_map homeomorphic_space homeomorphic_space_sym second_countable_open_map_image*)

lemma *second_countable_retraction_map_image*:
⟦*retraction_map* $X\ Y\ r$; *second_countable* X⟧ $\Longrightarrow$ *second_countable* Y
using *hereditary_imp_retractive_property homeomorphic_space_second_countability second_countable_subtopology* **by** *blast*

lemma *second_countable_imp_first_countable*:
second_countable $X \Longrightarrow$ *first_countable* X
by (*metis first_countable_def second_countable_def*)

lemma *first_countable_subtopology*:
assumes *first_countable* X
shows *first_countable* (*subtopology* $X\ S$)
unfolding *first_countable_def*
proof
fix x
assume $x \in$ *topspace* (*subtopology* $X\ S$)
then obtain $\mathcal{B}$ **where** *countable* $\mathcal{B}$ **and** $\mathcal{B}$: $\bigwedge V.\ V \in \mathcal{B} \Longrightarrow$ *openin* $X\ V$
$\bigwedge U.$ *openin* $X\ U \wedge x \in U \longrightarrow (\exists V \in \mathcal{B}.\ x \in V \wedge V \subseteq U)$
using *assms first_countable_def* **by** *force*
show $\exists \mathcal{B}.$ *countable* $\mathcal{B} \wedge (\forall V{\in}\mathcal{B}.$ *openin* (*subtopology* $X\ S$) $V) \wedge (\forall U.$ *openin* (*subtopology* $X\ S$) $U \wedge x \in U \longrightarrow (\exists V{\in}\mathcal{B}.\ x \in V \wedge V \subseteq U)$)
proof (*intro exI conjI strip*)
show *countable* $(((\cap)S)\ `\ \mathcal{B})$
using ‹*countable* $\mathcal{B}$› **by** *blast*
show *openin* (*subtopology* $X\ S$) V **if** $V \in ((\cap)S)\ `\ \mathcal{B}$ **for** V
using $\mathcal{B}$ *openin_subtopology_Int2 that* **by** *fastforce*
show $\exists V{\in}((\cap)S)\ `\ \mathcal{B}.\ x \in V \wedge V \subseteq U$
if *openin* (*subtopology* $X\ S$) $U \wedge x \in U$ **for** U
using *that* $\mathcal{B}(2)$ **by** (*clarsimp simp*: *openin_subtopology*) (*meson le_infI2*)
qed
qed

lemma *first_countable_discrete_topology*:
first_countable (*discrete_topology* U)
unfolding *first_countable_def topspace_discrete_topology openin_discrete_topology*
proof
fix x **assume** $x \in U$

show $\exists \mathcal{B}.$ *countable* $\mathcal{B} \wedge (\forall V{\in}\mathcal{B}.\ V \subseteq U) \wedge (\forall Ua.\ Ua \subseteq U \wedge x \in Ua \longrightarrow (\exists V{\in}\mathcal{B}.\ x \in V \wedge V \subseteq Ua))$
using ‹$x \in U$› **by** (*rule_tac* $x{=}\{\{x\}\}$ **in** *exI*) *auto*
qed

lemma *first_countable_open_map_image*:
assumes *continuous_map X Y f open_map X Y f*
and *fim*: f ‘ (*topspace X*) = *topspace Y* **and** *first_countable X*
shows *first_countable Y*
unfolding *first_countable_def*
proof
fix *y*
assume $y \in$ *topspace Y*
have *openXYf*: $\bigwedge U.$ *openin X U* $\longrightarrow$ *openin Y* (f ‘ U)
using *assms* **by** (*auto simp*: *open_map_def*)
then obtain x **where** x: $x \in$ *topspace X* $f\,x = y$
by (*metis* ‹$y \in$ *topspace Y*› *fim imageE*)
obtain $\mathcal{B}$ **where** $\mathcal{B}$: *countable* $\mathcal{B}$ $\bigwedge V.\ V \in \mathcal{B} \Longrightarrow$ *openin X V*
and $*$: $\bigwedge U.$ *openin X U* $\wedge\ x \in U \longrightarrow (\exists\, V \in \mathcal{B}.\ x \in V \wedge V \subseteq U)$
using *assms x first_countable_def* **by** *force*
show $\exists \mathcal{B}.$ *countable* $\mathcal{B} \wedge$
$(\forall V{\in}\mathcal{B}.$ *openin Y V*$) \wedge (\forall U.$ *openin Y U* $\wedge\ y \in U \longrightarrow (\exists V{\in}\mathcal{B}.\ y \in V \wedge V \subseteq U))$
proof (*intro exI conjI strip*)
fix *V* **assume** *openin Y V* $\wedge\ y \in V$
then have $\exists W{\in}\mathcal{B}.\ x \in W \wedge W \subseteq \{x \in$ *topspace X*$.\ f\,x \in V\}$
using $*$ [*of* $\{x \in$ *topspace X*$.\ f\,x \in V\}$] *assms openin_continuous_map_preimage x*
by *fastforce*
then show $\exists\, V' \in$ (*image f*) ‘ $\mathcal{B}.\ y \in V' \wedge V' \subseteq V$
using *image_mono x* **by** *auto*
qed (*use* $\mathcal{B}$ *openXYf* **in** *force*)+
qed

lemma *homeomorphic_space_first_countability*:
X homeomorphic_space Y $\Longrightarrow$ *first_countable X* $\longleftrightarrow$ *first_countable Y*
by (*meson first_countable_open_map_image homeomorphic_eq_everything_map homeomorphic_space homeomorphic_space_sym*)

lemma *first_countable_retraction_map_image*:
⟦*retraction_map X Y r*; *first_countable X*⟧ $\Longrightarrow$ *first_countable Y*
using *first_countable_subtopology hereditary_imp_retractive_property homeomorphic_space_first_countability* **by** *blast*

lemma *separable_space_open_subset*:
assumes *separable_space X openin X S*
shows *separable_space* (*subtopology X S*)
proof −
obtain *C* **where** *C*: *countable C C* $\subseteq$ *topspace X X closure_of C* = *topspace X*

by (*meson assms separable_space_def*)
then have $\bigwedge x\ T.\ [\![x \in \mathit{topspace}\ X;\ x \in T;\ \mathit{openin}\ (\mathit{subtopology}\ X\ S)\ T]\!] \Longrightarrow \exists y.\ y \in S \wedge y \in C \wedge y \in T$
by (*smt* (*verit*) ‹*openin X S*› *in_closure_of openin_open_subtopology subsetD*)
with *C* ‹*openin X S*› **show** *?thesis*
unfolding *separable_space_def*
by (*rule_tac x*=$S \cap C$ **in** *exI*) (*force simp*: *in_closure_of*)
qed

lemma *separable_space_continuous_map_image*:
assumes *separable_space X continuous_map X Y f*
and *fim*: *f* ‘ (*topspace X*) = *topspace Y*
shows *separable_space Y*
proof −
have *cont*: $\bigwedge S.\ f$ ‘ (*X closure_of S*) ⊆ *Y closure_of f* ‘ *S*
by (*simp add*: *assms continuous_map_image_closure_subset*)
obtain *C* **where** *C*: *countable C C* ⊆ *topspace X X closure_of C* = *topspace X*
by (*meson assms separable_space_def*)
then show *?thesis*
unfolding *separable_space_def*
by (*metis cont fim closure_of_subset_topspace countable_image image_mono subset_antisym*)
qed

lemma *separable_space_quotient_map_image*:
[*quotient_map X Y q*; *separable_space X*] ⟹ *separable_space Y*
by (*meson quotient_imp_continuous_map quotient_imp_surjective_map separable_space_continuous_map_image*)

lemma *separable_space_retraction_map_image*:
[*retraction_map X Y r*; *separable_space X*] ⟹ *separable_space Y*
using *retraction_imp_quotient_map separable_space_quotient_map_image* **by** *blast*

lemma *homeomorphic_separable_space*:
X homeomorphic_space Y ⟹ (*separable_space X* ⟷ *separable_space Y*)
by (*meson homeomorphic_eq_everything_map homeomorphic_maps_map homeomorphic_space_def separable_space_continuous_map_image*)

lemma *separable_space_discrete_topology*:
separable_space(*discrete_topology U*) ⟷ *countable U*
by (*metis countable_Int2 discrete_topology_closure_of dual_order.refl inf.orderE separable_space_def topspace_discrete_topology*)

lemma *second_countable_imp_separable_space*:
assumes *second_countable X*
shows *separable_space X*
proof −
obtain $\mathcal{B}$ **where** $\mathcal{B}$: *countable* $\mathcal{B}$ $\bigwedge V.\ V \in \mathcal{B} \Longrightarrow \mathit{openin}\ X\ V$

and $*$: $\bigwedge U\ x.\ openin\ X\ U \wedge x \in U \longrightarrow (\exists\, V \in \mathcal{B}.\ x \in V \wedge V \subseteq U)$
using *assms* **by** (*auto simp*: *second_countable_def*)
obtain *c* **where** *c*: $\bigwedge V.\ [\![V \in \mathcal{B};\ V \neq \{\}]\!] \Longrightarrow c\ V \in V$
by (*metis all_not_in_conv*)
then have $**$: $\bigwedge x.\ x \in topspace\ X \Longrightarrow x \in X\ closure\_of\ c\ `\ (\mathcal{B} - \{\{\}\})$
using $*$ **by** (*force simp*: *closure_of_def*)
show *?thesis*
unfolding *separable_space_def*
proof (*intro exI conjI*)
show *countable* (*c* ` ($\mathcal{B}$−{{}}))
using $\mathcal{B}(1)$ **by** *blast*
show (*c* ` ($\mathcal{B}$−{{}})) $\subseteq$ *topspace X*
using $\mathcal{B}(2)$ *c openin_subset* **by** *fastforce*
show *X closure_of* (*c* ` ($\mathcal{B}$−{{}})) = *topspace X*
by (*meson* $**$ *closure_of_subset_topspace subsetI subset_antisym*)
qed
qed

lemma *second_countable_imp_Lindelof_space*:
assumes *second_countable X*
shows *Lindelof_space X*
unfolding *Lindelof_space_def*
proof *clarify*
fix $\mathcal{U}$
assume $\forall\, U \in \mathcal{U}.\ openin\ X\ U$ **and** *UU*: $\bigcup\mathcal{U} = topspace\ X$
obtain $\mathcal{B}$ **where** $\mathcal{B}$: *countable* $\mathcal{B}$ $\bigwedge V.\ V \in \mathcal{B} \Longrightarrow openin\ X\ V$
and $*$: $\bigwedge U\ x.\ openin\ X\ U \wedge x \in U \longrightarrow (\exists\, V \in \mathcal{B}.\ x \in V \wedge V \subseteq U)$
using *assms* **by** (*auto simp*: *second_countable_def*)
define $\mathcal{B}'$ **where** $\mathcal{B}' = \{B \in \mathcal{B}.\ \exists\, U.\ U \in \mathcal{U} \wedge B \subseteq U\}$
have $\mathcal{B}'$: *countable* $\mathcal{B}'$ $\bigcup\mathcal{B}' = \bigcup\mathcal{U}$
using $\mathcal{B}$ **using** $*$ ‹$\forall\, U{\in}\mathcal{U}.\ openin\ X\ U$› **by** (*fastforce simp*: $\mathcal{B}'$*_def*)+
have $\bigwedge b.\ \exists\, U.\ b \in \mathcal{B}' \longrightarrow U \in \mathcal{U} \wedge b \subseteq U$
by (*simp add*: $\mathcal{B}'$*_def*)
then obtain *G* **where** *G*: $\bigwedge b.\ b \in \mathcal{B}' \longrightarrow G\ b \in \mathcal{U} \wedge b \subseteq G\ b$
by *metis*
with $\mathcal{B}'$ *UU* **show** $\exists \mathcal{V}.\ countable\ \mathcal{V} \wedge \mathcal{V} \subseteq \mathcal{U} \wedge \bigcup\mathcal{V} = topspace\ X$
by (*rule_tac x*=*G* ` $\mathcal{B}'$ **in** *exI*) *fastforce*
qed

5.6.6 Neigbourhood bases EXTRAS

Neigbourhood bases: useful for "local" properties of various kinds

lemma *openin_topology_neighbourhood_base_unique*:
openin X = *arbitrary union_of P* ⟷
($\forall\, u.\ P\ u \longrightarrow openin\ X\ u$) $\wedge$ *neighbourhood_base_of P X*
by (*smt* (*verit*, *best*) *open_neighbourhood_base_of openin_topology_base_unique*)

lemma *neighbourhood_base_at_topology_base*:
openin X = *arbitrary union_of b*

$\Longrightarrow$ (*neighbourhood_base_at x P X* $\longleftrightarrow$
(∀ *w*. *b w* ∧ *x* ∈ *w* $\longrightarrow$ (∃ *u v*. *openin X u* ∧ *P v* ∧ *x* ∈ *u* ∧ *u* ⊆ *v* ∧ *v* ⊆ *w*)))
apply (*simp add*: *neighbourhood_base_at_def*)
by (*smt* (*verit*, *del_insts*) *openin_topology_base_unique subset_trans*)

lemma *neighbourhood_base_of_unlocalized*:
assumes ⋀*S t*. *P S* ∧ *openin X t* ∧ (*t* ≠ {}) ∧ *t* ⊆ *S* $\Longrightarrow$ *P t*
shows *neighbourhood_base_of P X* $\longleftrightarrow$
(∀ *x* ∈ *topspace X*. ∃ *u v*. *openin X u* ∧ *P v* ∧ *x* ∈ *u* ∧ *u* ⊆ *v* ∧ *v* ⊆ *topspace X*)
apply (*simp add*: *neighbourhood_base_of_def*)
by (*smt* (*verit*, *ccfv_SIG*) *assms empty_iff neighbourhood_base_at_unlocalized*)

lemma *neighbourhood_base_at_discrete_topology*:
neighbourhood_base_at x P (*discrete_topology u*) $\longleftrightarrow$ *x* ∈ *u* $\Longrightarrow$ *P* {*x*}
apply (*simp add*: *neighbourhood_base_at_def*)
by (*smt* (*verit*) *empty_iff empty_subsetI insert_subset singletonI subsetD subset_singletonD*)

lemma *neighbourhood_base_of_discrete_topology*:
neighbourhood_base_of P (*discrete_topology u*) $\longleftrightarrow$ (∀ *x* ∈ *u*. *P* {*x*})
apply (*simp add*: *neighbourhood_base_of_def*)
using *neighbourhood_base_at_discrete_topology*[*of* _ *P u*]
by (*metis empty_subsetI insert_subset neighbourhood_base_at_def openin_discrete_topology singletonI*)

lemma *second_countable_neighbourhood_base_alt*:
second_countable X $\longleftrightarrow$
(∃ $\mathcal{B}$. *countable* $\mathcal{B}$ ∧ (∀ *V* ∈ $\mathcal{B}$. *openin X V*) ∧ *neighbourhood_base_of* (λ*A*. *A*∈$\mathcal{B}$) *X*)
by (*metis* (*full_types*) *openin_topology_neighbourhood_base_unique second_countable*)

lemma *first_countable_neighbourhood_base_alt*:
first_countable X $\longleftrightarrow$
(∀ *x* ∈ *topspace X*. ∃ $\mathcal{B}$. *countable* $\mathcal{B}$ ∧ (∀ *V* ∈ $\mathcal{B}$. *openin X V*) ∧ *neighbourhood_base_at x* (λ*V*. *V* ∈ $\mathcal{B}$) *X*)
unfolding *first_countable_def*
apply (*intro ball_cong refl ex_cong conj_cong*)
by (*metis* (*mono_tags*, *lifting*) *open_neighbourhood_base_at*)

lemma *second_countable_neighbourhood_base*:
second_countable X $\longleftrightarrow$
(∃ $\mathcal{B}$. *countable* $\mathcal{B}$ ∧ *neighbourhood_base_of* (λ*V*. *V* ∈ $\mathcal{B}$) *X*) (**is** *?lhs*=*?rhs*)
proof
assume *?lhs*
then show *?rhs*
using *second_countable_neighbourhood_base_alt* **by** *blast*
next

assume *?rhs*
then obtain $\mathcal{B}$ **where** *countable* $\mathcal{B}$
and $\mathcal{B}$: $\bigwedge W\ x.$ *openin* $X\ W \wedge x \in W \longrightarrow (\exists U.$ *openin* $X\ U \wedge (\exists V.\ V \in \mathcal{B} \wedge x \in U \wedge U \subseteq V \wedge V \subseteq W))$
by (*metis neighbourhood_base_of*)
then show *?lhs*
unfolding *second_countable_neighbourhood_base_alt neighbourhood_base_of*
apply (*rule_tac x*=$(\lambda u.\ X$ *interior_of* $u)$ ‘ $\mathcal{B}$ **in** *exI*)
by (*smt* (*verit*, *best*) *interior_of_eq interior_of_mono countable_image image_iff openin_interior_of*)
qed

lemma *first_countable_neighbourhood_base*:
first_countable $X \longleftrightarrow$
$(\forall x \in$ *topspace* $X.\ \exists \mathcal{B}.$ *countable* $\mathcal{B} \wedge$ *neighbourhood_base_at* $x\ (\lambda V.\ V \in \mathcal{B})\ X)$ (**is** *?lhs*=*?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (*metis first_countable_neighbourhood_base_alt*)
next
assume *R*: *?rhs*
show *?lhs*
unfolding *first_countable_neighbourhood_base_alt*
proof
fix x
assume $x \in$ *topspace* X
with *R* **obtain** $\mathcal{B}$ **where** *countable* $\mathcal{B}$ **and** $\mathcal{B}$: *neighbourhood_base_at* $x\ (\lambda V.\ V \in \mathcal{B})\ X$
by *blast*
then
show $\exists \mathcal{B}.$ *countable* $\mathcal{B} \wedge$ *Ball* $\mathcal{B}$ (*openin* X) $\wedge$ *neighbourhood_base_at* $x\ (\lambda V.\ V \in \mathcal{B})\ X$
unfolding *neighbourhood_base_at_def*
apply (*rule_tac x*=$(\lambda u.\ X$ *interior_of* $u)$ ‘ $\mathcal{B}$ **in** *exI*)
by (*smt* (*verit*, *best*) *countable_image image_iff interior_of_eq interior_of_mono openin_interior_of*)
qed
qed

5.6.7 T_0 spaces and the Kolmogorov quotient

definition *t0_space* **where**
t0_space $X \equiv$
$\forall x \in$ *topspace* $X.\ \forall y \in$ *topspace* $X.\ x \neq y \longrightarrow (\exists U.$ *openin* $X\ U \wedge (x \notin U \longleftrightarrow y \in U))$

lemma *t0_space_expansive*:
$[\![$*topspace* $Y =$ *topspace* X; $\bigwedge U.$ *openin* $X\ U \Longrightarrow$ *openin* $Y\ U]\!] \Longrightarrow$ *t0_space* X

$\Longrightarrow$ *t0_space Y*
by (*metis t0_space_def*)

lemma *t1_imp_t0_space*: *t1_space X* $\Longrightarrow$ *t0_space X*
by (*metis t0_space_def t1_space_def*)

lemma *t1_eq_symmetric_t0_space_alt*:
t1_space X $\longleftrightarrow$
t0_space X $\wedge$
($\forall x \in$ *topspace X*. $\forall y \in$ *topspace X*. *x* $\in$ *X closure_of* $\{y\} \longleftrightarrow y \in X$ *closure_of* $\{x\}$)
apply (*simp add*: *t0_space_def t1_space_def closure_of_def*)
by (*smt* (*verit*, *best*) *openin_topspace*)

lemma *t1_eq_symmetric_t0_space*:
t1_space X $\longleftrightarrow$ *t0_space X* $\wedge$ ($\forall x\ y$. *x* $\in$ *X closure_of* $\{y\} \longleftrightarrow y \in X$ *closure_of* $\{x\}$)
by (*auto simp*: *t1_eq_symmetric_t0_space_alt in_closure_of*)

lemma *Hausdorff_imp_t0_space*:
Hausdorff_space X $\Longrightarrow$ *t0_space X*
by (*simp add*: *Hausdorff_imp_t1_space t1_imp_t0_space*)

lemma *t0_space*:
t0_space X $\longleftrightarrow$
($\forall x \in$ *topspace X*. $\forall y \in$ *topspace X*. *x* $\neq$ *y* $\longrightarrow$ ($\exists C$. *closedin X C* $\wedge$ ($x \notin C \longleftrightarrow y \in C$)))
unfolding *t0_space_def* **by** (*metis Diff_iff closedin_def openin_closedin_eq*)

lemma *homeomorphic_t0_space*:
assumes *X homeomorphic_space Y*
shows *t0_space X* $\longleftrightarrow$ *t0_space Y*
proof −
obtain *f* **where** *f*: *homeomorphic_map X Y f* **and** *F*: *inj_on f* (*topspace X*) **and** *topspace Y* = *f* ‘ *topspace X*
by (*metis assms homeomorphic_imp_injective_map homeomorphic_imp_surjective_map homeomorphic_space*)
with *inj_on_image_mem_iff* [*OF F*]
show *?thesis*
apply (*simp add*: *t0_space_def homeomorphic_eq_everything_map continuous_map_def open_map_def inj_on_def*)
by (*smt* (*verit*) *mem_Collect_eq openin_subset*)
qed

lemma *t0_space_closure_of_sing*:
t0_space X $\longleftrightarrow$
($\forall x \in$ *topspace X*. $\forall y \in$ *topspace X*. *X closure_of* $\{x\}$ = *X closure_of* $\{y\}$ $\longrightarrow x = y$)
by (*simp add*: *t0_space_def closure_of_def set_eq_iff*) (*smt* (*verit*))

lemma *t0_space_discrete_topology*: *t0_space* (*discrete_topology S*)
by (*simp add*: *Hausdorff_imp_t0_space*)

lemma *t0_space_subtopology*: *t0_space X* ⟹ *t0_space* (*subtopology X U*)
by (*simp add*: *t0_space_def openin_subtopology*) (*metis Int_iff*)

lemma *t0_space_retraction_map_image*:
⟦*retraction_map X Y r*; *t0_space X*⟧ ⟹ *t0_space Y*
using *hereditary_imp_retractive_property homeomorphic_t0_space t0_space_subtopology*
by *blast*

lemma *XY*: {*x*}×{*y*} = {(*x*,*y*)}
by *simp*

lemma *t0_space_prod_topologyI*: ⟦*t0_space X*; *t0_space Y*⟧ ⟹ *t0_space* (*prod_topology X Y*)
by (*simp add*: *t0_space_closure_of_sing closure_of_Times closure_of_eq_empty_gen times_eq_iff flip*: *XY insert_Times_insert*)

lemma *t0_space_prod_topology_iff*:
t0_space (*prod_topology X Y*) ⟷ *prod_topology X Y* = *trivial_topology* ∨ *t0_space X* ∧ *t0_space Y* (**is** *?lhs*=*?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (*metis prod_topology_trivial_iff retraction_map_fst retraction_map_snd t0_space_retraction_map_image*)
qed (*metis t0_space_discrete_topology t0_space_prod_topologyI*)

proposition *t0_space_product_topology*:
t0_space (*product_topology X I*) ⟷ *product_topology X I* = *trivial_topology* ∨ (∀ *i* ∈ *I*. *t0_space* (*X i*))
(**is** *?lhs*=*?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (*meson retraction_map_product_projection t0_space_retraction_map_image*)
next
assume *R*: *?rhs*
show *?lhs*
proof (*cases product_topology X I* = *trivial_topology*)
case *True*
then show *?thesis*
by (*simp add*: *t0_space_def*)
next
case *False*
show *?thesis*

unfolding *t0_space*
proof (*intro strip*)
fix *x y*
assume *x*: $x \in$ *topspace* (*product_topology X I*)
and *y*: $y \in$ *topspace* (*product_topology X I*)
and $x \neq y$
then obtain *i* **where** $i \in I\ x\ i \neq y\ i$
by (*metis PiE_ext topspace_product_topology*)
then have *t0_space* (*X i*)
using *False R* **by** *blast*
then obtain *U* **where** *closedin* (*X i*) *U* ($x\ i \notin U \longleftrightarrow y\ i \in U$)
by (*metis t0_space PiE_mem* ‹$i \in I$› ‹$x\ i \neq y\ i$› *topspace_product_topology x y*)
with ‹$i \in I$› *x y* **show** $\exists U.$ *closedin* (*product_topology X I*) *U* $\wedge$ ($x \notin U$) = ($y \in U$)
by (*rule_tac x*=*PiE I* ($\lambda j.$ *if* $j = i$ *then U else topspace*(*X j*)) **in** *exI*)
(*simp add*: *closedin_product_topology PiE_iff*)
qed
qed
qed

5.6.8 Kolmogorov quotients

definition *Kolmogorov_quotient*
where *Kolmogorov_quotient X* $\equiv \lambda x.$ @$y.$ $\forall U.$ *openin X U* $\longrightarrow$ ($y \in U \longleftrightarrow x \in U$)

lemma *Kolmogorov_quotient_in_open*:
openin X U $\Longrightarrow$ (*Kolmogorov_quotient X x* $\in U \longleftrightarrow x \in U$)
by (*smt* (*verit*, *ccfv_SIG*) *Kolmogorov_quotient_def someI_ex*)

lemma *Kolmogorov_quotient_in_topspace*:
Kolmogorov_quotient X x $\in$ *topspace X* $\longleftrightarrow x \in$ *topspace X*
by (*simp add*: *Kolmogorov_quotient_in_open*)

lemma *Kolmogorov_quotient_in_closed*:
closedin X C $\Longrightarrow$ (*Kolmogorov_quotient X x* $\in C \longleftrightarrow x \in C$)
unfolding *closedin_def*
by (*meson DiffD2 DiffI Kolmogorov_quotient_in_open Kolmogorov_quotient_in_topspace in_mono*)

lemma *continuous_map_Kolmogorov_quotient*:
continuous_map X X (*Kolmogorov_quotient X*)
using *Kolmogorov_quotient_in_open openin_subopen openin_subset*
by (*fastforce simp*: *continuous_map_def Kolmogorov_quotient_in_topspace*)

lemma *open_map_Kolmogorov_quotient_explicit*:
openin X U $\Longrightarrow$ *Kolmogorov_quotient X* ‘ *U* = *Kolmogorov_quotient X* ‘ *topspace X* $\cap$ *U*

using *Kolmogorov_quotient_in_open openin_subset* **by** *fastforce*

lemma *open_map_Kolmogorov_quotient_gen*:
open_map (*subtopology X S*) (*subtopology X* (*Kolmogorov_quotient X ‘ S*))
(*Kolmogorov_quotient X*)
proof (*clarsimp simp*: *open_map_def openin_subtopology_alt image_iff*)
fix *U*
assume *openin X U*
then have *Kolmogorov_quotient X ‘* (*S* ∩ *U*) = *Kolmogorov_quotient X ‘ S* ∩
U
using *Kolmogorov_quotient_in_open* [*of X U*] **by** *auto*
then show ∃ *V*. *openin X V* ∧ *Kolmogorov_quotient X ‘* (*S* ∩ *U*) = *Kol-*
mogorov_quotient X ‘ S ∩ *V*
using ‹*openin X U*› **by** *blast*
qed

lemma *open_map_Kolmogorov_quotient*:
open_map X (*subtopology X* (*Kolmogorov_quotient X ‘ topspace X*))
(*Kolmogorov_quotient X*)
by (*metis open_map_Kolmogorov_quotient_gen subtopology_topspace*)

lemma *closed_map_Kolmogorov_quotient_explicit*:
closedin X U ⟹ *Kolmogorov_quotient X ‘ U* = *Kolmogorov_quotient X ‘*
topspace X ∩ *U*
using *closedin_subset* **by** (*fastforce simp*: *Kolmogorov_quotient_in_closed*)

lemma *closed_map_Kolmogorov_quotient_gen*:
closed_map (*subtopology X S*) (*subtopology X* (*Kolmogorov_quotient X ‘ S*))
(*Kolmogorov_quotient X*)
using *Kolmogorov_quotient_in_closed* **by** (*force simp*: *closed_map_def closedin_subtopology_alt*
image_iff)

lemma *closed_map_Kolmogorov_quotient*:
closed_map X (*subtopology X* (*Kolmogorov_quotient X ‘ topspace X*))
(*Kolmogorov_quotient X*)
by (*metis closed_map_Kolmogorov_quotient_gen subtopology_topspace*)

lemma *quotient_map_Kolmogorov_quotient_gen*:
quotient_map (*subtopology X S*) (*subtopology X* (*Kolmogorov_quotient X ‘ S*))
(*Kolmogorov_quotient X*)
proof (*intro continuous_open_imp_quotient_map*)
show *continuous_map* (*subtopology X S*) (*subtopology X* (*Kolmogorov_quotient*
X ‘ S)) (*Kolmogorov_quotient X*)
by (*simp add*: *continuous_map_Kolmogorov_quotient continuous_map_from_subtopology*
continuous_map_in_subtopology image_mono)
show *open_map* (*subtopology X S*) (*subtopology X* (*Kolmogorov_quotient X ‘ S*))
(*Kolmogorov_quotient X*)
using *open_map_Kolmogorov_quotient_gen* **by** *blast*

show *Kolmogorov_quotient X ‘ topspace (subtopology X S) = topspace (subtopology X (Kolmogorov_quotient X ‘ S))*
by (*force simp*: *Kolmogorov_quotient_in_open*)
qed

lemma *quotient_map_Kolmogorov_quotient*:
quotient_map X (subtopology X (Kolmogorov_quotient X ‘ topspace X)) (Kolmogorov_quotient X)
by (*metis quotient_map_Kolmogorov_quotient_gen subtopology_topspace*)

lemma *Kolmogorov_quotient_eq*:
Kolmogorov_quotient X x = Kolmogorov_quotient X y ⟷
(∀ U. openin X U ⟶ (x ∈ U ⟷ y ∈ U)) (**is** *?lhs=?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*metis Kolmogorov_quotient_in_open*)
next
assume *?rhs* **then show** *?lhs*
by (*simp add*: *Kolmogorov_quotient_def*)
qed

lemma *Kolmogorov_quotient_eq_alt*:
Kolmogorov_quotient X x = Kolmogorov_quotient X y ⟷
(∀ U. closedin X U ⟶ (x ∈ U ⟷ y ∈ U)) (**is** *?lhs=?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*metis Kolmogorov_quotient_in_closed*)
next
assume *?rhs* **then show** *?lhs*
by (*smt* (*verit*) *Diff_iff Kolmogorov_quotient_eq closedin_topspace in_mono openin_closedin_eq*)
qed

lemma *Kolmogorov_quotient_continuous_map*:
assumes *continuous_map X Y f t0_space Y* **and** *x*: *x ∈ topspace X*
shows *f (Kolmogorov_quotient X x) = f x*
using *assms* **unfolding** *continuous_map_def t0_space_def*
by (*smt* (*verit*, *ccfv_threshold*) *Kolmogorov_quotient_in_open Kolmogorov_quotient_in_topspace PiE mem_Collect_eq*)

lemma *t0_space_Kolmogorov_quotient*:
t0_space (subtopology X (Kolmogorov_quotient X ‘ topspace X))
apply (*clarsimp simp*: *t0_space_def*)
by (*smt* (*verit*, *best*) *Kolmogorov_quotient_eq imageE image_eqI open_map_Kolmogorov_quotient open_map_def*)

lemma *Kolmogorov_quotient_id*:
t0_space X ⟹ x ∈ topspace X ⟹ Kolmogorov_quotient X x = x
by (*metis Kolmogorov_quotient_in_open Kolmogorov_quotient_in_topspace t0_space_def*)

lemma *Kolmogorov_quotient_idemp*:
Kolmogorov_quotient X (Kolmogorov_quotient X x) = Kolmogorov_quotient X x
by (*simp add*: *Kolmogorov_quotient_eq Kolmogorov_quotient_in_open*)

lemma *retraction_maps_Kolmogorov_quotient*:
retraction_maps X
(*subtopology X (Kolmogorov_quotient X ‘ topspace X*))
(*Kolmogorov_quotient X*) *id*
unfolding *retraction_maps_def continuous_map_in_subtopology*
using *Kolmogorov_quotient_idemp continuous_map_Kolmogorov_quotient* **by** *force*

lemma *retraction_map_Kolmogorov_quotient*:
retraction_map X
(*subtopology X (Kolmogorov_quotient X ‘ topspace X*))
(*Kolmogorov_quotient X*)
using *retraction_map_def retraction_maps_Kolmogorov_quotient* **by** *blast*

lemma *retract_of_space_Kolmogorov_quotient_image*:
Kolmogorov_quotient X ‘ topspace X retract_of_space X
proof −
have *continuous_map X X* (*Kolmogorov_quotient X*)
by (*simp add*: *continuous_map_Kolmogorov_quotient*)
then have *Kolmogorov_quotient X ‘ topspace X* ⊆ *topspace X*
by (*simp add*: *continuous_map_image_subset_topspace*)
then show *?thesis*
by (*meson retract_of_space_retraction_maps retraction_maps_Kolmogorov_quotient*)
qed

lemma *Kolmogorov_quotient_lift_exists*:
assumes *S* ⊆ *topspace X t0_space Y* **and** *f*: *continuous_map* (*subtopology X S*) *Y f*
obtains *g* **where** *continuous_map* (*subtopology X* (*Kolmogorov_quotient X ‘ S*)) *Y g*
⋀*x*. *x* ∈ *S* ⟹ *g*(*Kolmogorov_quotient X x*) = *f x*
proof −
have ⋀*x y*. ⟦*x* ∈ *S*; *y* ∈ *S*; *Kolmogorov_quotient X x* = *Kolmogorov_quotient X y*⟧ ⟹ *f x* = *f y*
using *assms*
apply (*simp add*: *Kolmogorov_quotient_eq t0_space_def continuous_map_def Int_absorb1 openin_subtopology*)
by (*smt* (*verit*, *del_insts*) *Int_iff mem_Collect_eq Pi_iff*)
then obtain *g* **where** *g*: *continuous_map* (*subtopology X* (*Kolmogorov_quotient X ‘ S*)) *Y g*
g ‘ (*topspace X* ∩ *Kolmogorov_quotient X ‘ S*) = *f ‘ S*
⋀*x*. *x* ∈ *S* ⟹ *g* (*Kolmogorov_quotient X x*) = *f x*
using *quotient_map_lift_exists* [*OF quotient_map_Kolmogorov_quotient_gen*

[*of X S*] *f*]
by (*metis assms*(*1*) *topspace_subtopology topspace_subtopology_subset*)
show *?thesis*
proof qed (*use g* **in** *auto*)
qed

5.6.9 Closed diagonals and graphs

lemma *Hausdorff_space_closedin_diagonal*:
Hausdorff_space X ⟷ *closedin* (*prod_topology X X*) ((λ*x*. (*x*,*x*)) ‘ *topspace X*)
proof −
have §: ((λ*x*. (*x*, *x*)) ‘ *topspace X*) ⊆ *topspace X* × *topspace X*
by *auto*
show *?thesis*
apply (*simp add*: *closedin_def openin_prod_topology_alt Hausdorff_space_def disjnt_iff* §)
apply (*intro all_cong1 imp_cong ex_cong1 conj_cong refl*)
by (*force dest*!: *openin_subset*)+
qed

lemma *closed_map_diag_eq*:
closed_map X (*prod_topology X X*) (λ*x*. (*x*,*x*)) ⟷ *Hausdorff_space X*
proof −
have *section_map X* (*prod_topology X X*) (λ*x*. (*x*, *x*))
unfolding *section_map_def retraction_maps_def*
by (*smt* (*verit*) *continuous_map_fst continuous_map_of_fst continuous_map_on_empty continuous_map_pairwise fst_conv fst_diag_fst snd_diag_fst*)
then have *embedding_map X* (*prod_topology X X*) (λ*x*. (*x*, *x*))
by (*rule section_imp_embedding_map*)
then show *?thesis*
using *Hausdorff_space_closedin_diagonal embedding_imp_closed_map_eq* **by** *blast*
qed

lemma *proper_map_diag_eq* [*simp*]:
proper_map X (*prod_topology X X*) (λ*x*. (*x*,*x*)) ⟷ *Hausdorff_space X*
by (*simp add*: *closed_map_diag_eq inj_on_convol_ident injective_imp_proper_eq_closed_map*)

lemma *closedin_continuous_maps_eq*:
assumes *Hausdorff_space Y* **and** *f*: *continuous_map X Y f* **and** *g*: *continuous_map X Y g*
shows *closedin X* {*x* ∈ *topspace X*. *f x* = *g x*}
proof −
have §:{*x* ∈ *topspace X*. *f x* = *g x*} = {*x* ∈ *topspace X*. (*f x*,*g x*) ∈ ((λ*y*.(*y*,*y*)) ‘ *topspace Y*)}
using *f continuous_map_image_subset_topspace* **by** *fastforce*
show *?thesis*
unfolding §
proof (*intro closedin_continuous_map_preimage*)

```
    show continuous_map X (prod_topology Y Y) (λx. (f x, g x))
      by (simp add: continuous_map_pairedI f g)
    show closedin (prod_topology Y Y) ((λy. (y, y)) ' topspace Y)
      using Hausdorff_space_closedin_diagonal assms by blast
  qed
qed

lemma forall_in_closure_of:
  assumes x ∈ X closure_of S ⋀x. x ∈ S ⟹ P x
    and closedin X {x ∈ topspace X. P x}
  shows P x
 by (smt (verit, ccfv_threshold) Diff_iff assms closedin_def in_closure_of mem_Collect_eq)

lemma forall_in_closure_of_eq:
  assumes x: x ∈ X closure_of S
    and Y: Hausdorff_space Y
    and f: continuous_map X Y f and g: continuous_map X Y g
    and fg: ⋀x. x ∈ S ⟹ f x = g x
  shows f x = g x
proof −
  have closedin X {x ∈ topspace X. f x = g x}
    using Y closedin_continuous_maps_eq f g by blast
  then show ?thesis
    using forall_in_closure_of [OF x fg]
    by fastforce
qed

lemma retract_of_space_imp_closedin:
  assumes Hausdorff_space X and S: S retract_of_space X
  shows closedin X S
proof −
  obtain r where r: continuous_map X (subtopology X S) r ∀ x∈S. r x = x
    using assms by (meson retract_of_space_def)
  then have §: S = {x ∈ topspace X. r x = x}
     using S retract_of_space_imp_subset by (force simp: continuous_map_def
Pi_iff)
  show ?thesis
    unfolding §
    using r continuous_map_into_fulltopology assms
    by (force intro: closedin_continuous_maps_eq)
qed

lemma homeomorphic_maps_graph:
    homeomorphic_maps X (subtopology (prod_topology X Y) ((λx. (x, f x)) '
(topspace X)))
        (λx. (x, f x)) fst ⟷ continuous_map X Y f
   (is ?lhs=?rhs)
proof
  assume ?lhs
```

then
have *h*: *homeomorphic_map X* (*subtopology* (*prod_topology X Y*) ((λ*x*. (*x*, *f x*)) ‘ *topspace X*)) (λ*x*. (*x*, *f x*))
by (*auto simp*: *homeomorphic_maps_map*)
have *f* = *snd* ∘ (λ*x*. (*x*, *f x*))
by *force*
then show *?rhs*
by (*metis* (*no_types*, *lifting*) *h continuous_map_in_subtopology continuous_map_snd_of homeomorphic_eq_everything_map*)
next
assume *?rhs*
then show *?lhs*
unfolding *homeomorphic_maps_def*
by (*smt* (*verit*, *del_insts*) *continuous_map_eq continuous_map_subtopology_fst embedding_map_def*
embedding_map_graph homeomorphic_eq_everything_map image_cong image_iff prod.sel(*1*))
qed

5.6.10 KC spaces, those where all compact sets are closed.

definition *kc_space*
where *kc_space X* ≡ ∀ *S*. *compactin X S* ⟶ *closedin X S*

lemma *kc_space_euclidean*: *kc_space* (*euclidean* :: *'a*::*metric_space topology*)
by (*simp add*: *compact_imp_closed kc_space_def*)

lemma *kc_space_expansive*:
⟦*kc_space X*; *topspace Y* = *topspace X*; ⋀*U*. *openin X U* ⟹ *openin Y U*⟧
⟹ *kc_space Y*
by (*meson compactin_contractive kc_space_def topology_finer_closedin*)

lemma *compactin_imp_closedin_gen*:
⟦*kc_space X*; *compactin X S*⟧ ⟹ *closedin X S*
using *kc_space_def* **by** *blast*

lemma *Hausdorff_imp_kc_space*: *Hausdorff_space X* ⟹ *kc_space X*
by (*simp add*: *compactin_imp_closedin kc_space_def*)

lemma *kc_imp_t1_space*: *kc_space X* ⟹ *t1_space X*
by (*simp add*: *finite_imp_compactin kc_space_def t1_space_closedin_finite*)

lemma *kc_space_subtopology*:
kc_space X ⟹ *kc_space*(*subtopology X S*)
by (*metis closedin_Int_closure_of closure_of_eq compactin_subtopology inf.absorb2 kc_space_def*)

lemma *kc_space_discrete_topology*:
kc_space(*discrete_topology U*)

using *Hausdorff_space_discrete_topology compactin_imp_closedin kc_space_def*
by *blast*

lemma *kc_space_continuous_injective_map_preimage*:
assumes *kc_space Y continuous_map X Y f* **and** *injf*: *inj_on f* (*topspace X*)
shows *kc_space X*
unfolding *kc_space_def*
proof (*intro strip*)
fix *S*
assume *S*: *compactin X S*
have $S = \{x \in$ *topspace X*. *f x* $\in$ *f* ' *S*$\}$
using *S compactin_subset_topspace inj_onD* [*OF injf*] **by** *blast*
with *assms S* **show** *closedin X S*
by (*metis* (*no_types*, *lifting*) *Collect_cong closedin_continuous_map_preimage compactin_imp_closedin_gen image_compactin*)
qed

lemma *kc_space_retraction_map_image*:
assumes *retraction_map X Y r kc_space X*
shows *kc_space Y*
proof −
obtain *s* **where** *s*: *continuous_map X Y r continuous_map Y X s* $\bigwedge x$. *x* $\in$ *topspace Y* $\Longrightarrow$ *r* (*s x*) = *x*
using *assms* **by** (*force simp*: *retraction_map_def retraction_maps_def*)
then have *inj*: *inj_on s* (*topspace Y*)
by (*metis inj_on_def*)
show *?thesis*
unfolding *kc_space_def*
proof (*intro strip*)
fix *S*
assume *S*: *compactin Y S*
have $S = \{x \in$ *topspace Y*. *s x* $\in$ *s* ' *S*$\}$
using *S compactin_subset_topspace inj_onD* [*OF inj*] **by** *blast*
with *assms S* **show** *closedin Y S*
by (*meson compactin_imp_closedin_gen inj kc_space_continuous_injective_map_preimage s*(*2*))
qed
qed

lemma *homeomorphic_kc_space*:
X homeomorphic_space Y $\Longrightarrow$ *kc_space X* $\longleftrightarrow$ *kc_space Y*
by (*meson homeomorphic_eq_everything_map homeomorphic_space homeomorphic_space_sym kc_space_continuous_injective_map_preimage*)

lemma *compact_kc_eq_maximal_compact_space*:
assumes *compact_space X*
shows *kc_space X* $\longleftrightarrow$
($\forall Y$. *topspace Y* = *topspace X* $\wedge$ ($\forall S$. *openin X S* $\longrightarrow$ *openin Y S*) $\wedge$ *compact_space Y* $\longrightarrow$ *Y* = *X*) (**is** *?lhs=?rhs*)

proof
assume *?lhs*
then show *?rhs*
by (*metis closedin_compact_space compactin_contractive kc_space_def topology_eq topology_finer_closedin*)
next
assume *R*: *?rhs*
show *?lhs*
unfolding *kc_space_def*
proof (*intro strip*)
fix *S*
assume *S*: *compactin X S*
define *Y* **where**
Y ≡ *topology* (*arbitrary union_of* (*finite intersection_of* (λ*A*. *A* = *topspace X* − *S* ∨ *openin X A*)
relative_to (*topspace X*)))
have *topspace Y* = *topspace X*
by (*auto simp*: *Y_def*)
have *openin X T* ⟶ *openin Y T* **for** *T*
by (*simp add*: *Y_def arbitrary_union_of_inc finite_intersection_of_inc openin_subbase openin_subset relative_to_subset_inc*)
have *compact_space Y*
proof (*rule Alexander_subbase_alt*)
show ∃*F*′. *finite F*′ ∧ *F*′ ⊆ *C* ∧ *topspace X* ⊆ ⋃ *F*′
if *C*: *C* ⊆ *insert* (*topspace X* − *S*) (*Collect* (*openin X*)) **and** *sub*: *topspace X* ⊆ ⋃*C* **for** *C*
proof −
consider *C* ⊆ *Collect* (*openin X*) | *V* **where** *C* = *insert* (*topspace X* − *S*) *V* *V* ⊆ *Collect* (*openin X*)
using *C* **by** (*metis insert_Diff subset_insert_iff*)
then show *?thesis*
proof *cases*
case *1*
then show *?thesis*
by (*metis assms compact_space_alt mem_Collect_eq subsetD that*(*2*))
next
case *2*
then have *S* ⊆ ⋃*V*
using *S sub compactin_subset_topspace* **by** *blast*
with *2* **obtain** *F* **where** *finite F* ∧ *F* ⊆ *V* ∧ *S* ⊆ ⋃*F*
using *S* **unfolding** *compactin_def* **by** (*metis Ball_Collect*)
with *2* **show** *?thesis*
by (*rule_tac x*=*insert* (*topspace X* − *S*) *F* **in** *exI*) *blast*
qed
qed
qed (*auto simp*: *Y_def*)
have *Y* = *X*
using *R* ‹⋀*S*. *openin X S* ⟶ *openin Y S*› ‹*compact_space Y*› ‹*topspace Y* = *topspace X*› **by** *blast*

```
    moreover have openin Y (topspace X − S)
      by (simp add: Y_def arbitrary_union_of_inc finite_intersection_of_inc
openin_subbase relative_to_subset_inc)
    ultimately show closedin X S
     unfolding closedin_def using S compactin_subset_topspace by blast
  qed
qed

lemma continuous_imp_closed_map_gen:
  ⟦compact_space X; kc_space Y; continuous_map X Y f⟧ ⟹ closed_map X Y
f
 by (meson closed_map_def closedin_compact_space compactin_imp_closedin_gen
image_compactin)

lemma kc_space_compact_subtopologies:
  kc_space X ⟷ (∀K. compactin X K ⟶ kc_space(subtopology X K)) (is
?lhs=?rhs)
proof
  assume ?lhs
  then show ?rhs
   by (auto simp: kc_space_def closedin_closed_subtopology compactin_subtopology)
next
  assume R: ?rhs
  show ?lhs
    unfolding kc_space_def
  proof (intro strip)
    fix K
    assume K: compactin X K
    then have K ⊆ topspace X
      by (simp add: compactin_subset_topspace)
    moreover have X closure_of K ⊆ K
    proof
      fix x
      assume x: x ∈ X closure_of K
      have kc_space (subtopology X K)
        by (simp add: R ‹compactin X K›)
      have compactin X (insert x K)
       by (metis K x compactin_Un compactin_sing in_closure_of insert_is_Un)
      then show x ∈ K
     by (metis R x K Int_insert_left_if1 closedin_Int_closure_of compact_imp_compactin_subtopology

          insertCI kc_space_def subset_insertI)
    qed
    ultimately show closedin X K
      using closure_of_subset_eq by blast
  qed
qed

lemma kc_space_compact_prod_topology:
```

assumes *compact_space X*
shows *kc_space*(*prod_topology X X*) ⟷ *Hausdorff_space X* (**is** *?lhs*=*?rhs*)
proof
assume *L*: *?lhs*
show *?rhs*
unfolding *closed_map_diag_eq* [*symmetric*]
proof (*intro continuous_imp_closed_map_gen*)
show *continuous_map X* (*prod_topology X X*) (λ*x*. (*x*, *x*))
by (*intro continuous_intros*)
qed (*use L assms* **in** *auto*)
next
assume *?rhs* **then show** *?lhs*
by (*simp add*: *Hausdorff_imp_kc_space Hausdorff_space_prod_topology*)
qed

lemma *kc_space_prod_topology*:
kc_space(*prod_topology X X*) ⟷ (∀ *K*. *compactin X K* ⟶ *Hausdorff_space*(*subtopology X K*)) (**is** *?lhs*=*?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (*metis compactin_subspace kc_space_compact_prod_topology kc_space_subtopology subtopology_Times*)
next
assume *R*: *?rhs*
have *kc_space* (*subtopology* (*prod_topology X X*) *L*) **if** *compactin* (*prod_topology X X*) *L* **for** *L*
proof −
define *K* **where** *K* ≡ *fst* ‘ *L* ∪ *snd* ‘ *L*
have *L* ⊆ *K* × *K*
by (*force simp*: *K_def*)
have *compactin X K*
by (*metis K_def compactin_Un continuous_map_fst continuous_map_snd image_compactin that*)
then have *Hausdorff_space* (*subtopology X K*)
by (*simp add*: *R*)
then have *kc_space* (*prod_topology* (*subtopology X K*) (*subtopology X K*))
by (*simp add*: ‹*compactin X K*› *compact_space_subtopology kc_space_compact_prod_topology*)
then have *kc_space* (*subtopology* (*prod_topology* (*subtopology X K*) (*subtopology X K*)) *L*)
using *kc_space_subtopology* **by** *blast*
then show *?thesis*
using ‹*L* ⊆ *K* × *K*› *subtopology_Times subtopology_subtopology*
by (*metis* (*no_types*, *lifting*) *Sigma_cong inf.absorb_iff2*)
qed
then show *?lhs*
using *kc_space_compact_subtopologies* **by** *blast*
qed

lemma *kc_space_prod_topology_alt*:
kc_space(*prod_topology X X*) ⟷
kc_space X ∧
(∀ *K*. *compactin X K* ⟶ *Hausdorff_space*(*subtopology X K*))
using *Hausdorff_imp_kc_space kc_space_compact_subtopologies kc_space_prod_topology*
by *blast*

proposition *kc_space_prod_topology_left*:
assumes *X*: *kc_space X* **and** *Y*: *Hausdorff_space Y*
shows *kc_space* (*prod_topology X Y*)
unfolding *kc_space_def*
proof (*intro strip*)
fix *K*
assume *K*: *compactin* (*prod_topology X Y*) *K*
then have *K* ⊆ *topspace X* × *topspace Y*
using *compactin_subset_topspace topspace_prod_topology* **by** *blast*
moreover have ∃ *T*. *openin* (*prod_topology X Y*) *T* ∧ (*a*,*b*) ∈ *T* ∧ *T* ⊆ (*topspace X* × *topspace Y*) − *K*
if *ab*: (*a*,*b*) ∈ (*topspace X* × *topspace Y*) − *K* **for** *a b*
proof −
have *compactin Y* {*b*}
using *that* **by** *force*
moreover
have *compactin Y* {*y* ∈ *topspace Y*. (*a*,*y*) ∈ *K*}
proof −
have *compactin* (*prod_topology X Y*) (*K* ∩ {*a*} × *topspace Y*)
using *that compact_Int_closedin* [*OF K*]
by (*simp add*: *X closedin_prod_Times_iff compactin_imp_closedin_gen*)
moreover have *subtopology* (*prod_topology X Y*) (*K* ∩ {*a*} × *topspace Y*) *homeomorphic_space*
subtopology Y {*y* ∈ *topspace Y*. (*a*, *y*) ∈ *K*}
unfolding *homeomorphic_space_def homeomorphic_maps_def*
using *that*
apply (*rule_tac x*=*snd* **in** *exI*)
apply (*rule_tac x*=*Pair a* **in** *exI*)
by (*force simp*: *continuous_map_in_subtopology continuous_map_from_subtopology continuous_map_subtopology_snd continuous_map_paired*)
ultimately show *?thesis*
by (*simp add*: *compactin_subspace homeomorphic_compact_space*)
qed
moreover have *disjnt* {*b*} {*y* ∈ *topspace Y*. (*a*,*y*) ∈ *K*}
using *ab* **by** *force*
ultimately obtain *V U* **where** *VU*: *openin Y V openin Y U* {*b*} ⊆ *V* {*y* ∈ *topspace Y*. (*a*,*y*) ∈ *K*} ⊆ *U disjnt V U*
using *Hausdorff_space_compact_separation* [*OF Y*] **by** *blast*
define *V'* **where** *V'* ≡ *topspace Y* − *U*
have *W*: *closedin Y V'* {*y* ∈ *topspace Y*. (*a*,*y*) ∈ *K*} ⊆ *topspace Y* − *V' disjnt V* (*topspace Y* − *V'*)
using *VU* **by** (*auto simp*: *V'_def disjnt_iff*)

with *VU* **obtain** $V \subseteq$ *topspace Y* $V' \subseteq$ *topspace Y*
by (*meson closedin_subset openin_closedin_eq*)
then obtain $b \in V$ *disjnt* $\{y \in$ *topspace Y*. $(a,y) \in K\}$ V' $V \subseteq V'$
using *VU* **unfolding** *disjnt_iff V'_def* **by** *force*
define *C* **where** $C \equiv$ *image fst* $(K \cap \{z \in$ *topspace*(*prod_topology X Y*). *snd* $z \in V'\})$
have *closedin* (*prod_topology X Y*) $\{z \in$ *topspace* (*prod_topology X Y*). *snd z* $\in V'\}$
using *closedin_continuous_map_preimage* ‹*closedin Y V'*› *continuous_map_snd*
by *blast*
then have *compactin X C*
unfolding *C_def* **by** (*meson K compact_Int_closedin continuous_map_fst image_compactin*)
then have *closedin X C*
using *assms* **by** (*auto simp*: *kc_space_def*)
show *?thesis*
proof (*intro exI conjI*)
show *openin* (*prod_topology X Y*) ((*topspace X* − *C*) × *V*)
by (*simp add*: *VU* ‹*closedin X C*› *openin_diff openin_prod_Times_iff*)
have $a \notin C$
using *VU* **by** (*auto simp*: *C_def V'_def*)
then show $(a, b) \in$ (*topspace X* − *C*) × *V*
using ‹$a \notin C$› ‹$b \in V$› *that* **by** *blast*
show (*topspace X* − *C*) × *V* ⊆ *topspace X* × *topspace Y* − *K*
using ‹$V \subseteq V'$› ‹$V \subseteq$ *topspace Y*›
apply (*simp add*: *C_def*)
by (*smt* (*verit*, *ccfv_threshold*) *DiffE DiffI IntI SigmaE SigmaI image_eqI mem_Collect_eq prod.sel*(*1*) *snd_conv subset_iff*)
qed
qed
ultimately show *closedin* (*prod_topology X Y*) *K*
by (*metis surj_pair closedin_def openin_subopen topspace_prod_topology*)
qed

lemma *kc_space_prod_topology_right*:
⟦*Hausdorff_space X*; *kc_space Y*⟧ ⟹ *kc_space* (*prod_topology X Y*)
using *kc_space_prod_topology_left homeomorphic_kc_space homeomorphic_space_prod_topology_su*
by *blast*

5.6.11 Technical results about proper maps, perfect maps, etc

lemma *compact_imp_proper_map_gen*:
assumes *Y*: ⋀*S*. ⟦*S* ⊆ *topspace Y*; ⋀*K*. *compactin Y K* ⟹ *compactin Y* (*S* ∩ *K*)⟧
⟹ *closedin Y S*
and *fim*: *f* ‘ (*topspace X*) ⊆ *topspace Y*
and *f*: *continuous_map X Y f* ∨ *kc_space X*
and *YX*: ⋀*K*. *compactin Y K* ⟹ *compactin X* $\{x \in$ *topspace X*. *f x* $\in K\}$

```
  shows proper_map X Y f
  unfolding proper_map_alt closed_map_def
proof (intro conjI strip)
  fix C
  assume C: closedin X C
  show closedin Y (f ‘ C)
  proof (intro Y)
    show f ‘ C ⊆ topspace Y
      using C closedin_subset fim by blast
    fix K
    assume K: compactin Y K
    define A where A ≡ {x ∈ topspace X. f x ∈ K}
    have eq: f ‘ C ∩ K = f ‘ ({x ∈ topspace X. f x ∈ K} ∩ C)
      using C closedin_subset by auto
    show compactin Y (f ‘ C ∩ K)
      unfolding eq
    proof (rule image_compactin)
      show compactin (subtopology X A) ({x ∈ topspace X. f x ∈ K} ∩ C)
      proof (rule closedin_compact_space)
        show compact_space (subtopology X A)
          by (simp add: A_def K YX compact_space_subtopology)
        show closedin (subtopology X A) ({x ∈ topspace X. f x ∈ K} ∩ C)
          using A_def C closedin_subtopology by blast
      qed
      have continuous_map (subtopology X A) (subtopology Y K) f if kc_space X
        unfolding continuous_map_closedin
      proof (intro conjI strip)
        show f ∈ topspace (subtopology X A) → topspace (subtopology Y K)
          using A_def K compactin_subset_topspace by fastforce
      next
        fix C
        assume C: closedin (subtopology Y K) C
       show closedin (subtopology X A) {x ∈ topspace (subtopology X A). f x ∈ C}
        proof (rule compactin_imp_closedin_gen)
          show kc_space (subtopology X A)
            by (simp add: kc_space_subtopology that)
         have [simp]: {x ∈ topspace X. f x ∈ K ∧ f x ∈ C} = {x ∈ topspace X. f
x ∈ C}
            using C closedin_imp_subset by auto
          have compactin (subtopology Y K) C
           by (simp add: C K closedin_compact_space compact_space_subtopology)
          then have compactin X {x ∈ topspace X. x ∈ A ∧ f x ∈ C}
            by (auto simp: A_def compactin_subtopology dest: YX)
         then show compactin (subtopology X A) {x ∈ topspace (subtopology X A).
f x ∈ C}
            by (auto simp add: compactin_subtopology)
        qed
      qed
      with f show continuous_map (subtopology X A) Y f
```

using *continuous_map_from_subtopology continuous_map_in_subtopology*
by *blast*
qed
qed
qed (*simp add*: *YX*)

lemma *tube_lemma_left*:
assumes *W*: *openin* (*prod_topology X Y*) *W* **and** *C*: *compactin X C*
and *y*: $y \in$ *topspace Y* **and** *subW*: $C \times \{y\} \subseteq W$
shows $\exists\, U\ V.$ *openin X U* $\wedge$ *openin Y V* $\wedge\ C \subseteq U \wedge y \in V \wedge U \times V \subseteq W$
proof (*cases* $C = \{\}$)
case *True*
with *y* **show** *?thesis* **by** *auto*
next
case *False*
have $\exists\, U\ V.$ *openin X U* $\wedge$ *openin Y V* $\wedge\ x \in U \wedge y \in V \wedge U \times V \subseteq W$
if $x \in C$ **for** *x*
using *W openin_prod_topology_alt subW subsetD that* **by** *fastforce*
then obtain *U V* **where** *UV*: $\bigwedge x.\ x \in C \Longrightarrow$ *openin X* (*U x*) $\wedge$ *openin Y* (*V x*) $\wedge\ x \in U\ x \wedge y \in V\ x \wedge U\ x \times V\ x \subseteq W$
by *metis*
then obtain *D* **where** *D*: *finite D* $D \subseteq C\ C \subseteq \bigcup$ (*U ' D*)
using *compactinD* [*OF C*, *of U'C*]
by (*smt* (*verit*) *UN_I finite_subset_image imageE subsetI*)
show *?thesis*
proof (*intro exI conjI*)
show *openin X* ($\bigcup$ (*U ' D*)) *openin Y* ($\bigcap$ (*V ' D*))
using *D False UV* **by** *blast+*
show $y \in \bigcap$ (*V ' D*) $C \subseteq \bigcup$ (*U ' D*) $\bigcup(U$ ' $D) \times \bigcap(V$ ' $D) \subseteq W$
using *D UV* **by** *force+*
qed
qed

lemma *Wallace_theorem_prod_topology*:
assumes *compactin X K compactin Y L*
and *W*: *openin* (*prod_topology X Y*) *W* **and** *subW*: $K \times L \subseteq W$
obtains *U V* **where** *openin X U openin Y V K* $\subseteq U\ L \subseteq V\ U \times V \subseteq W$
proof −
have $\bigwedge y.\ y \in L \Longrightarrow \exists\, U\ V.$ *openin X U* $\wedge$ *openin Y V* $\wedge\ K \subseteq U \wedge y \in V \wedge U \times V \subseteq W$
proof (*intro tube_lemma_left assms*)
fix *y* **assume** $y \in L$
show $y \in$ *topspace Y*
using *assms* ‹$y \in L$› *compactin_subset_topspace* **by** *blast*
show $K \times \{y\} \subseteq W$
using ‹$y \in L$› *subW* **by** *force*
qed
then obtain *U V* **where** *UV*:
$\bigwedge y.\ y \in L \Longrightarrow$ *openin X* (*U y*) $\wedge$ *openin Y* (*V y*) $\wedge\ K \subseteq U\ y \wedge y \in V\ y$

```
∧ U y × V y ⊆ W
    by metis
  then obtain M where finite M M ⊆ L and M: L ⊆ ⋃ (V ‘ M)
    using ‹compactin Y L› unfolding compactin_def
    by (smt (verit) UN_iff finite_subset_image imageE subset_iff)
  show thesis
  proof (cases M={})
    case True
    with M have L={}
      by blast
    then show ?thesis
      using ‹compactin X K› compactin_subset_topspace that by fastforce
  next
    case False
    show ?thesis
    proof
      show openin X (⋂(U‘M))
        using False UV ‹M ⊆ L› ‹finite M› by blast
      show openin Y (⋃(V‘M))
        using UV ‹M ⊆ L› by blast
      show K ⊆ ⋂(U‘M)
        by (meson INF_greatest UV ‹M ⊆ L› subsetD)
      show L ⊆ ⋃(V‘M)
        by (simp add: M)
      show ⋂(U‘M) × ⋃(V‘M) ⊆ W
        using UV ‹M ⊆ L› by fastforce
    qed
  qed
qed

lemma proper_map_prod:
   proper_map (prod_topology X Y) (prod_topology X' Y') (λ(x,y). (f x, g y)) ⟷
    (prod_topology X Y) = trivial_topology ∨ proper_map X X' f ∧ proper_map
Y Y' g
   (is ?lhs ⟷ _ ∨ ?rhs)
proof (cases (prod_topology X Y) = trivial_topology)
  case True then show ?thesis by auto
next
  case False
  then have ne: topspace X ≠ {} topspace Y ≠ {}
    by auto
  define h where h ≡ λ(x,y). (f x, g y)
  have proper_map X X' f proper_map Y Y' g if ?lhs
  proof −
    have cm: closed_map X X' f closed_map Y Y' g
      using that False closed_map_prod proper_imp_closed_map by blast+
    show proper_map X X' f
    proof (clarsimp simp add: proper_map_def cm)
      fix y
```

```
    assume y: y ∈ topspace X'
    obtain z where z: z ∈ topspace Y
      using ne by blast
    then have eq: {x ∈ topspace X. f x = y} =
                  fst ` {u ∈ topspace X × topspace Y. h u = (y,g z)}
      by (force simp: h_def)
    show compactin X {x ∈ topspace X. f x = y}
      unfolding eq
    proof (intro image_compactin)
      have g z ∈ topspace Y'
        by (meson closed_map_def closedin_subset closedin_topspace cm image_subset_iff z)
      with y show compactin (prod_topology X Y) {u ∈ topspace X × topspace Y. (h u) = (y, g z)}
        using that by (simp add: h_def proper_map_def)
      show continuous_map (prod_topology X Y) X fst
        by (simp add: continuous_map_fst)
    qed
  qed
  show proper_map Y Y' g
  proof (clarsimp simp add: proper_map_def cm)
    fix y
    assume y: y ∈ topspace Y'
    obtain z where z: z ∈ topspace X
      using ne by blast
    then have eq: {x ∈ topspace Y. g x = y} =
                  snd ` {u ∈ topspace X × topspace Y. h u = (f z,y)}
      by (force simp: h_def)
    show compactin Y {x ∈ topspace Y. g x = y}
      unfolding eq
    proof (intro image_compactin)
      have f z ∈ topspace X'
        by (meson closed_map_def closedin_subset closedin_topspace cm image_subset_iff z)
      with y show compactin (prod_topology X Y) {u ∈ topspace X × topspace Y. (h u) = (f z, y)}
        using that by (simp add: proper_map_def h_def)
      show continuous_map (prod_topology X Y) Y snd
        by (simp add: continuous_map_snd)
    qed
  qed
 qed
 moreover
 { assume R: ?rhs
   then have fgim: f ∈ topspace X → topspace X' g ∈ topspace Y → topspace Y'
        and cm: closed_map X X' f closed_map Y Y' g
     by (auto simp: proper_map_def closed_map_imp_subset_topspace)
   have closed_map (prod_topology X Y) (prod_topology X' Y') h
```

unfolding *closed_map_fibre_neighbourhood imp_conjL*
proof (*intro conjI strip*)
show $h \in$ *topspace* (*prod_topology X Y*) $\to$ *topspace* (*prod_topology X' Y'*)
unfolding *h_def* **using** *fgim* **by** *auto*
fix *W w*
assume *W*: *openin* (*prod_topology X Y*) *W*
and *w*: $w \in$ *topspace* (*prod_topology X' Y'*)
and *subW*: $\{x \in$ *topspace* (*prod_topology X Y*). $h\ x = w\} \subseteq W$
then obtain $x'\ y'$ **where** *weq*: $w = (x',y')$ $x' \in$ *topspace* X' $y' \in$ *topspace* Y'
by *auto*
have *eq*: $\{u \in$ *topspace* $X \times$ *topspace* Y. $h\ u = (x',y')\} = \{x \in$ *topspace* X. $f\ x = x'\} \times \{y \in$ *topspace* Y. $g\ y = y'\}$
by (*auto simp*: *h_def*)
obtain $U\ V$ **where** *openin* $X\ U$ *openin* $Y\ V$ $U \times V \subseteq W$
and *U*: $\{x \in$ *topspace* X. $f\ x = x'\} \subseteq U$
and *V*: $\{x \in$ *topspace* Y. $g\ x = y'\} \subseteq V$
proof (*rule Wallace_theorem_prod_topology*)
show *compactin* X $\{x \in$ *topspace* X. $f\ x = x'\}$ *compactin* Y $\{x \in$ *topspace* Y. $g\ x = y'\}$
using *R weq* **unfolding** *proper_map_def closed_map_fibre_neighbourhood* **by** *fastforce+*
show $\{x \in$ *topspace* X. $f\ x = x'\} \times \{x \in$ *topspace* Y. $g\ x = y'\} \subseteq W$
using *weq subW* **by** (*auto simp*: *h_def*)
qed (*use W* **in** *auto*)
obtain U' **where** *openin* $X'\ U'\ x' \in U'$ **and** U': $\{x \in$ *topspace* X. $f\ x \in U'\} \subseteq U$
using *cm U* ‹*openin X U*› *weq* **unfolding** *closed_map_fibre_neighbourhood* **by** *meson*
obtain V' **where** *openin* $Y'\ V'\ y' \in V'$ **and** V': $\{x \in$ *topspace* Y. $g\ x \in V'\} \subseteq V$
using *cm V* ‹*openin Y V*› *weq* **unfolding** *closed_map_fibre_neighbourhood* **by** *meson*
show $\exists\, V$. *openin* (*prod_topology X' Y'*) $V \wedge w \in V \wedge \{x \in$ *topspace* (*prod_topology X Y*). $h\ x \in V\} \subseteq W$
proof (*intro conjI exI*)
show *openin* (*prod_topology X' Y'*) $(U' \times V')$
by (*simp add*: ‹*openin X' U'*› ‹*openin Y' V'*› *openin_prod_Times_iff*)
show $w \in U' \times V'$
using ‹$x' \in U'$› ‹$y' \in V'$› *weq* **by** *blast*
show $\{x \in$ *topspace* (*prod_topology X Y*). $h\ x \in U' \times V'\} \subseteq W$
using ‹$U \times V \subseteq W$› U' V' *h_def* **by** *auto*
qed
qed
moreover
have *compactin* (*prod_topology X Y*) $\{u \in$ *topspace* $X \times$ *topspace* Y. $h\ u = (w, z)\}$
if $w \in$ *topspace* X' **and** $z \in$ *topspace* Y' **for** $w\ z$
proof −
have *eq*: $\{u \in$ *topspace* $X \times$ *topspace* Y. $h\ u = (w,z)\} =$

```
        {u ∈ topspace X. f u = w} × {y. y ∈ topspace Y ∧ g y = z}
    by (auto simp: h_def)
   show ?thesis
    using R that by (simp add: eq compactin_Times proper_map_def)
  qed
  ultimately have ?lhs
   by (auto simp: h_def proper_map_def)
 }
 ultimately show ?thesis using False by metis
qed

lemma proper_map_paired:
 assumes Hausdorff_space X ∧ proper_map X Y f ∧ proper_map X Z g ∨
     Hausdorff_space Y ∧ continuous_map X Y f ∧ proper_map X Z g ∨
     Hausdorff_space Z ∧ proper_map X Y f ∧ continuous_map X Z g
 shows proper_map X (prod_topology Y Z) (λx. (f x,g x))
 using assms
proof (elim disjE conjE)
 assume §: Hausdorff_space X proper_map X Y f proper_map X Z g
 have eq: (λx. (f x, g x)) = (λ(x, y). (f x, g y)) ∘ (λx. (x, x))
  by auto
 show proper_map X (prod_topology Y Z) (λx. (f x, g x))
  unfolding eq
 proof (rule proper_map_compose)
  show proper_map X (prod_topology X X) (λx. (x,x))
   by (simp add: §)
  show proper_map (prod_topology X X) (prod_topology Y Z) (λ(x,y). (f x, g
y))
   by (simp add: § proper_map_prod)
 qed
next
 assume §: Hausdorff_space Y continuous_map X Y f proper_map X Z g
 have eq: (λx. (f x, g x)) = (λ(x,y). (x,g y)) ∘ (λx. (f x,x))
  by auto
 show proper_map X (prod_topology Y Z) (λx. (f x, g x))
  unfolding eq
 proof (rule proper_map_compose)
  show proper_map X (prod_topology Y X) (λx. (f x,x))
   by (simp add: § proper_map_paired_continuous_map_left)
  show proper_map (prod_topology Y X) (prod_topology Y Z) (λ(x,y). (x,g y))
   by (simp add: § proper_map_prod proper_map_id [unfolded id_def])
 qed
next
 assume §: Hausdorff_space Z proper_map X Y f continuous_map X Z g
 have eq: (λx. (f x, g x)) = (λ(x,y). (f x,y)) ∘ (λx. (x,g x))
  by auto
 show proper_map X (prod_topology Y Z) (λx. (f x, g x))
  unfolding eq
 proof (rule proper_map_compose)
```

```
    show proper_map X (prod_topology X Z) (λx. (x, g x))
      using § proper_map_paired_continuous_map_right by auto
    show proper_map (prod_topology X Z) (prod_topology Y Z) (λ(x,y). (f x,y))
      by (simp add: § proper_map_prod proper_map_id [unfolded id_def])
  qed
qed

lemma proper_map_pairwise:
  assumes
    Hausdorff_space X ∧ proper_map X Y (fst ∘ f) ∧ proper_map X Z (snd ∘ f)
∨
    Hausdorff_space Y ∧ continuous_map X Y (fst ∘ f) ∧ proper_map X Z (snd
∘ f) ∨
    Hausdorff_space Z ∧ proper_map X Y (fst ∘ f) ∧ continuous_map X Z (snd
∘ f)
  shows proper_map X (prod_topology Y Z) f
  using proper_map_paired [OF assms] by (simp add: o_def)

lemma proper_map_from_composition_right:
   assumes Hausdorff_space Y proper_map X Z (g ∘ f) and contf: continu-
ous_map X Y f
    and contg: continuous_map Y Z g
  shows proper_map X Y f
proof −
  define YZ where YZ ≡ subtopology (prod_topology Y Z) ((λx. (x, g x)) ‘
topspace Y)
  have proper_map X Y (fst ∘ (λx. (f x, (g ∘ f) x)))
  proof (rule proper_map_compose)
    have [simp]: x ∈ topspace X ⟹ f x ∈ topspace Y for x
      using contf continuous_map_preimage_topspace by auto
    show proper_map X YZ (λx. (f x, (g ∘ f) x))
      unfolding YZ_def
      using assms
        by (force intro!: proper_map_into_subtopology proper_map_paired simp:
o_def image_iff)+
    show proper_map YZ Y fst
      using contg
     by (simp flip: homeomorphic_maps_graph add: YZ_def homeomorphic_maps_map
homeomorphic_imp_proper_map)
  qed
  moreover have fst ∘ (λx. (f x, (g ∘ f) x)) = f
    by auto
  ultimately show ?thesis
    by auto
qed

lemma perfect_map_from_composition_right:
  ⟦Hausdorff_space Y; perfect_map X Z (g ∘ f);
    continuous_map X Y f; continuous_map Y Z g; f ‘ topspace X = topspace Y⟧
```

$\Longrightarrow$ *perfect_map X Y f*
by (*meson perfect_map_def proper_map_from_composition_right*)

lemma *perfect_map_from_composition_right_inj*:
⟦*perfect_map X Z* (*g* ∘ *f*); *f* ‘ *topspace X* = *topspace Y*;
continuous_map X Y f; *continuous_map Y Z g*; *inj_on g* (*topspace Y*)⟧
$\Longrightarrow$ *perfect_map X Y f*
by (*meson continuous_map_openin_preimage_eq perfect_map_def proper_map_from_composition_r*

5.6.12 Regular spaces

Regular spaces are *not* a priori assumed to be Hausdorff or T_1

definition *regular_space*
where *regular_space X* ≡
∀ *C a*. *closedin X C* ∧ *a* ∈ *topspace X* − *C*
⟶ (∃ *U V*. *openin X U* ∧ *openin X V* ∧ *a* ∈ *U* ∧ *C* ⊆ *V* ∧ *disjnt U V*)

lemma *homeomorphic_regular_space_aux*:
assumes *hom*: *X homeomorphic_space Y* **and** *X*: *regular_space X*
shows *regular_space Y*
proof −
obtain *f g* **where** *hmf*: *homeomorphic_map X Y f* **and** *hmg*: *homeomorphic_map Y X g*
and *fg*: (∀ *x* ∈ *topspace X*. *g*(*f x*) = *x*) ∧ (∀ *y* ∈ *topspace Y*. *f*(*g y*) = *y*)
using *assms X homeomorphic_maps_map homeomorphic_space_def* **by** *fastforce*
show *?thesis*
unfolding *regular_space_def*
proof *clarify*
fix *C a*
assume *Y*: *closedin Y C a* ∈ *topspace Y* **and** *a* ∉ *C*
then obtain *closedin X* (*g* ‘ *C*) *g a* ∈ *topspace X g a* ∉ *g* ‘ *C*
using ‹*closedin Y C*› *hmg homeomorphic_map_closedness_eq*
by (*smt* (*verit*, *ccfv_SIG*) *fg homeomorphic_imp_surjective_map image_iff in_mono*)
then obtain *S T* **where** *ST*: *openin X S openin X T g a* ∈ *S g*‘*C* ⊆ *T disjnt S T*
using *X* **unfolding** *regular_space_def* **by** (*metis DiffI*)
then have *openin Y* (*f*‘*S*) *openin Y* (*f*‘*T*)
by (*meson hmf homeomorphic_map_openness_eq*)+
moreover have *a* ∈ *f*‘*S* ∧ *C* ⊆ *f*‘*T*
using *ST* **by** (*smt* (*verit*, *best*) *Y closedin_subset fg image_eqI subset_iff*)
moreover have *disjnt* (*f*‘*S*) (*f*‘*T*)
using *ST* **by** (*smt* (*verit*, *ccfv_SIG*) *disjnt_iff fg image_iff openin_subset subsetD*)
ultimately show ∃ *U V*. *openin Y U* ∧ *openin Y V* ∧ *a* ∈ *U* ∧ *C* ⊆ *V* ∧ *disjnt U V*
by *metis*

```
  qed
qed

lemma homeomorphic_regular_space:
   X homeomorphic_space Y
        ⟹ (regular_space X ⟷ regular_space Y)
  by (meson homeomorphic_regular_space_aux homeomorphic_space_sym)

lemma regular_space:
   regular_space X ⟷
        (∀ C a. closedin X C ∧ a ∈ topspace X − C
              ⟶ (∃ U. openin X U ∧ a ∈ U ∧ disjnt C (X closure_of U)))
  unfolding regular_space_def
proof (intro all_cong1 imp_cong refl ex_cong1)
  fix C a U
  assume C: closedin X C ∧ a ∈ topspace X − C
  show (∃ V. openin X U ∧ openin X V ∧ a ∈ U ∧ C ⊆ V ∧ disjnt U V)
    ⟷ (openin X U ∧ a ∈ U ∧ disjnt C (X closure_of U)) (is ?lhs=?rhs)
  proof
    assume ?lhs
    then show ?rhs
      by (smt (verit, best) disjnt_iff in_closure_of subsetD)
  next
    assume R: ?rhs
    then have disjnt U (topspace X − X closure_of U)
      by (meson DiffD2 closure_of_subset disjnt_iff openin_subset subsetD)
    moreover have C ⊆ topspace X − X closure_of U
      by (meson C DiffI R closedin_subset disjnt_iff subset_eq)
    ultimately show ?lhs
      using R by (rule_tac x=topspace X − X closure_of U in exI) auto
    qed
qed

lemma neighbourhood_base_of_closedin:
  neighbourhood_base_of (closedin X) X ⟷ regular_space X (is ?lhs=?rhs)
proof −
  have ?lhs ⟷ (∀ W x. openin X W ∧ x ∈ W ⟶
                (∃ U. openin X U ∧ (∃ V. closedin X V ∧ x ∈ U ∧ U ⊆ V ∧ V ⊆
W)))
    by (simp add: neighbourhood_base_of)
  also have ... ⟷ (∀ W x. closedin X W ∧ x ∈ topspace X − W ⟶
                  (∃ U. openin X U ∧ (∃ V. closedin X V ∧ x ∈ U ∧ U ⊆ V ∧ V
⊆ topspace X − W)))
   by (smt (verit) Diff_Diff_Int closedin_def inf.absorb_iff2 openin_closedin_eq)
  also have ... ⟷ ?rhs
  proof −
    have §: (∃ V. closedin X V ∧ x ∈ U ∧ U ⊆ V ∧ V ⊆ topspace X − W)
        ⟷ (∃ V. openin X V ∧ x ∈ U ∧ W ⊆ V ∧ disjnt U V) (is ?lhs=?rhs)
      if openin X U  closedin X W x ∈ topspace X x ∉ W for W U x
```

proof
assume *?lhs* **with** ‹*closedin X W*› **show** *?rhs*
unfolding *closedin_def* **by** (*smt* (*verit*) *Diff_mono disjnt_Diff1 double_diff subset_eq*)
next
assume *?rhs* **with** ‹*openin X U*› **show** *?lhs*
unfolding *openin_closedin_eq disjnt_def*
by (*smt* (*verit*) *Diff_Diff_Int Diff_disjoint Diff_eq_empty_iff Int_Diff inf.orderE*)
qed
show *?thesis*
unfolding *regular_space_def*
by (*intro all_cong1 ex_cong1 imp_cong refl*) (*metis* § *DiffE*)
qed
finally show *?thesis* **.**
qed

lemma *regular_space_discrete_topology* [*simp*]:
regular_space (*discrete_topology S*)
using *neighbourhood_base_of_closedin neighbourhood_base_of_discrete_topology*
by *fastforce*

lemma *regular_space_subtopology*:
regular_space X ⟹ *regular_space* (*subtopology X S*)
unfolding *regular_space_def openin_subtopology_alt closedin_subtopology_alt disjnt_iff*
by *clarsimp* (*smt* (*verit*, *best*) *inf.orderE inf_le1 le_inf_iff*)

lemma *regular_space_retraction_map_image*:
⟦*retraction_map X Y r*; *regular_space X*⟧ ⟹ *regular_space Y*
using *hereditary_imp_retractive_property homeomorphic_regular_space regular_space_subtopology* **by** *blast*

lemma *regular_t0_imp_Hausdorff_space*:
⟦*regular_space X*; *t0_space X*⟧ ⟹ *Hausdorff_space X*
apply (*clarsimp simp*: *regular_space_def t0_space Hausdorff_space_def*)
by (*metis disjnt_sym subsetD*)

lemma *regular_t0_eq_Hausdorff_space*:
regular_space X ⟹ (*t0_space X* ⟷ *Hausdorff_space X*)
using *Hausdorff_imp_t0_space regular_t0_imp_Hausdorff_space* **by** *blast*

lemma *regular_t1_imp_Hausdorff_space*:
⟦*regular_space X*; *t1_space X*⟧ ⟹ *Hausdorff_space X*
by (*simp add*: *regular_t0_imp_Hausdorff_space t1_imp_t0_space*)

lemma *regular_t1_eq_Hausdorff_space*:
regular_space X ⟹ *t1_space X* ⟷ *Hausdorff_space X*

using *regular_t0_imp_Hausdorff_space t1_imp_t0_space t1_or_Hausdorff_space*
by *blast*

lemma *compact_Hausdorff_imp_regular_space*:
assumes *compact_space X Hausdorff_space X*
shows *regular_space X*
unfolding *regular_space_def*
proof *clarify*
fix *S a*
assume *closedin X S* **and** $a \in$ *topspace X* **and** $a \notin S$
then show $\exists U\ V.$ *openin X U* $\wedge$ *openin X V* $\wedge$ $a \in U \wedge S \subseteq V \wedge$ *disjnt U V*
using *assms* **unfolding** *Hausdorff_space_compact_sets*
by (*metis closedin_compact_space compactin_sing disjnt_empty1 insert_subset disjnt_insert1*)
qed

lemma *neighbourhood_base_of_closed_Hausdorff_space*:
regular_space X $\wedge$ *Hausdorff_space X* $\longleftrightarrow$
neighbourhood_base_of ($\lambda C.$ *closedin X C* $\wedge$ *Hausdorff_space*(*subtopology X C*)) *X* (**is** *?lhs=?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*simp add*: *Hausdorff_space_subtopology neighbourhood_base_of_closedin*)
next
assume *?rhs* **then show** *?lhs*
by (*metis* (*mono_tags*, *lifting*) *Hausdorff_space_closed_neighbourhood neighbourhood_base_of neighbourhood_base_of_closedin openin_topspace*)
qed

lemma *locally_compact_imp_kc_eq_Hausdorff_space*:
neighbourhood_base_of (*compactin X*) *X* $\Longrightarrow$ *kc_space X* $\longleftrightarrow$ *Hausdorff_space X*
by (*metis Hausdorff_imp_kc_space kc_imp_t1_space kc_space_def neighbourhood_base_of_closedin neighbourhood_base_of_mono regular_t1_imp_Hausdorff_space*)

lemma *regular_space_compact_closed_separation*:
assumes *X*: *regular_space X*
and *S*: *compactin X S*
and *T*: *closedin X T*
and *disjnt S T*
shows $\exists U\ V.$ *openin X U* $\wedge$ *openin X V* $\wedge$ $S \subseteq U \wedge T \subseteq V \wedge$ *disjnt U V*
proof (*cases S*={})
case *True*
then show *?thesis*
by (*meson T closedin_def disjnt_empty1 empty_subsetI openin_empty openin_topspace*)
next
case *False*
then have $\exists U\ V.\ x \in S \longrightarrow$ *openin X U* $\wedge$ *openin X V* $\wedge$ $x \in U \wedge T \subseteq V \wedge$ *disjnt U V* **for** *x*

```
    using assms unfolding regular_space_def
    by (smt (verit) Diff_iff compactin_subset_topspace disjnt_iff subsetD)
  then obtain U V where UV: ⋀x. x ∈ S ⟹ openin X (U x) ∧ openin X (V
x) ∧ x ∈ (U x) ∧ T ⊆ (V x) ∧ disjnt (U x) (V x)
    by metis
  then obtain ℱ where finite ℱ ℱ ⊆ U ‘ S S ⊆ ⋃ ℱ
    using S unfolding compactin_def by (smt (verit) UN_iff image_iff subsetI)
  then obtain K where finite K K ⊆ S and K: S ⊆ ⋃(U ‘ K)
    by (metis finite_subset_image)
  show ?thesis
  proof (intro exI conjI)
    show openin X (⋃(U ‘ K))
      using ‹K ⊆ S› UV by blast
    show openin X (⋂(V ‘ K))
      using False K UV ‹K ⊆ S› ‹finite K› by blast
    show S ⊆ ⋃(U ‘ K)
      by (simp add: K)
    show T ⊆ ⋂(V ‘ K)
      using UV ‹K ⊆ S› by blast
    show disjnt (⋃(U ‘ K)) (⋂(V ‘ K))
     by (smt (verit) Inter_iff UN_E UV ‹K ⊆ S› disjnt_iff image_eqI subset_iff)
  qed
qed

lemma regular_space_compact_closed_sets:
   regular_space X ⟷
        (∀ S T. compactin X S ∧ closedin X T ∧ disjnt S T
            ⟶ (∃ U V. openin X U ∧ openin X V ∧ S ⊆ U ∧ T ⊆ V ∧ disjnt U
V)) (is ?lhs=?rhs)
proof
  assume ?lhs then show ?rhs
    using regular_space_compact_closed_separation by fastforce
next
  assume R: ?rhs
  show ?lhs
    unfolding regular_space
  proof clarify
    fix S x
    assume closedin X S and x ∈ topspace X and x ∉ S
    then obtain U V where openin X U ∧ openin X V ∧ {x} ⊆ U ∧ S ⊆ V ∧
disjnt U V
      by (metis R compactin_sing disjnt_empty1 disjnt_insert1)
    then show ∃ U. openin X U ∧ x ∈ U ∧ disjnt S (X closure_of U)
      by (smt (verit, best) disjnt_iff in_closure_of insert_subset subsetD)
  qed
qed


lemma regular_space_prod_topology:
```

regular_space (prod_topology X Y) ⟷
X = trivial_topology ∨ Y = trivial_topology ∨ regular_space X ∧ regular_space Y (**is** *?lhs=?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (*metis regular_space_retraction_map_image retraction_map_fst retraction_map_snd*)
next
assume *R*: *?rhs*
show *?lhs*
proof (*cases X = trivial_topology ∨ Y = trivial_topology*)
case *True* **then show** *?thesis* **by** *auto*
next
case *False*
then have *regular_space X regular_space Y*
using *R* **by** *auto*
show *?thesis*
unfolding *neighbourhood_base_of_closedin* [*symmetric*] *neighbourhood_base_of*
proof *clarify*
fix *W x y*
assume *W*: *openin (prod_topology X Y) W* **and** *(x,y) ∈ W*
then obtain *U V* **where** *U*: *openin X U x ∈ U* **and** *V*: *openin Y V y ∈ V*
and *U × V ⊆ W*
by (*metis openin_prod_topology_alt*)
obtain *D1 C1* **where** *1*: *openin X D1 closedin X C1 x ∈ D1 D1 ⊆ C1 C1 ⊆ U*
by (*metis ‹regular_space X› U neighbourhood_base_of neighbourhood_base_of_closedin*)
obtain *D2 C2* **where** *2*: *openin Y D2 closedin Y C2 y ∈ D2 D2 ⊆ C2 C2 ⊆ V*
by (*metis ‹regular_space Y› V neighbourhood_base_of neighbourhood_base_of_closedin*)
show *∃ U V. openin (prod_topology X Y) U ∧ closedin (prod_topology X Y) V ∧*
(x,y) ∈ U ∧ U ⊆ V ∧ V ⊆ W
proof (*intro conjI exI*)
show *openin (prod_topology X Y) (D1 × D2)*
by (*simp add*: *1 2 openin_prod_Times_iff*)
show *closedin (prod_topology X Y) (C1 × C2)*
by (*simp add*: *1 2 closedin_prod_Times_iff*)
qed (*use 1 2 ‹U × V ⊆ W›* **in** *auto*)
qed
qed
qed

lemma *regular_space_product_topology*:
regular_space (product_topology X I) ⟷
(product_topology X I) = trivial_topology ∨ (∀ i ∈ I. regular_space (X i)) (**is** *?lhs=?rhs*)
proof

assume *?lhs*
then show *?rhs*
by (*meson regular_space_retraction_map_image retraction_map_product_projection*)
next
assume *R*: *?rhs*
show *?lhs*
proof (*cases product_topology X I = trivial_topology*)
case *True*
then show *?thesis*
by *auto*
next
case *False*
then obtain *x* **where** *x*: $x \in$ *topspace* (*product_topology X I*)
by (*meson ex_in_conv null_topspace_iff_trivial*)
define $\mathcal{F}$ **where** $\mathcal{F} \equiv \{Pi_E\ I\ U\ |U.$ *finite* $\{i \in I.\ U\ i \neq$ *topspace* $(X\ i)\}$
$\wedge\ (\forall\, i{\in}I.$ *openin* $(X\ i)\ (U\ i))\}$
have *oo*: *openin* (*product_topology X I*) = *arbitrary union_of* $(\lambda W.\ W \in \mathcal{F})$
by (*simp add*: $\mathcal{F}$*_def openin_product_topology product_topology_base_alt*)
have $\exists\, U\ V.$ *openin* (*product_topology X I*) $U \wedge$ *closedin* (*product_topology X I*) $V \wedge x \in U \wedge U \subseteq V \wedge V \subseteq Pi_E\ I\ W$
if *fin*: *finite* $\{i \in I.\ W\ i \neq$ *topspace* $(X\ i)\}$
and *opeW*: $\bigwedge k.\ k \in I \Longrightarrow$ *openin* $(X\ k)\ (W\ k)$ **and** *x*: $x \in$ *PiE I W* **for** *W x*
proof −
have $\bigwedge i.\ i \in I \Longrightarrow \exists\, U\ V.$ *openin* $(X\ i)\ U \wedge$ *closedin* $(X\ i)\ V \wedge x\ i \in U \wedge U \subseteq V \wedge V \subseteq W\ i$
by (*metis False PiE_iff R neighbourhood_base_of neighbourhood_base_of_closedin opeW x*)
then obtain *U C* **where** *UC*:
$\bigwedge i.\ i \in I \Longrightarrow$ *openin* $(X\ i)\ (U\ i) \wedge$ *closedin* $(X\ i)\ (C\ i) \wedge x\ i \in U\ i \wedge U\ i \subseteq C\ i \wedge C\ i \subseteq W\ i$
by *metis*
define *PI* **where** $PI \equiv \lambda V.$ *PiE I* $(\lambda i.$ *if* $W\ i =$ *topspace*$(X\ i)$ *then topspace*$(X\ i)$ *else* $V\ i)$
show *?thesis*
proof (*intro conjI exI*)
have $\forall\, i{\in}I.\ W\ i \neq$ *topspace* $(X\ i) \longrightarrow$ *openin* $(X\ i)\ (U\ i)$
using *UC* **by** *force*
with *fin* **show** *openin* (*product_topology X I*) (*PI U*)
by (*simp add*: *Collect_mono_iff PI_def openin_PiE_gen rev_finite_subset*)
show *closedin* (*product_topology X I*) (*PI C*)
by (*simp add*: *UC closedin_product_topology PI_def*)
show $x \in$ *PI U*
using *UC x* **by** (*fastforce simp*: *PI_def*)
show *PI U* $\subseteq$ *PI C*
by (*smt* (*verit*) *UC Orderings.order_eq_iff PiE_mono PI_def*)
show *PI C* $\subseteq Pi_E\ I\ W$
by (*simp add*: *UC PI_def subset_PiE*)
qed

```
    qed
  then have neighbourhood_base_of (closedin (product_topology X I)) (product_topology
X I)
      unfolding neighbourhood_base_of_topology_base [OF oo] by (force simp:
\mathcal{F}_def)
    then show ?thesis
      by (simp add: neighbourhood_base_of_closedin)
  qed
qed

lemma closed_map_paired_gen_aux:
  assumes regular_space Y and f: closed_map Z X f and g: closed_map Z Y g
    and clo: \bigwedgey. y ∈ topspace X ⟹ closedin Z {x ∈ topspace Z. f x = y}
    and contg: continuous_map Z Y g
  shows closed_map Z (prod_topology X Y) (λx. (f x, g x))
  unfolding closed_map_def
proof (intro strip)
  fix C assume closedin Z C
  then have C ⊆ topspace Z
    by (simp add: closedin_subset)
  have f ∈ topspace Z → topspace X g ∈ topspace Z → topspace Y
    by (simp_all add: assms closed_map_imp_subset_topspace)
  show closedin (prod_topology X Y) ((λx. (f x, g x)) ‘ C)
    unfolding closedin_def topspace_prod_topology
  proof (intro conjI)
    have closedin Y (g ‘ C)
      using ‹closedin Z C› assms(3) closed_map_def by blast
    with assms show (λx. (f x, g x)) ‘ C ⊆ topspace X × topspace Y
      by (smt (verit) SigmaI ‹closedin Z C› closed_map_def closedin_subset im-
age_subset_iff)
    have ∗: ∃T. openin (prod_topology X Y) T ∧ (y1,y2) ∈ T ∧ T ⊆ topspace X
× topspace Y − (λx. (f x, g x)) ‘ C
      if (y1,y2) ∉ (λx. (f x, g x)) ‘ C and y1: y1 ∈ topspace X and y2: y2 ∈
topspace Y
      for y1 y2
    proof −
      define A where A ≡ topspace Z − (C ∩ {x ∈ topspace Z. f x = y1})
      have A: openin Z A {x ∈ topspace Z. g x = y2} ⊆ A
        using that ‹closedin Z C› clo that(2) by (auto simp: A_def)
      obtain V0 where openin Y V0 ∧ y2 ∈ V0 and UA: {x ∈ topspace Z. g x
∈ V0} ⊆ A
        using g A y2 unfolding closed_map_fibre_neighbourhood by blast
      then obtain V V' where VV: openin Y V ∧ closedin Y V' ∧ y2 ∈ V ∧ V
⊆ V' and V' ⊆ V0
          by (metis (no_types, lifting) ‹regular_space Y› neighbourhood_base_of
neighbourhood_base_of_closedin)
      with UA have subA: {x ∈ topspace Z. g x ∈ V'} ⊆ A
        by blast
      show ?thesis
```

```
    proof −
      define B where B ≡ topspace Z − (C ∩ {x ∈ topspace Z. g x ∈ V'})
      have openin Z B
           using VV ‹closedin Z C› contg by (fastforce simp: B_def continuous_map_closedin)
      have {x ∈ topspace Z. f x = y1} ⊆ B
        using A_def subA by (auto simp: A_def B_def)
      then obtain U where openin X U y1 ∈ U and subB: {x ∈ topspace Z. f x ∈ U} ⊆ B
        using ‹openin Z B› y1 f unfolding closed_map_fibre_neighbourhood by meson
      show ?thesis
      proof (intro conjI exI)
        show openin (prod_topology X Y) (U × V)
          by (metis VV ‹openin X U› openin_prod_Times_iff)
        show (y1, y2) ∈ U × V
          by (simp add: VV ‹y1 ∈ U›)
        show U × V ⊆ topspace X × topspace Y − (λx. (f x, g x)) ‘ C
          using VV ‹C ⊆ topspace Z› ‹openin X U› subB
          by (force simp: image_iff B_def subset_iff dest: openin_subset)
      qed
    qed
  qed
  then show openin (prod_topology X Y) (topspace X × topspace Y − (λx. (f x, g x)) ‘ C)
    by (smt (verit, ccfv_threshold) Diff_iff SigmaE openin_subopen)
 qed
qed


lemma closed_map_paired_gen:
 assumes f: closed_map Z X f and g: closed_map Z Y g
 and D: (regular_space X ∧ continuous_map Z X f ∧ (∀ z ∈ topspace Y. closedin Z {x ∈ topspace Z. g x = z})
       ∨ regular_space Y ∧ continuous_map Z Y g ∧ (∀ y ∈ topspace X. closedin Z {x ∈ topspace Z. f x = y}))
        (is ?RSX ∨ ?RSY)
     shows closed_map Z (prod_topology X Y) (λx. (f x, g x))
 using D
proof
 assume RSX: ?RSX
 have eq: (λx. (f x, g x)) = (λ(x,y). (y,x)) ∘ (λx. (g x, f x))
   by auto
 show ?thesis
   unfolding eq
 proof (rule closed_map_compose)
   show closed_map Z (prod_topology Y X) (λx. (g x, f x))
     using RSX closed_map_paired_gen_aux f g by fastforce
   show closed_map (prod_topology Y X) (prod_topology X Y) (λ(x, y). (y, x))
```

using *homeomorphic_imp_closed_map homeomorphic_map_swap* **by** *blast*
qed
qed (*blast intro*: *assms closed_map_paired_gen_aux*)

lemma *closed_map_paired*:
assumes *closed_map Z X f* **and** *contf*: *continuous_map Z X f*
closed_map Z Y g **and** *contg*: *continuous_map Z Y g*
and *D*: *t1_space X* ∧ *regular_space Y* ∨ *regular_space X* ∧ *t1_space Y*
shows *closed_map Z* (*prod_topology X Y*) (λ*x*. (*f x*, *g x*))
proof (*rule closed_map_paired_gen*)
show *regular_space X* ∧ *continuous_map Z X f* ∧ (∀ *z*∈*topspace Y*. *closedin Z* {*x* ∈ *topspace Z*. *g x* = *z*}) ∨ *regular_space Y* ∧ *continuous_map Z Y g* ∧ (∀ *y*∈*topspace X*. *closedin Z* {*x* ∈ *topspace Z*. *f x* = *y*})
using *D contf contg*
by (*smt* (*verit*, *del_insts*) *Collect_cong closedin_continuous_map_preimage t1_space_closedin_singleton singleton_iff*)
qed (*use assms* **in** *auto*)

lemma *closed_map_pairwise*:
assumes *closed_map Z X* (*fst* ∘ *f*) *continuous_map Z X* (*fst* ∘ *f*)
closed_map Z Y (*snd* ∘ *f*) *continuous_map Z Y* (*snd* ∘ *f*)
t1_space X ∧ *regular_space Y* ∨ *regular_space X* ∧ *t1_space Y*
shows *closed_map Z* (*prod_topology X Y*) *f*
proof −
have *closed_map Z* (*prod_topology X Y*) (λ*a*. ((*fst* ∘ *f*) *a*, (*snd* ∘ *f*) *a*))
using *assms closed_map_paired* **by** *blast*
then show *?thesis*
by *auto*
qed

lemma *continuous_imp_proper_map*:
⟦*compact_space X*; *kc_space Y*; *continuous_map X Y f*⟧ ⟹ *proper_map X Y f*
by (*simp add*: *continuous_closed_imp_proper_map continuous_imp_closed_map_gen kc_imp_t1_space*)

lemma *tube_lemma_right*:
assumes *W*: *openin* (*prod_topology X Y*) *W* **and** *C*: *compactin Y C*
and *x*: *x* ∈ *topspace X* **and** *subW*: {*x*} × *C* ⊆ *W*
shows ∃ *U V*. *openin X U* ∧ *openin Y V* ∧ *x* ∈ *U* ∧ *C* ⊆ *V* ∧ *U* × *V* ⊆ *W*
proof (*cases C* = {})
case *True*
with *x* **show** *?thesis* **by** *auto*
next
case *False*
have ∃ *U V*. *openin X U* ∧ *openin Y V* ∧ *x* ∈ *U* ∧ *y* ∈ *V* ∧ *U* × *V* ⊆ *W*
if *y* ∈ *C* **for** *y*
using *W openin_prod_topology_alt subW subsetD that* **by** *fastforce*

```
  then obtain U V where UV: ⋀y. y ∈ C ⟹ openin X (U y) ∧ openin Y (V
y) ∧ x ∈ U y ∧ y ∈ V y ∧ U y × V y ⊆ W
    by metis
  then obtain D where D: finite D D ⊆ C C ⊆ ⋃ (V ‘ D)
    using compactinD [OF C, of V‘C]
    by (smt (verit) UN_I finite_subset_image imageE subsetI)
  show ?thesis
  proof (intro exI conjI)
    show openin X (⋂ (U ‘ D)) openin Y (⋃ (V ‘ D))
      using D False UV by blast+
    show x ∈ ⋂ (U ‘ D) C ⊆ ⋃ (V ‘ D) ⋂ (U ‘ D) × ⋃ (V ‘ D) ⊆ W
      using D UV by force+
  qed
qed


lemma closed_map_fst:
  assumes compact_space Y
  shows closed_map (prod_topology X Y) X fst
proof −
  have ∗: {x ∈ topspace X × topspace Y. fst x ∈ U} = U × topspace Y
    if U ⊆ topspace X for U
    using that by force
  have ∗∗: ⋀U y. ⟦openin (prod_topology X Y) U; y ∈ topspace X;
           {x ∈ topspace X × topspace Y. fst x = y} ⊆ U⟧
           ⟹ ∃ V. openin X V ∧ y ∈ V ∧ V × topspace Y ⊆ U
     using tube_lemma_right[of X Y _ topspace Y] assms by (fastforce simp:
compact_space_def)
  show ?thesis
    unfolding closed_map_fibre_neighbourhood
    by (force simp: ∗ openin_subset cong: conj_cong intro: ∗∗)
qed

lemma closed_map_snd:
  assumes compact_space X
  shows closed_map (prod_topology X Y) Y snd
proof −
  have snd = fst o prod.swap
    by force
  moreover have closed_map (prod_topology X Y) Y (fst o prod.swap)
  proof (rule closed_map_compose)
    show closed_map (prod_topology X Y) (prod_topology Y X) prod.swap
      by (metis (no_types, lifting) homeomorphic_imp_closed_map homeomor-
phic_map_eq homeomorphic_map_swap prod.swap_def split_beta)
    show closed_map (prod_topology Y X) Y fst
      by (simp add: closed_map_fst assms)
  qed
  ultimately show ?thesis
    by metis
```

qed

lemma *closed_map_paired_closed_map_right*:
⟦*closed_map X Y f*; *regular_space X*;
⋀*y*. *y* ∈ *topspace Y* ⟹ *closedin X* {*x* ∈ *topspace X*. *f x* = *y*}⟧
⟹ *closed_map X* (*prod_topology X Y*) (λ*x*. (*x*, *f x*))
by (*rule closed_map_paired_gen* [*OF closed_map_id*, *unfolded id_def*]) *auto*

lemma *closed_map_paired_closed_map_left*:
assumes *closed_map X Y f regular_space X*
⋀*y*. *y* ∈ *topspace Y* ⟹ *closedin X* {*x* ∈ *topspace X*. *f x* = *y*}
shows *closed_map X* (*prod_topology Y X*) (λ*x*. (*f x*, *x*))
proof −
have *eq*: (λ*x*. (*f x*, *x*)) = (λ(*x*,*y*). (*y*,*x*)) ∘ (λ*x*. (*x*, *f x*))
by *auto*
show *?thesis*
unfolding *eq*
proof (*rule closed_map_compose*)
show *closed_map X* (*prod_topology X Y*) (λ*x*. (*x*, *f x*))
by (*simp add*: *assms closed_map_paired_closed_map_right*)
show *closed_map* (*prod_topology X Y*) (*prod_topology Y X*) (λ(*x*, *y*). (*y*, *x*))
using *homeomorphic_imp_closed_map homeomorphic_map_swap* **by** *blast*
qed
qed

lemma *closed_map_imp_closed_graph*:
assumes *closed_map X Y f regular_space X*
⋀*y*. *y* ∈ *topspace Y* ⟹ *closedin X* {*x* ∈ *topspace X*. *f x* = *y*}
shows *closedin* (*prod_topology X Y*) ((λ*x*. (*x*, *f x*)) ‘ *topspace X*)
using *assms closed_map_def closed_map_paired_closed_map_right* **by** *blast*

lemma *proper_map_paired_closed_map_right*:
assumes *closed_map X Y f regular_space X*
⋀*y*. *y* ∈ *topspace Y* ⟹ *closedin X* {*x* ∈ *topspace X*. *f x* = *y*}
shows *proper_map X* (*prod_topology X Y*) (λ*x*. (*x*, *f x*))
by (*simp add*: *assms closed_injective_imp_proper_map inj_on_def closed_map_paired_closed_map_right*)

lemma *proper_map_paired_closed_map_left*:
assumes *closed_map X Y f regular_space X*
⋀*y*. *y* ∈ *topspace Y* ⟹ *closedin X* {*x* ∈ *topspace X*. *f x* = *y*}
shows *proper_map X* (*prod_topology Y X*) (λ*x*. (*f x*, *x*))
by (*simp add*: *assms closed_injective_imp_proper_map inj_on_def closed_map_paired_closed_map_left*)

proposition *regular_space_continuous_proper_map_image*:
assumes *regular_space X* **and** *contf*: *continuous_map X Y f* **and** *pmapf*:
proper_map X Y f
and *fim*: *f* ‘ (*topspace X*) = *topspace Y*
shows *regular_space Y*
unfolding *regular_space_def*

proof *clarify*
fix *C y*
assume *closedin Y C* **and** *y* ∈ *topspace Y* **and** *y* ∉ *C*
have *closed_map X Y f* (∀ *y* ∈ *topspace Y*. *compactin X* {*x* ∈ *topspace X*. *f x* = *y*})
using *pmapf proper_map_def* **by** *force+*
moreover have *closedin X* {*z* ∈ *topspace X*. *f z* ∈ *C*}
using ‹*closedin Y C*› *contf continuous_map_closedin* **by** *fastforce*
moreover have *disjnt* {*z* ∈ *topspace X*. *f z* = *y*} {*z* ∈ *topspace X*. *f z* ∈ *C*}
using ‹*y* ∉ *C*› *disjnt_iff* **by** *blast*
ultimately
obtain *U V* **where** *UV*: *openin X U openin X V* {*z* ∈ *topspace X*. *f z* = *y*} ⊆ *U* {*z* ∈ *topspace X*. *f z* ∈ *C*} ⊆ *V*
and *dUV*: *disjnt U V*
using ‹*y* ∈ *topspace Y*› ‹*regular_space X*› **unfolding** *regular_space_compact_closed_sets*
by *meson*

have ∗: ⋀*U T*. *openin X U* ∧ *T* ⊆ *topspace Y* ∧ {*x* ∈ *topspace X*. *f x* ∈ *T*} ⊆ *U* ⟶
(∃ *V*. *openin Y V* ∧ *T* ⊆ *V* ∧ {*x* ∈ *topspace X*. *f x* ∈ *V*} ⊆ *U*)
using ‹*closed_map X Y f*› **unfolding** *closed_map_preimage_neighbourhood* **by** *blast*
obtain *V1* **where** *V1*: *openin Y V1* ∧ *y* ∈ *V1* **and** *sub1*: {*x* ∈ *topspace X*. *f x* ∈ *V1*} ⊆ *U*
using ∗ [*of U* {*y*}] *UV* ‹*y* ∈ *topspace Y*› **by** *auto*
moreover
obtain *V2* **where** *openin Y V2* ∧ *C* ⊆ *V2* **and** *sub2*: {*x* ∈ *topspace X*. *f x* ∈ *V2*} ⊆ *V*
by (*smt* (*verit*, *ccfv_SIG*) ∗ *UV* ‹*closedin Y C*› *closedin_subset mem_Collect_eq subset_iff*)
moreover have *disjnt V1 V2*
proof −
have ⋀*x*. ⟦∀ *x*. *x* ∈ *U* ⟶ *x* ∉ *V*; *x* ∈ *V1*; *x* ∈ *V2*⟧ ⟹ *False*
by (*smt* (*verit*) *V1 fim image_iff mem_Collect_eq openin_subset sub1 sub2 subsetD*)
with *dUV* **show** *?thesis* **by** (*auto simp*: *disjnt_iff*)
qed
ultimately show ∃ *U V*. *openin Y U* ∧ *openin Y V* ∧ *y* ∈ *U* ∧ *C* ⊆ *V* ∧ *disjnt U V*
by *meson*
qed

lemma *regular_space_perfect_map_image*:
⟦*regular_space X*; *perfect_map X Y f*⟧ ⟹ *regular_space Y*
by (*meson perfect_map_def regular_space_continuous_proper_map_image*)

proposition *regular_space_perfect_map_image_eq*:
assumes *Hausdorff_space X* **and** *perf*: *perfect_map X Y f*
shows *regular_space X* ⟷ *regular_space Y* (**is** *?lhs*=*?rhs*)

proof
assume *?lhs*
then show *?rhs*
using *perf regular_space_perfect_map_image* **by** *blast*
next
assume *R*: *?rhs*
have *continuous_map X Y f proper_map X Y f* **and** *fim*: *f ' (topspace X) = topspace Y*
using *perf* **by** (*auto simp*: *perfect_map_def*)
then have *closed_map X Y f* **and** *preYf*: (∀ *y* ∈ *topspace Y*. *compactin X* {*x* ∈ *topspace X*. *f x* = *y*})
by (*simp_all add*: *proper_map_def*)
show *?lhs*
unfolding *regular_space_def*
proof *clarify*
fix *C x*
assume *closedin X C* **and** *x* ∈ *topspace X* **and** *x* ∉ *C*
obtain *U1 U2* **where** *openin X U1 openin X U2* {*x*} ⊆ *U1* **and** *disjnt U1 U2*
and *subV*: *C* ∩ {*z* ∈ *topspace X*. *f z* = *f x*} ⊆ *U2*
proof (*rule Hausdorff_space_compact_separation* [*of X* {*x*} *C* ∩ {*z* ∈ *topspace X*. *f z* = *f x*}, *OF* ‹*Hausdorff_space X*›])
show *compactin X* {*x*}
by (*simp add*: ‹*x* ∈ *topspace X*›)
show *compactin X* (*C* ∩ {*z* ∈ *topspace X*. *f z* = *f x*})
using ‹*closedin X C*› *fim* ‹*x* ∈ *topspace X*› *closed_Int_compactin preYf* **by** *fastforce*
show *disjnt* {*x*} (*C* ∩ {*z* ∈ *topspace X*. *f z* = *f x*})
using ‹*x* ∉ *C*› **by** *force*
qed
have *closedin Y* (*f ' (C − U2)*)
using ‹*closed_map X Y f*› ‹*closedin X C*› ‹*openin X U2*› *closed_map_def* **by** *blast*
moreover
have *f x* ∈ *topspace Y − f ' (C − U2)*
using ‹*closedin X C*› ‹*continuous_map X Y f*› ‹*x* ∈ *topspace X*› *closedin_subset continuous_map_def subV*
by (*fastforce simp*: *Pi_iff*)
ultimately
obtain *V1 V2* **where** *VV*: *openin Y V1 openin Y V2 f x* ∈ *V1*
and *subV2*: *f ' (C − U2)* ⊆ *V2* **and** *disjnt V1 V2*
by (*meson R regular_space_def*)
show ∃ *U U'*. *openin X U* ∧ *openin X U'* ∧ *x* ∈ *U* ∧ *C* ⊆ *U'* ∧ *disjnt U U'*
proof (*intro exI conjI*)
show *openin X* (*U1* ∩ {*x* ∈ *topspace X*. *f x* ∈ *V1*})
using *VV*(*1*) ‹*continuous_map X Y f*› ‹*openin X U1*› *continuous_map* **by** *fastforce*
show *openin X* (*U2* ∪ {*x* ∈ *topspace X*. *f x* ∈ *V2*})
using *VV*(*2*) ‹*continuous_map X Y f*› ‹*openin X U2*› *continuous_map* **by** *fastforce*

show $x \in U1 \cap \{x \in topspace\ X.\ f\ x \in V1\}$
using $VV(3)$ ‹$x \in topspace\ X$› ‹$\{x\} \subseteq U1$› **by** *auto*
show $C \subseteq U2 \cup \{x \in topspace\ X.\ f\ x \in V2\}$
using ‹*closedin X C*› *closedin_subset subV2* **by** *auto*
show *disjnt* $(U1 \cap \{x \in topspace\ X.\ f\ x \in V1\})$ $(U2 \cup \{x \in topspace\ X.\ f\ x \in V2\})$
using ‹*disjnt U1 U2*› ‹*disjnt V1 V2*› **by** (*auto simp*: *disjnt_iff*)
qed
qed
qed

5.6.13 Locally compact spaces

definition *locally_compact_space*
where *locally_compact_space X* ≡
$\forall x \in topspace\ X.\ \exists U\ K.\ openin\ X\ U \wedge compactin\ X\ K \wedge x \in U \wedge U \subseteq K$

lemma *homeomorphic_locally_compact_spaceD*:
assumes *X*: *locally_compact_space X* **and** *X homeomorphic_space Y*
shows *locally_compact_space Y*
proof −
obtain *f* **where** *hmf*: *homeomorphic_map X Y f*
using *assms homeomorphic_space* **by** *blast*
then have *eq*: *topspace Y* = *f* ‘ (*topspace X*)
by (*simp add*: *homeomorphic_imp_surjective_map*)
have $\exists V\ K.\ openin\ Y\ V \wedge compactin\ Y\ K \wedge f\ x \in V \wedge V \subseteq K$
if $x \in topspace\ X$ *openin X U compactin X K* $x \in U\ U \subseteq K$ **for** *x U K*
using *that*
by (*meson hmf homeomorphic_map_compactness_eq homeomorphic_map_openness_eq image_mono image_eqI*)
with *X* **show** *?thesis*
by (*smt* (*verit*) *eq image_iff locally_compact_space_def*)
qed

lemma *homeomorphic_locally_compact_space*:
assumes *X homeomorphic_space Y*
shows *locally_compact_space X* ⟷ *locally_compact_space Y*
by (*meson assms homeomorphic_locally_compact_spaceD homeomorphic_space_sym*)

lemma *locally_compact_space_retraction_map_image*:
assumes *retraction_map X Y r* **and** *X*: *locally_compact_space X*
shows *locally_compact_space Y*
proof −
obtain *s* **where** *s*: *retraction_maps X Y r s*
using *assms retraction_map_def* **by** *blast*
obtain *T* **where** *T retract_of_space X* **and** *Teq*: *T* = *s* ‘ *topspace Y*
using *retraction_maps_section_image1 s* **by** *blast*
then obtain *r* **where** *r*: *continuous_map X* (*subtopology X T*) *r* $\forall x \in T.\ r\ x = x$

by (*meson retract_of_space_def*)
have *locally_compact_space* (*subtopology X T*)
unfolding *locally_compact_space_def openin_subtopology_alt*
proof *clarsimp*
fix *x*
assume *x* ∈ *topspace X x* ∈ *T*
obtain *U K* **where** *UK*: *openin X U* ∧ *compactin X K* ∧ *x* ∈ *U* ∧ *U* ⊆ *K*
by (*meson X* ‹*x* ∈ *topspace X*› *locally_compact_space_def*)
then have *compactin* (*subtopology X T*) (*r* ‘ *K*) ∧ *T* ∩ *U* ⊆ *r* ‘ *K*
by (*smt* (*verit*) *IntD1 image_compactin image_iff inf_le2 r subset_iff*)
then show ∃ *U*. *openin X U* ∧ (∃ *K*. *compactin* (*subtopology X T*) *K* ∧ *x* ∈ *U* ∧ *T* ∩ *U* ⊆ *K*)
using *UK* **by** *auto*
qed
with *Teq* **show** *?thesis*
using *homeomorphic_locally_compact_space retraction_maps_section_image2 s* **by** *blast*
qed

lemma *compact_imp_locally_compact_space*:
compact_space X ⟹ *locally_compact_space X*
using *compact_space_def locally_compact_space_def* **by** *blast*

lemma *neighbourhood_base_imp_locally_compact_space*:
neighbourhood_base_of (*compactin X*) *X* ⟹ *locally_compact_space X*
by (*metis locally_compact_space_def neighbourhood_base_of openin_topspace*)

lemma *locally_compact_imp_neighbourhood_base*:
assumes *loc*: *locally_compact_space X* **and** *reg*: *regular_space X*
shows *neighbourhood_base_of* (*compactin X*) *X*
unfolding *neighbourhood_base_of*
proof *clarify*
fix *W x*
assume *openin X W* **and** *x* ∈ *W*
then obtain *U K* **where** *openin X U compactin X K x* ∈ *U U* ⊆ *K*
by (*metis loc locally_compact_space_def openin_subset subsetD*)
moreover have *openin X* (*U* ∩ *W*) ∧ *x* ∈ *U* ∩ *W*
using ‹*openin X W*› ‹*x* ∈ *W*› ‹*openin X U*› ‹*x* ∈ *U*› **by** *blast*
then have ∃ *u′ v*. *openin X u′* ∧ *closedin X v* ∧ *x* ∈ *u′* ∧ *u′* ⊆ *v* ∧ *v* ⊆ *U* ∧ *v* ⊆ *W*
using *reg*
by (*metis le_infE neighbourhood_base_of neighbourhood_base_of_closedin*)
then show ∃ *U V*. *openin X U* ∧ *compactin X V* ∧ *x* ∈ *U* ∧ *U* ⊆ *V* ∧ *V* ⊆ *W*
by (*meson* ‹*U* ⊆ *K*› ‹*compactin X K*› *closed_compactin subset_trans*)
qed

lemma *Hausdorff_regular*: ⟦*Hausdorff_space X*; *neighbourhood_base_of* (*compactin X*) *X*⟧ ⟹ *regular_space X*
by (*metis compactin_imp_closedin neighbourhood_base_of_closedin neighbour-*

hood_base_of_mono)

lemma *locally_compact_Hausdorff_imp_regular_space*:
assumes *loc*: *locally_compact_space X* **and** *Hausdorff_space X*
shows *regular_space X*
unfolding *neighbourhood_base_of_closedin* [*symmetric*] *neighbourhood_base_of*
proof *clarify*
fix *W x*
assume *openin X W* **and** $x \in W$
then have $x \in$ *topspace X*
using *openin_subset* **by** *blast*
then obtain *U K* **where** *openin X U compactin X K* **and** *UK*: $x \in U$ $U \subseteq K$
by (*meson loc locally_compact_space_def*)
with ‹*Hausdorff_space X*› **have** *regular_space* (*subtopology X K*)
using *Hausdorff_space_subtopology compact_Hausdorff_imp_regular_space compact_space_subtopology* **by** *blast*
then have $\exists U'\ V'.$ *openin* (*subtopology X K*) $U' \wedge$ *closedin* (*subtopology X K*) $V' \wedge x \in U' \wedge U' \subseteq V' \wedge V' \subseteq K \cap W$
unfolding *neighbourhood_base_of_closedin* [*symmetric*] *neighbourhood_base_of*
by (*meson IntI* ‹$U \subseteq K$› ‹*openin X W*› ‹$x \in U$› ‹$x \in W$› *openin_subtopology_Int2 subsetD*)
then obtain *V C* **where** *openin X V closedin X C* **and** *VC*: $x \in K \cap V$ $K \cap V \subseteq K \cap C$ $K \cap C \subseteq K \cap W$
by (*metis Int_commute closedin_subtopology openin_subtopology*)
show $\exists U\ V.$ *openin X U* $\wedge$ *closedin X V* $\wedge$ $x \in U \wedge U \subseteq V \wedge V \subseteq W$
proof (*intro conjI exI*)
show *openin X* (*U* $\cap$ *V*)
using ‹*openin X U*› ‹*openin X V*› **by** *blast*
show *closedin X* (*K* $\cap$ *C*)
using ‹*closedin X C*› ‹*compactin X K*› *compactin_imp_closedin* ‹*Hausdorff_space X*› **by** *blast*
qed (*use UK VC* **in** *auto*)
qed

lemma *locally_compact_space_neighbourhood_base*:
Hausdorff_space X $\vee$ *regular_space X*
$\Longrightarrow$ *locally_compact_space X* $\longleftrightarrow$ *neighbourhood_base_of* (*compactin X*) *X*
by (*metis locally_compact_imp_neighbourhood_base locally_compact_Hausdorff_imp_regular_space neighbourhood_base_imp_locally_compact_space*)

lemma *locally_compact_Hausdorff_or_regular*:
locally_compact_space X $\wedge$ (*Hausdorff_space X* $\vee$ *regular_space X*) $\longleftrightarrow$ *locally_compact_space X* $\wedge$ *regular_space X*
using *locally_compact_Hausdorff_imp_regular_space* **by** *blast*

lemma *locally_compact_space_compact_closedin*:
assumes *Hausdorff_space X* $\vee$ *regular_space X*

shows *locally_compact_space X* ⟷
(∀ *x* ∈ *topspace X*. ∃ *U K*. *openin X U* ∧ *compactin X K* ∧ *closedin X K* ∧ *x* ∈ *U* ∧ *U* ⊆ *K*)
using *locally_compact_Hausdorff_or_regular* **unfolding** *locally_compact_space_def*
by (*metis assms closed_compactin inf.absorb_iff2 le_infE neighbourhood_base_of neighbourhood_base_of_closedin*)

lemma *locally_compact_space_compact_closure_of*:
assumes *Hausdorff_space X* ∨ *regular_space X*
shows *locally_compact_space X* ⟷
(∀ *x* ∈ *topspace X*. ∃ *U*. *openin X U* ∧ *compactin X* (*X closure_of U*) ∧ *x* ∈ *U*) (**is** *?lhs*=*?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*metis assms closed_compactin closedin_closure_of closure_of_eq closure_of_mono locally_compact_space_compact_closedin*)
next
assume *?rhs* **then show** *?lhs*
by (*meson closure_of_subset locally_compact_space_def openin_subset*)
qed

lemma *locally_compact_space_neighbourhood_base_closedin*:
assumes *Hausdorff_space X* ∨ *regular_space X*
shows *locally_compact_space X* ⟷ *neighbourhood_base_of* (λ*C*. *compactin X C* ∧ *closedin X C*) *X* (**is** *?lhs*=*?rhs*)
proof
assume *L*: *?lhs*
then have *regular_space X*
using *assms locally_compact_Hausdorff_imp_regular_space* **by** *blast*
with *L* **have** *neighbourhood_base_of* (*compactin X*) *X*
by (*simp add*: *locally_compact_imp_neighbourhood_base*)
with ‹*regular_space X*› **show** *?rhs*
by (*smt* (*verit*, *ccfv_threshold*) *closed_compactin neighbourhood_base_of subset_trans neighbourhood_base_of_closedin*)
next
assume *?rhs* **then show** *?lhs*
using *neighbourhood_base_imp_locally_compact_space neighbourhood_base_of_mono*
by *blast*
qed

lemma *locally_compact_space_neighbourhood_base_closure_of*:
assumes *Hausdorff_space X* ∨ *regular_space X*
shows *locally_compact_space X* ⟷ *neighbourhood_base_of* (λ*T*. *compactin X* (*X closure_of T*)) *X*
(**is** *?lhs*=*?rhs*)
proof
assume *L*: *?lhs*
then have *regular_space X*
using *assms locally_compact_Hausdorff_imp_regular_space* **by** *blast*

with *L* **have** *neighbourhood_base_of* (λ*A*. *compactin X A* ∧ *closedin X A*) *X*
using *locally_compact_space_neighbourhood_base_closedin* **by** *blast*
then show *?rhs*
by (*simp add*: *closure_of_closedin neighbourhood_base_of_mono*)
next
assume *?rhs* **then show** *?lhs*
unfolding *locally_compact_space_def neighbourhood_base_of*
by (*meson closure_of_subset openin_topspace subset_trans*)
qed

lemma *locally_compact_space_neighbourhood_base_open_closure_of*:
assumes *Hausdorff_space X* ∨ *regular_space X*
shows *locally_compact_space X* ⟷
neighbourhood_base_of (λ*U*. *openin X U* ∧ *compactin X* (*X closure_of U*)) *X*
(**is** *?lhs*=*?rhs*)
proof
assume *L*: *?lhs*
then have *regular_space X*
using *assms locally_compact_Hausdorff_imp_regular_space* **by** *blast*
then have *neighbourhood_base_of* (λ*T*. *compactin X* (*X closure_of T*)) *X*
using *L locally_compact_space_neighbourhood_base_closure_of* **by** *auto*
with *L* **show** *?rhs*
unfolding *neighbourhood_base_of*
by (*meson closed_compactin closure_of_closure_of closure_of_eq closure_of_mono subset_trans*)
next
assume *?rhs* **then show** *?lhs*
unfolding *locally_compact_space_def neighbourhood_base_of*
by (*meson closure_of_subset openin_topspace subset_trans*)
qed

lemma *locally_compact_space_compact_closed_compact*:
assumes *Hausdorff_space X* ∨ *regular_space X*
shows *locally_compact_space X* ⟷
(∀ *K*. *compactin X K*
⟶ (∃ *U L*. *openin X U* ∧ *compactin X L* ∧ *closedin X L* ∧ *K* ⊆ *U* ∧ *U* ⊆ *L*))
(**is** *?lhs*=*?rhs*)
proof
assume *L*: *?lhs*
then obtain *U L* **where** *UL*: ∀ *x* ∈ *topspace X*. *openin X* (*U x*) ∧ *compactin X* (*L x*) ∧ *closedin X* (*L x*) ∧ *x* ∈ *U x* ∧ *U x* ⊆ *L x*
unfolding *locally_compact_space_compact_closedin* [*OF assms*]
by *metis*
show *?rhs*
proof *clarify*
fix *K*
assume *compactin X K*

then have $K \subseteq$ *topspace* X
by (*simp add*: *compactin_subset_topspace*)
then have $*$: $(\forall U{\in}U\ `\ K.\ \mathit{openin}\ X\ U) \wedge K \subseteq \bigcup\ (U\ `\ K)$
using *UL* **by** *blast*
with ‹*compactin* X K› **obtain** *KF* **where** *KF*: *finite KF* $KF \subseteq K$ $K \subseteq \bigcup(U\ `\ KF)$
by (*metis compactinD finite_subset_image*)
show $\exists U\ L.$ *openin* $X\ U \wedge$ *compactin* $X\ L \wedge$ *closedin* $X\ L \wedge K \subseteq U \wedge U \subseteq L$
proof (*intro conjI exI*)
show *openin* X $(\bigcup\ (U\ `\ KF))$
using $*$ ‹$KF \subseteq K$› **by** *fastforce*
show *compactin* X $(\bigcup\ (L\ `\ KF))$
by (*smt* (*verit*) *UL* ‹$K \subseteq$ *topspace* X› *KF compactin_Union finite_imageI imageE subset_iff*)
show *closedin* X $(\bigcup\ (L\ `\ KF))$
by (*smt* (*verit*) *UL* ‹$K \subseteq$ *topspace* X› *KF closedin_Union finite_imageI imageE subsetD*)
qed (*use UL* ‹$K \subseteq$ *topspace* X› *KF* **in** *auto*)
qed
next
assume *?rhs* **then show** *?lhs*
by (*metis compactin_sing insert_subset locally_compact_space_def*)
qed

lemma *locally_compact_regular_space_neighbourhood_base*:
locally_compact_space $X \wedge$ *regular_space* $X \longleftrightarrow$
neighbourhood_base_of $(\lambda C.$ *compactin* $X\ C \wedge$ *closedin* $X\ C)\ X$
using *locally_compact_space_neighbourhood_base_closedin neighbourhood_base_of_closedin neighbourhood_base_of_mono* **by** *blast*

lemma *locally_compact_kc_space*:
neighbourhood_base_of (*compactin* X) $X \wedge$ *kc_space* $X \longleftrightarrow$
locally_compact_space $X \wedge$ *Hausdorff_space* X
using *Hausdorff_imp_kc_space locally_compact_imp_kc_eq_Hausdorff_space locally_compact_space_neighbourhood_base* **by** *blast*

lemma *locally_compact_kc_space_alt*:
neighbourhood_base_of (*compactin* X) $X \wedge$ *kc_space* $X \longleftrightarrow$
locally_compact_space $X \wedge$ *Hausdorff_space* $X \wedge$ *regular_space* X
using *Hausdorff_regular locally_compact_kc_space* **by** *blast*

lemma *locally_compact_kc_imp_regular_space*:
$[\![$*neighbourhood_base_of* (*compactin* X) X; *kc_space* $X]\!] \Longrightarrow$ *regular_space* X
using *Hausdorff_regular locally_compact_imp_kc_eq_Hausdorff_space* **by** *blast*

lemma *kc_locally_compact_space*:
kc_space X
$\Longrightarrow$ *neighbourhood_base_of* (*compactin* X) $X \longleftrightarrow$ *locally_compact_space* $X \wedge$ *Hausdorff_space* $X \wedge$ *regular_space* X

using *Hausdorff_regular locally_compact_kc_space* **by** *blast*

lemma *locally_compact_space_closed_subset*:
assumes *loc*: *locally_compact_space X* **and** *closedin X S*
shows *locally_compact_space* (*subtopology X S*)
proof (*clarsimp simp*: *locally_compact_space_def*)
fix *x* **assume** *x*: $x \in$ *topspace X* $x \in S$
then obtain *U K* **where** *UK*: *openin X U* $\wedge$ *compactin X K* $\wedge$ $x \in U \wedge U \subseteq K$
by (*meson loc locally_compact_space_def*)
show $\exists U.$ *openin* (*subtopology X S*) *U* $\wedge$
$(\exists K.$ *compactin* (*subtopology X S*) *K* $\wedge$ $x \in U \wedge U \subseteq K)$
proof (*intro conjI exI*)
show *openin* (*subtopology X S*) ($S \cap U$)
by (*simp add*: *UK openin_subtopology_Int2*)
show *compactin* (*subtopology X S*) ($S \cap K$)
by (*simp add*: *UK assms*(*2*) *closed_Int_compactin compactin_subtopology*)
qed (*use UK x* **in** *auto*)
qed

lemma *locally_compact_space_open_subset*:
assumes *X*: *Hausdorff_space X* $\vee$ *regular_space X* **and** *loc*: *locally_compact_space X* **and** *openin X S*
shows *locally_compact_space* (*subtopology X S*)
proof (*clarsimp simp*: *locally_compact_space_def*)
fix *x* **assume** *x*: $x \in$ *topspace X* $x \in S$
then obtain *U K* **where** *UK*: *openin X U compactin X K* $x \in U$ $U \subseteq K$
by (*meson loc locally_compact_space_def*)
moreover have *reg*: *regular_space X*
using *X loc locally_compact_Hausdorff_imp_regular_space* **by** *blast*
moreover have *openin X* ($U \cap S$)
by (*simp add*: *UK* ‹*openin X S*› *openin_Int*)
ultimately obtain *V C*
where *VC*: *openin X V closedin X C* $x \in V$ $V \subseteq C$ $C \subseteq U$ $C \subseteq S$
by (*metis* ‹$x \in S$› *IntI le_inf_iff neighbourhood_base_of neighbourhood_base_of_closedin*)
show $\exists U.$ *openin* (*subtopology X S*) *U* $\wedge$
$(\exists K.$ *compactin* (*subtopology X S*) *K* $\wedge$ $x \in U \wedge U \subseteq K)$
proof (*intro conjI exI*)
show *openin* (*subtopology X S*) *V*
using *VC* **by** (*meson* ‹*openin X S*› *openin_open_subtopology order_trans*)
show *compactin* (*subtopology X S*) ($C \cap K$)
using *UK VC closed_Int_compactin compactin_subtopology* **by** *fastforce*
qed (*use UK VC x* **in** *auto*)
qed

lemma *locally_compact_space_discrete_topology*:
locally_compact_space (*discrete_topology U*)
by (*simp add*: *neighbourhood_base_imp_locally_compact_space neighbourhood_base_of_discrete_topo*

lemma *locally_compact_space_continuous_open_map_image*:
⟦*continuous_map X X' f*; *open_map X X' f*;
f ‘ topspace X = topspace X'; *locally_compact_space X*⟧ ⟹ *locally_compact_space X'*
unfolding *locally_compact_space_def open_map_def*
by (*smt* (*verit*, *ccfv_SIG*) *image_compactin image_iff image_mono*)

lemma *locally_compact_subspace_openin_closure_of*:
assumes *Hausdorff_space X* **and** *S*: *S* ⊆ *topspace X*
and *loc*: *locally_compact_space* (*subtopology X S*)
shows *openin* (*subtopology X* (*X closure_of S*)) *S*
unfolding *openin_subopen* [**where** *S*=*S*]
proof *clarify*
fix *a* **assume** *a* ∈ *S*
then obtain *T K* **where** ∗: *openin X T compactin X K K* ⊆ *S a* ∈ *S a* ∈ *T S* ∩ *T* ⊆ *K*
using *loc* **unfolding** *locally_compact_space_def*
by (*metis IntE S compactin_subtopology inf_commute openin_subtopology topspace_subtopology_subset*)
have *T* ∩ *X closure_of S* ⊆ *X closure_of* (*T* ∩ *S*)
by (*simp add*: ∗(*1*) *openin_Int_closure_of_subset*)
also have ... ⊆ *S*
using ∗ ‹*Hausdorff_space X*› **by** (*metis closure_of_minimal compactin_imp_closedin order.trans inf_commute*)
finally have *T* ∩ *X closure_of S* ⊆ *T* ∩ *S* **by** *simp*
then have *openin* (*subtopology X* (*X closure_of S*)) (*T* ∩ *S*)
unfolding *openin_subtopology* **using** ‹*openin X T*› *S closure_of_subset* **by** *fastforce*
with ∗ **show** ∃ *T*. *openin* (*subtopology X* (*X closure_of S*)) *T* ∧ *a* ∈ *T* ∧ *T* ⊆ *S*
by *blast*
qed

lemma *locally_compact_subspace_closed_Int_openin*:
⟦*Hausdorff_space X* ∧ *S* ⊆ *topspace X* ∧ *locally_compact_space*(*subtopology X S*)⟧
⟹ ∃ *C U*. *closedin X C* ∧ *openin X U* ∧ *C* ∩ *U* = *S*
by (*metis closedin_closure_of inf_commute locally_compact_subspace_openin_closure_of openin_subtopology*)

lemma *locally_compact_subspace_open_in_closure_of_eq*:
assumes *Hausdorff_space X* **and** *loc*: *locally_compact_space X*
shows *openin* (*subtopology X* (*X closure_of S*)) *S* ⟷ *S* ⊆ *topspace X* ∧ *locally_compact_space*(*subtopology X S*) (**is** *?lhs*=*?rhs*)
proof
assume *L*: *?lhs*
then obtain *S* ⊆ *topspace X regular_space X*
using *assms locally_compact_Hausdorff_imp_regular_space openin_subset* **by** *fastforce*
then have *locally_compact_space* (*subtopology* (*subtopology X* (*X closure_of S*))

S)
by (*simp add*: *L loc locally_compact_space_closed_subset locally_compact_space_open_subset regular_space_subtopology*)
then show *?rhs*
by (*metis L inf.orderE inf_commute le_inf_iff openin_subset subtopology_subtopology topspace_subtopology*)
next
assume *?rhs* **then show** *?lhs*
using *assms locally_compact_subspace_openin_closure_of* **by** *blast*
qed

lemma *locally_compact_subspace_closed_Int_openin_eq*:
assumes *Hausdorff_space X* **and** *loc*: *locally_compact_space X*
shows (∃ *C U*. *closedin X C* ∧ *openin X U* ∧ *C* ∩ *U* = *S*) ⟷ *S* ⊆ *topspace X* ∧ *locally_compact_space*(*subtopology X S*) (**is** *?lhs*=*?rhs*)
proof
assume *L*: *?lhs*
then obtain *C U* **where** *closedin X C openin X U* **and** *Seq*: *S* = *C* ∩ *U*
by *blast*
then have *C* ⊆ *topspace X*
by (*simp add*: *closedin_subset*)
have *locally_compact_space* (*subtopology* (*subtopology X C*) (*topspace* (*subtopology X C*) ∩ *U*))
proof (*rule locally_compact_space_open_subset*)
show *locally_compact_space* (*subtopology X C*)
by (*simp add*: ‹*closedin X C*› *loc locally_compact_space_closed_subset*)
show *openin* (*subtopology X C*) (*topspace* (*subtopology X C*) ∩ *U*)
by (*simp add*: ‹*openin X U*› *Int_left_commute inf_commute openin_Int openin_subtopology_Int2*)
qed (*simp add*: *Hausdorff_space_subtopology* ‹*Hausdorff_space X*›)
then show *?rhs*
by (*metis Seq* ‹*C* ⊆ *topspace X*› *inf.coboundedI1 subtopology_subtopology subtopology_topspace*)
next
assume *?rhs* **then show** *?lhs*
using *assms locally_compact_subspace_closed_Int_openin* **by** *blast*
qed

lemma *dense_locally_compact_openin_Hausdorff_space*:
⟦*Hausdorff_space X*; *S* ⊆ *topspace X*; *X closure_of S* = *topspace X*;
locally_compact_space (*subtopology X S*)⟧ ⟹ *openin X S*
by (*metis locally_compact_subspace_openin_closure_of subtopology_topspace*)

lemma *locally_compact_space_prod_topology*:
locally_compact_space (*prod_topology X Y*) ⟷
(*prod_topology X Y*) = *trivial_topology* ∨
locally_compact_space X ∧ *locally_compact_space Y* (**is** *?lhs*=*?rhs*)
proof (*cases* (*prod_topology X Y*) = *trivial_topology*)
case *True*

```
  then show ?thesis
    using locally_compact_space_discrete_topology by force
next
  case False
  then obtain w z where wz: w ∈ topspace X z ∈ topspace Y
    by fastforce
  show ?thesis
  proof
    assume L: ?lhs then show ?rhs
    by (metis locally_compact_space_retraction_map_image prod_topology_trivial_iff
retraction_map_fst retraction_map_snd)
  next
    assume R: ?rhs
    show ?lhs
      unfolding locally_compact_space_def
    proof clarsimp
      fix x y
      assume x ∈ topspace X and y ∈ topspace Y
      obtain U C where openin X U compactin X C x ∈ U U ⊆ C
        by (meson False R ‹x ∈ topspace X› locally_compact_space_def)
      obtain V D where openin Y V compactin Y D y ∈ V V ⊆ D
        by (meson False R ‹y ∈ topspace Y› locally_compact_space_def)
      show ∃ U. openin (prod_topology X Y) U ∧ (∃ K. compactin (prod_topology
X Y) K ∧ (x, y) ∈ U ∧ U ⊆ K)
      proof (intro exI conjI)
        show openin (prod_topology X Y) (U × V)
          by (simp add: ‹openin X U› ‹openin Y V› openin_prod_Times_iff)
        show compactin (prod_topology X Y) (C × D)
          by (simp add: ‹compactin X C› ‹compactin Y D› compactin_Times)
        show (x, y) ∈ U × V
          by (simp add: ‹x ∈ U› ‹y ∈ V›)
        show U × V ⊆ C × D
          by (simp add: Sigma_mono ‹U ⊆ C› ‹V ⊆ D›)
      qed
    qed
  qed
qed

lemma locally_compact_space_product_topology:
   locally_compact_space(product_topology X I) ⟷
        product_topology X I = trivial_topology ∨
        finite {i ∈ I. ¬ compact_space(X i)} ∧ (∀ i ∈ I. locally_compact_space(X
i)) (is ?lhs=?rhs)
proof (cases (product_topology X I) = trivial_topology)
  case True
  then show ?thesis
    by (simp add: locally_compact_space_def)
next
  case False
```

show *?thesis*
proof
assume *L*: *?lhs*
obtain *z* **where** *z*: $z \in$ *topspace* (*product_topology X I*)
using *False*
by (*meson ex_in_conv null_topspace_iff_trivial*)
with *L z* **obtain** *U C* **where** *openin* (*product_topology X I*) *U compactin* (*product_topology X I*) *C z* $\in$ *U U* $\subseteq$ *C*
by (*meson locally_compact_space_def*)
then obtain *V* **where** *finV*: *finite* $\{i \in I.\ V\ i \neq$ *topspace* $(X\ i)\}$ **and** $\forall\, i \in I.$ *openin* $(X\ i)$ $(V\ i)$
and *z* $\in$ *PiE I V PiE I V* $\subseteq$ *U*
by (*auto simp*: *openin_product_topology_alt*)
have *compact_space* $(X\ i)$ **if** $i \in I$ $V\ i =$ *topspace* $(X\ i)$ **for** *i*
proof −
have *compactin* $(X\ i)$ $((\lambda x.\ x\ i)$ ‘ $C)$
using ‹*compactin* (*product_topology X I*) *C*› *image_compactin*
by (*metis continuous_map_product_projection* ‹*i* $\in$ *I*›)
moreover have *V i* $\subseteq$ $(\lambda x.\ x\ i)$ ‘ *C*
proof −
have *V i* $\subseteq$ $(\lambda x.\ x\ i)$ ‘ Pi_E *I V*
by (*metis* ‹*z* $\in$ Pi_E *I V*› *empty_iff image_projection_PiE order_refl* ‹*i* $\in$ *I*›)
also have ... $\subseteq$ $(\lambda x.\ x\ i)$ ‘ *C*
using ‹*U* $\subseteq$ *C*› ‹Pi_E *I V* $\subseteq$ *U*› **by** *blast*
finally show *?thesis* **.**
qed
ultimately show *?thesis*
by (*metis closed_compactin closedin_topspace compact_space_def that*(*2*))
qed
with *finV* **have** *finite* $\{i \in I.\ \neg$ *compact_space* $(X\ i)\}$
by (*metis* (*mono_tags*, *lifting*) *mem_Collect_eq finite_subset subsetI*)
moreover have *locally_compact_space* $(X\ i)$ **if** $i \in I$ **for** *i*
by (*meson False L locally_compact_space_retraction_map_image retraction_map_product_projection that*)
ultimately show *?rhs* **by** *metis*
next
assume *R*: *?rhs*
show *?lhs*
unfolding *locally_compact_space_def*
proof *clarsimp*
fix *z*
assume *z*: $z \in (\Pi_E\ i \in I.$ *topspace* $(X\ i))$
have $\exists\, U\ C.$ *openin* $(X\ i)$ *U* $\wedge$ *compactin* $(X\ i)$ *C* $\wedge$ *z i* $\in$ *U* $\wedge$ *U* $\subseteq$ *C* $\wedge$
$(compact\_space(X\ i) \longrightarrow U = topspace(X\ i) \wedge C = topspace(X\ i))$
if $i \in I$ **for** *i*
using *that R z* **unfolding** *locally_compact_space_def compact_space_def*
by (*metis* (*no_types*, *lifting*) *False PiE_mem openin_topspace set_eq_subset*)

```
    then obtain U C where UC: ⋀i. i ∈ I ⟹
        openin (X i) (U i) ∧ compactin (X i) (C i) ∧ z i ∈ U i ∧ U i ⊆ C i ∧
          (compact_space(X i) ⟶ U i = topspace(X i) ∧ C i = topspace(X
i))
      by metis
   show ∃U. openin (product_topology X I) U ∧ (∃K. compactin (product_topology
X I) K ∧ z ∈ U ∧ U ⊆ K)
    proof (intro exI conjI)
      show openin (product_topology X I) (Pi_E I U)
     by (smt (verit) Collect_cong False R UC compactin_subspace openin_PiE_gen
subset_antisym subtopology_topspace)
      show compactin (product_topology X I) (Pi_E I C)
       by (simp add: UC compactin_PiE)
     qed (use UC z in blast)+
   qed
  qed
qed

lemma locally_compact_space_sum_topology:
  locally_compact_space (sum_topology X I) ⟷ (∀i∈I. locally_compact_space
(X i)) (is ?lhs=?rhs)
proof
  assume ?lhs then show ?rhs
   by (metis closed_map_component_injection embedding_map_imp_homeomorphic_space
embedding_map_component_injection
      embedding_imp_closed_map_eq homeomorphic_locally_compact_space lo-
cally_compact_space_closed_subset)
next
  assume R: ?rhs
  show ?lhs
    unfolding locally_compact_space_def
  proof clarsimp
    fix i y
    assume i ∈ I and y: y ∈ topspace (X i)
    then obtain U K where UK: openin (X i) U compactin (X i) K y ∈ U U ⊆
K
      using R by (fastforce simp: locally_compact_space_def)
  then show ∃U. openin (sum_topology X I) U ∧ (∃K. compactin (sum_topology
X I) K ∧ (i, y) ∈ U ∧ U ⊆ K)
      by (metis ‹i ∈ I› continuous_map_component_injection image_compactin
image_mono
        imageI open_map_component_injection open_map_def)
  qed
qed

lemma locally_compact_space_euclidean:
  locally_compact_space (euclidean::'a::heine_borel topology)
  unfolding locally_compact_space_def
proof (intro strip)
```

```
  fix x::'a
  assume x ∈ topspace euclidean
  have ball x 1 ⊆ cball x 1
    by auto
  then show ∃ U K. openin euclidean U ∧ compactin euclidean K ∧ x ∈ U ∧ U
⊆ K
    by (metis Elementary_Metric_Spaces.open_ball centre_in_ball compact_cball
compactin_euclidean_iff open_openin zero_less_one)
qed

lemma locally_compact_Euclidean_space:
  locally_compact_space(Euclidean_space n)
 using homeomorphic_locally_compact_space [OF homeomorphic_Euclidean_space_product_topology]

 using locally_compact_space_product_topology locally_compact_space_euclidean
by fastforce

proposition quotient_map_prod_right:
  assumes loc: locally_compact_space Z
    and reg: Hausdorff_space Z ∨ regular_space Z
    and f: quotient_map X Y f
  shows quotient_map (prod_topology Z X) (prod_topology Z Y) (λ(x,y). (x,f y))
proof −
  define h where h ≡ (λ(x::'a,y). (x,f y))
  have continuous_map (prod_topology Z X) Y (f o snd)
    by (simp add: continuous_map_of_snd f quotient_imp_continuous_map)
  then have cmh: continuous_map (prod_topology Z X) (prod_topology Z Y) h
     by (simp add: h_def continuous_map_paired split_def continuous_map_fst
o_def)
  have fim: f ‘ topspace X = topspace Y
    by (simp add: f quotient_imp_surjective_map)
  moreover
  have openin (prod_topology Z X) {u ∈ topspace Z × topspace X. h u ∈ W}
   ⟷ openin (prod_topology Z Y) W   (is ?lhs=?rhs)
    if W: W ⊆ topspace Z × topspace Y for W
  proof
    define S where S ≡ {u ∈ topspace Z × topspace X. h u ∈ W}
    assume ?lhs
    then have L: openin (prod_topology Z X) S
      using S_def by blast
    have ∃ T. openin (prod_topology Z Y) T ∧ (x0, z0) ∈ T ∧ T ⊆ W
      if §: (x0,z0) ∈ W for x0 z0
    proof −
      have x0: x0 ∈ topspace Z
        using W that by blast
      obtain y0 where y0: y0 ∈ topspace X f y0 = z0
        by (metis W fim imageE insert_absorb insert_subset mem_Sigma_iff §)
      then have (x0, y0) ∈ S
        by (simp add: S_def h_def that x0)
```

```
      have continuous_map Z (prod_topology Z X) (λx. (x, y0))
        by (simp add: continuous_map_paired y0)
      with openin_continuous_map_preimage [OF _ L]
      have ope_ZS: openin Z {x ∈ topspace Z. (x,y0) ∈ S}
        by blast
      obtain U U' where openin Z U compactin Z U' closedin Z U'
        x0 ∈ U  U ⊆ U' U' ⊆ {x ∈ topspace Z. (x,y0) ∈ S}
        using loc ope_ZS x0 ‹(x0, y0) ∈ S›
        by (force simp: locally_compact_space_neighbourhood_base_closedin [OF
reg]
            neighbourhood_base_of)
      then have D: U' × {y0} ⊆ S
        by (auto simp: )
      define V where V ≡ {z ∈ topspace Y. U' × {y ∈ topspace X. f y = z} ⊆
S}
      have z0 ∈ V
        using D y0 Int_Collect fim by (fastforce simp: h_def V_def S_def)
      have openin X {x ∈ topspace X. f x ∈ V} ⟹ openin Y V
        using f unfolding V_def quotient_map_def subset_iff
        by (smt (verit, del_insts) Collect_cong mem_Collect_eq)
      moreover have openin X {x ∈ topspace X. f x ∈ V}
      proof −
        let ?Z = subtopology Z U'
        have ∗: {x ∈ topspace X. f x ∈ V} = topspace X − snd ` (U' × topspace
X − S)
          by (force simp: V_def S_def h_def simp flip: fim)
        have compact_space ?Z
          using ‹compactin Z U'› compactin_subspace by auto
        moreover have closedin (prod_topology ?Z X) (U' × topspace X − S)
          by (simp add: L ‹closedin Z U'› closedin_closed_subtopology closedin_diff
closedin_prod_Times_iff
              prod_topology_subtopology(1))
        ultimately show ?thesis
          using ∗ closed_map_snd closed_map_def by fastforce
      qed
      ultimately have openin Y V
        by metis
      show ?thesis
      proof (intro conjI exI)
        show openin (prod_topology Z Y) (U × V)
          by (simp add: openin_prod_Times_iff ‹openin Z U› ‹openin Y V›)
        show (x0, z0) ∈ U × V
          by (simp add: ‹x0 ∈ U› ‹z0 ∈ V›)
        show U × V ⊆ W
          using ‹U ⊆ U'› by (force simp: V_def S_def h_def simp flip: fim)
      qed
    qed
    with openin_subopen show ?rhs by force
  next
```

```
    assume ?rhs then show ?lhs
      using openin_continuous_map_preimage cmh by fastforce
  qed
  ultimately show ?thesis
    by (fastforce simp: image_iff quotient_map_def h_def)
qed

lemma quotient_map_prod_left:
  assumes loc: locally_compact_space Z
    and reg: Hausdorff_space Z ∨ regular_space Z
    and f: quotient_map X Y f
  shows quotient_map (prod_topology X Z) (prod_topology Y Z) (λ(x,y). (f x,y))
proof −
  have (λ(x,y). (f x,y)) = prod.swap ∘ (λ(x,y). (x,f y)) ∘ prod.swap
    by force
  then
  show ?thesis
    apply (rule ssubst)
  proof (intro quotient_map_compose)
    show quotient_map (prod_topology X Z) (prod_topology Z X) prod.swap
      quotient_map (prod_topology Z Y) (prod_topology Y Z) prod.swap
      using homeomorphic_map_def homeomorphic_map_swap quotient_map_eq
by fastforce+
    show quotient_map (prod_topology Z X) (prod_topology Z Y) (λ(x, y). (x, f
y))
      by (simp add: f loc quotient_map_prod_right reg)
  qed
qed

lemma locally_compact_space_perfect_map_preimage:
  assumes locally_compact_space X' and f: perfect_map X X' f
  shows locally_compact_space X
  unfolding locally_compact_space_def
proof (intro strip)
  fix x
  assume x: x ∈ topspace X
  then obtain U K where openin X' U compactin X' K f x ∈ U U ⊆ K
    using assms unfolding locally_compact_space_def perfect_map_def
    by (metis (no_types, lifting) continuous_map_closedin Pi_iff)
  show ∃ U K. openin X U ∧ compactin X K ∧ x ∈ U ∧ U ⊆ K
  proof (intro exI conjI)
    have continuous_map X X' f
      using f perfect_map_def by blast
    then show openin X {x ∈ topspace X. f x ∈ U}
      by (simp add: ‹openin X' U› continuous_map)
    show compactin X {x ∈ topspace X. f x ∈ K}
     using ‹compactin X' K› f perfect_imp_proper_map proper_map_alt by blast
  qed (use x ‹f x ∈ U› ‹U ⊆ K› in auto)
qed
```

5.6.14 Special characterizations of classes of functions into and out of R

lemma *monotone_map_into_euclideanreal_alt*:
assumes *continuous_map X euclideanreal f*
shows $(\forall k.\ is\_interval\ k \longrightarrow connectedin\ X\ \{x \in topspace\ X.\ f\ x \in k\}) \longleftrightarrow$
connected_space X $\wedge$ *monotone_map X euclideanreal f* (**is** *?lhs=?rhs*)
proof
assume *L*: *?lhs*
show *?rhs*
proof
show *connected_space X*
using *L connected_space_subconnected* **by** *blast*
have *connectedin X* $\{x \in topspace\ X.\ f\ x \in \{y\}\}$ **for** *y*
by (*metis L is_interval_1 nle_le singletonD*)
then show *monotone_map X euclideanreal f*
by (*simp add*: *monotone_map*)
qed
next
assume *R*: *?rhs*
then
have $*$: *False*
if $a < b$ *closedin X U closedin X V U* $\neq \{\}$ *V* $\neq \{\}$ *disjnt U V*
and *UV*: $\{x \in topspace\ X.\ f\ x \in \{a..b\}\} = U \cup V$
and *dis*: *disjnt U* $\{x \in topspace\ X.\ f\ x = b\}$ *disjnt V* $\{x \in topspace\ X.\ f\ x = a\}$
for *a b U V*
proof −
define *E1* **where** $E1 \equiv U \cup \{x \in topspace\ X.\ f\ x \in \{c.\ c \leq a\}\}$
define *E2* **where** $E2 \equiv V \cup \{x \in topspace\ X.\ f\ x \in \{c.\ b \leq c\}\}$
have *closedin X* $\{x \in topspace\ X.\ f\ x \leq a\}$ *closedin X* $\{x \in topspace\ X.\ b \leq f\ x\}$
using *assms continuous_map_upper_lower_semicontinuous_le* **by** *blast+*
then have *closedin X E1 closedin X E2*
unfolding *E1_def E2_def* **using** *that* **by** *auto*
moreover
have $E1 \cap E2 = \{\}$
unfolding *E1_def E2_def* **using** ‹$a<b$› ‹*disjnt U V*› *dis UV*
by (*simp add*: *disjnt_def set_eq_iff*) (*smt* (*verit*))
have $topspace\ X \subseteq E1 \cup E2$
unfolding *E1_def E2_def* **using** *UV* **by** *fastforce*
have $E1 = \{\} \vee E2 = \{\}$
using *R connected_space_closedin*
using ‹$E1 \cap E2 = \{\}$› ‹*closedin X E1*› ‹*closedin X E2*› ‹$topspace\ X \subseteq E1 \cup E2$› **by** *blast*
then show *False*
using *E1_def E2_def* ‹$U \neq \{\}$› ‹$V \neq \{\}$› **by** *fastforce*
qed
show *?lhs*
proof (*intro strip*)

fix K :: *real set*
assume *is_interval K*
have *False*
if $a \in K$ $b \in K$ **and** *clo*: *closedin X U closedin X V*
and *UV*: $\{x.\ x \in$ *topspace X* $\wedge\ f\ x \in K\} \subseteq U \cup V$
$U \cap V \cap \{x.\ x \in$ *topspace X* $\wedge\ f\ x \in K\} = \{\}$
and *nondis*: $\neg$ *disjnt U* $\{x.\ x \in$ *topspace X* $\wedge\ f\ x = a\}$
$\neg$ *disjnt V* $\{x.\ x \in$ *topspace X* $\wedge\ f\ x = b\}$
for a b U V
proof −
have $\forall y.$ *connectedin X* $\{x.\ x \in$ *topspace X* $\wedge\ f\ x = y\}$
using *R monotone_map* **by** *fastforce*
then have $**$: *False* **if** $p \in U \wedge q \in V \wedge f\ p = f\ q \wedge f\ q \in K$ **for** p q
unfolding *connectedin_closedin*
using ‹$a \in K$› ‹$b \in K$› *UV clo that*
by (*smt* (*verit*, *ccfv_threshold*) *closedin_subset disjoint_iff mem_Collect_eq subset_iff*)
consider $a < b \mid a = b \mid b < a$
by *linarith*
then show *?thesis*
proof *cases*
case *1*
define *W* **where** $W \equiv \{x \in$ *topspace X*$.\ f\ x \in \{a..b\}\}$
have *closedin X W*
unfolding *W_def*
by (*metis* (*no_types*) *assms closed_real_atLeastAtMost closed_closedin continuous_map_closedin*)
show *?thesis*
proof (*rule* $*$ [*OF 1* , *of U* $\cap$ *W V* $\cap$ *W*])
show *closedin X* $(U \cap W)$ *closedin X* $(V \cap W)$
using ‹*closedin X W*› *clo* **by** *auto*
show $U \cap W \neq \{\}$ $V \cap W \neq \{\}$
using *nondis 1* **by** (*auto simp*: *disjnt_iff W_def*)
show *disjnt* $(U \cap W)$ $(V \cap W)$
using ‹*is_interval K*› **unfolding** *is_interval_1 disjnt_iff W_def*
by (*metis* (*mono_tags*, *lifting*) ‹$a \in K$› ‹$b \in K$› $**$ *Int_Collect atLeastAtMost_iff*)
have $\bigwedge x.\ [\![x \in$ *topspace X*; $a \le f\ x$; $f\ x \le b]\!] \Longrightarrow x \in U \vee x \in V$
using ‹$a \in K$› ‹$b \in K$› ‹*is_interval K*› *UV* **unfolding** *is_interval_1 disjnt_iff*
by *blast*
then show $\{x \in$ *topspace X*$.\ f\ x \in \{a..b\}\} = U \cap W \cup V \cap W$
by (*auto simp*: *W_def*)
show *disjnt* $(U \cap W)$ $\{x \in$ *topspace X*$.\ f\ x = b\}$ *disjnt* $(V \cap W)$ $\{x \in$ *topspace X*$.\ f\ x = a\}$
using $**$ ‹$a \in K$› ‹$b \in K$› *nondis* **by** (*force simp*: *disjnt_iff*)+
qed
next
case *2*

```
      then show ?thesis
        using ** nondis ‹b ∈ K› by (force simp add: disjnt_iff)
      next
        case 3
        define W where W ≡ {x ∈ topspace X. f x ∈ {b..a}}
        have closedin X W
          unfolding W_def
          by (metis (no_types) assms closed_real_atLeastAtMost closed_closedin
continuous_map_closedin)
        show ?thesis
        proof (rule * [OF 3, of V ∩ W U ∩ W])
          show closedin X (U ∩ W) closedin X (V ∩ W)
            using ‹closedin X W› clo by auto
          show U ∩ W ≠ {} V ∩ W ≠ {}
            using nondis 3 by (auto simp: disjnt_iff W_def)
          show disjnt (V ∩ W) (U ∩ W)
            using ‹is_interval K› unfolding is_interval_1 disjnt_iff W_def
                by (metis (mono_tags, lifting) ‹a ∈ K› ‹b ∈ K› ** Int_Collect
atLeastAtMost_iff)
          have ⋀x. ⟦x ∈ topspace X; b ≤ f x; f x ≤ a⟧ ⟹ x ∈ U ∨ x ∈ V
            using ‹a ∈ K› ‹b ∈ K› ‹is_interval K› UV unfolding is_interval_1
disjnt_iff
            by blast
          then show {x ∈ topspace X. f x ∈ {b..a}} = V ∩ W ∪ U ∩ W
            by (auto simp: W_def)
          show disjnt (V ∩ W) {x ∈ topspace X. f x = a} disjnt (U ∩ W) {x ∈
topspace X. f x = b}
            using ** ‹a ∈ K› ‹b ∈ K› nondis by (force simp: disjnt_iff)+
        qed
      qed
    qed
    then show connectedin X {x ∈ topspace X. f x ∈ K}
      unfolding connectedin_closedin disjnt_iff by blast
  qed
qed

lemma monotone_map_into_euclideanreal:
  ⟦connected_space X; continuous_map X euclideanreal f⟧
   ⟹ monotone_map X euclideanreal f ⟷
      (∀k. is_interval k ⟶ connectedin X {x ∈ topspace X. f x ∈ k})
  by (simp add: monotone_map_into_euclideanreal_alt)

lemma monotone_map_euclideanreal_alt:
  (∀I::real set. is_interval I ⟶ is_interval {x::real. x ∈ S ∧ f x ∈ I}) ⟷
   is_interval S ∧ (mono_on S f ∨ antimono_on S f) (is ?lhs=?rhs)
proof
  assume L [rule_format]: ?lhs
  show ?rhs
  proof
```

show *is_interval S*
using *L is_interval_1* **by** *auto*
have *False* **if** $a \in S$ $b \in S$ $c \in S$ $a<b$ $b<c$ **and** *d*: $f\ a < f\ b \wedge f\ c < f\ b \vee f\ a > f\ b \wedge f\ c > f\ b$ **for** *a b c*
using *d*
proof
assume $f\ a < f\ b \wedge f\ c < f\ b$
then show *False*
using *L* [*of* $\{y.\ \ y < f\ b\}$] **unfolding** *is_interval_1*
by (*smt* (*verit*, *best*) *mem_Collect_eq that*)
next
assume $f\ b < f\ a \wedge f\ b < f\ c$
then show *False*
using *L* [*of* $\{y.\ \ y > f\ b\}$] **unfolding** *is_interval_1*
by (*smt* (*verit*, *best*) *mem_Collect_eq that*)
qed
then show *mono_on S f* $\vee$ *monotone_on S* $(\leq)$ $(\geq)$ *f*
unfolding *monotone_on_def* **by** (*smt* (*verit*))
qed
next
assume *?rhs* **then show** *?lhs*
unfolding *is_interval_1 monotone_on_def* **by** *simp meson*
qed

lemma *monotone_map_euclideanreal*:
fixes *S* :: *real set*
shows
$[\![$*is_interval S*; *continuous_on S f*$]\!] \Longrightarrow$
monotone_map (*top_of_set S*) *euclideanreal f* $\longleftrightarrow$ (*mono_on S f* $\vee$ *monotone_on S* $(\leq)$ $(\geq)$ *f*)
using *monotone_map_euclideanreal_alt*
by (*simp add*: *monotone_map_into_euclideanreal connectedin_subtopology is_interval_connected_1*)

lemma *injective_eq_monotone_map*:
fixes *f* :: *real* $\Rightarrow$ *real*
assumes *is_interval S continuous_on S f*
shows *inj_on f S* $\longleftrightarrow$ *strict_mono_on S f* $\vee$ *strict_antimono_on S f*
by (*metis assms injective_imp_monotone_map monotone_map_euclideanreal strict_antimono_iff_antimono*
strict_mono_iff_mono top_greatest topspace_euclidean topspace_euclidean_subtopology)

5.6.15 Normal spaces

definition *normal_space*
where *normal_space X* $\equiv$
$\forall\, S\ T.$ *closedin X S* $\wedge$ *closedin X T* $\wedge$ *disjnt S T*
$\longrightarrow$ $(\exists\, U\ V.$ *openin X U* $\wedge$ *openin X V* $\wedge\ S \subseteq U \wedge\ T \subseteq V \wedge$ *disjnt U V*)

lemma *normal_space_retraction_map_image*:
assumes *r*: *retraction_map X Y r* **and** *X*: *normal_space X*
shows *normal_space Y*
unfolding *normal_space_def*
proof *clarify*
fix *S T*
assume *closedin Y S* **and** *closedin Y T* **and** *disjnt S T*
obtain *r'* **where** *r'*: *retraction_maps X Y r r'*
using *r retraction_map_def* **by** *blast*
have *closedin X* {*x* ∈ *topspace X*. *r x* ∈ *S*} *closedin X* {*x* ∈ *topspace X*. *r x* ∈ *T*}
using *closedin_continuous_map_preimage* ‹*closedin Y S*› ‹*closedin Y T*› *r'*
by (*auto simp*: *retraction_maps_def*)
moreover
have *disjnt* {*x* ∈ *topspace X*. *r x* ∈ *S*} {*x* ∈ *topspace X*. *r x* ∈ *T*}
using ‹*disjnt S T*› **by** (*auto simp*: *disjnt_def*)
ultimately
obtain *U V* **where** *UV*: *openin X U* ∧ *openin X V* ∧ {*x* ∈ *topspace X*. *r x* ∈ *S*} ⊆ *U* ∧ {*x* ∈ *topspace X*. *r x* ∈ *T*} ⊆ *V disjnt U V*
by (*meson X normal_space_def*)
show ∃ *U V*. *openin Y U* ∧ *openin Y V* ∧ *S* ⊆ *U* ∧ *T* ⊆ *V* ∧ *disjnt U V*
proof (*intro exI conjI*)
show *openin Y* {*x* ∈ *topspace Y*. *r' x* ∈ *U*} *openin Y* {*x* ∈ *topspace Y*. *r' x* ∈ *V*}
using *openin_continuous_map_preimage UV r'*
by (*auto simp*: *retraction_maps_def*)
show *S* ⊆ {*x* ∈ *topspace Y*. *r' x* ∈ *U*} *T* ⊆ {*x* ∈ *topspace Y*. *r' x* ∈ *V*}
using *openin_continuous_map_preimage UV r'* ‹*closedin Y S*› ‹*closedin Y T*›
by (*auto simp add*: *closedin_def continuous_map_closedin retraction_maps_def subset_iff Pi_iff*)
show *disjnt* {*x* ∈ *topspace Y*. *r' x* ∈ *U*} {*x* ∈ *topspace Y*. *r' x* ∈ *V*}
using ‹*disjnt U V*› **by** (*auto simp*: *disjnt_def*)
qed
qed

lemma *homeomorphic_normal_space*:
X homeomorphic_space Y ⟹ *normal_space X* ⟷ *normal_space Y*
unfolding *homeomorphic_space_def*
by (*meson homeomorphic_imp_retraction_maps homeomorphic_maps_sym normal_space_retraction_map_image retraction_map_def*)

lemma *normal_space*:
normal_space X ⟷
(∀ *S T*. *closedin X S* ∧ *closedin X T* ∧ *disjnt S T*
⟶ (∃ *U*. *openin X U* ∧ *S* ⊆ *U* ∧ *disjnt T* (*X closure_of U*)))
proof −
have (∃ *V*. *openin X U* ∧ *openin X V* ∧ *S* ⊆ *U* ∧ *T* ⊆ *V* ∧ *disjnt U V*) ⟷

openin X U ∧ S ⊆ U ∧ disjnt T (X closure_of U)
(**is** *?lhs=?rhs*)
if *closedin X S closedin X T disjnt S T* **for** *S T U*
proof
show *?lhs ⟹ ?rhs*
by (*smt* (*verit*, *best*) *disjnt_iff in_closure_of subsetD*)
assume *R*: *?rhs*
then have (*U ∪ S*) *∩* (*topspace X − X closure_of U*) = {}
by (*metis Diff_eq_empty_iff Int_Diff Int_Un_eq*(*4*) *closure_of_subset inf.orderE openin_subset*)
moreover have *T ⊆ topspace X − X closure_of U*
by (*meson DiffI R closedin_subset disjnt_iff subsetD subsetI that*(*2*))
ultimately show *?lhs*
by (*metis R closedin_closure_of closedin_def disjnt_def sup.orderE*)
qed
then show *?thesis*
unfolding *normal_space_def* **by** *meson*
qed

lemma *normal_space_alt*:
normal_space X ⟷
(*∀ S U. closedin X S ∧ openin X U ∧ S ⊆ U ⟶* (*∃ V. openin X V ∧ S ⊆ V ∧ X closure_of V ⊆ U*))
proof −
have *∃ V. openin X V ∧ S ⊆ V ∧ X closure_of V ⊆ U*
if *⋀T. closedin X T ⟶ disjnt S T ⟶* (*∃ U. openin X U ∧ S ⊆ U ∧ disjnt T* (*X closure_of U*))
closedin X S openin X U S ⊆ U
for *S U*
using *that*
by (*smt* (*verit*) *Diff_eq_empty_iff Int_Diff closure_of_subset_topspace disjnt_def inf.orderE inf_commute openin_closedin_eq*)
moreover have *∃ U. openin X U ∧ S ⊆ U ∧ disjnt T* (*X closure_of U*)
if *⋀U. openin X U ∧ S ⊆ U ⟶* (*∃ V. openin X V ∧ S ⊆ V ∧ X closure_of V ⊆ U*)
and *closedin X S closedin X T disjnt S T*
for *S T*
using *that*
by (*smt* (*verit*) *Diff_Diff_Int Diff_eq_empty_iff Int_Diff closedin_def disjnt_def inf.absorb_iff2 inf.orderE*)
ultimately show *?thesis*
by (*fastforce simp*: *normal_space*)
qed

lemma *normal_space_closures*:
normal_space X ⟷
(*∀ S T. S ⊆ topspace X ∧ T ⊆ topspace X ∧*
disjnt (*X closure_of S*) (*X closure_of T*)
⟶ (*∃ U V. openin X U ∧ openin X V ∧ S ⊆ U ∧ T ⊆ V ∧ disjnt U*

```
V))
  (is ?lhs=?rhs)
proof
  show ?lhs ⟹ ?rhs
   by (meson closedin_closure_of closure_of_subset normal_space_def order.trans)
  show ?rhs ⟹ ?lhs
    by (metis closedin_subset closure_of_eq normal_space_def)
qed

lemma normal_space_disjoint_closures:
   normal_space X ⟷
    (∀ S T. closedin X S ∧ closedin X T ∧ disjnt S T
          ⟶ (∃ U V. openin X U ∧ openin X V ∧ S ⊆ U ∧ T ⊆ V ∧
                    disjnt (X closure_of U) (X closure_of V)))
  (is ?lhs=?rhs)
proof
  show ?lhs ⟹ ?rhs
    by (metis closedin_closure_of normal_space)
  show ?rhs ⟹ ?lhs
    by (smt (verit) closure_of_subset disjnt_iff normal_space openin_subset sub-
set_eq)
qed

lemma normal_space_dual:
   normal_space X ⟷
    (∀ U V. openin X U ⟶ openin X V ∧ U ∪ V = topspace X
          ⟶ (∃ S T. closedin X S ∧ closedin X T ∧ S ⊆ U ∧ T ⊆ V ∧ S ∪ T =
topspace X))
  (is _ = ?rhs)
proof −
  have normal_space X ⟷
       (∀ U V. closedin X U ⟶ closedin X V ⟶ disjnt U V ⟶
            (∃ S T. ¬ (openin X S ∧ openin X T ⟶
                    ¬ (U ⊆ S ∧ V ⊆ T ∧ disjnt S T))))
    unfolding normal_space_def by meson
  also have ... ⟷ (∀ U V. openin X U ⟶ openin X V ∧ disjnt (topspace X −
U) (topspace X − V) ⟶
            (∃ S T. ¬ (openin X S ∧ openin X T ⟶
                     ¬ (topspace X − U ⊆ S ∧ topspace X − V ⊆ T ∧ disjnt S
T))))
    by (auto simp: all_closedin)
  also have ... ⟷ ?rhs
  proof −
    have ∗: disjnt (topspace X − U) (topspace X − V) ⟷ U ∪ V = topspace X
      if U ⊆ topspace X V ⊆ topspace X for U V
      using that by (auto simp: disjnt_iff)
    show ?thesis
      using ex_closedin ∗
       apply (simp add: ex_closedin ∗ [OF openin_subset openin_subset] cong:
```

conj_cong)
apply (*intro all_cong1 ex_cong1 imp_cong refl*)
by (*smt* (*verit*, *best*) ∗ *Diff_Diff_Int Diff_subset Diff_subset_conv inf.orderE inf_commute openin_subset sup_commute*)
qed
finally show *?thesis* **.**
qed

lemma *normal_t1_imp_Hausdorff_space*:
assumes *normal_space X t1_space X*
shows *Hausdorff_space X*
unfolding *Hausdorff_space_def*
proof *clarify*
fix *x y*
assume *xy*: $x \in$ *topspace X* $y \in$ *topspace X* $x \neq y$
then have *disjnt* $\{x\}$ $\{y\}$
by (*auto simp*: *disjnt_iff*)
then show $\exists$ *U V*. *openin X U* $\wedge$ *openin X V* $\wedge$ $x \in U \wedge y \in V \wedge$ *disjnt U V*
using *assms xy closedin_t1_singleton normal_space_def*
by (*metis singletonI subsetD*)
qed

lemma *normal_t1_eq_Hausdorff_space*:
normal_space X $\Longrightarrow$ *t1_space X* $\longleftrightarrow$ *Hausdorff_space X*
using *normal_t1_imp_Hausdorff_space t1_or_Hausdorff_space* **by** *blast*

lemma *normal_t1_imp_regular_space*:
$[\![$*normal_space X*; *t1_space X*$]\!] \Longrightarrow$ *regular_space X*
by (*metis compactin_imp_closedin normal_space_def normal_t1_eq_Hausdorff_space regular_space_compact_closed_sets*)

lemma *compact_Hausdorff_or_regular_imp_normal_space*:
$[\![$*compact_space X*; *Hausdorff_space X* $\vee$ *regular_space X*$]\!]$
$\Longrightarrow$ *normal_space X*
by (*metis Hausdorff_space_compact_sets closedin_compact_space normal_space_def regular_space_compact_closed_sets*)

lemma *normal_space_discrete_topology*:
normal_space(*discrete_topology U*)
by (*metis discrete_topology_closure_of inf_le2 normal_space_alt*)

lemma *normal_space_fsigmas*:
normal_space X $\longleftrightarrow$
($\forall$ *S T*. *fsigma_in X S* $\wedge$ *fsigma_in X T* $\wedge$ *separatedin X S T*
$\longrightarrow$ ($\exists$ *U B*. *openin X U* $\wedge$ *openin X B* $\wedge$ $S \subseteq U \wedge T \subseteq B \wedge$ *disjnt U B*)) (**is** *?lhs=?rhs*)
proof
assume *L*: *?lhs*

show *?rhs*
proof *clarify*
fix *S T*
assume *fsigma_in X S*
then obtain *C* **where** *C*: $\bigwedge n.$ *closedin X* (*C n*) $\bigwedge n.$ *C n* $\subseteq$ *C* (*Suc n*) $\bigcup$ (*range C*) = *S*
by (*meson fsigma_in_ascending*)
assume *fsigma_in X T*
then obtain *D* **where** *D*: $\bigwedge n.$ *closedin X* (*D n*) $\bigwedge n.$ *D n* $\subseteq$ *D* (*Suc n*) $\bigcup$ (*range D*) = *T*
by (*meson fsigma_in_ascending*)
assume *separatedin X S T*
have $\bigwedge n.$ *disjnt* (*D n*) (*X closure_of S*)
by (*metis D*(3) ‹*separatedin X S T*› *disjnt_Union1 disjnt_def rangeI separatedin_def*)
then have $\bigwedge n.\ \exists V\ V'.$ *openin X V* $\wedge$ *openin X V'* $\wedge$ *D n* $\subseteq$ *V* $\wedge$ *X closure_of S* $\subseteq$ *V'* $\wedge$ *disjnt V V'*
by (*metis D*(1) *L closedin_closure_of normal_space_def*)
then obtain *V V'* **where** *V*: $\bigwedge n.$ *openin X* (*V n*) **and** $\bigwedge n.$ *openin X* (*V' n*) $\bigwedge n.$ *disjnt* (*V n*) (*V' n*)
and *DV*: $\bigwedge n.$ *D n* $\subseteq$ *V n*
and *subV'*: $\bigwedge n.$ *X closure_of S* $\subseteq$ *V' n*
by *metis*
then have *VV*: *V' n* $\cap$ *X closure_of V n* = {} **for** *n*
using *openin_Int_closure_of_eq_empty* [*of X V' n V n*] **by** (*simp add*: *Int_commute disjnt_def*)
have $\bigwedge n.$ *disjnt* (*C n*) (*X closure_of T*)
by (*metis C*(3) ‹*separatedin X S T*› *disjnt_Union1 disjnt_def rangeI separatedin_def*)
then have $\bigwedge n.\ \exists U\ U'.$ *openin X U* $\wedge$ *openin X U'* $\wedge$ *C n* $\subseteq$ *U* $\wedge$ *X closure_of T* $\subseteq$ *U'* $\wedge$ *disjnt U U'*
by (*metis C*(1) *L closedin_closure_of normal_space_def*)
then obtain *U U'* **where** *U*: $\bigwedge n.$ *openin X* (*U n*) **and** $\bigwedge n.$ *openin X* (*U' n*) $\bigwedge n.$ *disjnt* (*U n*) (*U' n*)
and *CU*: $\bigwedge n.$ *C n* $\subseteq$ *U n*
and *subU'*: $\bigwedge n.$ *X closure_of T* $\subseteq$ *U' n*
by *metis*
then have *UU*: *U' n* $\cap$ *X closure_of U n* = {} **for** *n*
using *openin_Int_closure_of_eq_empty* [*of X U' n U n*] **by** (*simp add*: *Int_commute disjnt_def*)
show $\exists U\ B.$ *openin X U* $\wedge$ *openin X B* $\wedge$ *S* $\subseteq$ *U* $\wedge$ *T* $\subseteq$ *B* $\wedge$ *disjnt U B*
proof (*intro conjI exI*)
have $\bigwedge S\ n.$ *closedin X* ($\bigcup m{\le}n.$ *X closure_of V m*)
by (*force intro*: *closedin_Union*)
then show *openin X* ($\bigcup n.$ *U n* − ($\bigcup m{\le}n.$ *X closure_of V m*))
using *U* **by** *blast*
have $\bigwedge S\ n.$ *closedin X* ($\bigcup m{\le}n.$ *X closure_of U m*)
by (*force intro*: *closedin_Union*)
then show *openin X* ($\bigcup n.$ *V n* − ($\bigcup m{\le}n.$ *X closure_of U m*))

using *V* **by** *blast*
have *S* ⊆ *topspace X*
by (*simp add*: ‹*fsigma_in X S*› *fsigma_in_subset*)
then show *S* ⊆ (⋃ *n*. *U n* − (⋃ *m*≤*n*. *X closure_of V m*))
apply (*clarsimp simp*: *Ball_def*)
by (*metis VV C*(*3*) *CU IntI UN_E closure_of_subset empty_iff subV*′ *subsetD*)
have *T* ⊆ *topspace X*
by (*simp add*: ‹*fsigma_in X T*› *fsigma_in_subset*)
then show *T* ⊆ (⋃ *n*. *V n* − (⋃ *m*≤*n*. *X closure_of U m*))
apply (*clarsimp simp*: *Ball_def*)
by (*metis UU D*(*3*) *DV IntI UN_E closure_of_subset empty_iff subU*′ *subsetD*)
have ⋀*x m n*. ⟦*x* ∈ *U n*; *x* ∈ *V m*; ∀ *k*≤*m*. *x* ∉ *X closure_of U k*⟧ ⟹ ∃ *k*≤*n*. *x* ∈ *X closure_of V k*
by (*meson U V closure_of_subset nat_le_linear openin_subset subsetD*)
then show *disjnt* (⋃ *n*. *U n* − (⋃ *m*≤*n*. *X closure_of V m*)) (⋃ *n*. *V n* − (⋃ *m*≤*n*. *X closure_of U m*))
by (*force simp*: *disjnt_iff*)
qed
qed
next
show *?rhs* ⟹ *?lhs*
by (*simp add*: *closed_imp_fsigma_in normal_space_def separatedin_closed_sets*)
qed

lemma *normal_space_fsigma_subtopology*:
assumes *normal_space X fsigma_in X S*
shows *normal_space* (*subtopology X S*)
unfolding *normal_space_fsigmas*
proof *clarify*
fix *T U*
assume *fsigma_in* (*subtopology X S*) *T*
and *fsigma_in* (*subtopology X S*) *U*
and *TU*: *separatedin* (*subtopology X S*) *T U*
then obtain *A B* **where** *openin X A* ∧ *openin X B* ∧ *T* ⊆ *A* ∧ *U* ⊆ *B* ∧ *disjnt A B*
by (*metis assms fsigma_in_fsigma_subtopology normal_space_fsigmas separatedin_subtopology*)
then
show ∃ *A B*. *openin* (*subtopology X S*) *A* ∧ *openin* (*subtopology X S*) *B* ∧ *T* ⊆ *A* ∧
U ⊆ *B* ∧ *disjnt A B*
using *TU*
by (*force simp add*: *separatedin_subtopology openin_subtopology_alt disjnt_iff*)
qed

lemma *normal_space_closed_subtopology*:
assumes *normal_space X closedin X S*

shows *normal_space* (*subtopology X S*)
by (*simp add*: *assms closed_imp_fsigma_in normal_space_fsigma_subtopology*)

lemma *normal_space_continuous_closed_map_image*:
assumes *normal_space X* **and** *contf*: *continuous_map X Y f*
and *clof*: *closed_map X Y f* **and** *fim*: *f ‘ topspace X = topspace Y*
shows *normal_space Y*
unfolding *normal_space_def*
proof *clarify*
fix *S T*
assume *closedin Y S* **and** *closedin Y T* **and** *disjnt S T*
have *closedin X* {*x* ∈ *topspace X*. *f x* ∈ *S*} *closedin X* {*x* ∈ *topspace X*. *f x* ∈ *T*}
using ‹*closedin Y S*› ‹*closedin Y T*› *closedin_continuous_map_preimage contf*
by *auto*
moreover
have *disjnt* {*x* ∈ *topspace X*. *f x* ∈ *S*} {*x* ∈ *topspace X*. *f x* ∈ *T*}
using ‹*disjnt S T*› **by** (*auto simp*: *disjnt_iff*)
ultimately
obtain *U V* **where** *closedin X U closedin X V*
and *subXU*: {*x* ∈ *topspace X*. *f x* ∈ *S*} ⊆ *topspace X* − *U*
and *subXV*: {*x* ∈ *topspace X*. *f x* ∈ *T*} ⊆ *topspace X* − *V*
and *dis*: *disjnt* (*topspace X* − *U*) (*topspace X* − *V*)
using ‹*normal_space X*› **by** (*force simp add*: *normal_space_def ex_openin*)
have *closedin Y* (*f ‘ U*) *closedin Y* (*f ‘ V*)
using ‹*closedin X U*› ‹*closedin X V*› *clof closed_map_def* **by** *blast*+
moreover have *S* ⊆ *topspace Y* − *f ‘ U*
using ‹*closedin Y S*› ‹*closedin X U*› *subXU* **by** (*force dest*: *closedin_subset*)
moreover have *T* ⊆ *topspace Y* − *f ‘ V*
using ‹*closedin Y T*› ‹*closedin X V*› *subXV* **by** (*force dest*: *closedin_subset*)
moreover have *disjnt* (*topspace Y* − *f ‘ U*) (*topspace Y* − *f ‘ V*)
using *fim dis* **by** (*force simp add*: *disjnt_iff*)
ultimately show ∃ *U V*. *openin Y U* ∧ *openin Y V* ∧ *S* ⊆ *U* ∧ *T* ⊆ *V* ∧ *disjnt U V*
by (*force simp add*: *ex_openin*)
qed

5.6.16 Hereditary topological properties

definition *hereditarily*
where *hereditarily P X* ≡
∀ *S*. *S* ⊆ *topspace X* ⟶ *P*(*subtopology X S*)

lemma *hereditarily*:
hereditarily P X ⟷ (∀ *S*. *P*(*subtopology X S*))
by (*metis Int_lower1 hereditarily_def subtopology_restrict*)

lemma *hereditarily_mono*:
⟦*hereditarily P X*; ⋀*x*. *P x* ⟹ *Q x*⟧ ⟹ *hereditarily Q X*

by (*simp add*: *hereditarily*)

lemma *hereditarily_inc*:
hereditarily P X $\Longrightarrow$ *P X*
by (*metis hereditarily subtopology_topspace*)

lemma *hereditarily_subtopology*:
hereditarily P X $\Longrightarrow$ *hereditarily P* (*subtopology X S*)
by (*simp add*: *hereditarily subtopology_subtopology*)

lemma *hereditarily_normal_space_continuous_closed_map_image*:
assumes *X*: *hereditarily normal_space X* **and** *contf*: *continuous_map X Y f*
and *clof*: *closed_map X Y f* **and** *fim*: *f ' (topspace X) = topspace Y*
shows *hereditarily normal_space Y*
unfolding *hereditarily_def*
proof (*intro strip*)
fix *T*
assume *T* $\subseteq$ *topspace Y*
then have *nx*: *normal_space* (*subtopology X* {*x* $\in$ *topspace X*. *f x* $\in$ *T*})
by (*meson X hereditarily*)
moreover have *continuous_map* (*subtopology X* {*x* $\in$ *topspace X*. *f x* $\in$ *T*})
(*subtopology Y T*) *f*
by (*simp add*: *contf continuous_map_from_subtopology continuous_map_in_subtopology image_subset_iff*)
moreover have *closed_map* (*subtopology X* {*x* $\in$ *topspace X*. *f x* $\in$ *T*}) (*subtopology Y T*) *f*
by (*simp add*: *clof closed_map_restriction*)
ultimately show *normal_space* (*subtopology Y T*)
using *fim normal_space_continuous_closed_map_image* **by** *fastforce*
qed

lemma *homeomorphic_hereditarily_normal_space*:
X homeomorphic_space Y
$\Longrightarrow$ (*hereditarily normal_space X* $\longleftrightarrow$ *hereditarily normal_space Y*)
by (*meson hereditarily_normal_space_continuous_closed_map_image homeomorphic_eq_everything_map*
homeomorphic_space homeomorphic_space_sym)

lemma *hereditarily_normal_space_retraction_map_image*:
⟦*retraction_map X Y r*; *hereditarily normal_space X*⟧ $\Longrightarrow$ *hereditarily normal_space Y*
by (*smt* (*verit*) *hereditarily_subtopology hereditary_imp_retractive_property homeomorphic_hereditarily_normal_space*)

5.6.17 Limits in a topological space

lemma *limitin_const_iff*:
assumes *t1_space X* ¬ *trivial_limit F*
shows *limitin X* (λ*k*. *a*) *l F* $\longleftrightarrow$ *l* $\in$ *topspace X* $\wedge$ *a* = *l* (**is** *?lhs=?rhs*)

proof
assume *?lhs* **then show** *?rhs*
using *assms* **unfolding** *limitin_def t1_space_def* **by** (*metis eventually_const openin_topspace*)
next
assume *?rhs* **then show** *?lhs*
using *assms* **by** (*auto simp*: *limitin_def t1_space_def*)
qed

lemma *compactin_sequence_with_limit*:
assumes *lim*: *limitin X σ l sequentially* **and** $S \subseteq$ *range σ* **and** *SX*: $S \subseteq$ *topspace X*
shows *compactin X* (*insert l S*)
unfolding *compactin_def*
proof (*intro conjI strip*)
show *insert l S* $\subseteq$ *topspace X*
by (*meson SX insert_subset lim limitin_topspace*)
fix $\mathcal{U}$
assume §: *Ball* $\mathcal{U}$ (*openin X*) $\wedge$ *insert l S* $\subseteq \bigcup \mathcal{U}$
have $\exists V.$ *finite V* $\wedge$ $V \subseteq \mathcal{U} \wedge (\exists t \in V.\ l \in t) \wedge S \subseteq \bigcup V$
if $*$: $\forall x \in S.\ \exists T \in \mathcal{U}.\ x \in T$ **and** $T \in \mathcal{U}$ $l \in T$ **for** T
proof −
obtain V **where** V: $\bigwedge x.\ x \in S \Longrightarrow V\ x \in \mathcal{U} \wedge x \in V\ x$
using $*$ **by** *metis*
obtain N **where** N: $\bigwedge n.\ N \leq n \Longrightarrow \sigma\ n \in T$
by (*meson* § ‹$T \in \mathcal{U}$› ‹$l \in T$› *lim limitin_sequentially*)
show *?thesis*
proof (*intro conjI exI*)
have $x \in T$
if $x \in S$ **and** $\forall A.\ (\forall x \in S.\ (\forall n{\leq}N.\ x \neq \sigma\ n) \vee A \neq V\ x) \vee x \notin A$ **for** x
by (*metis* (*no_types*) *N V that assms*(*2*) *imageE nle_le subsetD*)
then show $S \subseteq \bigcup$ (*insert T* (*V* ‘ (*S* ∩ *σ* ‘ {*0*..*N*})))
by *force*
qed (*use V that* **in** *auto*)
qed
then show $\exists \mathcal{F}.$ *finite* $\mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge$ *insert l S* $\subseteq \bigcup \mathcal{F}$
by (*smt* (*verit*, *best*) *Union_iff* § *insert_subset subsetD*)
qed

lemma *limitin_Hausdorff_unique*:
assumes *limitin X f l1 F limitin X f l2 F* ¬ *trivial_limit F Hausdorff_space X*
shows *l1* = *l2*
proof (*rule ccontr*)
assume *l1* ≠ *l2*
with *assms* **obtain** *U V* **where** *openin X U openin X V l1* ∈ *U l2* ∈ *V disjnt U V*
by (*metis Hausdorff_space_def limitin_topspace*)
then have *eventually* (λ*x*. *f x* ∈ *U*) *F eventually* (λ*x*. *f x* ∈ *V*) *F*
using *assms* **by** (*fastforce simp*: *limitin_def*)+

then have $\exists x.\ f\ x \in U \wedge f\ x \in V$
using *assms eventually_elim2 filter_eq_iff* **by** *fastforce*
with *assms* ‹*disjnt U V*› **show** *False*
by (*meson disjnt_iff*)
qed

lemma *limitin_kc_unique*:
assumes *kc_space X* **and** *lim1*: *limitin X f l1 sequentially* **and** *lim2*: *limitin X f l2 sequentially*
shows *l1* = *l2*
proof (*rule ccontr*)
assume *l1* ≠ *l2*
define *A* **where** *A* ≡ *insert l1* (*range f* − {*l2*})
have *l1* ∈ *topspace X*
using *lim1 limitin_def* **by** *fastforce*
moreover have *compactin X* (*insert l1* (*topspace X* ∩ (*range f* − {*l2*})))
by (*meson Diff_subset compactin_sequence_with_limit inf_le1 inf_le2 lim1 subset_trans*)
ultimately have *compactin X* (*topspace X* ∩ *A*)
by (*simp add*: *A_def*)
then have *OXA*: *openin X* (*topspace X* − *A*)
by (*metis Diff_Diff_Int Diff_subset* ‹*kc_space X*› *kc_space_def openin_closedin_eq*)
have *l2* ∈ *topspace X* − *A*
using ‹*l1* ≠ *l2*› *A_def lim2 limitin_topspace* **by** *fastforce*
then have ∀$_F$ *x in sequentially*. *f x* = *l2*
using *limitinD* [*OF lim2 OXA*] **by** (*auto simp*: *A_def eventually_conj_iff*)
then show *False*
using *limitin_transform_eventually* [*OF _ lim1*]
limitin_const_iff [*OF kc_imp_t1_space trivial_limit_sequentially*]
using ‹*l1* ≠ *l2*› ‹*kc_space X*› **by** *fastforce*
qed

lemma *limitin_closedin*:
assumes *lim*: *limitin X f l F*
and *closedin X S* **and** *ev*: *eventually* (λ*x*. *f x* ∈ *S*) *F* ¬ *trivial_limit F*
shows *l* ∈ *S*
proof (*rule ccontr*)
assume *l* ∉ *S*
have ∀$_F$ *x in F*. *f x* ∈ *topspace X* − *S*
by (*metis Diff_iff* ‹*l* ∉ *S*› ‹*closedin X S*› *closedin_def lim limitin_def*)
with *ev eventually_elim2 trivial_limit_def* **show** *False*
by *force*
qed

5.6.18 Quasi-components

definition *quasi_component_of* :: *'a topology* ⇒ *'a* ⇒ *'a* ⇒ *bool*
where
quasi_component_of X x y ≡

$x \in$ *topspace* $X \wedge y \in$ *topspace* $X \wedge$
$(\forall T.$ *closedin* $X\ T \wedge$ *openin* $X\ T \longrightarrow (x \in T \longleftrightarrow y \in T))$

abbreviation *quasi_component_of_set* $S\ x \equiv$ *Collect* (*quasi_component_of* $S\ x$)

definition *quasi_components_of* :: $'a$ *topology* $\Rightarrow$ ($'a$ *set*) *set*
where
quasi_components_of $X =$ *quasi_component_of_set* X ‘ *topspace* X

lemma *quasi_component_in_topspace*:
quasi_component_of $X\ x\ y \Longrightarrow x \in$ *topspace* $X \wedge y \in$ *topspace* X
by (*simp add*: *quasi_component_of_def*)

lemma *quasi_component_of_refl* [*simp*]:
quasi_component_of $X\ x\ x \longleftrightarrow x \in$ *topspace* X
by (*simp add*: *quasi_component_of_def*)

lemma *quasi_component_of_sym*:
quasi_component_of $X\ x\ y \longleftrightarrow$ *quasi_component_of* $X\ y\ x$
by (*meson quasi_component_of_def*)

lemma *quasi_component_of_trans*:
⟦*quasi_component_of* $X\ x\ y$; *quasi_component_of* $X\ y\ z$⟧ $\Longrightarrow$ *quasi_component_of* $X\ x\ z$
by (*simp add*: *quasi_component_of_def*)

lemma *quasi_component_of_subset_topspace*:
quasi_component_of_set $X\ x \subseteq$ *topspace* X
using *quasi_component_of_def* **by** *fastforce*

lemma *quasi_component_of_eq_empty*:
quasi_component_of_set $X\ x = \{\} \longleftrightarrow (x \notin$ *topspace* $X)$
using *quasi_component_of_def* **by** *fastforce*

lemma *quasi_component_of*:
quasi_component_of $X\ x\ y \longleftrightarrow$
$x \in$ *topspace* $X \wedge y \in$ *topspace* $X \wedge (\forall T.\ x \in T \wedge$ *closedin* $X\ T \wedge$ *openin* $X\ T \longrightarrow y \in T)$
unfolding *quasi_component_of_def* **by** (*metis Diff_iff closedin_def openin_closedin_eq*)

lemma *quasi_component_of_alt*:
quasi_component_of $X\ x\ y \longleftrightarrow$
$x \in$ *topspace* $X \wedge y \in$ *topspace* $X \wedge$
$\neg (\exists U\ V.$ *openin* $X\ U \wedge$ *openin* $X\ V \wedge U \cup V =$ *topspace* $X \wedge$ *disjnt* $U\ V \wedge x \in U \wedge y \in V)$
(**is** *?lhs* = *?rhs*)
proof
show *?lhs* $\Longrightarrow$ *?rhs*

unfolding *quasi_component_of_def*
by (*metis disjnt_iff separatedin_full separatedin_open_sets*)
show *?rhs* $\Longrightarrow$ *?lhs*
unfolding *quasi_component_of_def*
by (*metis Diff_disjoint Diff_iff Un_Diff_cancel closedin_def disjnt_def inf_commute sup.orderE sup_commute*)
qed

lemma *quasi_components_lepoll_topspace*: *quasi_components_of X* $\lesssim$ *topspace X*
by (*simp add*: *image_lepoll quasi_components_of_def*)

lemma *quasi_component_of_separated*:
quasi_component_of X x y $\longleftrightarrow$
$x \in$ *topspace X* $\wedge$ $y \in$ *topspace X* $\wedge$
$\neg$ ($\exists$ *U V*. *separatedin X U V* $\wedge$ $U \cup V =$ *topspace X* $\wedge$ $x \in U \wedge y \in V$)
by (*meson quasi_component_of_alt separatedin_full separatedin_open_sets*)

lemma *quasi_component_of_subtopology*:
quasi_component_of (*subtopology X s*) *x y* $\Longrightarrow$ *quasi_component_of X x y*
unfolding *quasi_component_of_def*
by (*simp add*: *closedin_subtopology*) (*metis Int_iff inf_commute openin_subtopology_Int2*)

lemma *quasi_component_of_mono*:
quasi_component_of (*subtopology X S*) *x y* $\wedge$ $S \subseteq T$
$\Longrightarrow$ *quasi_component_of* (*subtopology X T*) *x y*
by (*metis inf.absorb_iff2 quasi_component_of_subtopology subtopology_subtopology*)

lemma *quasi_component_of_equiv*:
quasi_component_of X x y $\longleftrightarrow$
$x \in$ *topspace X* $\wedge$ $y \in$ *topspace X* $\wedge$ *quasi_component_of X x* = *quasi_component_of X y*
using *quasi_component_of_def* **by** *fastforce*

lemma *quasi_component_of_disjoint* [*simp*]:
disjnt (*quasi_component_of_set X x*) (*quasi_component_of_set X y*) $\longleftrightarrow$ $\neg$ (*quasi_component_of X x y*)
by (*metis disjnt_iff quasi_component_of_equiv mem_Collect_eq*)

lemma *quasi_component_of_eq*:
quasi_component_of X x = *quasi_component_of X y* $\longleftrightarrow$
($x \notin$ *topspace X* $\wedge$ $y \notin$ *topspace X*)
$\vee$ $x \in$ *topspace X* $\wedge$ $y \in$ *topspace X* $\wedge$ *quasi_component_of X x y*
by (*metis Collect_empty_eq_bot quasi_component_of_eq_empty quasi_component_of_equiv*)

lemma *topspace_imp_quasi_components_of*:
assumes $x \in$ *topspace X*
obtains *C* **where** $C \in$ *quasi_components_of X x* $\in C$
by (*metis assms imageI mem_Collect_eq quasi_component_of_refl quasi_components_of_def*)

lemma *Union_quasi_components_of*: $\bigcup$ (*quasi_components_of X*) = *topspace X*
by (*auto simp*: *quasi_components_of_def quasi_component_of_def*)

lemma *pairwise_disjoint_quasi_components_of*:
pairwise disjnt (*quasi_components_of X*)
by (*auto simp*: *quasi_components_of_def quasi_component_of_def disjoint_def*)

lemma *complement_quasi_components_of_Union*:
assumes *C* ∈ *quasi_components_of X*
shows *topspace X* − *C* = $\bigcup$ (*quasi_components_of X* − {*C*}) (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs*
using *Union_quasi_components_of* **by** *fastforce*
show *?rhs* ⊆ *?lhs*
using *assms*
using *quasi_component_of_equiv* **by** (*fastforce simp add*: *quasi_components_of_def image_iff subset_iff*)
qed

lemma *nonempty_quasi_components_of*:
C ∈ *quasi_components_of X* ⟹ *C* ≠ {}
by (*metis imageE quasi_component_of_eq_empty quasi_components_of_def*)

lemma *quasi_components_of_subset*:
C ∈ *quasi_components_of X* ⟹ *C* ⊆ *topspace X*
using *Union_quasi_components_of* **by** *force*

lemma *quasi_component_in_quasi_components_of*:
quasi_component_of_set X a ∈ *quasi_components_of X* ⟷ *a* ∈ *topspace X*
by (*metis* (*no_types*, *lifting*) *image_iff quasi_component_of_eq_empty quasi_components_of_def*)

lemma *quasi_components_of_eq_empty* [*simp*]:
quasi_components_of X = {} ⟷ *X* = *trivial_topology*
by (*simp add*: *quasi_components_of_def*)

lemma *quasi_components_of_empty_space* [*simp*]:
quasi_components_of trivial_topology = {}
by *simp*

lemma *quasi_component_of_set*:
quasi_component_of_set X x =
(*if x* ∈ *topspace X*
then $\bigcap$ {*t*. *closedin X t* ∧ *openin X t* ∧ *x* ∈ *t*}
else {})
by (*auto simp*: *quasi_component_of*)

lemma *closedin_quasi_component_of*: *closedin X* (*quasi_component_of_set X*

x)
by (*auto simp*: *quasi_component_of_set*)

lemma *closedin_quasi_components_of*:
$C \in$ *quasi_components_of* $X \Longrightarrow$ *closedin* X C
by (*auto simp*: *quasi_components_of_def closedin_quasi_component_of*)

lemma *openin_finite_quasi_components*:
⟦*finite*(*quasi_components_of* X); $C \in$ *quasi_components_of* X⟧ $\Longrightarrow$ *openin* X C
apply (*simp add:openin_closedin_eq quasi_components_of_subset complement_quasi_components_of*
by (*meson DiffD1 closedin_Union closedin_quasi_components_of finite_Diff*)

lemma *quasi_component_of_eq_overlap*:
quasi_component_of X x = *quasi_component_of* X y $\longleftrightarrow$
($x \notin$ *topspace* $X \wedge y \notin$ *topspace* X) $\vee$
$\neg$ (*quasi_component_of_set* X $x \cap$ *quasi_component_of_set* X $y = \{\}$)
using *quasi_component_of_equiv* **by** *fastforce*

lemma *quasi_component_of_nonoverlap*:
quasi_component_of_set X $x \cap$ *quasi_component_of_set* X $y = \{\} \longleftrightarrow$
($x \notin$ *topspace* X) $\vee$ ($y \notin$ *topspace* X) $\vee$
$\neg$ (*quasi_component_of* X x = *quasi_component_of* X y)
by (*metis inf.idem quasi_component_of_eq_empty quasi_component_of_eq_overlap*)

lemma *quasi_component_of_overlap*:
$\neg$ (*quasi_component_of_set* X $x \cap$ *quasi_component_of_set* X $y = \{\}$) $\longleftrightarrow$
$x \in$ *topspace* $X \wedge y \in$ *topspace* $X \wedge$ *quasi_component_of* X x = *quasi_component_of* X y
by (*meson quasi_component_of_nonoverlap*)

lemma *quasi_components_of_disjoint*:
⟦$C \in$ *quasi_components_of* X; $D \in$ *quasi_components_of* X⟧ $\Longrightarrow$ *disjnt* C D $\longleftrightarrow$ $C \neq D$
by (*metis disjnt_self_iff_empty nonempty_quasi_components_of pairwiseD pairwise_disjoint_quasi_components_of*)

lemma *quasi_components_of_overlap*:
⟦$C \in$ *quasi_components_of* X; $D \in$ *quasi_components_of* X⟧ $\Longrightarrow$ $\neg$ ($C \cap D = \{\}$) $\longleftrightarrow$ $C = D$
by (*metis disjnt_def quasi_components_of_disjoint*)

lemma *pairwise_separated_quasi_components_of*:
pairwise (*separatedin* X) (*quasi_components_of* X)
by (*metis closedin_quasi_components_of pairwise_def pairwise_disjoint_quasi_components_of separatedin_closed_sets*)

lemma *finite_quasi_components_of_finite*:
finite(*topspace* X) $\Longrightarrow$ *finite*(*quasi_components_of* X)
by (*simp add*: *Union_quasi_components_of finite_UnionD*)

lemma *connected_imp_quasi_component_of*:
assumes *connected_component_of X x y*
shows *quasi_component_of X x y*
proof −
have *x* ∈ *topspace X y* ∈ *topspace X*
by (*meson assms connected_component_of_equiv*)+
with *assms* **show** *?thesis*
apply (*clarsimp simp add*: *quasi_component_of connected_component_of_def*)
by (*meson connectedin_clopen_cases disjnt_iff subsetD*)
qed

lemma *connected_component_subset_quasi_component_of*:
connected_component_of_set X x ⊆ *quasi_component_of_set X x*
using *connected_imp_quasi_component_of* **by** *force*

lemma *quasi_component_as_connected_component_Union*:
quasi_component_of_set X x =
⋃ (*connected_component_of_set X* ‘ *quasi_component_of_set X x*)
(**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs*
using *connected_component_of_refl quasi_component_of* **by** *fastforce*
show *?rhs* ⊆ *?lhs*
apply (*rule SUP_least*)
by (*simp add*: *connected_component_subset_quasi_component_of quasi_component_of_equiv*)
qed

lemma *quasi_components_as_connected_components_Union*:
assumes *C* ∈ *quasi_components_of X*
obtains $\mathcal{T}$ **where** $\mathcal{T}$ ⊆ *connected_components_of X* ⋃$\mathcal{T}$ = *C*
proof −
obtain *x* **where** *x* ∈ *topspace X* **and** *Ceq*: *C* = *quasi_component_of_set X x*
by (*metis assms imageE quasi_components_of_def*)
define $\mathcal{T}$ **where** $\mathcal{T}$ ≡ *connected_component_of_set X* ‘ *quasi_component_of_set*
X x
show *thesis*
proof
show $\mathcal{T}$ ⊆ *connected_components_of X*
by (*simp add*: $\mathcal{T}$*_def connected_components_of_def image_mono quasi_component_of_subset_topspace*)
show ⋃$\mathcal{T}$ = *C*
by (*metis* $\mathcal{T}$*_def Ceq quasi_component_as_connected_component_Union*)
qed
qed

lemma *path_imp_quasi_component_of*:
path_component_of X x y ⟹ *quasi_component_of X x y*
by (*simp add*: *connected_imp_quasi_component_of path_imp_connected_component_of*)

lemma *path_component_subset_quasi_component_of*:
path_component_of_set X x ⊆ quasi_component_of_set X x
by (*simp add*: *Collect_mono path_imp_quasi_component_of*)

lemma *connected_space_iff_quasi_component*:
connected_space X ⟷ (∀ x ∈ topspace X. ∀ y ∈ topspace X. quasi_component_of X x y)
unfolding *connected_space_clopen_in closedin_def quasi_component_of*
by *blast*

lemma *connected_space_imp_quasi_component_of*:
⟦connected_space X; a ∈ topspace X; b ∈ topspace X⟧ ⟹ quasi_component_of X a b
by (*simp add*: *connected_space_iff_quasi_component*)

lemma *connected_space_quasi_component_set*:
connected_space X ⟷ (∀ x ∈ topspace X. quasi_component_of_set X x = topspace X)
by (*metis Ball_Collect connected_space_iff_quasi_component quasi_component_of_subset_topspace subset_antisym*)

lemma *connected_space_iff_quasi_components_eq*:
connected_space X ⟷
(∀ C ∈ quasi_components_of X. ∀ D ∈ quasi_components_of X. C = D)
apply (*simp add*: *quasi_components_of_def*)
by (*metis connected_space_iff_quasi_component mem_Collect_eq quasi_component_of_equiv*)

lemma *quasi_components_of_subset_sing*:
quasi_components_of X ⊆ {S} ⟷ connected_space X ∧ (X = trivial_topology ∨ topspace X = S)
proof (*cases quasi_components_of X = {}*)
case *True*
then show *?thesis*
by (*simp add*: *subset_singleton_iff*)
next
case *False*
then show *?thesis*
apply (*simp add*: *connected_space_iff_quasi_components_eq subset_iff Ball_def*)
by (*metis False Union_quasi_components_of ccpo_Sup_singleton insert_iff is_singletonE is_singletonI'*)
qed

lemma *connected_space_iff_quasi_components_subset_sing*:
connected_space X ⟷ (∃ a. quasi_components_of X ⊆ {a})
by (*simp add*: *quasi_components_of_subset_sing*)

lemma *quasi_components_of_eq_singleton*:
quasi_components_of X = {S} ⟷
connected_space X ∧ ¬ (X = trivial_topology) ∧ S = topspace X

by (*metis empty_not_insert quasi_components_of_eq_empty quasi_components_of_subset_sing subset_singleton_iff*)

lemma *quasi_components_of_connected_space*:
connected_space X
⟹ *quasi_components_of X* = (*if X* = *trivial_topology then* {} *else* {*topspace X*})
by (*simp add*: *quasi_components_of_eq_singleton*)

lemma *separated_between_singletons*:
separated_between X {*x*} {*y*} ⟷
x ∈ *topspace X* ∧ *y* ∈ *topspace X* ∧ ¬ (*quasi_component_of X x y*)
proof (*cases x* ∈ *topspace X* ∧ *y* ∈ *topspace X*)
case *True*
then show *?thesis*
by (*auto simp add*: *separated_between_def quasi_component_of_alt*)
qed (*use separated_between_imp_subset* **in** *blast*)

lemma *quasi_component_nonseparated*:
quasi_component_of X x y ⟷ *x* ∈ *topspace X* ∧ *y* ∈ *topspace X* ∧ ¬ (*separated_between X* {*x*} {*y*})
by (*metis quasi_component_of_equiv separated_between_singletons*)

lemma *separated_between_quasi_component_pointwise_left*:
assumes *C* ∈ *quasi_components_of X*
shows *separated_between X C S* ⟷ (∃ *x* ∈ *C*. *separated_between X* {*x*} *S*) (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⟹ *?rhs*
using *assms quasi_components_of_disjoint separated_between_mono* **by** *fast-force*
next
assume *?rhs*
then obtain *y* **where** *separated_between X* {*y*} *S* **and** *y* ∈ *C*
by *metis*
with *assms* **show** *?lhs*
by (*force simp add*: *separated_between quasi_components_of_def quasi_component_of_def*)
qed

lemma *separated_between_quasi_component_pointwise_right*:
C ∈ *quasi_components_of X* ⟹ *separated_between X S C* ⟷ (∃ *x* ∈ *C*. *separated_between X S* {*x*})
by (*simp add*: *separated_between_quasi_component_pointwise_left separated_between_sym*)

lemma *separated_between_quasi_component_point*:
assumes *C* ∈ *quasi_components_of X*
shows *separated_between X C* {*x*} ⟷ *x* ∈ *topspace X* − *C* (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⟹ *?rhs*

by (*meson DiffI disjnt_insert2 insert_subset separated_between_imp_disjoint separated_between_imp_subset*)
next
assume *?rhs*
with *assms* **show** *?lhs*
unfolding *quasi_components_of_def image_iff Diff_iff separated_between_quasi_component_pointu*
[*OF assms*]
by (*metis mem_Collect_eq quasi_component_of_refl separated_between_singletons*)
qed

lemma *separated_between_point_quasi_component*:
$C \in$ *quasi_components_of* $X \Longrightarrow$ *separated_between* X $\{x\}$ $C \longleftrightarrow x \in$ *topspace* $X - C$
by (*simp add*: *separated_between_quasi_component_point separated_between_sym*)

lemma *separated_between_quasi_component_compact*:
$[\![C \in$ *quasi_components_of* X; *compactin* X $K]\!] \Longrightarrow$ (*separated_between* X C K $\longleftrightarrow$ *disjnt* C K)
unfolding *disjnt_iff*
using *compactin_subset_topspace quasi_components_of_subset separated_between_pointwise_right separated_between_quasi_component_point* **by** *fastforce*

lemma *separated_between_compact_quasi_component*:
$[\![$*compactin* X K; $C \in$ *quasi_components_of* $X]\!] \Longrightarrow$ *separated_between* X K C $\longleftrightarrow$ *disjnt* K C
using *disjnt_sym separated_between_quasi_component_compact separated_between_sym*
by *blast*

lemma *separated_between_quasi_components*:
assumes C: $C \in$ *quasi_components_of* X **and** D: $D \in$ *quasi_components_of* X
shows *separated_between* X C $D \longleftrightarrow$ *disjnt* C D (**is** *?lhs* = *?rhs*)
proof
show *?lhs* $\Longrightarrow$ *?rhs*
by (*simp add*: *separated_between_imp_disjoint*)
next
assume *?rhs*
obtain x y **where** x: C = *quasi_component_of_set* X x **and** $x \in C$
and y: D = *quasi_component_of_set* X y **and** $y \in D$
using *assms* **by** (*auto simp*: *quasi_components_of_def*)
then have *separated_between* X $\{x\}$ $\{y\}$
using ‹*disjnt* C D› *separated_between_singletons* **by** *fastforce*
with ‹$x \in C$› ‹$y \in D$› **show** *?lhs*
by (*auto simp*: *assms separated_between_quasi_component_pointwise_left separated_between_quasi_component_pointwise_right*)
qed

lemma *quasi_eq_connected_component_of_eq*:
quasi_component_of X x = *connected_component_of* X x $\longleftrightarrow$
connectedin X (*quasi_component_of_set* X x) (**is** *?lhs* = *?rhs*)

proof (*cases* $x \in$ *topspace* X)
case *True*
show *?thesis*
proof
show *?lhs* $\Longrightarrow$ *?rhs*
by (*simp add*: *connectedin_connected_component_of*)
next
assume *?rhs*
then have $\bigwedge y$. *quasi_component_of* X x y = *connected_component_of* X x y
by (*metis connected_component_of_def connected_imp_quasi_component_of mem_Collect_eq quasi_component_of_equiv*)
then show *?lhs*
by *force*
qed
next
case *False*
then show *?thesis*
by (*metis Collect_empty_eq_bot connected_component_of_eq_empty connectedin_empty quasi_component_of_eq_empty*)
qed

lemma *connected_quasi_component_of*:
assumes $C \in$ *quasi_components_of* X
shows $C \in$ *connected_components_of* $X \longleftrightarrow$ *connectedin* X C (**is** *?lhs* = *?rhs*)
proof
show *?lhs* $\Longrightarrow$ *?rhs*
using *assms*
by (*simp add*: *connectedin_connected_components_of*)
next
assume *?rhs*
with *assms* **show** *?lhs*
unfolding *quasi_components_of_def connected_components_of_def image_iff*
by (*metis quasi_eq_connected_component_of_eq*)
qed

lemma *quasi_component_of_clopen_cases*:
$[\![C \in$ *quasi_components_of* X; *closedin* X T; *openin* X $T]\!] \Longrightarrow C \subseteq T \vee$ *disjnt* C T
by (*smt* (*verit*) *disjnt_iff image_iff mem_Collect_eq quasi_component_of_def quasi_components_of_def subset_iff*)

lemma *quasi_components_of_set*:
assumes $C \in$ *quasi_components_of* X
shows $\bigcap \{T.$ *closedin* X $T \wedge$ *openin* X $T \wedge C \subseteq T\} = C$ (**is** *?lhs* = *?rhs*)
proof
have $x \in C$ **if** $x \in \bigcap \{T.$ *closedin* X $T \wedge$ *openin* X $T \wedge C \subseteq T\}$ **for** x
proof (*rule ccontr*)
assume $x \notin C$
have $x \in$ *topspace* X

```
    using assms quasi_components_of_subset that by force
  then have separated_between X C {x}
    by (simp add: ‹x ∉ C› assms separated_between_quasi_component_point)
  with that show False
    by (auto simp: separated_between)
 qed
 then show ?lhs ⊆ ?rhs
  by auto
qed blast

lemma open_quasi_eq_connected_components_of:
 assumes openin X C
 shows C ∈ quasi_components_of X ⟷ C ∈ connected_components_of X (is
?lhs = ?rhs)
proof (cases closedin X C)
 case True
 show ?thesis
 proof
  assume L: ?lhs
  have T = {} ∨ T = topspace X ∩ C
   if openin (subtopology X C) T closedin (subtopology X C) T for T
  proof −
   have C ⊆ T ∨ disjnt C T
   by (meson L True assms closedin_trans_full openin_trans_full quasi_component_of_clopen_cases
that)
   with that show ?thesis
     by (metis Int_absorb2 True closedin_imp_subset closure_of_subset_eq
disjnt_def inf_absorb2)
  qed
  with L assms show ?rhs
   by (simp add: connected_quasi_component_of connected_space_clopen_in
connectedin_def openin_subset)
 next
  assume ?rhs
 then obtain x where x ∈ topspace X and x: C = connected_component_of_set
X x
   by (metis connected_components_of_def imageE)
  have C = quasi_component_of_set X x
  using True assms connected_component_of_refl connected_imp_quasi_component_of
quasi_component_of_def x by fastforce
  then show ?lhs
   using ‹x ∈ topspace X› quasi_components_of_def by fastforce
 qed
next
 case False
 then show ?thesis
  using closedin_connected_components_of closedin_quasi_components_of by
blast
qed
```

lemma *quasi_component_of_continuous_image*:
assumes *f*: *continuous_map X Y f* **and** *qc*: *quasi_component_of X x y*
shows *quasi_component_of Y (f x) (f y)*
unfolding *quasi_component_of_def*
proof (*intro strip conjI*)
show *f x ∈ topspace Y f y ∈ topspace Y*
using *assms* **by** (*simp_all add*: *continuous_map_def quasi_component_of_def Pi_iff*)
fix *T*
assume *closedin Y T ∧ openin Y T*
with *assms* **show** (*f x ∈ T*) = (*f y ∈ T*)
by (*smt* (*verit*) *continuous_map_closedin continuous_map_def mem_Collect_eq quasi_component_of_def*)
qed

lemma *quasi_component_of_discrete_topology*:
quasi_component_of (*discrete_topology U*) *x* = (*if x ∈ U then* {*x*} *else* {})
proof −
have *quasi_component_of_set* (*discrete_topology U*) *y* = {*y*} **if** *y ∈ U* **for** *y*
using *that*
apply (*simp add*: *set_eq_iff quasi_component_of_def*)
by (*metis Set.set_insert insertE subset_insertI*)
then show *?thesis*
by (*simp add*: *quasi_component_of*)
qed

lemma *quasi_components_of_discrete_topology*:
quasi_components_of (*discrete_topology U*) = (λ*x*. {*x*}) ‘ *U*
by (*auto simp add*: *quasi_components_of_def quasi_component_of_discrete_topology*)

lemma *homeomorphic_map_quasi_component_of*:
assumes *hmf*: *homeomorphic_map X Y f* **and** *x ∈ topspace X*
shows *quasi_component_of_set Y* (*f x*) = *f* ‘ (*quasi_component_of_set X x*)
proof −
obtain *g* **where** *hmg*: *homeomorphic_map Y X g*
and *contf*: *continuous_map X Y f* **and** *contg*: *continuous_map Y X g*
and *fg*: (∀ *x ∈ topspace X*. *g*(*f x*) = *x*) ∧ (∀ *y ∈ topspace Y*. *f*(*g y*) = *y*)
by (*smt* (*verit*, *best*) *hmf homeomorphic_map_maps homeomorphic_maps_def*)
show *?thesis*
proof
show *quasi_component_of_set Y* (*f x*) ⊆ *f* ‘ *quasi_component_of_set X x*
using *quasi_component_of_continuous_image* [*OF contg*]
‹*x ∈ topspace X*› *fg image_iff quasi_component_of_subset_topspace* **by** *fastforce*
show *f* ‘ *quasi_component_of_set X x* ⊆ *quasi_component_of_set Y* (*f x*)
using *quasi_component_of_continuous_image* [*OF contf*] **by** *blast*
qed
qed

lemma *homeomorphic_map_quasi_components_of*:
assumes *homeomorphic_map X Y f*
shows *quasi_components_of Y = image (image f) (quasi_components_of X)*
using *assms*
proof −
have ∃ *x*∈*topspace X*. *quasi_component_of_set Y y* = *f* ‘ *quasi_component_of_set X x*
if *y* ∈ *topspace Y* **for** *y*
by (*metis that assms homeomorphic_imp_surjective_map homeomorphic_map_quasi_component_of image_iff*)
moreover have ∃ *x*∈*topspace Y*. *f* ‘ *quasi_component_of_set X u* = *quasi_component_of_set Y x*
if *u* ∈ *topspace X* **for** *u*
by (*metis that assms homeomorphic_imp_surjective_map homeomorphic_map_quasi_component_of imageI*)
ultimately show *?thesis*
by (*auto simp*: *quasi_components_of_def image_iff*)
qed

lemma *openin_quasi_component_of_locally_connected_space*:
assumes *locally_connected_space X*
shows *openin X* (*quasi_component_of_set X x*)
proof −
have ∗: *openin X* (*connected_component_of_set X x*)
by (*simp add*: *assms openin_connected_component_of_locally_connected_space*)
moreover have *connected_component_of_set X x* = *quasi_component_of_set X x*
using ∗ *closedin_connected_component_of connected_component_of_refl connected_imp_quasi_component_of*
quasi_component_of_def **by** *fastforce*
ultimately show *?thesis*
by *simp*
qed

lemma *openin_quasi_components_of_locally_connected_space*:
locally_connected_space X ∧ *c* ∈ *quasi_components_of X*
⟹ *openin X c*
by (*smt* (*verit*, *best*) *image_iff openin_quasi_component_of_locally_connected_space quasi_components_of_def*)

lemma *quasi_eq_connected_components_of_alt*:
quasi_components_of X = *connected_components_of X* ⟷ (∀ *C* ∈ *quasi_components_of X*. *connectedin X C*)
(**is** *?lhs* = *?rhs*)
proof
assume *R*: *?rhs*
moreover have *connected_components_of X* ⊆ *quasi_components_of X*

using *R* **unfolding** *quasi_components_of_def connected_components_of_def*
by (*force simp flip*: *quasi_eq_connected_component_of_eq*)
ultimately show *?lhs*
using *connected_quasi_component_of* **by** *blast*
qed (*use connected_quasi_component_of* **in** *blast*)

lemma *connected_subset_quasi_components_of_pointwise*:
connected_components_of X ⊆ *quasi_components_of X* ⟷
(∀ *x* ∈ *topspace X*. *quasi_component_of X x* = *connected_component_of X x*)
(**is** *?lhs* = *?rhs*)
proof
assume *L*: *?lhs*
have *connectedin X* (*quasi_component_of_set X x*) **if** *x* ∈ *topspace X* **for** *x*
proof −
have ∃ *y*∈*topspace X*. *connected_component_of_set X x* = *quasi_component_of_set X y*
using *L that* **by** (*force simp*: *quasi_components_of_def connected_components_of_def image_subset_iff*)
then show *?thesis*
by (*metis connected_component_of_equiv connectedin_connected_component_of mem_Collect_eq quasi_component_of_eq*)
qed
then show *?rhs*
by (*simp add*: *quasi_eq_connected_component_of_eq*)
qed (*simp add*: *connected_components_of_def quasi_components_of_def*)

lemma *quasi_subset_connected_components_of_pointwise*:
quasi_components_of X ⊆ *connected_components_of X* ⟷
(∀ *x* ∈ *topspace X*. *quasi_component_of X x* = *connected_component_of X x*)
by (*simp add*: *connected_quasi_component_of image_subset_iff quasi_components_of_def quasi_eq_connected_component_of_eq*)

lemma *quasi_eq_connected_components_of_pointwise*:
quasi_components_of X = *connected_components_of X* ⟷
(∀ *x* ∈ *topspace X*. *quasi_component_of X x* = *connected_component_of X x*)
using *connected_subset_quasi_components_of_pointwise quasi_subset_connected_components_of_pointwise*
by *fastforce*

lemma *quasi_eq_connected_components_of_pointwise_alt*:
quasi_components_of X = *connected_components_of X* ⟷
(∀ *x*. *quasi_component_of X x* = *connected_component_of X x*)
unfolding *quasi_eq_connected_components_of_pointwise*
by (*metis connectedin_empty quasi_component_of_eq_empty quasi_eq_connected_component_of_eq*)

lemma *quasi_eq_connected_components_of_inclusion*:
quasi_components_of X = *connected_components_of X* ⟷
connected_components_of X ⊆ *quasi_components_of X* ∨
quasi_components_of X ⊆ *connected_components_of X*
by (*simp add*: *connected_subset_quasi_components_of_pointwise dual_order.eq_iff*

quasi_subset_connected_components_of_pointwise)

lemma *quasi_eq_connected_components_of*:
finite(*connected_components_of X*) ∨
finite(*quasi_components_of X*) ∨
locally_connected_space X ∨
compact_space X ∧ (*Hausdorff_space X* ∨ *regular_space X* ∨ *normal_space X*)
⟹ *quasi_components_of X* = *connected_components_of X*
proof (*elim disjE*)
show *quasi_components_of X* = *connected_components_of X*
if *finite* (*connected_components_of X*)
unfolding *quasi_eq_connected_components_of_inclusion*
using *that open_in_finite_connected_components open_quasi_eq_connected_components_of*
by *blast*
show *quasi_components_of X* = *connected_components_of X*
if *finite* (*quasi_components_of X*)
unfolding *quasi_eq_connected_components_of_inclusion*
using *that open_quasi_eq_connected_components_of openin_finite_quasi_components*
by *blast*
show *quasi_components_of X* = *connected_components_of X*
if *locally_connected_space X*
unfolding *quasi_eq_connected_components_of_inclusion*
using *that open_quasi_eq_connected_components_of openin_quasi_components_of_locally_connect*
by *auto*
show *quasi_components_of X* = *connected_components_of X*
if *compact_space X* ∧ (*Hausdorff_space X* ∨ *regular_space X* ∨ *normal_space X*)
proof −
show *?thesis*
unfolding *quasi_eq_connected_components_of_alt*
proof (*intro strip*)
fix *C*
assume *C*: *C* ∈ *quasi_components_of X*
then have *cloC*: *closedin X C*
by (*simp add*: *closedin_quasi_components_of*)
have *normal_space X*
using *that compact_Hausdorff_or_regular_imp_normal_space* **by** *blast*
show *connectedin X C*
proof (*clarsimp simp add*: *connectedin_def connected_space_closedin_eq closedin_closed_subtopology cloC closedin_subset* [*OF cloC*])
fix *S T*
assume *S* ⊆ *C* **and** *closedin X S* **and** *S* ∩ *T* = {} **and** *SUT*: *S* ∪ *T* = *topspace X* ∩ *C*
and *T*: *T* ⊆ *C T* ≠ {} **and** *closedin X T*
with ‹*normal_space X*› **obtain** *U V* **where** *UV*: *openin X U openin X V S* ⊆ *U T* ⊆ *V disjnt U V*
by (*meson disjnt_def normal_space_def*)

```
      moreover have compactin X (topspace X − (U ∪ V))
      using UV that by (intro closedin_compact_space closedin_diff openin_Un)
auto
       ultimately have separated_between X C (topspace X − (U ∪ V)) ⟷
disjnt C (topspace X − (U ∪ V))
     by (simp add: ‹C ∈ quasi_components_of X› separated_between_quasi_component_compact)
      moreover have disjnt C (topspace X − (U ∪ V))
        using UV SUT disjnt_def by fastforce
      ultimately have separated_between X C (topspace X − (U ∪ V))
        by simp
       then obtain A B where openin X A openin X B A ∪ B = topspace X
disjnt A B C ⊆ A
                    and subB: topspace X − (U ∪ V) ⊆ B
        by (meson separated_between_def)
      have B ∪ U = topspace X − (A ∩ V)
      proof
        show B ∪ U ⊆ topspace X − A ∩ V
          using ‹openin X U› ‹disjnt U V› ‹disjnt A B› ‹openin X B› disjnt_iff
openin_closedin_eq by fastforce
        show topspace X − A ∩ V ⊆ B ∪ U
          using ‹A ∪ B = topspace X› subB by fastforce
      qed
      then have closedin X (B ∪ U)
        using ‹openin X V› ‹openin X A› by auto
      then have C ⊆ B ∪ U ∨ disjnt C (B ∪ U)
        using quasi_component_of_clopen_cases [OF C] ‹openin X U› ‹openin
X B› by blast
      with UV show S = {}
          by (metis UnE ‹C ⊆ A› ‹S ⊆ C› T ‹disjnt A B› all_not_in_conv
disjnt_Un2 disjnt_iff subset_eq)
    qed
   qed
  qed
qed


lemma quasi_eq_connected_component_of:
  finite(connected_components_of X) ∨
     finite(quasi_components_of X) ∨
     locally_connected_space X ∨
     compact_space X ∧ (Hausdorff_space X ∨ regular_space X ∨ normal_space
X)
     ⟹ quasi_component_of X x = connected_component_of X x
 by (metis quasi_eq_connected_components_of quasi_eq_connected_components_of_pointwise_alt)
```

5.6.19 Additional quasicomponent and continuum properties like Boundary Bumping

```
lemma cut_wire_fence_theorem_gen:
```

```
  assumes compact_space X and X: Hausdorff_space X ∨ regular_space X ∨
normal_space X
    and S: compactin X S and T: closedin X T
    and dis: ⋀C. connectedin X C ⟹ disjnt C S ∨ disjnt C T
  shows separated_between X S T
  proof −
  have x ∈ topspace X if x ∈ S and T = {} for x
    using that S compactin_subset_topspace by auto
  moreover have separated_between X {x} {y} if x ∈ S and y ∈ T for x y
  proof (cases x ∈ topspace X ∧ y ∈ topspace X)
    case True
    then have ¬ connected_component_of X x y
      by (meson dis connected_component_of_def disjnt_iff that)
    with True X ‹compact_space X› show ?thesis
    by (metis quasi_component_nonseparated quasi_eq_connected_component_of)
  next
    case False
    then show ?thesis
      using S T compactin_subset_topspace closedin_subset that by blast
  qed
  ultimately show ?thesis
    using assms
   by (simp add: separated_between_pointwise_left separated_between_pointwise_right

              closedin_compact_space closedin_subset)
qed

lemma cut_wire_fence_theorem:
   ⟦compact_space X; Hausdorff_space X; closedin X S; closedin X T;
     ⋀C. connectedin X C ⟹ disjnt C S ∨ disjnt C T⟧
       ⟹ separated_between X S T
  by (simp add: closedin_compact_space cut_wire_fence_theorem_gen)

lemma separated_between_from_closed_subtopology:
  assumes XC: separated_between (subtopology X C) S (X frontier_of C)
    and ST: separated_between (subtopology X C) S T
  shows separated_between X S T
proof −
  obtain U where clo: closedin (subtopology X C) U and ope: openin (subtopology
X C) U
             and S ⊆ U and sub: X frontier_of C ∪ T ⊆ topspace (subtopology X
C) − U
    by (meson assms separated_between separated_between_Un)
  then have X frontier_of C ∪ T ⊆ topspace X ∩ C − U
    by auto
  have closedin X (topspace X ∩ C)
   by (metis XC frontier_of_restrict frontier_of_subset_eq inf_le1 separated_between_imp_subset
topspace_subtopology)
  then have closedin X U
```

by (*metis clo closedin_closed_subtopology subtopology_restrict*)
moreover have *openin* (*subtopology X C*) *U* ⟷ *openin X U* ∧ *U* ⊆ *C*
using *disjnt_iff sub* **by** (*force intro*!: *openin_subset_topspace_eq*)
with *ope* **have** *openin X U*
by *blast*
moreover have *T* ⊆ *topspace X* − *U*
using *ope openin_closedin_eq sub* **by** *auto*
ultimately show *?thesis*
using ‹*S* ⊆ *U*› *separated_between* **by** *blast*
qed

lemma *separated_between_from_closed_subtopology_frontier*:
separated_between (*subtopology X T*) *S* (*X frontier_of T*)
⟹ *separated_between X S* (*X frontier_of T*)
using *separated_between_from_closed_subtopology* **by** *blast*

lemma *separated_between_from_frontier_of_closed_subtopology*:
assumes *separated_between* (*subtopology X T*) *S* (*X frontier_of T*)
shows *separated_between X S* (*topspace X* − *T*)
proof −
have *disjnt S* (*topspace X* − *T*)
using *assms disjnt_iff separated_between_imp_subset* **by** *fastforce*
then show *?thesis*
by (*metis Diff_subset assms frontier_of_complement separated_between_from_closed_subtopology separated_between_frontier_of_eq*′)
qed

lemma *separated_between_compact_connected_component*:
assumes *locally_compact_space X Hausdorff_space X*
and *C*: *C* ∈ *connected_components_of X*
and *compactin X C closedin X T disjnt C T*
shows *separated_between X C T*
proof −
have *Csub*: *C* ⊆ *topspace X*
by (*simp add*: *assms*(*4*) *compactin_subset_topspace*)
have *Hausdorff_space* (*subtopology X* (*topspace X* − *T*))
using *Hausdorff_space_subtopology assms*(*2*) **by** *blast*
moreover have *compactin* (*subtopology X* (*topspace X* − *T*)) *C*
using *assms Csub* **by** (*metis Diff_Int_distrib Diff_empty compact_imp_compactin_subtopology disjnt_def le_iff_inf*)
moreover have *locally_compact_space* (*subtopology X* (*topspace X* − *T*))
by (*meson assms closedin_def locally_compact_Hausdorff_imp_regular_space locally_compact_space_open_subset*)
ultimately
obtain *N L* **where** *openin X N compactin X L closedin X L C* ⊆ *N N* ⊆ *L*
and *Lsub*: *L* ⊆ *topspace X* − *T*
using ‹*Hausdorff_space X*› ‹*closedin X T*›
apply (*simp add*: *locally_compact_space_compact_closed_compact compactin_subtopology*)
by (*meson closedin_def compactin_imp_closedin openin_trans_full*)

```
  then have disC: disjnt C (topspace X - L)
    by (meson DiffD2 disjnt_iff subset_iff)
  have separated_between (subtopology X L) C (X frontier_of L)
  proof (rule cut_wire_fence_theorem)
    show compact_space (subtopology X L)
      by (simp add: ‹compactin X L› compact_space_subtopology)
    show Hausdorff_space (subtopology X L)
      by (simp add: Hausdorff_space_subtopology ‹Hausdorff_space X›)
    show closedin (subtopology X L) C
       by (meson ‹C ⊆ N› ‹N ⊆ L› ‹Hausdorff_space X› ‹compactin X C›
closedin_subset_topspace compactin_imp_closedin subset_trans)
    show closedin (subtopology X L) (X frontier_of L)
      by (simp add: ‹closedin X L› closedin_frontier_of closedin_subset_topspace
frontier_of_subset_closedin)
    show disjnt D C ∨ disjnt D (X frontier_of L)
      if connectedin (subtopology X L) D for D
    proof (rule ccontr)
      assume ¬ (disjnt D C ∨ disjnt D (X frontier_of L))
      moreover have connectedin X D
        using connectedin_subtopology that by blast
      ultimately show False
        using that connected_components_of_maximal [of C X D] C
        apply (simp add: disjnt_iff)
        by (metis Diff_eq_empty_iff ‹C ⊆ N› ‹N ⊆ L› ‹openin X N› disjoint_iff
frontier_of_openin_straddle_Int(2) subsetD)
    qed
  qed
  then have separated_between X (X frontier_of C) (topspace X - L)
   using separated_between_from_frontier_of_closed_subtopology separated_between_frontier_of_eq
by blast
  with ‹closedin X T›
    separated_between_frontier_of [OF Csub disC]
  show ?thesis
     unfolding separated_between by (smt (verit) Diff_iff Lsub closedin_subset
subset_iff)
qed

lemma wilder_locally_compact_component_thm:
  assumes locally_compact_space X Hausdorff_space X
    and C ∈ connected_components_of X compactin X C openin X W C ⊆ W
  obtains U V where openin X U openin X V disjnt U V U ∪ V = topspace X
C ⊆ U U ⊆ W
proof -
  have closedin X (topspace X - W)
    using ‹openin X W› by blast
  moreover have disjnt C (topspace X - W)
    using ‹C ⊆ W› disjnt_def by fastforce
  ultimately have separated_between X C (topspace X - W)
    using separated_between_compact_connected_component assms by blast
```

then show *thesis*
by (*smt* (*verit*, *del_insts*) *DiffI disjnt_iff openin_subset separated_between_def subset_iff that*)
qed

lemma *compact_quasi_eq_connected_components_of*:
assumes *locally_compact_space X Hausdorff_space X compactin X C*
shows $C \in$ *quasi_components_of X* $\longleftrightarrow$ $C \in$ *connected_components_of X*
proof −
have *compactin X* (*connected_component_of_set X x*)
if $x \in$ *topspace X compactin X* (*quasi_component_of_set X x*) **for** *x*
proof (*rule closed_compactin*)
show *compactin X* (*quasi_component_of_set X x*)
by (*simp add*: *that*)
show *connected_component_of_set X x* $\subseteq$ *quasi_component_of_set X x*
by (*simp add*: *connected_component_subset_quasi_component_of*)
show *closedin X* (*connected_component_of_set X x*)
by (*simp add*: *closedin_connected_component_of*)
qed
moreover have *connected_component_of X x* = *quasi_component_of X x*
if §: $x \in$ *topspace X compactin X* (*connected_component_of_set X x*) **for** *x*
proof −
have $\bigwedge y$. *connected_component_of X x y* $\Longrightarrow$ *quasi_component_of X x y*
by (*simp add*: *connected_imp_quasi_component_of*)
moreover have *False* **if** *non*: ¬ *connected_component_of X x y* **and** *quasi*: *quasi_component_of X x y* **for** *y*
proof −
have $y \in$ *topspace X*
by (*meson quasi_component_of_equiv that*)
then have *closedin X* {*y*}
by (*simp add*: ‹*Hausdorff_space X*› *compactin_imp_closedin*)
moreover have *disjnt* (*connected_component_of_set X x*) {*y*}
by (*simp add*: *non*)
moreover have ¬ *separated_between X* (*connected_component_of_set X x*) {*y*}
using § *quasi separated_between_pointwise_left*
by (*fastforce simp*: *quasi_component_nonseparated connected_component_of_refl*)
ultimately show *False*
using *assms* **by** (*metis* § *connected_component_in_connected_components_of separated_between_compact_connected_component*)
qed
ultimately show *?thesis*
by *blast*
qed
ultimately show *?thesis*
using ‹*compactin X C*› **unfolding** *connected_components_of_def image_iff quasi_components_of_def* **by** *metis*
qed

lemma *boundary_bumping_theorem_closed_gen*:
assumes *connected_space X locally_compact_space X Hausdorff_space X closedin X S*
S ≠ *topspace X* **and** *C*: *compactin X C C* ∈ *connected_components_of* (*subtopology X S*)
shows *C* ∩ *X frontier_of S* ≠ {}
proof
assume §: *C* ∩ *X frontier_of S* = {}
consider *C* ≠ {} *X frontier_of S* ⊆ *topspace X* | *C* ⊆ *topspace X S* = {}
using *C* **by** (*metis frontier_of_subset_topspace nonempty_connected_components_of*)
then show *False*
proof *cases*
case *1*
have *separated_between* (*subtopology X S*) *C* (*X frontier_of S*)
proof (*rule separated_between_compact_connected_component*)
show *compactin* (*subtopology X S*) *C*
using *C compact_imp_compactin_subtopology connected_components_of_subset* **by** *fastforce*
show *closedin* (*subtopology X S*) (*X frontier_of S*)
by (*simp add*: ‹*closedin X S*› *closedin_frontier_of closedin_subset_topspace frontier_of_subset_closedin*)
show *disjnt C* (*X frontier_of S*)
using § **by** (*simp add*: *disjnt_def*)
qed (*use assms Hausdorff_space_subtopology locally_compact_space_closed_subset* **in** *auto*)
then have *separated_between X C* (*X frontier_of S*)
using *separated_between_from_closed_subtopology* **by** *auto*
then have *X frontier_of S* = {}
using ‹*C* ≠ {}› ‹*connected_space X*› *connected_space_separated_between* **by** *blast*
moreover have *C* ⊆ *S*
using *C connected_components_of_subset* **by** *fastforce*
ultimately show *False*
using *1 assms* **by** (*metis closedin_subset connected_space_eq_frontier_eq_empty subset_empty*)
next
case *2*
then show *False*
using *C connected_components_of_eq_empty* **by** *fastforce*
qed
qed

lemma *boundary_bumping_theorem_closed*:
assumes *connected_space X compact_space X Hausdorff_space X closedin X S*
S ≠ *topspace X C* ∈ *connected_components_of*(*subtopology X S*)
shows *C* ∩ *X frontier_of S* ≠ {}
by (*meson assms boundary_bumping_theorem_closed_gen closedin_compact_space closedin_connected_components_of*

closedin_trans_full compact_imp_locally_compact_space)

lemma *intermediate_continuum_exists*:
assumes *connected_space X locally_compact_space X Hausdorff_space X*
and *C*: *compactin X C connectedin X C C ≠ {} C ≠ topspace X*
and *U*: *openin X U C ⊆ U*
obtains *D* **where** *compactin X D connectedin X D C ⊂ D D ⊂ U*
proof −
have *C ⊆ topspace X*
by (*simp add*: *C compactin_subset_topspace*)
with *C* **obtain** *a* **where** *a*: *a ∈ topspace X a ∉ C*
by *blast*
moreover have *compactin* (*subtopology X* (*U* − {*a*})) *C*
by (*simp add*: *C U a compact_imp_compactin_subtopology subset_Diff_insert*)
moreover have *Hausdorff_space* (*subtopology X* (*U* − {*a*}))
using *Hausdorff_space_subtopology assms*(*3*) **by** *blast*
moreover
have *locally_compact_space* (*subtopology X* (*U* − {*a*}))
by (*rule locally_compact_space_open_subset*)
(*auto simp*: *locally_compact_Hausdorff_imp_regular_space open_in_Hausdorff_delete assms*)
ultimately obtain *V K* **where** *V*: *openin X V a ∉ V V ⊆ U* **and** *K*: *compactin X K a ∉ K K ⊆ U*
and *cloK*: *closedin* (*subtopology X* (*U* − {*a*})) *K* **and** *C ⊆ V V ⊆ K*
using *locally_compact_space_compact_closed_compact* [*of subtopology X* (*U* − {*a*})] *assms*
by (*smt* (*verit*, *del_insts*) *Diff_empty compactin_subtopology open_in_Hausdorff_delete openin_open_subtopology subset_Diff_insert*)
then obtain *D* **where** *D*: *D ∈ connected_components_of* (*subtopology X K*) **and** *C ⊆ D*
using *C*
by (*metis compactin_subset_topspace connected_component_in_connected_components_of connected_component_of_maximal connectedin_subtopology subset_empty subset_eq topspace_subtopology_subset*)
show *thesis*
proof
have *cloD*: *closedin* (*subtopology X K*) *D*
by (*simp add*: *D closedin_connected_components_of*)
then have *XKD*: *compactin* (*subtopology X K*) *D*
by (*simp add*: *K closedin_compact_space compact_space_subtopology*)
then show *compactin X D*
using *compactin_subtopology_imp_compact* **by** *blast*
show *connectedin X D*
using *D connectedin_connected_components_of connectedin_subtopology* **by** *blast*
have *K ≠ topspace X*
using *K a* **by** *blast*

```
    moreover have V ⊆ X interior_of K
      by (simp add: ‹openin X V› ‹V ⊆ K› interior_of_maximal)
    ultimately have C ≠ D
      using boundary_bumping_theorem_closed_gen [of X K C] D ‹C ⊆ V›
      by (auto simp add: assms K compactin_imp_closedin frontier_of_def)
    then show C ⊂ D
      using ‹C ⊆ D› by blast
    have D ⊆ U
      using K(3) ‹closedin (subtopology X K) D› closedin_imp_subset by blast
    moreover have D ≠ U
      using K XKD ‹C ⊂ D› assms
    by (metis ‹K ≠ topspace X› cloD closedin_imp_subset compactin_imp_closedin
connected_space_clopen_in
              inf_bot_left inf_le2 subset_antisym)
    ultimately
    show D ⊂ U by blast
  qed
qed

lemma boundary_bumping_theorem_gen:
  assumes X: connected_space X locally_compact_space X Hausdorff_space X
   and S ⊂ topspace X and C: C ∈ connected_components_of(subtopology X S)
   and compC: compactin X (X closure_of C)
 shows X frontier_of C ∩ X frontier_of S ≠ {}
proof −
  have Csub: C ⊆ topspace X C ⊆ S and connectedin X C
    using C connectedin_connected_components_of connectedin_subset_topspace
connectedin_subtopology
    by fastforce+
  have C ≠ {}
    using C nonempty_connected_components_of by blast
  obtain X interior_of C ⊆ X interior_of S X closure_of C ⊆ X closure_of S
    by (simp add: Csub closure_of_mono interior_of_mono)
  moreover have False if X closure_of C ⊆ X interior_of S
  proof −
    have X closure_of C = C
      by (meson C closedin_connected_component_of_subtopology closure_of_eq
interior_of_subset order_trans that)
    with that have C ⊆ X interior_of S
      by simp
   then obtain D where  compactin X D and connectedin X D and C ⊂ D and
D ⊂ X interior_of S
      using intermediate_continuum_exists assms  ‹X closure_of C = C› compC
Csub
      by (metis ‹C ≠ {}› ‹connectedin X C› openin_interior_of psubsetE)
    then have D ⊆ C
    by (metis C ‹C ≠ {}› connected_components_of_maximal connectedin_subtopology
disjnt_def inf.orderE interior_of_subset order_trans psubsetE)
    then show False
```

using ‹$C \subset D$› **by** *blast*
qed
ultimately show *?thesis*
by (*smt* (*verit*, *ccfv_SIG*) *DiffI disjoint_iff_not_equal frontier_of_def subset_eq*)
qed

lemma *boundary_bumping_theorem*:
⟦*connected_space X*; *compact_space X*; *Hausdorff_space X*; $S \subset$ *topspace X*;
$C \in$ *connected_components_of* (*subtopology X S*)⟧
$\Longrightarrow X$ *frontier_of* $C \cap X$ *frontier_of* $S \neq \{\}$
by (*simp add*: *boundary_bumping_theorem_gen closedin_compact_space compact_imp_locally_compact_space*)

5.6.20 Compactly generated spaces (k-spaces)

These don't have to be Hausdorff

definition *k_space* **where**
k_space X ≡
$\forall S.\ S \subseteq$ *topspace X* ⟶
(*closedin X S* ⟷ ($\forall K.$ *compactin X K* ⟶ *closedin* (*subtopology X K*) ($K \cap S$)))

lemma *k_space*:
k_space X ⟷
($\forall S.\ S \subseteq$ *topspace X* ∧
($\forall K.$ *compactin X K* ⟶ *closedin* (*subtopology X K*) ($K \cap S$)) ⟶ *closedin X S*)
by (*metis closedin_subtopology inf_commute k_space_def*)

lemma *k_space_open*:
k_space X ⟷
($\forall S.\ S \subseteq$ *topspace X* ∧
($\forall K.$ *compactin X K* ⟶ *openin* (*subtopology X K*) ($K \cap S$)) ⟶ *openin X S*)
proof −
have *openin X S*
if *k_space X S* ⊆ *topspace X*
and $\forall K.$ *compactin X K* ⟶ *openin* (*subtopology X K*) ($K \cap S$) **for** *S*
using *that* **unfolding** *k_space openin_closedin_eq*
by (*metis Diff_Int_distrib2 Diff_subset inf_commute topspace_subtopology*)
moreover have *k_space X*
if $\forall S.\ S \subseteq$ *topspace X* ∧ ($\forall K.$ *compactin X K* ⟶ *openin* (*subtopology X K*) ($K \cap S$)) ⟶ *openin X S*
unfolding *k_space openin_closedin_eq*
by (*simp add*: *Diff_Int_distrib closedin_def inf_commute that*)
ultimately show *?thesis*
by *blast*
qed

lemma *k_space_alt*:
  k_space X $\longleftrightarrow$
   ($\forall$ *S*. *S* $\subseteq$ *topspace X*
       $\longrightarrow$ (*openin X S* $\longleftrightarrow$ ($\forall$ *K*. *compactin X K* $\longrightarrow$ *openin* (*subtopology X K*) (*K* $\cap$ *S*))))
  by (*meson k_space_open openin_subtopology_Int2*)

lemma *k_space_quotient_map_image*:
  assumes *q*: *quotient_map X Y q* **and** *X*: *k_space X*
  shows *k_space Y*
  unfolding *k_space*
proof *clarify*
  fix *S*
  assume *S* $\subseteq$ *topspace Y* **and** *S*: $\forall$ *K*. *compactin Y K* $\longrightarrow$ *closedin* (*subtopology Y K*) (*K* $\cap$ *S*)
  then have *iff*: *closedin X* {*x* $\in$ *topspace X*. *q x* $\in$ *S*} $\longleftrightarrow$ *closedin Y S*
    using *q quotient_map_closedin* **by** *fastforce*
  have *closedin* (*subtopology X K*) (*K* $\cap$ {*x* $\in$ *topspace X*. *q x* $\in$ *S*}) **if** *compactin X K* **for** *K*
  proof −
    have {*x* $\in$ *topspace X*. *q x* $\in$ *q* ‘ *K*} $\cap$ *K* = *K*
      using *compactin_subset_topspace that* **by** *blast*
    then have ∗: *subtopology X K* = *subtopology* (*subtopology X* {*x* $\in$ *topspace X*. *q x* $\in$ *q* ‘ *K*}) *K*
      by (*simp add*: *subtopology_subtopology*)
    have ∗∗: *K* $\cap$ {*x* $\in$ *topspace X*. *q x* $\in$ *S*} =
            K $\cap$ {*x* $\in$ *topspace* (*subtopology X* {*x* $\in$ *topspace X*. *q x* $\in$ *q* ‘ *K*}). *q x* $\in$ *q* ‘ *K* $\cap$ *S*}
      by *auto*
    have *K* $\subseteq$ *topspace X*
      by (*simp add*: *compactin_subset_topspace that*)
    show *?thesis*
      unfolding ∗ ∗∗
   proof (*intro closedin_continuous_map_preimage closedin_subtopology_Int_closed*)
       show *continuous_map* (*subtopology X* {*x* $\in$ *topspace X*. *q x* $\in$ *q* ‘ *K*}) (*subtopology Y* (*q* ‘ *K*)) *q*
     by (*auto simp add*: *continuous_map_in_subtopology continuous_map_from_subtopology q quotient_imp_continuous_map*)
      show *closedin* (*subtopology Y* (*q* ‘ *K*)) (*q* ‘ *K* $\cap$ *S*)
       by (*meson S image_compactin q quotient_imp_continuous_map that*)
    qed
  qed
  then have *closedin X* {*x* $\in$ *topspace X*. *q x* $\in$ *S*}
    by (*metis* (*no_types*, *lifting*) *X k_space mem_Collect_eq subsetI*)
  with *iff* **show** *closedin Y S* **by** *simp*
qed

lemma *k_space_retraction_map_image*:

⟦*retraction_map X Y r*; *k_space X*⟧ ⟹ *k_space Y*
using *k_space_quotient_map_image retraction_imp_quotient_map* **by** *blast*

lemma *homeomorphic_k_space*:
X homeomorphic_space Y ⟹ *k_space X* ⟷ *k_space Y*
by (*meson homeomorphic_map_def homeomorphic_space homeomorphic_space_sym k_space_quotient_map_image*)

lemma *k_space_perfect_map_image*:
⟦*k_space X*; *perfect_map X Y f*⟧ ⟹ *k_space Y*
using *k_space_quotient_map_image perfect_imp_quotient_map* **by** *blast*

lemma *locally_compact_imp_k_space*:
assumes *locally_compact_space X*
shows *k_space X*
unfolding *k_space*
proof *clarify*
fix *S*
assume *S* ⊆ *topspace X* **and** *S*: ∀ *K*. *compactin X K* ⟶ *closedin* (*subtopology X K*) (*K* ∩ *S*)
have *False* **if** *non*: ¬ (*X closure_of S* ⊆ *S*)
proof −
obtain *x* **where** *x* ∈ *X closure_of S x* ∉ *S*
using *non* **by** *blast*
then have *x* ∈ *topspace X*
by (*simp add*: *in_closure_of*)
then obtain *K U* **where** *openin X U compactin X K x* ∈ *U U* ⊆ *K*
by (*meson assms locally_compact_space_def*)
then show *False*
using ‹*x* ∈ *X closure_of S*› *openin_Int_closure_of_eq* [*OF* ‹*openin X U*›]
by (*smt* (*verit*, *ccfv_threshold*) *Int_iff S* ‹*x* ∉ *S*› *closedin_Int_closure_of inf.orderE inf_assoc*)
qed
then show *closedin X S*
using *S* ‹*S* ⊆ *topspace X*› *closure_of_subset_eq* **by** *blast*
qed

lemma *compact_imp_k_space*:
compact_space X ⟹ *k_space X*
by (*simp add*: *compact_imp_locally_compact_space locally_compact_imp_k_space*)

lemma *k_space_discrete_topology*: *k_space*(*discrete_topology U*)
by (*simp add*: *k_space_open*)

lemma *k_space_closed_subtopology*:
assumes *k_space X closedin X C*
shows *k_space* (*subtopology X C*)
unfolding *k_space compactin_subtopology*
proof *clarsimp*

fix S
assume *Ssub*: $S \subseteq$ *topspace* X $S \subseteq C$
and S: $\forall K.$ *compactin* X $K \wedge K \subseteq C \longrightarrow$ *closedin* (*subtopology* (*subtopology* X C) K) ($K \cap S$)
have *closedin* (*subtopology* X K) ($K \cap S$) **if** *compactin* X K **for** K
proof −
have *closedin* (*subtopology* (*subtopology* X C) ($K \cap C$)) (($K \cap C$) $\cap S$)
by (*simp add*: S ‹*closedin* X C› *compact_Int_closedin that*)
then show *?thesis*
using ‹*closedin* X C› *Ssub* **by** (*auto simp add*: *closedin_subtopology*)
qed
then show *closedin* (*subtopology* X C) S
by (*metis Ssub* ‹*k_space* X› *closedin_subset_topspace k_space_def*)
qed

lemma *k_space_subtopology*:
assumes *1*: $\bigwedge T.$ $[\![T \subseteq$ *topspace* X; $T \subseteq S$; $\bigwedge K.$ *compactin* X $K \Longrightarrow$ *closedin* (*subtopology* X ($K \cap S$)) ($K \cap T$)$]\!] \Longrightarrow$ *closedin* (*subtopology* X S) T
assumes *2*: $\bigwedge K.$ *compactin* X $K \Longrightarrow$ *k_space*(*subtopology* X ($K \cap S$))
shows *k_space* (*subtopology* X S)
unfolding *k_space*
proof (*intro conjI strip*)
fix U
assume §: $U \subseteq$ *topspace* (*subtopology* X S) $\wedge$ ($\forall K.$ *compactin* (*subtopology* X S) $K \longrightarrow$ *closedin* (*subtopology* (*subtopology* X S) K) ($K \cap U$))
have *closedin* (*subtopology* X ($K \cap S$)) ($K \cap U$) **if** *compactin* X K **for** K
proof −
have $K \cap U \subseteq$ *topspace* (*subtopology* X ($K \cap S$))
using § **by** *auto*
moreover
have $\bigwedge K'.$ *compactin* (*subtopology* X ($K \cap S$)) $K' \Longrightarrow$ *closedin* (*subtopology* (*subtopology* X ($K \cap S$)) K') ($K' \cap K \cap U$)
by (*metis* § *compactin_subtopology inf.orderE inf_commute subtopology_subtopology*)
ultimately show *?thesis*
by (*metis* (*no_types*, *opaque_lifting*) *2 inf.assoc k_space_def that*)
qed
then show *closedin* (*subtopology* X S) U
using *1* § **by** *auto*
qed

lemma *k_space_subtopology_open*:
assumes *1*: $\bigwedge T.$ $[\![T \subseteq$ *topspace* X; $T \subseteq S$; $\bigwedge K.$ *compactin* X $K \Longrightarrow$ *openin* (*subtopology* X ($K \cap S$)) ($K \cap T$)$]\!] \Longrightarrow$ *openin* (*subtopology* X S) T
assumes *2*: $\bigwedge K.$ *compactin* X $K \Longrightarrow$ *k_space*(*subtopology* X ($K \cap S$))
shows *k_space* (*subtopology* X S)
unfolding *k_space_open*
proof (*intro conjI strip*)

```
  fix U
  assume §: U ⊆ topspace (subtopology X S) ∧ (∀K. compactin (subtopology X S)
K ⟶ openin (subtopology (subtopology X S) K) (K ∩ U))
  have openin (subtopology X (K ∩ S)) (K ∩ U) if compactin X K for K
  proof −
    have K ∩ U ⊆ topspace (subtopology X (K ∩ S))
      using § by auto
    moreover
     have ⋀K'. compactin (subtopology X (K ∩ S)) K' ⟹ openin (subtopology
(subtopology X (K ∩ S)) K') (K' ∩ K ∩ U)
    by (metis § compactin_subtopology inf.orderE inf_commute subtopology_subtopology)
    ultimately show ?thesis
      by (metis (no_types, opaque_lifting) 2 inf.assoc k_space_open that)
  qed
  then show openin (subtopology X S) U
    using 1 § by auto
qed


lemma k_space_open_subtopology_aux:
  assumes kc_space X compact_space X openin X V
  shows k_space (subtopology X V)
proof (clarsimp simp: k_space subtopology_subtopology compactin_subtopology Int_absorb1)
  fix S
  assume S ⊆ topspace X
    and S ⊆ V
    and S: ∀K. compactin X K ∧ K ⊆ V ⟶ closedin (subtopology X K) (K ∩
S)
  then have V ⊆ topspace X
    using assms openin_subset by blast
  have S = V ∩ ((topspace X − V) ∪ S)
    using ‹S ⊆ V› by auto
  moreover have closedin (subtopology X V) (V ∩ ((topspace X − V) ∪ S))
  proof (intro closedin_subtopology_Int_closed compactin_imp_closedin_gen ‹kc_space
X›)
    show compactin X (topspace X − V ∪ S)
      unfolding compactin_def
    proof (intro conjI strip)
      show topspace X − V ∪ S ⊆ topspace X
        by (simp add: ‹S ⊆ topspace X›)
      fix 𝒰
      assume 𝒰: Ball 𝒰 (openin X) ∧ topspace X − V ∪ S ⊆ ⋃𝒰
      moreover
      have compactin X (topspace X − V)
        using assms closedin_compact_space by blast
      ultimately obtain 𝒢 where finite 𝒢 𝒢 ⊆ 𝒰 and 𝒢: topspace X − V ⊆ ⋃𝒢
        unfolding compactin_def using ‹V ⊆ topspace X› by (metis le_sup_iff)
      then have topspace X − ⋃𝒢 ⊆ V
        by blast
```

then have *closedin* (*subtopology X* (*topspace X* − ⋃𝒢)) ((*topspace X* − ⋃𝒢) ∩ *S*)
by (*meson S* 𝒰 ‹𝒢 ⊆ 𝒰› ‹*compact_space X*› *closedin_compact_space openin_Union openin_closedin_eq subset_iff*)
then have *compactin X* ((*topspace X* − ⋃𝒢) ∩ *S*)
by (*meson* 𝒰 ‹𝒢 ⊆ 𝒰›‹*compact_space X*› *closedin_compact_space closedin_trans_full openin_Union openin_closedin_eq subset_iff*)
then obtain ℋ **where** *finite* ℋ ℋ ⊆ 𝒰 (*topspace X* − ⋃𝒢) ∩ *S* ⊆ ⋃ℋ
unfolding *compactin_def* **by** (*smt* (*verit*, *best*) 𝒰 *inf_le2 subset_trans sup.boundedE*)
with 𝒢 **have** *topspace X* − *V* ∪ *S* ⊆ ⋃(𝒢 ∪ ℋ)
using ‹*S* ⊆ *topspace X*› **by** *auto*
then show ∃ℱ. *finite* ℱ ∧ ℱ ⊆ 𝒰 ∧ *topspace X* − *V* ∪ *S* ⊆ ⋃ℱ
by (*metis* ‹𝒢 ⊆ 𝒰› ‹ℋ ⊆ 𝒰› ‹*finite* 𝒢› ‹*finite* ℋ› *finite_Un le_sup_iff*)
qed
qed
ultimately show *closedin* (*subtopology X V*) *S*
by *metis*
qed

lemma *k_space_open_subtopology*:
assumes *X*: *kc_space X* ∨ *Hausdorff_space X* ∨ *regular_space X* **and** *k_space X openin X S*
shows *k_space*(*subtopology X S*)
proof (*rule k_space_subtopology_open*)
fix *T*
assume *T* ⊆ *topspace X*
and *T* ⊆ *S*
and *T*: ⋀*K*. *compactin X K* ⟹ *openin* (*subtopology X* (*K* ∩ *S*)) (*K* ∩ *T*)
have *openin* (*subtopology X K*) (*K* ∩ *T*) **if** *compactin X K* **for** *K*
by (*smt* (*verit*, *ccfv_threshold*) *T assms*(*3*) *inf_assoc inf_commute openin_Int openin_subtopology that*)
then show *openin* (*subtopology X S*) *T*
by (*metis* ‹*T* ⊆ *S*› ‹*T* ⊆ *topspace X*› *assms*(*2*) *k_space_alt subset_openin_subtopology*)
next
fix *K*
assume *compactin X K*
then have *KS*: *openin* (*subtopology X K*) (*K* ∩ *S*)
by (*simp add*: ‹*openin X S*› *openin_subtopology_Int2*)
have *XK*: *compact_space* (*subtopology X K*)
by (*simp add*: ‹*compactin X K*› *compact_space_subtopology*)
show *k_space* (*subtopology X* (*K* ∩ *S*))
using *X*
proof (*rule disjE*)
assume *kc_space X*
then show *k_space* (*subtopology X* (*K* ∩ *S*))
using *k_space_open_subtopology_aux* [*of subtopology X K K* ∩ *S*]
by (*simp add*: *KS XK kc_space_subtopology subtopology_subtopology*)

```
  next
    assume Hausdorff_space X ∨ regular_space X
    then have locally_compact_space (subtopology (subtopology X K) (K ∩ S))
      using locally_compact_space_open_subset Hausdorff_space_subtopology KS
XK
        compact_imp_locally_compact_space regular_space_subtopology by blast
    then show k_space (subtopology X (K ∩ S))
      by (simp add: locally_compact_imp_k_space subtopology_subtopology)
  qed
qed

lemma k_kc_space_subtopology:
  ⟦k_space X; kc_space X; openin X S ∨ closedin X S⟧ ⟹ k_space(subtopology
X S) ∧ kc_space(subtopology X S)
 by (metis k_space_closed_subtopology k_space_open_subtopology kc_space_subtopology)


lemma k_space_as_quotient_explicit:
  k_space X ⟷ quotient_map (sum_topology (subtopology X) {K. compactin X
K}) X snd
proof −
  have [simp]: {x ∈ topspace X. x ∈ K ∧ x ∈ U} = K ∩ U if U ⊆ topspace X
for K U
    using that by blast
  show ?thesis
    apply (simp add: quotient_map_def openin_sum_topology snd_image_Sigma
k_space_alt)
   by (smt (verit, del_insts) Union_iff compactin_sing inf.orderE mem_Collect_eq
singletonI subsetI)
qed

lemma k_space_as_quotient:
  fixes X :: 'a topology
  shows k_space X ⟷ (∃ q. ∃ Y:: ('a set ∗ 'a) topology. locally_compact_space
Y ∧ quotient_map Y X q)
        (is ?lhs=?rhs)
proof
  show k_space X if ?rhs
    using that k_space_quotient_map_image locally_compact_imp_k_space by
blast
next
  assume k_space X
  show ?rhs
  proof (intro exI conjI)
    show locally_compact_space (sum_topology (subtopology X) {K. compactin X
K})
    by (simp add: compact_imp_locally_compact_space compact_space_subtopology
locally_compact_space_sum_topology)
    show quotient_map (sum_topology (subtopology X) {K. compactin X K}) X
```

snd
    using ‹*k_space X*› *k_space_as_quotient_explicit* **by** *blast*
  qed
qed

lemma *k_space_prod_topology_left*:
  assumes *X*: *locally_compact_space X Hausdorff_space X* ∨ *regular_space X* **and** *k_space Y*
  shows *k_space* (*prod_topology X Y*)
proof −
  obtain *q* **and** *Z* :: (*'b set* ∗ *'b*) *topology* **where** *locally_compact_space Z* **and** *q*: *quotient_map Z Y q*
    using ‹*k_space Y*› *k_space_as_quotient* **by** *blast*
  then show *?thesis*
    using *quotient_map_prod_right* [*OF X q*] *X k_space_quotient_map_image locally_compact_imp_k_space*
        locally_compact_space_prod_topology **by** *blast*
qed

Essentially the same proof

lemma *k_space_prod_topology_right*:
  assumes *k_space X* **and** *Y*: *locally_compact_space Y Hausdorff_space Y* ∨ *regular_space Y*
  shows *k_space* (*prod_topology X Y*)
proof −
  obtain *q* **and** *Z* :: (*'a set* ∗ *'a*) *topology* **where** *locally_compact_space Z* **and** *q*: *quotient_map Z X q*
    using ‹*k_space X*› *k_space_as_quotient* **by** *blast*
  then show *?thesis*
   using *quotient_map_prod_left* [*OF Y q*] **using** *Y k_space_quotient_map_image locally_compact_imp_k_space*
        locally_compact_space_prod_topology **by** *blast*
qed

lemma *continuous_map_from_k_space*:
 assumes *k_space X* **and** *f*: ⋀*K*. *compactin X K* ⟹ *continuous_map*(*subtopology X K*) *Y f*
  shows *continuous_map X Y f*
proof −
  have ⋀*x*. *x* ∈ *topspace X* ⟹ *f x* ∈ *topspace Y*
   by (*metis compactin_absolute compactin_sing f image_compactin image_empty image_insert*)
  moreover have *closedin X* {*x* ∈ *topspace X*. *f x* ∈ *C*} **if** *closedin Y C* **for** *C*
  proof −
    have {*x* ∈ *topspace X*. *f x* ∈ *C*} ⊆ *topspace X*
      by *fastforce*
    moreover
    have *eq*: *K* ∩ {*x* ∈ *topspace X*. *f x* ∈ *C*} = {*x*. *x* ∈ *topspace*(*subtopology X K*)

$\wedge f\ x \in (f\ `\ K \cap C)\}$ **for** K
by *auto*
have *closedin* (*subtopology X K*) ($K \cap \{x \in$ *topspace X*. $f\ x \in C\}$) **if** *compactin X K* **for** K
unfolding *eq*
proof (*rule closedin_continuous_map_preimage*)
show *continuous_map* (*subtopology X K*) (*subtopology Y* (*f`K*)) *f*
by (*simp add*: *continuous_map_in_subtopology f image_mono that*)
show *closedin* (*subtopology Y* (*f`K*)) (*f* ` $K \cap C$)
using ‹*closedin Y C*› *closedin_subtopology* **by** *blast*
qed
ultimately show *?thesis*
using ‹*k_space X*› *k_space* **by** *blast*
qed
ultimately show *?thesis*
by (*simp add*: *continuous_map_closedin*)
qed

lemma *closed_map_into_k_space*:
assumes *k_space Y* **and** *fim*: $f \in$ (*topspace X*) $\to$ *topspace Y*
and *f*: $\bigwedge K$. *compactin Y K*
$\Longrightarrow$ *closed_map*(*subtopology X* $\{x \in$ *topspace X*. $f\ x \in K\}$) (*subtopology Y K*) *f*
shows *closed_map X Y f*
unfolding *closed_map_def*
proof (*intro strip*)
fix *C*
assume *closedin X C*
have *closedin* (*subtopology Y K*) ($K \cap f$ ` *C*)
if *compactin Y K* **for** *K*
proof −
have *eq*: $K \cap f$ ` $C = f$ ` ($\{x \in$ *topspace X*. $f\ x \in K\} \cap C$)
using ‹*closedin X C*› *closedin_subset* **by** *auto*
show *?thesis*
unfolding *eq*
by (*metis* (*no_types*, *lifting*) ‹*closedin X C*› *closed_map_def closedin_subtopology f inf_commute that*)
qed
then show *closedin Y* (*f* ` *C*)
using ‹*k_space Y*› **unfolding** *k_space*
by (*meson* ‹*closedin X C*› *closedin_subset order.trans fim funcset_image subset_image_iff*)
qed

Essentially the same proof

lemma *open_map_into_k_space*:
assumes *k_space Y* **and** *fim*: $f \in$ (*topspace X*) $\to$ *topspace Y*
and *f*: $\bigwedge K$. *compactin Y K*
$\Longrightarrow$ *open_map* (*subtopology X* $\{x \in$ *topspace X*. $f\ x \in K\}$) (*subtopology*

```
Y K) f
  shows open_map X Y f
  unfolding open_map_def
proof (intro strip)
  fix C
  assume openin X C
  have openin (subtopology Y K) (K ∩ f ‘ C)
    if compactin Y K for K
  proof −
    have eq: K ∩ f ‘ C = f ‘ ({x ∈ topspace X. f x ∈ K} ∩ C)
      using ‹openin X C› openin_subset by auto
    show ?thesis
      unfolding eq
     by (metis (no_types, lifting) ‹openin X C› open_map_def openin_subtopology
f inf_commute that)
  qed
  then show openin Y (f ‘ C)
    using ‹k_space Y› unfolding k_space_open
     by (meson ‹openin X C› openin_subset order.trans fim funcset_image sub-
set_image_iff)
qed

lemma quotient_map_into_k_space:
  fixes f :: 'a ⇒ 'b
  assumes k_space Y and cmf: continuous_map X Y f
    and fim: f ‘ (topspace X) = topspace Y
    and f: ⋀k. compactin Y k
                ⟹ quotient_map (subtopology X {x ∈ topspace X. f x ∈ k})
                                (subtopology Y k) f
  shows quotient_map X Y f
proof −
  have closedin Y C
    if C ⊆ topspace Y and K: closedin X {x ∈ topspace X. f x ∈ C} for C
  proof −
    have closedin (subtopology Y K) (K ∩ C) if compactin Y K for K
    proof −
      define Kf where Kf ≡ {x ∈ topspace X. f x ∈ K}
      have ∗: K ∩ C ⊆ topspace Y ∧ K ∩ C ⊆ K
        using ‹C ⊆ topspace Y› by blast
     then have eq: closedin (subtopology X Kf) (Kf ∩ {x ∈ topspace X. f x ∈ C})
=
                  closedin (subtopology Y K) (K ∩ C)
        using f [OF that] ∗ unfolding quotient_map_closedin Kf_def
       by (smt (verit, ccfv_SIG) Collect_cong Int_def compactin_subset_topspace
mem_Collect_eq that topspace_subtopology topspace_subtopology_subset)
      have dd: {x ∈ topspace X ∩ Kf. f x ∈ K ∩ C} = Kf ∩ {x ∈ topspace X. f x
∈ C}
        by (auto simp add: Kf_def)
      have closedin (subtopology X Kf) {x ∈ topspace X. x ∈ Kf ∧ f x ∈ K ∧ f x
```

```
∈ C}
      using K closedin_subtopology by (fastforce simp add: Kf_def)
    with K closedin_subtopology_Int_closed eq show ?thesis
      by blast
  qed
  then show ?thesis
    using ‹k_space Y› that unfolding k_space by blast
 qed
 moreover have closedin X {x ∈ topspace X. f x ∈ K}
  if K ⊆ topspace Y closedin Y K for K
  using that cmf continuous_map_closedin by fastforce
 ultimately show ?thesis
  unfolding quotient_map_closedin using fim by blast
qed

lemma quotient_map_into_k_space_eq:
 assumes k_space Y kc_space Y
 shows quotient_map X Y f ⟷
       continuous_map X Y f ∧ f ' (topspace X) = topspace Y ∧
       (∀K. compactin Y K
        ⟶ quotient_map (subtopology X {x ∈ topspace X. f x ∈ K}) (subtopology
Y K) f)
 using assms
 by (auto simp: kc_space_def intro: quotient_map_into_k_space quotient_map_restriction
     dest: quotient_imp_continuous_map quotient_imp_surjective_map)

lemma open_map_into_k_space_eq:
 assumes k_space Y
 shows open_map X Y f ⟷
       f ∈ (topspace X) → topspace Y ∧
       (∀k. compactin Y k
           ⟶ open_map (subtopology X {x ∈ topspace X. f x ∈ k}) (subtopology
Y k) f)
     (is ?lhs=?rhs)
proof
 show ?lhs ⟹ ?rhs
  by (simp add: open_map_imp_subset_topspace open_map_restriction)
 show ?rhs ⟹ ?lhs
  by (simp add: assms open_map_into_k_space)
qed

lemma closed_map_into_k_space_eq:
 assumes k_space Y
 shows closed_map X Y f ⟷
       f ∈ (topspace X) → topspace Y ∧
       (∀k. compactin Y k
           ⟶ closed_map (subtopology X {x ∈ topspace X. f x ∈ k}) (subtopology
Y k) f)
     (is ?lhs ⟷ ?rhs)
```

proof
show *?lhs* $\Longrightarrow$ *?rhs*
by (*simp add*: *closed_map_imp_subset_topspace closed_map_restriction*)
show *?rhs* $\Longrightarrow$ *?lhs*
by (*simp add*: *assms closed_map_into_k_space*)
qed

lemma *proper_map_into_k_space*:
assumes *k_space Y* **and** *fim*: $f \in (\mathit{topspace}\ X) \to \mathit{topspace}\ Y$
and *f*: $\bigwedge K.\ \mathit{compactin}\ Y\ K$
$\Longrightarrow \mathit{proper\_map}\ (\mathit{subtopology}\ X\ \{x \in \mathit{topspace}\ X.\ f\ x \in K\})$
$(\mathit{subtopology}\ Y\ K)\ f$
shows *proper_map X Y f*
proof −
have *closed_map X Y f*
by (*meson assms closed_map_into_k_space fim proper_map_def*)
with *f topspace_subtopology_subset* **show** *?thesis*
apply (*simp add*: *proper_map_alt*)
by (*smt* (*verit*, *best*) *Collect_cong compactin_absolute*)
qed

lemma *proper_map_into_k_space_eq*:
assumes *k_space Y*
shows *proper_map X Y f* $\longleftrightarrow$
$f \in (\mathit{topspace}\ X) \to \mathit{topspace}\ Y \wedge$
$(\forall K.\ \mathit{compactin}\ Y\ K$
$\longrightarrow \mathit{proper\_map}\ (\mathit{subtopology}\ X\ \{x \in \mathit{topspace}\ X.\ f\ x \in K\})\ (\mathit{subtopology}$
$Y\ K)\ f)$
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof
show *?lhs* $\Longrightarrow$ *?rhs*
by (*simp add*: *proper_map_imp_subset_topspace proper_map_restriction*)
show *?rhs* $\Longrightarrow$ *?lhs*
by (*simp add*: *assms funcset_image proper_map_into_k_space*)
qed

lemma *compact_imp_proper_map*:
assumes *k_space Y kc_space Y* **and** *fim*: $f \in (\mathit{topspace}\ X) \to \mathit{topspace}\ Y$
and *f*: *continuous_map X Y f* $\vee$ *kc_space X*
and *comp*: $\bigwedge K.\ \mathit{compactin}\ Y\ K \Longrightarrow \mathit{compactin}\ X\ \{x \in \mathit{topspace}\ X.\ f\ x \in K\}$
shows *proper_map X Y f*
proof (*rule compact_imp_proper_map_gen*)
fix *S*
assume $S \subseteq \mathit{topspace}\ Y$
and $\bigwedge K.\ \mathit{compactin}\ Y\ K \Longrightarrow \mathit{compactin}\ Y\ (S \cap K)$
with *assms* **show** *closedin Y S*
by (*simp add*: *closedin_subset_topspace inf_commute k_space kc_space_def*)
qed (*use assms* **in** *auto*)

lemma *proper_eq_compact_map*:
assumes *k_space Y kc_space Y*
and *f*: *continuous_map X Y f* ∨ *kc_space X*
shows *proper_map X Y f* ⟷
f ∈ (*topspace X*) → *topspace Y* ∧
(∀ *K*. *compactin Y K* ⟶ *compactin X* {*x* ∈ *topspace X*. *f x* ∈ *K*})
(**is** *?lhs* ⟷ *?rhs*)
proof
show *?lhs* ⟹ *?rhs*
using ‹*k_space Y*› *compactin_proper_map_preimage proper_map_into_k_space_eq*
by *blast*
qed (*use assms compact_imp_proper_map* **in** *auto*)

lemma *compact_imp_perfect_map*:
assumes *k_space Y kc_space Y* **and** *f* ‘ (*topspace X*) = *topspace Y*
and *continuous_map X Y f*
and ⋀*K*. *compactin Y K* ⟹ *compactin X* {*x* ∈ *topspace X*. *f x* ∈ *K*}
shows *perfect_map X Y f*
by (*simp add*: *assms compact_imp_proper_map perfect_map_def flip*: *image_subset_iff_funcset*)

end

5.7 Abstract Metric Spaces

theory *Abstract_Metric_Spaces*
imports *Elementary_Metric_Spaces Abstract_Limits Abstract_Topological_Spaces*
begin

locale *Metric_space* =
fixes *M* :: ′*a set* **and** *d* :: ′*a* ⇒ ′*a* ⇒ *real*
assumes *nonneg* [*simp*]: ⋀*x y*. *0* ≤ *d x y*
assumes *commute*: ⋀*x y*. *d x y* = *d y x*
assumes *zero* [*simp*]: ⋀*x y*. ⟦*x* ∈ *M*; *y* ∈ *M*⟧ ⟹ *d x y* = *0* ⟷ *x*=*y*
assumes *triangle*: ⋀*x y z*. ⟦*x* ∈ *M*; *y* ∈ *M*; *z* ∈ *M*⟧ ⟹ *d x z* ≤ *d x y* + *d y z*

Link with the type class version

interpretation *Met_TC*: *Metric_space UNIV dist*
by (*simp add*: *dist_commute dist_triangle Metric_space.intro*)

context *Metric_space*
begin

lemma *subspace*: *M* ′ ⊆ *M* ⟹ *Metric_space M* ′ *d*
by (*simp add*: *commute in_mono Metric_space.intro triangle*)

lemma *abs_mdist* [*simp*] : |*d x y*| = *d x y*
by *simp*

lemma *mdist_pos_less*: $[\![x \neq y; x \in M; y \in M]\!] \Longrightarrow 0 < d\ x\ y$
by (*metis less_eq_real_def nonneg zero*)

lemma *mdist_zero* [*simp*]: $x \in M \Longrightarrow d\ x\ x = 0$
by *simp*

lemma *mdist_pos_eq* [*simp*]: $[\![x \in M; y \in M]\!] \Longrightarrow 0 < d\ x\ y \longleftrightarrow x \neq y$
using *mdist_pos_less zero* **by** *fastforce*

lemma *triangle'*: $[\![x \in M; y \in M; z \in M]\!] \Longrightarrow d\ x\ z \leq d\ x\ y + d\ z\ y$
by (*simp add*: *commute triangle*)

lemma *triangle''*: $[\![x \in M; y \in M; z \in M]\!] \Longrightarrow d\ x\ z \leq d\ y\ x + d\ y\ z$
by (*simp add*: *commute triangle*)

lemma *mdist_reverse_triangle*: $[\![x \in M; y \in M; z \in M]\!] \Longrightarrow |d\ x\ y - d\ y\ z| \leq d\ x\ z$
by (*smt* (*verit*) *commute triangle*)

Open and closed balls

definition *mball* **where** $mball\ x\ r \equiv \{y.\ x \in M \wedge y \in M \wedge d\ x\ y < r\}$
definition *mcball* **where** $mcball\ x\ r \equiv \{y.\ x \in M \wedge y \in M \wedge d\ x\ y \leq r\}$

lemma *in_mball* [*simp*]: $y \in mball\ x\ r \longleftrightarrow x \in M \wedge y \in M \wedge d\ x\ y < r$
by (*simp add*: *mball_def*)

lemma *centre_in_mball_iff* [*iff*]: $x \in mball\ x\ r \longleftrightarrow x \in M \wedge 0 < r$
using *in_mball mdist_zero* **by** *force*

lemma *mball_subset_mspace*: $mball\ x\ r \subseteq M$
by *auto*

lemma *mball_eq_empty*: $mball\ x\ r = \{\} \longleftrightarrow (x \notin M) \vee r \leq 0$
by (*smt* (*verit*, *best*) *Collect_empty_eq centre_in_mball_iff mball_def nonneg*)

lemma *mball_subset*: $[\![d\ x\ y + a \leq b; y \in M]\!] \Longrightarrow mball\ x\ a \subseteq mball\ y\ b$
by (*smt* (*verit*) *commute in_mball subsetI triangle*)

lemma *disjoint_mball*: $r + r' \leq d\ x\ x' \Longrightarrow disjnt\ (mball\ x\ r)\ (mball\ x'\ r')$
by (*smt* (*verit*) *commute disjnt_iff in_mball triangle*)

lemma *mball_subset_concentric*: $r \leq s \Longrightarrow mball\ x\ r \subseteq mball\ x\ s$
by *auto*

lemma *in_mcball* [*simp*]: $y \in mcball\ x\ r \longleftrightarrow x \in M \wedge y \in M \wedge d\ x\ y \leq r$
by (*simp add*: *mcball_def*)

lemma *centre_in_mcball_iff* [*iff*]: $x \in mcball\ x\ r \longleftrightarrow x \in M \wedge 0 \leq r$
using *mdist_zero* **by** *force*

lemma *mcball_eq_empty*: *mcball x r* = {} ⟷ (*x* ∉ *M*) ∨ *r* < *0*
by (*smt* (*verit*, *best*) *Collect_empty_eq centre_in_mcball_iff empty_iff mcball_def nonneg*)

lemma *mcball_subset_mspace*: *mcball x r* ⊆ *M*
by *auto*

lemma *mball_subset_mcball*: *mball x r* ⊆ *mcball x r*
by *auto*

lemma *mcball_subset*: ⟦*d x y* + *a* ≤ *b*; *y* ∈ *M*⟧ ⟹ *mcball x a* ⊆ *mcball y b*
by (*smt* (*verit*) *in_mcball mdist_reverse_triangle subsetI*)

lemma *mcball_subset_concentric*: *r* ≤ *s* ⟹ *mcball x r* ⊆ *mcball x s*
by *force*

lemma *mcball_subset_mball*: ⟦*d x y* + *a* < *b*; *y* ∈ *M*⟧ ⟹ *mcball x a* ⊆ *mball y b*
by (*smt* (*verit*) *commute in_mball in_mcball subsetI triangle*)

lemma *mcball_subset_mball_concentric*: *a* < *b* ⟹ *mcball x a* ⊆ *mball x b*
by *force*

end

5.7.1 Metric topology

context *Metric_space*
begin

definition *mopen* **where**
mopen U ≡ *U* ⊆ *M* ∧ (∀ *x*. *x* ∈ *U* ⟶ (∃ *r*>*0*. *mball x r* ⊆ *U*))

definition *mtopology* :: *'a topology* **where**
mtopology ≡ *topology mopen*

lemma *is_topology_metric_topology* [*iff*]: *istopology mopen*
proof −
have ⋀*S T*. ⟦*mopen S*; *mopen T*⟧ ⟹ *mopen* (*S* ∩ *T*)
by (*smt* (*verit*, *del_insts*) *Int_iff in_mball mopen_def subset_eq*)
moreover have ⋀*K*. (∀ *K*∈*K*. *mopen K*) ⟶ *mopen* (⋃*K*)
using *mopen_def* **by** *fastforce*
ultimately show *?thesis*
by (*simp add*: *istopology_def*)
qed

lemma *openin_mtopology*: *openin mtopology U* ⟷ *U* ⊆ *M* ∧ (∀ *x*. *x* ∈ *U* ⟶ (∃ *r*>*0*. *mball x r* ⊆ *U*))
by (*simp add*: *mopen_def mtopology_def*)

lemma *topspace_mtopology* [*simp*]: *topspace mtopology = M*
by (*meson order.refl mball_subset_mspace openin_mtopology openin_subset openin_topspace subset_antisym zero_less_one*)

lemma *subtopology_mspace* [*simp*]: *subtopology mtopology M = mtopology*
by (*metis subtopology_topspace topspace_mtopology*)

lemma *open_in_mspace* [*iff*]: *openin mtopology M*
by (*metis openin_topspace topspace_mtopology*)

lemma *closedin_mspace* [*iff*]: *closedin mtopology M*
by (*metis closedin_topspace topspace_mtopology*)

lemma *openin_mball* [*iff*]: *openin mtopology* (*mball x r*)
proof −
have $\bigwedge y.\ [\![x \in M;\ d\ x\ y < r]\!] \Longrightarrow \exists s{>}0.\ mball\ y\ s \subseteq mball\ x\ r$
by (*metis add_diff_cancel_left' add_diff_eq commute less_add_same_cancel1 mball_subset order_refl*)
then show *?thesis*
by (*auto simp*: *openin_mtopology*)
qed

lemma *mtopology_base*:
mtopology = topology(*arbitrary union_of* ($\lambda U.\ \exists x \in M.\ \exists r{>}0.\ U = mball\ x\ r$))
proof −
have $\bigwedge S.\ \exists x\ r.\ x \in M \wedge 0 < r \wedge S = mball\ x\ r \Longrightarrow openin\ mtopology\ S$
using *openin_mball* **by** *blast*
moreover have $\bigwedge U\ x.\ [\![openin\ mtopology\ U;\ x \in U]\!] \Longrightarrow \exists B.\ (\exists x\ r.\ x \in M \wedge 0 < r \wedge B = mball\ x\ r) \wedge x \in B \wedge B \subseteq U$
by (*metis centre_in_mball_iff in_mono openin_mtopology*)
ultimately show *?thesis*
by (*smt* (*verit*) *topology_base_unique*)
qed

lemma *closedin_metric*:
closedin mtopology C $\longleftrightarrow$ $C \subseteq M \wedge (\forall x.\ x \in M - C \longrightarrow (\exists r{>}0.\ disjnt\ C\ (mball\ x\ r)))$ (**is** *?lhs = ?rhs*)
proof
show *?lhs* $\Longrightarrow$ *?rhs*
unfolding *closedin_def openin_mtopology*
by (*metis Diff_disjoint disjnt_def disjnt_subset2 topspace_mtopology*)
show *?rhs* $\Longrightarrow$ *?lhs*
unfolding *closedin_def openin_mtopology disjnt_def*
by (*metis Diff_subset Diff_triv Int_Diff Int_commute inf.absorb_iff2 mball_subset_mspace topspace_mtopology*)
qed

lemma *closedin_mcball* [*iff*]: *closedin mtopology* (*mcball x r*)
proof −
have ∃ *ra*>*0*. *disjnt* (*mcball x r*) (*mball y ra*) **if** *x* ∉ *M* **for** *y*
by (*metis disjnt_empty1 gt_ex mcball_eq_empty that*)
moreover have *disjnt* (*mcball x r*) (*mball y* (*d x y* − *r*)) **if** *y* ∈ *M d x y* > *r* **for** *y*
using *that disjnt_iff in_mball in_mcball mdist_reverse_triangle* **by** *force*
ultimately show *?thesis*
using *closedin_metric mcball_subset_mspace* **by** *fastforce*
qed

lemma *mball_iff_mcball*: (∃ *r*>*0*. *mball x r* ⊆ *U*) = (∃ *r*>*0*. *mcball x r* ⊆ *U*)
by (*meson dense mball_subset_mcball mcball_subset_mball_concentric order_trans*)

lemma *openin_mtopology_mcball*:
openin mtopology U ⟷ *U* ⊆ *M* ∧ (∀ *x*. *x* ∈ *U* ⟶ (∃ *r*. *0* < *r* ∧ *mcball x r* ⊆ *U*))
by (*simp add*: *mball_iff_mcball openin_mtopology*)

lemma *metric_derived_set_of*:
mtopology derived_set_of S = {*x* ∈ *M*. ∀ *r*>*0*. ∃ *y*∈*S*. *y*≠*x* ∧ *y* ∈ *mball x r*} (**is** *?lhs*=*?rhs*)
proof
show *?lhs* ⊆ *?rhs*
unfolding *openin_mtopology derived_set_of_def*
by *clarsimp* (*metis in_mball openin_mball openin_mtopology zero*)
show *?rhs* ⊆ *?lhs*
unfolding *openin_mtopology derived_set_of_def*
by *clarify* (*metis subsetD topspace_mtopology*)
qed

lemma *metric_closure_of*:
mtopology closure_of S = {*x* ∈ *M*. ∀ *r*>*0*. ∃ *y* ∈ *S*. *y* ∈ *mball x r*}
proof −
have ⋀*x r*. ⟦*0* < *r*; *x* ∈ *mtopology closure_of S*⟧ ⟹ ∃ *y*∈*S*. *y* ∈ *mball x r*
by (*metis centre_in_mball_iff in_closure_of openin_mball topspace_mtopology*)
moreover have ⋀*x T*. ⟦*x* ∈ *M*; ∀ *r*>*0*. ∃ *y*∈*S*. *y* ∈ *mball x r*⟧ ⟹ *x* ∈ *mtopology closure_of S*
by (*smt* (*verit*) *in_closure_of in_mball openin_mtopology subsetD topspace_mtopology*)
ultimately show *?thesis*
by (*auto simp*: *in_closure_of*)
qed

lemma *metric_closure_of_alt*:
mtopology closure_of S = {*x* ∈ *M*. ∀ *r*>*0*. ∃ *y* ∈ *S*. *y* ∈ *mcball x r*}
proof −
have ⋀*x r*. ⟦∀ *r*>*0*. *x* ∈ *M* ∧ (∃ *y*∈*S*. *y* ∈ *mcball x r*); *0* < *r*⟧ ⟹ ∃ *y*∈*S*. *y* ∈ *M* ∧ *d x y* < *r*
by (*meson dense in_mcball le_less_trans*)

then show *?thesis*
by (*fastforce simp*: *metric_closure_of in_closure_of*)
qed

lemma *metric_interior_of*:
mtopology interior_of S = {*x* ∈ *M*. ∃ε>0. *mball x* ε ⊆ *S*} (**is** *?lhs*=*?rhs*)
proof
show *?lhs* ⊆ *?rhs*
using *interior_of_maximal_eq openin_mtopology* **by** *fastforce*
show *?rhs* ⊆ *?lhs*
using *interior_of_def openin_mball* **by** *fastforce*
qed

lemma *metric_interior_of_alt*:
mtopology interior_of S = {*x* ∈ *M*. ∃ε>0. *mcball x* ε ⊆ *S*}
by (*fastforce simp*: *mball_iff_mcball metric_interior_of*)

lemma *in_interior_of_mball*:
x ∈ *mtopology interior_of S* ⟷ *x* ∈ *M* ∧ (∃ε>0. *mball x* ε ⊆ *S*)
using *metric_interior_of* **by** *force*

lemma *in_interior_of_mcball*:
x ∈ *mtopology interior_of S* ⟷ *x* ∈ *M* ∧ (∃ε>0. *mcball x* ε ⊆ *S*)
using *metric_interior_of_alt* **by** *force*

lemma *Hausdorff_space_mtopology*: *Hausdorff_space mtopology*
unfolding *Hausdorff_space_def*
proof *clarify*
fix *x y*
assume *x*: *x* ∈ *topspace mtopology* **and** *y*: *y* ∈ *topspace mtopology* **and** *x* ≠ *y*
then have *gt0*: *d x y* / 2 > 0
by *auto*
have *disjnt* (*mball x* (*d x y* / 2)) (*mball y* (*d x y* / 2))
by (*simp add*: *disjoint_mball*)
then show ∃ *U V*. *openin mtopology U* ∧ *openin mtopology V* ∧ *x* ∈ *U* ∧ *y* ∈ *V* ∧ *disjnt U V*
by (*metis centre_in_mball_iff gt0 openin_mball topspace_mtopology x y*)
qed

5.7.2 Bounded sets

definition *mbounded* **where** *mbounded S* ⟷ (∃ *x B*. *S* ⊆ *mcball x B*)

lemma *mbounded_pos*: *mbounded S* ⟷ (∃ *x B*. 0 < *B* ∧ *S* ⊆ *mcball x B*)
proof −
have ∃ *x' r'*. 0 < *r'* ∧ *S* ⊆ *mcball x' r'* **if** *S* ⊆ *mcball x r* **for** *x r*
by (*metis gt_ex less_eq_real_def linorder_not_le mcball_subset_concentric order_trans that*)
then show *?thesis*

by (*auto simp*: *mbounded_def*)
qed

lemma *mbounded_alt*:
mbounded S $\longleftrightarrow$ *S* $\subseteq$ *M* $\wedge$ ($\exists$ *B*. $\forall$ *x* $\in$ *S*. $\forall$ *y* $\in$ *S*. *d x y* $\leq$ *B*)
proof −
have $\bigwedge$*x B*. *S* $\subseteq$ *mcball x B* $\Longrightarrow$ $\forall$ *x*$\in$*S*. $\forall$ *y*$\in$*S*. *d x y* $\leq$ *2* $*$ *B*
by (*smt* (*verit*, *best*) *commute in_mcball subsetD triangle*)
then show *?thesis*
apply (*auto simp*: *mbounded_def subset_iff*)
apply *blast+*
done
qed

lemma *mbounded_alt_pos*:
mbounded S $\longleftrightarrow$ *S* $\subseteq$ *M* $\wedge$ ($\exists$ *B*>*0*. $\forall$ *x* $\in$ *S*. $\forall$ *y* $\in$ *S*. *d x y* $\leq$ *B*)
by (*smt* (*verit*, *del_insts*) *gt_ex mbounded_alt*)

lemma *mbounded_subset*: $[\![$*mbounded T*; *S* $\subseteq$ *T*$]\!]$ $\Longrightarrow$ *mbounded S*
by (*meson mbounded_def order_trans*)

lemma *mbounded_subset_mspace*: *mbounded S* $\Longrightarrow$ *S* $\subseteq$ *M*
by (*simp add*: *mbounded_alt*)

lemma *mbounded*:
mbounded S $\longleftrightarrow$ *S* = {} $\vee$ ($\forall$ *x* $\in$ *S*. *x* $\in$ *M*) $\wedge$ ($\exists$ *y B*. *y* $\in$ *M* $\wedge$ ($\forall$ *x* $\in$ *S*. *d y x* $\leq$ *B*))
by (*meson all_not_in_conv in_mcball mbounded_def subset_iff*)

lemma *mbounded_empty* [*iff*]: *mbounded* {}
by (*simp add*: *mbounded*)

lemma *mbounded_mcball*: *mbounded* (*mcball x r*)
using *mbounded_def* **by** *auto*

lemma *mbounded_mball* [*iff*]: *mbounded* (*mball x r*)
by (*meson mball_subset_mcball mbounded_def*)

lemma *mbounded_insert*: *mbounded* (*insert a S*) $\longleftrightarrow$ *a* $\in$ *M* $\wedge$ *mbounded S*
proof −
have $\bigwedge$*y B*. $[\![$*y* $\in$ *M*; $\forall$ *x*$\in$*S*. *d y x* $\leq$ *B*$]\!]$
$\Longrightarrow$ $\exists$ *y*. *y* $\in$ *M* $\wedge$ ($\exists$ *B* $\geq$ *d y a*. $\forall$ *x*$\in$*S*. *d y x* $\leq$ *B*)
by (*metis order.trans nle_le*)
then show *?thesis*
by (*auto simp*: *mbounded*)
qed

lemma *mbounded_Int*: *mbounded S* $\Longrightarrow$ *mbounded* (*S* $\cap$ *T*)

by (*meson inf_le1 mbounded_subset*)

lemma *mbounded_Un*: *mbounded* (*S* ∪ *T*) ⟷ *mbounded S* ∧ *mbounded T* (**is** *?lhs=?rhs*)
proof
assume *R*: *?rhs*
show *?lhs*
proof (*cases S={}* ∨ *T={}*)
case *True* **then show** *?thesis*
using *R* **by** *auto*
next
case *False*
obtain *x y B C* **where** *S* ⊆ *mcball x B T* ⊆ *mcball y C B* > *0 C* > *0 x* ∈ *M y* ∈ *M*
using *R mbounded_pos*
by (*metis False mcball_eq_empty subset_empty*)
then have *S* ∪ *T* ⊆ *mcball x* (*B* + *C* + *d x y*)
by (*smt* (*verit*) *commute dual_order.trans le_supI mcball_subset mdist_pos_eq*)
then show *?thesis*
using *mbounded_def* **by** *blast*
qed
next
show *?lhs* ⟹ *?rhs*
using *mbounded_def* **by** *auto*
qed

lemma *mbounded_Union*:
⟦*finite* ℱ; ⋀*X*. *X* ∈ ℱ ⟹ *mbounded X*⟧ ⟹ *mbounded* (⋃ℱ)
by (*induction* ℱ *rule*: *finite_induct*) (*auto simp*: *mbounded_Un*)

lemma *mbounded_closure_of*:
mbounded S ⟹ *mbounded* (*mtopology closure_of S*)
by (*meson closedin_mcball closure_of_minimal mbounded_def*)

lemma *mbounded_closure_of_eq*:
S ⊆ *M* ⟹ (*mbounded* (*mtopology closure_of S*) ⟷ *mbounded S*)
by (*metis closure_of_subset mbounded_closure_of mbounded_subset topspace_mtopology*)

lemma *maxdist_thm*:
assumes *mbounded S*
and *x* ∈ *S*
and *y* ∈ *S*
shows *d x y* = (*SUP z*∈*S*. |*d x z* − *d z y*|)
proof −
have |*d x z* − *d z y*| ≤ *d x y* **if** *z* ∈ *S* **for** *z*
by (*metis all_not_in_conv assms mbounded mdist_reverse_triangle that*)
moreover have *d x y* ≤ *r*
if ⋀*z*. *z* ∈ *S* ⟹ |*d x z* − *d z y*| ≤ *r* **for** *r* :: *real*

using *that assms mbounded_subset_mspace mdist_zero* **by** *fastforce*
ultimately show *?thesis*
by (*intro cSup_eq* [*symmetric*]) *auto*
qed

lemma *metric_eq_thm*: $[\![S \subseteq M; x \in S; y \in S]\!] \Longrightarrow (x = y) = (\forall z{\in}S.\ d\ x\ z = d\ y\ z)$
by (*metis commute subset_iff zero*)

lemma *compactin_imp_mbounded*:
assumes *compactin mtopology S*
shows *mbounded S*
proof −
have $S \subseteq M$
and *com*: $\bigwedge\mathcal{U}.\ [\![\forall U{\in}\mathcal{U}.\ \mathit{openin}\ \mathit{mtopology}\ U; S \subseteq \bigcup\mathcal{U}]\!] \Longrightarrow \exists\mathcal{F}.\ \mathit{finite}\ \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge S \subseteq \bigcup\mathcal{F}$
using *assms* **by** (*auto simp*: *compactin_def mbounded_def*)
show *?thesis*
proof (*cases* $S = \{\}$)
case *False*
with ‹$S \subseteq M$› **obtain** a **where** $a \in S\ a \in M$
by *blast*
with ‹$S \subseteq M$› *gt_ex* **have** $S \subseteq \bigcup(\mathit{range}\ (\mathit{mball}\ a))$
by *force*
then obtain $\mathcal{F}$ **where** *finite* $\mathcal{F}$ $\mathcal{F} \subseteq \mathit{range}\ (\mathit{mball}\ a)$ $S \subseteq \bigcup\mathcal{F}$
by (*metis* (*no_types*, *opaque_lifting*) *com imageE openin_mball*)
then show *?thesis*
using *mbounded_Union mbounded_subset* **by** *fastforce*
qed *auto*
qed

end

lemma *mcball_eq_cball* [*simp*]: *Met_TC.mcball* = *cball*
by *force*

lemma *mball_eq_ball* [*simp*]: *Met_TC.mball* = *ball*
by *force*

lemma *mopen_eq_open* [*simp*]: *Met_TC.mopen* = *open*
by (*force simp*: *open_contains_ball Met_TC.mopen_def*)

lemma *limitin_iff_tendsto* [*iff*]: *limitin Met_TC.mtopology* σ x F = *tendsto* σ x F
by (*simp add*: *Met_TC.mtopology_def*)

lemma *mtopology_is_euclidean* [*simp*]: *Met_TC.mtopology* = *euclidean*

by (*simp add*: *Met_TC.mtopology_def*)

lemma *mbounded_iff_bounded* [*iff*]: *Met_TC.mbounded A* ⟷ *bounded A*
by (*metis Met_TC.mbounded UNIV_I all_not_in_conv bounded_def*)

5.7.3 Subspace of a metric space

locale *Submetric* = *Metric_space* +
fixes *A*
assumes *subset*: *A* ⊆ *M*

sublocale *Submetric* ⊆ *sub*: *Metric_space A d*
by (*simp add*: *subset subspace*)

context *Submetric*
begin

lemma *mball_submetric_eq*: *sub.mball a r* = (*if a* ∈ *A then A* ∩ *mball a r else* {})
and *mcball_submetric_eq*: *sub.mcball a r* = (*if a* ∈ *A then A* ∩ *mcball a r else* {})
using *subset* **by** *force*+

lemma *mtopology_submetric*: *sub.mtopology* = *subtopology mtopology A*
unfolding *topology_eq*
proof (*intro allI iffI*)
fix *S*
assume *openin sub.mtopology S*
then have ∃ *T*. *openin* (*subtopology mtopology A*) *T* ∧ *x* ∈ *T* ∧ *T* ⊆ *S* **if** *x* ∈ *S* **for** *x*
by (*metis mball_submetric_eq openin_mball openin_subtopology_Int2 sub.centre_in_mball_iff sub.openin_mtopology subsetD that*)
then show *openin* (*subtopology mtopology A*) *S*
by (*meson openin_subopen*)
next
fix *S*
assume *openin* (*subtopology mtopology A*) *S*
then obtain *T* **where** *openin mtopology T S* = *T* ∩ *A*
by (*meson openin_subtopology*)
then have *mopen T*
by (*simp add*: *mopen_def openin_mtopology*)
then have *sub.mopen* (*T* ∩ *A*)
unfolding *sub.mopen_def mopen_def*
by (*metis inf.coboundedI2 mball_submetric_eq Int_iff* ‹*S* = *T* ∩ *A*› *inf.bounded_iff subsetI*)
then show *openin sub.mtopology S*
using ‹*S* = *T* ∩ *A*› *sub.mopen_def sub.openin_mtopology* **by** *force*
qed

lemma *mbounded_submetric*: *sub.mbounded T* $\longleftrightarrow$ *mbounded T* $\wedge$ $T \subseteq A$
by (*meson mbounded_alt sub.mbounded_alt subset subset_trans*)

end

lemma (**in** *Metric_space*) *submetric_empty* [*iff*]: *Submetric M d* {}
proof qed *auto*

5.7.4 Abstract type of metric spaces

typedef *'a metric* = {(*M*::*'a set*,*d*). *Metric_space M d*}
morphisms *dest_metric metric*
proof −
have *Metric_space* {} ($\lambda x\ y.\ 0$)
by (*auto simp*: *Metric_space_def*)
then show *?thesis*
by *blast*
qed

definition *mspace* **where** *mspace m* ≡ *fst* (*dest_metric m*)

definition *mdist* **where** *mdist m* ≡ *snd* (*dest_metric m*)

lemma *Metric_space_mspace_mdist* [*iff*]: *Metric_space* (*mspace m*) (*mdist m*)
by (*metis Product_Type.Collect_case_prodD dest_metric mdist_def mspace_def*)

lemma *mdist_nonneg* [*simp*]: $\bigwedge x\ y.\ 0 \le$ *mdist m x y*
by (*metis Metric_space_def Metric_space_mspace_mdist*)

lemma *mdist_commute*: $\bigwedge x\ y.$ *mdist m x y* = *mdist m y x*
by (*metis Metric_space_def Metric_space_mspace_mdist*)

lemma *mdist_zero* [*simp*]: $\bigwedge x\ y.\ [\![x \in$ *mspace m*; $y \in$ *mspace m*$]\!] \Longrightarrow$ *mdist m x y* = *0* $\longleftrightarrow$ *x*=*y*
by (*meson Metric_space.zero Metric_space_mspace_mdist*)

lemma *mdist_triangle*: $\bigwedge x\ y\ z.\ [\![x \in$ *mspace m*; $y \in$ *mspace m*; $z \in$ *mspace m*$]\!]$
$\Longrightarrow$ *mdist m x z* ≤ *mdist m x y* + *mdist m y z*
by (*meson Metric_space.triangle Metric_space_mspace_mdist*)

lemma (**in** *Metric_space*) *mspace_metric*[*simp*]:
mspace (*metric* (*M*,*d*)) = *M*
by (*simp add*: *metric_inverse mspace_def subspace*)

lemma (**in** *Metric_space*) *mdist_metric*[*simp*]:
mdist (*metric* (*M*,*d*)) = *d*
by (*simp add*: *mdist_def metric_inverse subspace*)

lemma *metric_collapse* [*simp*]: *metric* (*mspace m*, *mdist m*) = *m*

by (*simp add*: *dest_metric_inverse mdist_def mspace_def*)

definition *mtopology_of* :: *'a metric* ⇒ *'a topology*
where *mtopology_of* ≡ λ*m*. *Metric_space.mtopology* (*mspace m*) (*mdist m*)

lemma *topspace_mtopology_of* [*simp*]: *topspace* (*mtopology_of m*) = *mspace m*
by (*simp add*: *Metric_space.topspace_mtopology Metric_space_mspace_mdist mtopology_of_def*)

lemma (**in** *Metric_space*) *mtopology_of* [*simp*]:
mtopology_of (*metric* (*M*,*d*)) = *mtopology*
by (*simp add*: *mtopology_of_def*)

definition *mball_of* ≡ λ*m*. *Metric_space.mball* (*mspace m*) (*mdist m*)

lemma *in_mball_of* [*simp*]: *y* ∈ *mball_of m x r* ⟷ *x* ∈ *mspace m* ∧ *y* ∈ *mspace m* ∧ *mdist m x y* < *r*
by (*simp add*: *Metric_space.in_mball mball_of_def*)

lemma (**in** *Metric_space*) *mball_of* [*simp*]:
mball_of (*metric* (*M*,*d*)) = *mball*
by (*simp add*: *mball_of_def*)

definition *mcball_of* ≡ λ*m*. *Metric_space.mcball* (*mspace m*) (*mdist m*)

lemma *in_mcball_of* [*simp*]: *y* ∈ *mcball_of m x r* ⟷ *x* ∈ *mspace m* ∧ *y* ∈ *mspace m* ∧ *mdist m x y* ≤ *r*
by (*simp add*: *Metric_space.in_mcball mcball_of_def*)

lemma (**in** *Metric_space*) *mcball_of* [*simp*]:
mcball_of (*metric* (*M*,*d*)) = *mcball*
by (*simp add*: *mcball_of_def*)

definition *euclidean_metric* ≡ *metric* (*UNIV*,*dist*)

lemma *mspace_euclidean_metric* [*simp*]: *mspace euclidean_metric* = *UNIV*
by (*simp add*: *euclidean_metric_def*)

lemma *mdist_euclidean_metric* [*simp*]: *mdist euclidean_metric* = *dist*
by (*simp add*: *euclidean_metric_def*)

lemma *mtopology_of_euclidean* [*simp*]: *mtopology_of euclidean_metric* = *euclidean*
by (*simp add*: *Met_TC.mtopology_def mtopology_of_def*)

Allows reference to the current metric space within the locale as a value

definition (**in** *Metric_space*) *Self* ≡ *metric* (*M*,*d*)

lemma (**in** *Metric_space*) *mspace_Self* [*simp*]: *mspace Self* = *M*
by (*simp add*: *Self_def*)

lemma (**in** *Metric_space*) *mdist_Self* [*simp*]: *mdist Self* = *d*
by (*simp add*: *Self_def*)

Subspace of a metric space

definition *submetric* **where**
submetric ≡ λ*m S*. *metric* (*S* ∩ *mspace m*, *mdist m*)

lemma *mspace_submetric* [*simp*]: *mspace* (*submetric m S*) = *S* ∩ *mspace m*
unfolding *submetric_def*
by (*meson Metric_space.subspace inf_le2 Metric_space_mspace_mdist Metric_space.mspace_metric*)

lemma *mdist_submetric* [*simp*]: *mdist* (*submetric m S*) = *mdist m*
unfolding *submetric_def*
by (*meson Metric_space.subspace inf_le2 Metric_space.mdist_metric Metric_space_mspace_mdist*)

lemma *submetric_UNIV* [*simp*]: *submetric m UNIV* = *m*
by (*simp add*: *submetric_def dest_metric_inverse mdist_def mspace_def*)

lemma *submetric_submetric* [*simp*]:
submetric (*submetric m S*) *T* = *submetric m* (*S* ∩ *T*)
by (*metis submetric_def Int_assoc inf_commute mdist_submetric mspace_submetric*)

lemma *submetric_mspace* [*simp*]:
submetric m (*mspace m*) = *m*
by (*simp add*: *submetric_def dest_metric_inverse mdist_def mspace_def*)

lemma *submetric_restrict*:
submetric m S = *submetric m* (*mspace m* ∩ *S*)
by (*metis submetric_mspace submetric_submetric*)

lemma *mtopology_of_submetric*: *mtopology_of* (*submetric m A*) = *subtopology* (*mtopology_of m*) *A*
proof −
interpret *Submetric mspace m mdist m A* ∩ *mspace m*
using *Metric_space_mspace_mdist Submetric.intro Submetric_axioms.intro inf_le2* **by** *blast*
have *sub.mtopology* = *subtopology* (*mtopology_of m*) *A*
by (*metis inf_commute mtopology_of_def mtopology_submetric subtopology_mspace subtopology_subtopology*)
then show *?thesis*
by (*simp add*: *submetric_def*)
qed

5.7.5 The discrete metric

locale *discrete_metric* =
fixes *M* :: *'a set*

definition (**in** *discrete_metric*) *dd* :: *'a* ⇒ *'a* ⇒ *real*
where *dd* ≡ λ*x y*::*'a*. *if x*=*y then 0 else 1*

lemma *metric_M_dd*: *Metric_space M discrete_metric.dd*
by (*simp add*: *discrete_metric.dd_def Metric_space.intro*)

sublocale *discrete_metric* ⊆ *disc*: *Metric_space M dd*
by (*simp add*: *metric_M_dd*)

lemma (**in** *discrete_metric*) *mopen_singleton*:
assumes *x* ∈ *M* **shows** *disc.mopen* {*x*}
proof −
have *disc.mball x* (*1*/*2*) ⊆ {*x*}
by (*smt* (*verit*) *dd_def disc.in_mball less_divide_eq_1_pos singleton_iff subsetI*)
with *assms* **show** *?thesis*
using *disc.mopen_def half_gt_zero_iff zero_less_one* **by** *blast*
qed

lemma (**in** *discrete_metric*) *mtopology_discrete_metric*:
disc.mtopology = *discrete_topology M*
proof −
have ⋀*x*. *x* ∈ *M* ⟹ *openin disc.mtopology* {*x*}
by (*simp add*: *disc.mtopology_def mopen_singleton*)
then show *?thesis*
by (*metis disc.topspace_mtopology discrete_topology_unique*)
qed

lemma (**in** *discrete_metric*) *discrete_ultrametric*:
dd x z ≤ *max* (*dd x y*) (*dd y z*)
by (*simp add*: *dd_def*)

lemma (**in** *discrete_metric*) *dd_le1*: *dd x y* ≤ *1*
by (*simp add*: *dd_def*)

lemma (**in** *discrete_metric*) *mbounded_discrete_metric*: *disc.mbounded S* ⟷ *S* ⊆ *M*
by (*meson dd_le1 disc.mbounded_alt*)

5.7.6 Metrizable spaces

definition *metrizable_space* **where**
metrizable_space X ≡ ∃ *M d*. *Metric_space M d* ∧ *X* = *Metric_space.mtopology M d*

lemma (**in** *Metric_space*) *metrizable_space_mtopology*: *metrizable_space mtopol-*

ogy
using *local.Metric_space_axioms metrizable_space_def* **by** *blast*

lemma (**in** *Metric_space*) *first_countable_mtopology*: *first_countable mtopology*
proof (*clarsimp simp add*: *first_countable_def*)
fix *x*
assume $x \in M$
define $\mathcal{B}$ **where** $\mathcal{B} \equiv$ *mball x ' {r ∈ ℚ. 0 < r}*
show $\exists \mathcal{B}$. *countable* $\mathcal{B} \wedge (\forall V{\in}\mathcal{B}.$ *openin mtopology V*$) \wedge (\forall U.$ *openin mtopology* $U \wedge x \in U \longrightarrow (\exists V{\in}\mathcal{B}.\ x \in V \wedge V \subseteq U))$
proof (*intro exI conjI ballI*)
show *countable* $\mathcal{B}$
by (*simp add*: $\mathcal{B}$*_def countable_rat*)
show $\forall U.$ *openin mtopology* $U \wedge x \in U \longrightarrow (\exists V{\in}\mathcal{B}.\ x \in V \wedge V \subseteq U)$
proof *clarify*
fix *U*
assume *openin mtopology U* **and** $x \in U$
then obtain *r* **where** *r>0* **and** *r*: *mball x r* $\subseteq U$
by (*meson openin_mtopology*)
then obtain *q* **where** $q \in$ *Rats* $0 < q$ $q < r$
using *Rats_dense_in_real* **by** *blast*
then show $\exists V{\in}\mathcal{B}.\ x \in V \wedge V \subseteq U$
unfolding $\mathcal{B}$*_def* **using** ‹$x \in M$› *r* **by** *fastforce*
qed
qed (*auto simp*: $\mathcal{B}$*_def*)
qed

lemma *metrizable_imp_first_countable*:
metrizable_space X $\Longrightarrow$ *first_countable X*
by (*force simp add*: *metrizable_space_def Metric_space.first_countable_mtopology*)

lemma *openin_mtopology_eq_open* [*simp*]: *openin Met_TC.mtopology* = *open*
by (*simp add*: *Met_TC.mtopology_def*)

lemma *closedin_mtopology_eq_closed* [*simp*]: *closedin Met_TC.mtopology* = *closed*
proof −
have (*euclidean*::'*a topology*) = *Met_TC.mtopology*
by (*simp add*: *Met_TC.mtopology_def*)
then show *?thesis*
using *closed_closedin* **by** *fastforce*
qed

lemma *compactin_mtopology_eq_compact* [*simp*]: *compactin Met_TC.mtopology*
= *compact*
by (*simp add*: *compactin_def compact_eq_Heine_Borel fun_eq_iff*) *meson*

lemma *metrizable_space_discrete_topology* [*simp*]:
metrizable_space(*discrete_topology U*)
by (*metis discrete_metric.mtopology_discrete_metric metric_M_dd metrizable_space_def*)

lemma *empty_metrizable_space*: *metrizable_space trivial_topology*
by *simp*

lemma *metrizable_space_subtopology*:
assumes *metrizable_space X*
shows *metrizable_space*(*subtopology X S*)
proof −
obtain *M d* **where** *Metric_space M d* **and** *X*: *X* = *Metric_space.mtopology M d*
using *assms metrizable_space_def* **by** *blast*
then interpret *Submetric M d M* ∩ *S*
by (*simp add*: *Submetric.intro Submetric_axioms_def*)
show *?thesis*
unfolding *metrizable_space_def*
by (*metis X mtopology_submetric sub.Metric_space_axioms subtopology_restrict topspace_mtopology*)
qed

lemma *homeomorphic_metrizable_space_aux*:
assumes *X homeomorphic_space Y metrizable_space X*
shows *metrizable_space Y*
proof −
obtain *M d* **where** *Metric_space M d* **and** *X*: *X* = *Metric_space.mtopology M d*
using *assms* **by** (*auto simp*: *metrizable_space_def*)
then interpret *m*: *Metric_space M d*
by *simp*
obtain *f g* **where** *hmf*: *homeomorphic_map X Y f* **and** *hmg*: *homeomorphic_map Y X g*
and *fg*: (∀ *x* ∈ *M*. *g*(*f x*) = *x*) ∧ (∀ *y* ∈ *topspace Y*. *f*(*g y*) = *y*)
using *assms X homeomorphic_maps_map homeomorphic_space_def* **by** *fastforce*
define *d′* **where** *d′ x y* ≡ *d* (*g x*) (*g y*) **for** *x y*
interpret *m′*: *Metric_space topspace Y d′*
unfolding *d′_def*
proof
show (*d* (*g x*) (*g y*) = *0*) = (*x* = *y*) **if** *x* ∈ *topspace Y y* ∈ *topspace Y* **for** *x y*
by (*metis fg X hmg homeomorphic_imp_surjective_map imageI m.topspace_mtopology m.zero that*)
show *d* (*g x*) (*g z*) ≤ *d* (*g x*) (*g y*) + *d* (*g y*) (*g z*)
if *x* ∈ *topspace Y* **and** *y* ∈ *topspace Y* **and** *z* ∈ *topspace Y* **for** *x y z*
by (*metis X that hmg homeomorphic_eq_everything_map imageI m.topspace_mtopology m.triangle*)
qed (*auto simp*: *m.nonneg m.commute*)
have *Y* = *Metric_space.mtopology* (*topspace Y*) *d′*
unfolding *topology_eq*
proof (*intro allI*)
fix *S*

```
    have openin m'.mtopology S if S: S ⊆ topspace Y and openin X (g ‘ S)
      unfolding m'.openin_mtopology
    proof (intro conjI that strip)
      fix y
      assume y ∈ S
      then obtain r where r>0 and r: m.mball (g y) r ⊆ g ‘ S
        using X ‹openin X (g ‘ S)› m.openin_mtopology using ‹y ∈ S› by auto
      then have g ‘ m'.mball y r ⊆ m.mball (g y) r
        using X d'_def hmg homeomorphic_imp_surjective_map by fastforce
      with S fg have m'.mball y r ⊆ S
        by (smt (verit, del_insts) image_iff m'.in_mball r subset_iff)
      then show ∃r>0. m'.mball y r ⊆ S
        using ‹0 < r› by blast
    qed
    moreover have openin X (g ‘ S) if ope': openin m'.mtopology S
    proof −
      have ∃r>0. m.mball (g y) r ⊆ g ‘ S if y ∈ S for y
      proof −
        have y: y ∈ topspace Y
          using m'.openin_mtopology ope' that by blast
        obtain r where r > 0 and r: m'.mball y r ⊆ S
          using ope' by (meson ‹y ∈ S› m'.openin_mtopology)
        moreover have ⋀x. ⟦x ∈ M; d (g y) x < r⟧ ⟹ ∃u. u ∈ topspace Y ∧ d'
y u < r ∧ x = g u
          using fg X d'_def hmf homeomorphic_imp_surjective_map by fastforce
        ultimately have m.mball (g y) r ⊆ g ‘ m'.mball y r
          using y by (force simp: m'.openin_mtopology)
        then show ?thesis
          using ‹0 < r› r by blast
      qed
      then show ?thesis
       using X hmg homeomorphic_imp_surjective_map m.openin_mtopology ope'
openin_subset by fastforce
    qed
    ultimately have (S ⊆ topspace Y ∧ openin X (g ‘ S)) = openin m'.mtopology
S
      using m'.topspace_mtopology openin_subset by blast
    then show openin Y S = openin m'.mtopology S
      by (simp add: m'.mopen_def homeomorphic_map_openness_eq [OF hmg])
  qed
  then show ?thesis
    using m'.metrizable_space_mtopology by force
qed

lemma homeomorphic_metrizable_space:
  assumes X homeomorphic_space Y
  shows metrizable_space X ⟷ metrizable_space Y
   using assms homeomorphic_metrizable_space_aux homeomorphic_space_sym
by metis
```

lemma *metrizable_space_retraction_map_image*:
  retraction_map X Y r ∧ *metrizable_space X*
    ⟹ *metrizable_space Y*
 using *hereditary_imp_retractive_property metrizable_space_subtopology homeomorphic_metrizable_space*
 by *blast*

lemma *metrizable_imp_Hausdorff_space*:
  metrizable_space X ⟹ *Hausdorff_space X*
 by (*metis Metric_space.Hausdorff_space_mtopology metrizable_space_def*)

lemma *metrizable_imp_t1_space*:
  metrizable_space X ⟹ *t1_space X*
 by (*simp add*: *Hausdorff_imp_t1_space metrizable_imp_Hausdorff_space*)

lemma *closed_imp_gdelta_in*:
 assumes *X*: *metrizable_space X* **and** *S*: *closedin X S*
 shows *gdelta_in X S*
proof −
 obtain *M d* **where** *Metric_space M d* **and** *Xeq*: *X* = *Metric_space.mtopology M d*
   using *X metrizable_space_def* **by** *blast*
 then interpret *M*: *Metric_space M d*
   by *blast*
 have *S* ⊆ *M*
   using *M.closedin_metric* ‹*X* = *M.mtopology*› *S* **by** *blast*
 show *?thesis*
 proof (*cases S* = {})
   case *True*
   then show *?thesis*
     by *simp*
 next
   case *False*
   have ∃ *y*∈*S*. *d x y* < *inverse* (*1* + *real n*) **if** *x* ∈ *S* **for** *x n*
     using ‹*S* ⊆ *M*› *M.mdist_zero* [*of x*] *that* **by** *force*
   moreover
   have *x* ∈ *S* **if** *x* ∈ *M* **and** §: ⋀*n*. ∃ *y*∈*S*. *d x y* < *inverse*(*Suc n*) **for** *x*
   proof −
     have ∗: ∃ *y*∈*S*. *d x y* < *ε* **if** *ε* > *0* **for** *ε*
       by (*metis* § *that not0_implies_Suc order_less_le order_less_le_trans real_arch_inverse*)
     have *closedin M.mtopology S*
       using *S* **by** (*simp add*: *Xeq*)
     then show *?thesis*
       apply (*simp add*: *M.closedin_metric*)

```
      by (metis * ‹x ∈ M› M.in_mball disjnt_insert1 insert_absorb subsetD)
    qed
    ultimately have Seq: S = ⋂(range (λn. {x∈M. ∃ y∈S. d x y < inverse(Suc
n)}))
      using ‹S ⊆ M› by force
    have openin M.mtopology {xa ∈ M. ∃ y∈S. d xa y < inverse (1 + real n)} for
n
    proof (clarsimp simp: M.openin_mtopology)
      fix x y
      assume x ∈ M y ∈ S and dxy: d x y < inverse (1 + real n)
      then have ⋀z. ⟦z ∈ M; d x z < inverse (1 + real n) − d x y⟧ ⟹ ∃ y∈S. d
z y < inverse (1 + real n)
        by (smt (verit) M.commute M.triangle ‹S ⊆ M› in_mono)
      with dxy show ∃ r>0. M.mball x r ⊆ {z ∈ M. ∃ y∈S. d z y < inverse (1 +
real n)}
        by (rule_tac x=inverse(Suc n) − d x y in exI) auto
    qed
    then show ?thesis
      apply (subst Seq)
      apply (force simp: Xeq intro: gdelta_in_Inter open_imp_gdelta_in)
      done
  qed
qed

lemma open_imp_fsigma_in:
  ⟦metrizable_space X; openin X S⟧ ⟹ fsigma_in X S
  by (meson closed_imp_gdelta_in fsigma_in_gdelta_in openin_closedin openin_subset)

lemma metrizable_space_euclidean:
  metrizable_space (euclidean :: 'a::metric_space topology)
  using Met_TC.metrizable_space_mtopology by auto

lemma (in Metric_space) regular_space_mtopology:
   regular_space mtopology
unfolding regular_space_def
proof clarify
  fix C a
  assume C: closedin mtopology C and a: a ∈ topspace mtopology and a ∉ C
  have openin mtopology (topspace mtopology − C)
    by (simp add: C openin_diff)
  then obtain r where r>0 and r: mball a r ⊆ topspace mtopology − C
    unfolding openin_mtopology using ‹a ∉ C› a by auto
  show ∃ U V. openin mtopology U ∧ openin mtopology V ∧ a ∈ U ∧ C ⊆ V ∧
disjnt U V
  proof (intro exI conjI)
    show a ∈ mball a (r/2)
      using ‹0 < r› a by force
    show C ⊆ topspace mtopology − mcball a (r/2)
      using C ‹0 < r› r by (fastforce simp: closedin_metric)
```

qed (*auto simp*: *openin_mball closedin_mcball openin_diff disjnt_iff*)
qed

lemma *metrizable_imp_regular_space*:
metrizable_space X $\Longrightarrow$ *regular_space X*
by (*metis Metric_space.regular_space_mtopology metrizable_space_def*)

lemma *regular_space_euclidean*:
regular_space (*euclidean* :: *'a*::*metric_space topology*)
by (*simp add*: *metrizable_imp_regular_space metrizable_space_euclidean*)

5.7.7 Limits at a point in a topological space

lemma (**in** *Metric_space*) *eventually_atin_metric*:
eventually P (*atin mtopology a*) $\longleftrightarrow$
$(a \in M \longrightarrow (\exists \delta{>}0.\ \forall x.\ x \in M \wedge 0 < d\ x\ a \wedge d\ x\ a < \delta \longrightarrow P\ x))$ (**is** *?lhs=?rhs*)
proof (*cases a* $\in$ *M*)
case *True*
show *?thesis*
proof
assume *L*: *?lhs*
with *True* **obtain** *U* **where** *openin mtopology U a* $\in$ *U* **and** *U*: $\forall x{\in}U - \{a\}.$ *P x*
by (*auto simp*: *eventually_atin*)
then obtain *r* **where** *r*>*0* **and** *mball a r* $\subseteq$ *U*
by (*meson openin_mtopology*)
with *U* **show** *?rhs*
by (*smt* (*verit*, *ccfv_SIG*) *commute in_mball insert_Diff_single insert_iff subset_iff*)
next
assume *?rhs*
then obtain δ **where** $\delta{>}0$ **and** δ: $\forall x.\ x \in M \wedge 0 < d\ x\ a \wedge d\ x\ a < \delta \longrightarrow P\ x$
using *True* **by** *blast*
then have $\forall x \in$ *mball a* $\delta - \{a\}.$ *P x*
by (*simp add*: *commute*)
then show *?lhs*
unfolding *eventually_atin openin_mtopology*
by (*metis True* ‹*0* < δ› *centre_in_mball_iff openin_mball openin_mtopology*)

qed
qed *auto*

5.7.8 Normal spaces and metric spaces

lemma (**in** *Metric_space*) *normal_space_mtopology*:
normal_space mtopology
unfolding *normal_space_def*
proof *clarify*

fix S T
assume *closedin mtopology S*
then have $\bigwedge x.\ x \in M - S \Longrightarrow (\exists r{>}0.\ \mathit{mball}\ x\ r \subseteq M - S)$
by (*simp add*: *closedin_def openin_mtopology*)
then obtain δ **where** *d0*: $\bigwedge x.\ x \in M - S \Longrightarrow \delta\ x > 0 \wedge \mathit{mball}\ x\ (\delta\ x) \subseteq M - S$
by *metis*
assume *closedin mtopology T*
then have $\bigwedge x.\ x \in M - T \Longrightarrow (\exists r{>}0.\ \mathit{mball}\ x\ r \subseteq M - T)$
by (*simp add*: *closedin_def openin_mtopology*)
then obtain ε **where** *e*: $\bigwedge x.\ x \in M - T \Longrightarrow \varepsilon\ x > 0 \wedge \mathit{mball}\ x\ (\varepsilon\ x) \subseteq M - T$
by *metis*
assume *disjnt S T*
have $S \subseteq M\ T \subseteq M$
using ‹*closedin mtopology S*› ‹*closedin mtopology T*› *closedin_metric* **by** *blast+*
have δ: $\bigwedge x.\ x \in T \Longrightarrow \delta\ x > 0 \wedge \mathit{mball}\ x\ (\delta\ x) \subseteq M - S$
by (*meson DiffI* ‹$T \subseteq M$› ‹*disjnt S T*› *d0 disjnt_iff subsetD*)
have ε: $\bigwedge x.\ x \in S \Longrightarrow \varepsilon\ x > 0 \wedge \mathit{mball}\ x\ (\varepsilon\ x) \subseteq M - T$
by (*meson Diff_iff* ‹$S \subseteq M$› ‹*disjnt S T*› *disjnt_iff e subsetD*)
show $\exists U\ V.$ *openin mtopology U* $\wedge$ *openin mtopology V* $\wedge$ $S \subseteq U \wedge T \subseteq V \wedge$ *disjnt U V*
proof (*intro exI conjI*)
show *openin mtopology* $(\bigcup x{\in}S.\ \mathit{mball}\ x\ (\varepsilon\ x\ /\ 2))$ *openin mtopology* $(\bigcup x{\in}T.\ \mathit{mball}\ x\ (\delta\ x\ /\ 2))$
by *force+*
show $S \subseteq (\bigcup x{\in}S.\ \mathit{mball}\ x\ (\varepsilon\ x\ /\ 2))$
using ε ‹$S \subseteq M$› **by** *force*
show $T \subseteq (\bigcup x{\in}T.\ \mathit{mball}\ x\ (\delta\ x\ /\ 2))$
using δ ‹$T \subseteq M$› **by** *force*
show *disjnt* $(\bigcup x{\in}S.\ \mathit{mball}\ x\ (\varepsilon\ x\ /\ 2))$ $(\bigcup x{\in}T.\ \mathit{mball}\ x\ (\delta\ x\ /\ 2))$
using ε δ
apply (*clarsimp simp*: *disjnt_iff subset_iff*)
by (*smt* (*verit*, *ccfv_SIG*) *field_sum_of_halves triangle'*)
qed
qed

lemma *metrizable_imp_normal_space*:
metrizable_space X $\Longrightarrow$ *normal_space X*
by (*metis Metric_space.normal_space_mtopology metrizable_space_def*)

5.7.9 Topological limitin in metric spaces

lemma (**in** *Metric_space*) *limitin_mspace*:
limitin mtopology f l F $\Longrightarrow$ $l \in M$
using *limitin_topspace* **by** *fastforce*

lemma (**in** *Metric_space*) *limitin_metric_unique*:
$[\![$*limitin mtopology f l1 F*; *limitin mtopology f l2 F*; $F \neq$ *bot*$]\!] \Longrightarrow$ *l1* $=$ *l2*

by (*meson Hausdorff_space_mtopology limitin_Hausdorff_unique*)

lemma (**in** *Metric_space*) *limitin_metric*:
limitin mtopology f l F ⟷ *l* ∈ *M* ∧ (∀ε>*0*. *eventually* (λ*x*. *f x* ∈ *M* ∧ *d* (*f x*) *l* < ε) *F*)
(**is** *?lhs=?rhs*)
proof
assume *L*: *?lhs*
show *?rhs*
unfolding *limitin_def*
proof (*intro conjI strip*)
show *l* ∈ *M*
using *L limitin_mspace* **by** *blast*
fix ε::*real*
assume ε>*0*
then have ∀ *F x in F*. *f x* ∈ *mball l* ε
using *L openin_mball* **by** (*fastforce simp*: *limitin_def*)
then show ∀ *F x in F*. *f x* ∈ *M* ∧ *d* (*f x*) *l* < ε
using *commute eventually_mono* **by** *fastforce*
qed
next
assume *R*: *?rhs*
then show *?lhs*
by (*force simp*: *limitin_def commute openin_mtopology subset_eq elim*: *eventually_mono*)
qed

lemma (**in** *Metric_space*) *limit_metric_sequentially*:
limitin mtopology f l sequentially ⟷
l ∈ *M* ∧ (∀ε>*0*. ∃ *N*. ∀ *n*≥*N*. *f n* ∈ *M* ∧ *d* (*f n*) *l* < ε)
by (*auto simp*: *limitin_metric eventually_sequentially*)

lemma (**in** *Submetric*) *limitin_submetric_iff*:
limitin sub.mtopology f l F ⟷
l ∈ *A* ∧ *eventually* (λ*x*. *f x* ∈ *A*) *F* ∧ *limitin mtopology f l F* (**is** *?lhs=?rhs*)
by (*simp add*: *limitin_subtopology mtopology_submetric*)

lemma (**in** *Metric_space*) *metric_closedin_iff_sequentially_closed*:
closedin mtopology S ⟷
S ⊆ *M* ∧ (∀σ *l*. *range* σ ⊆ *S* ∧ *limitin mtopology* σ *l sequentially* ⟶ *l* ∈ *S*)
(**is** *?lhs=?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*force simp*: *closedin_metric limitin_closedin range_subsetD*)
next
assume *R*: *?rhs*
show *?lhs*
unfolding *closedin_metric*
proof (*intro conjI strip*)

```
    show S ⊆ M
      using R by blast
    fix x
    assume x ∈ M − S
    have False if ∀r>0. ∃y. y ∈ M ∧ y ∈ S ∧ d x y < r
    proof −
      have ∀n. ∃y. y ∈ M ∧ y ∈ S ∧ d x y < inverse(Suc n)
        using that by auto
     then obtain σ where σ: ⋀n. σ n ∈ M ∧ σ n ∈ S ∧ d x (σ n) < inverse(Suc
n)
        by metis
      then have range σ ⊆ M
        by blast
      have ∃N. ∀n≥N. d x (σ n) < ε if ε>0 for ε
      proof −
        have real (Suc (nat ⌈inverse ε⌉)) ≥ inverse ε
          by linarith
        then have ∀n ≥ nat ⌈inverse ε⌉. d x (σ n) < ε
            by (metis σ inverse_inverse_eq inverse_le_imp_le nat_ceiling_le_eq
nle_le not_less_eq_eq order.strict_trans2 that)
        then show ?thesis ..
      qed
      with σ have limitin mtopology σ x sequentially
        using ‹x ∈ M − S› commute limit_metric_sequentially by auto
      then show ?thesis
        by (metis R DiffD2 σ image_subset_iff ‹x ∈ M − S›)
    qed
    then show ∃r>0. disjnt S (mball x r)
      by (meson disjnt_iff in_mball)
  qed
qed

lemma (in Metric_space) limit_atin_metric:
   limitin X f y (atin mtopology x) ⟷
      y ∈ topspace X ∧
      (x ∈ M
       ⟶ (∀V. openin X V ∧ y ∈ V
             ⟶ (∃δ>0. ∀x'. x' ∈ M ∧ 0 < d x' x ∧ d x' x < δ ⟶ f x' ∈ V)))
  by (force simp: limitin_def eventually_atin_metric)

lemma (in Metric_space) limitin_metric_dist_null:
   limitin mtopology f l F ⟷ l ∈ M ∧ eventually (λx. f x ∈ M) F ∧ ((λx. d (f
x) l) ⟶ 0) F
  by (simp add: limitin_metric tendsto_iff eventually_conj_iff all_conj_distrib
imp_conjR gt_ex)
```

5.7.10 Cauchy sequences and complete metric spaces

```
context Metric_space
```

begin

definition *MCauchy* :: (*nat* ⇒ *'a*) ⇒ *bool*
where *MCauchy σ* ≡ *range σ* ⊆ *M* ∧ (∀ *ε*>*0*. ∃ *N*. ∀ *n n'*. *N* ≤ *n* ⟶ *N* ≤ *n'* ⟶ *d* (*σ n*) (*σ n'*) < *ε*)

definition *mcomplete*
where *mcomplete* ≡ (∀ *σ*. *MCauchy σ* ⟶ (∃ *x*. *limitin mtopology σ x sequentially*))

lemma *mcomplete_empty* [*iff*]: *Metric_space.mcomplete* {} *d*
by (*simp add*: *Metric_space.MCauchy_def Metric_space.mcomplete_def subspace*)

lemma *MCauchy_imp_MCauchy_suffix*: *MCauchy σ* ⟹ *MCauchy* (*σ* ∘ (+)*n*)
unfolding *MCauchy_def image_subset_iff comp_apply*
by (*metis UNIV_I add.commute trans_le_add1*)

lemma *mcomplete*:
mcomplete ⟷
(∀ *σ*. (∀ *F n in sequentially*. *σ n* ∈ *M*) ∧
(∀ *ε*>*0*. ∃ *N*. ∀ *n n'*. *N* ≤ *n* ⟶ *N* ≤ *n'* ⟶ *d* (*σ n*) (*σ n'*) < *ε*) ⟶
(∃ *x*. *limitin mtopology σ x sequentially*)) (**is** *?lhs*=*?rhs*)
proof
assume *L*: *?lhs*
show *?rhs*
proof *clarify*
fix *σ*
assume ∀ *F n in sequentially*. *σ n* ∈ *M*
and *σ*: ∀ *ε*>*0*. ∃ *N*. ∀ *n n'*. *N* ≤ *n* ⟶ *N* ≤ *n'* ⟶ *d* (*σ n*) (*σ n'*) < *ε*
then obtain *N* **where** ⋀*n*. *n*≥*N* ⟹ *σ n* ∈ *M*
by (*auto simp*: *eventually_sequentially*)
with *σ* **have** *MCauchy* (*σ* ∘ (+)*N*)
unfolding *MCauchy_def image_subset_iff comp_apply* **by** (*meson le_add1 trans_le_add2*)
then obtain *x* **where** *limitin mtopology* (*σ* ∘ (+)*N*) *x sequentially*
using *L MCauchy_imp_MCauchy_suffix mcomplete_def* **by** *blast*
then have *limitin mtopology σ x sequentially*
unfolding *o_def* **by** (*auto simp*: *add.commute limitin_sequentially_offset_rev*)
then show ∃ *x*. *limitin mtopology σ x sequentially* **..**
qed
qed (*simp add*: *mcomplete_def MCauchy_def image_subset_iff*)

lemma *mcomplete_empty_mspace*: *M* = {} ⟹ *mcomplete*
using *MCauchy_def mcomplete_def* **by** *blast*

lemma *MCauchy_const* [*simp*]: *MCauchy* (λ*n*. *a*) ⟷ *a* ∈ *M*
using *MCauchy_def mdist_zero* **by** *auto*

lemma *convergent_imp_MCauchy*:
assumes *range* $\sigma \subseteq M$ **and** *lim*: *limitin mtopology* σ *l sequentially*
shows *MCauchy* σ
unfolding *MCauchy_def image_subset_iff*
proof (*intro conjI strip*)
fix ε::*real*
assume $\varepsilon > 0$
then have $\forall_F$ *n in sequentially*. $\sigma\ n \in M \wedge d\ (\sigma\ n)\ l < \varepsilon/2$
using *half_gt_zero lim limitin_metric* **by** *blast*
then obtain *N* **where** $\bigwedge n.\ n \geq N \Longrightarrow \sigma\ n \in M \wedge d\ (\sigma\ n)\ l < \varepsilon/2$
by (*force simp*: *eventually_sequentially*)
then show $\exists N.\ \forall n\ n'.\ N \leq n \longrightarrow N \leq n' \longrightarrow d\ (\sigma\ n)\ (\sigma\ n') < \varepsilon$
by (*smt* (*verit*) *limitin_mspace mdist_reverse_triangle field_sum_of_halves lim*)
qed (*use assms* **in** *blast*)

lemma *mcomplete_alt*:
mcomplete $\longleftrightarrow$ $(\forall \sigma.$ *MCauchy* $\sigma \longleftrightarrow$ *range* $\sigma \subseteq M \wedge (\exists x.$ *limitin mtopology* σ *x sequentially*$))$
using *MCauchy_def convergent_imp_MCauchy mcomplete_def* **by** *blast*

lemma *MCauchy_subsequence*:
assumes *strict_mono r MCauchy* σ
shows *MCauchy* $(\sigma \circ r)$
proof −
have $d\ (\sigma\ (r\ n))\ (\sigma\ (r\ n')) < \varepsilon$
if $N \leq n\ N \leq n'$ *strict_mono r* $\forall n\ n'.\ N \leq n \longrightarrow N \leq n' \longrightarrow d\ (\sigma\ n)\ (\sigma\ n') < \varepsilon$
for $\varepsilon\ N\ n\ n'$
using *that* **by** (*meson le_trans strict_mono_imp_increasing*)
moreover have *range* $(\lambda x.\ \sigma\ (r\ x)) \subseteq M$
using *MCauchy_def assms* **by** *blast*
ultimately show *?thesis*
using *assms* **by** (*simp add*: *MCauchy_def*) *metis*
qed

lemma *MCauchy_offset*:
assumes *cau*: *MCauchy* $(\sigma \circ (+)k)$ **and** σ: $\bigwedge n.\ n < k \Longrightarrow \sigma\ n \in M$
shows *MCauchy* σ
unfolding *MCauchy_def image_subset_iff*
proof (*intro conjI strip*)
fix *n*
show $\sigma\ n \in M$
using *assms*
unfolding *MCauchy_def image_subset_iff*
by (*metis UNIV_I comp_apply le_iff_add linorder_not_le*)
next
fix ε :: *real*

assume $\varepsilon > 0$
obtain N **where** $\forall n\ n'.\ N \leq n \longrightarrow N \leq n' \longrightarrow d\ ((\sigma \circ (+)k)\ n)\ ((\sigma \circ (+)k)\ n') < \varepsilon$
using *cau* ‹$\varepsilon > 0$› **by** (*fastforce simp*: *MCauchy_def*)
then show $\exists N.\ \forall n\ n'.\ N \leq n \longrightarrow N \leq n' \longrightarrow d\ (\sigma\ n)\ (\sigma\ n') < \varepsilon$
unfolding *o_def*
apply (*rule_tac x=k+N* **in** *exI*)
by (*smt* (*verit*, *del_insts*) *add.assoc le_add1 less_eqE*)
qed

lemma *MCauchy_convergent_subsequence*:
assumes *cau*: *MCauchy* σ **and** *strict_mono r*
and *lim*: *limitin mtopology* $(\sigma \circ r)$ *a sequentially*
shows *limitin mtopology* σ *a sequentially*
unfolding *limitin_metric*
proof (*intro conjI strip*)
show $a \in M$
by (*meson assms limitin_mspace*)
fix ε :: *real*
assume $\varepsilon > 0$
then obtain *N1* **where** *N1*: $\bigwedge n\ n'.\ [\![n \geq N1;\ n' \geq N1]\!] \Longrightarrow d\ (\sigma\ n)\ (\sigma\ n') < \varepsilon/2$
using *cau* **unfolding** *MCauchy_def* **by** (*meson half_gt_zero*)
obtain *N2* **where** *N2*: $\bigwedge n.\ n \geq N2 \Longrightarrow (\sigma \circ r)\ n \in M \wedge d\ ((\sigma \circ r)\ n)\ a < \varepsilon/2$
by (*metis* (*no_types*, *lifting*) *lim* ‹$\varepsilon > 0$› *half_gt_zero limit_metric_sequentially*)
have $\sigma\ n \in M \wedge d\ (\sigma\ n)\ a < \varepsilon$ **if** $n \geq max\ N1\ N2$ **for** n
proof (*intro conjI*)
show $\sigma\ n \in M$
using *MCauchy_def cau* **by** *blast*
have $N1 \leq r\ n$
by (*meson* ‹*strict_mono r*› *le_trans max.cobounded1 strict_mono_imp_increasing that*)
then show $d\ (\sigma\ n)\ a < \varepsilon$
using *N1*[*of n r n*] *N2*[*of n*] ‹$\sigma\ n \in M$› ‹$a \in M$› *triangle that* **by** *fastforce*
qed
then show $\forall_F\ n$ *in sequentially*. $\sigma\ n \in M \wedge d\ (\sigma\ n)\ a < \varepsilon$
using *eventually_sequentially* **by** *blast*
qed

lemma *MCauchy_interleaving_gen*:
MCauchy $(\lambda n.$ *if even n then* $x(n$ *div 2*$)$ *else* $y(n$ *div 2*$))$ $\longleftrightarrow$
$(MCauchy\ x \wedge MCauchy\ y \wedge (\lambda n.\ d\ (x\ n)\ (y\ n)) \longrightarrow 0)$ (**is** *?lhs=?rhs*)
proof
assume *L*: *?lhs*
have *evens*: *strict_mono* $(\lambda n{::}nat.\ 2 * n)$ **and** *odds*: *strict_mono* $(\lambda n{::}nat.$ *Suc* $(2 * n))$
by (*auto simp*: *strict_mono_def*)
show *?rhs*
proof (*intro conjI*)
show *MCauchy x MCauchy y*

```
    using MCauchy_subsequence [OF evens L] MCauchy_subsequence [OF odds
L] by (auto simp: o_def)
   show (λn. d (x n) (y n)) ⟶ 0
    unfolding LIMSEQ_iff
   proof (intro strip)
    fix ε :: real
    assume ε > 0
    then obtain N where N:
     ⋀n n'. ⟦n≥N; n'≥N⟧ ⟹ d (if even n then x (n div 2) else y (n div 2))
                           (if even n' then x (n' div 2) else y (n' div 2)) < ε
      using L MCauchy_def by fastforce
    have d (x n) (y n) < ε if n≥N for n
      using N [of 2*n Suc(2*n)] that by auto
    then show ∃N. ∀n≥N. norm (d (x n) (y n) − 0) < ε
      by auto
   qed
  qed
next
  assume R: ?rhs
  show ?lhs
   unfolding MCauchy_def
  proof (intro conjI strip)
   show range (λn. if even n then x (n div 2) else y (n div 2)) ⊆ M
    using R by (auto simp: MCauchy_def)
   fix ε :: real
   assume ε > 0
   obtain Nx where Nx: ⋀n n'. ⟦n≥Nx; n'≥Nx⟧ ⟹ d (x n) (x n') < ε/2
    by (meson half_gt_zero MCauchy_def R ‹ε > 0›)
   obtain Ny where Ny: ⋀n n'. ⟦n≥Ny; n'≥Ny⟧ ⟹ d (y n) (y n') < ε/2
    by (meson half_gt_zero MCauchy_def R ‹ε > 0›)
   obtain Nxy where Nxy: ⋀n. n≥Nxy ⟹ d (x n) (y n) < ε/2
    using R ‹ε > 0› half_gt_zero unfolding LIMSEQ_iff
    by (metis abs_mdist diff_zero real_norm_def)
   define N where N ≡ 2 * Max{Nx,Ny,Nxy}
   show ∃N. ∀n n'. N ≤ n ⟶ N ≤ n' ⟶ d (if even n then x (n div 2) else y
(n div 2)) (if even n' then x (n' div 2) else y (n' div 2)) < ε
   proof (intro exI strip)
    fix n n'
    assume N ≤ n and N ≤ n'
    then have n div 2 ≥ Nx n div 2 ≥ Ny n div 2 ≥ Nxy n' div 2 ≥ Nx n' div
2 ≥ Ny
     by (auto simp: N_def)
    then have dxyn: d (x (n div 2)) (y (n div 2)) < ε/2
         and dxnn': d (x (n div 2)) (x (n' div 2)) < ε/2
         and dynn': d (y (n div 2)) (y (n' div 2)) < ε/2
     using Nx Ny Nxy by blast+
    have inM: x (n div 2) ∈ M x (n' div 2) ∈ My (n div 2) ∈ M y (n' div 2) ∈
M
     using MCauchy_def R by blast+
```

```
    show d (if even n then x (n div 2) else y (n div 2)) (if even n' then x (n' div
2) else y (n' div 2)) < ε
     proof (cases even n)
      case nt: True
      show ?thesis
      proof (cases even n')
       case True
       with ‹ε > 0› nt dxnn' show ?thesis by auto
      next
       case False
       with nt dxyn dynn' inM triangle show ?thesis
        by fastforce
      qed
     next
      case nf: False
      show ?thesis
      proof (cases even n')
       case True
       then show ?thesis
       by (smt (verit) ‹ε > 0› dxyn dxnn' triangle commute inM field_sum_of_halves)
      next
       case False
       with ‹ε > 0› nf dynn' show ?thesis by auto
      qed
     qed
   qed
  qed
qed

lemma MCauchy_interleaving:
   MCauchy (λn. if even n then σ(n div 2) else a) ⟷
    range σ ⊆ M ∧ limitin mtopology σ a sequentially  (is ?lhs=?rhs)
proof −
  have ?lhs ⟷ (MCauchy σ ∧ a ∈ M ∧ (λn. d (σ n) a) ⟶ 0)
    by (simp add: MCauchy_interleaving_gen [where y = λn. a])
  also have ... = ?rhs
     by (metis MCauchy_def always_eventually convergent_imp_MCauchy lim-
itin_metric_dist_null range_subsetD)
  finally show ?thesis .
qed

lemma mcomplete_nest:
   mcomplete ⟷
     (∀ C::nat ⇒'a set. (∀ n. closedin mtopology (C n)) ∧
         (∀ n. C n ≠ {}) ∧ decseq C ∧ (∀ε>0. ∃ n a. C n ⊆ mcball a ε)
         ⟶ ⋂ (range C) ≠ {}) (is ?lhs=?rhs)
proof
  assume L: ?lhs
  show ?rhs
```

```
    unfolding imp_conjL
  proof (intro strip)
    fix C :: nat ⇒ 'a set
    assume clo: ∀ n. closedin mtopology (C n)
      and ne: ∀ n. C n ≠ ({}::'a set)
      and dec: decseq C
      and cover [rule_format]: ∀ ε>0. ∃ n a. C n ⊆ mcball a ε
    obtain σ where σ: ⋀n. σ n ∈ C n
      by (meson ne empty_iff set_eq_iff)
    have MCauchy σ
      unfolding MCauchy_def
    proof (intro conjI strip)
      show range σ ⊆ M
        using σ clo metric_closedin_iff_sequentially_closed by auto
      fix ε :: real
      assume ε > 0
      then obtain N a where N: C N ⊆ mcball a (ε/3)
        using cover by fastforce
      have d (σ m) (σ n) < ε if N ≤ m N ≤ n for m n
      proof −
        have d a (σ m) ≤ ε/3 d a (σ n) ≤ ε/3
          using dec N σ that by (fastforce simp: decseq_def)+
        then have d (σ m) (σ n) ≤ ε/3 + ε/3
          using triangle σ commute dec decseq_def subsetD that N
          by (smt (verit, ccfv_threshold) in_mcball)
        also have ... < ε
          using ‹ε > 0› by auto
        finally show ?thesis .
      qed
      then show ∃ N. ∀ m n. N ≤ m ⟶ N ≤ n ⟶ d (σ m) (σ n) < ε
        by blast
    qed
    then obtain x where x: limitin mtopology σ x sequentially
      using L mcomplete_def by blast
    have x ∈ C n for n
    proof (rule limitin_closedin [OF x])
      show closedin mtopology (C n)
        by (simp add: clo)
      show ∀ F x in sequentially. σ x ∈ C n
        by (metis σ dec decseq_def eventually_sequentiallyI subsetD)
    qed auto
    then show ⋂ (range C) ≠ {}
      by blast
qed
next
  assume R: ?rhs
  show ?lhs
    unfolding mcomplete_def
  proof (intro strip)
```

fix σ
assume *MCauchy* σ
then have *range* $\sigma \subseteq M$
using *MCauchy_def* **by** *blast*
define C **where** $C \equiv \lambda n.$ *mtopology closure_of* $(\sigma$ ‘ $\{n..\})$
have $\forall\, n.$ *closedin mtopology* $(C\ n)$
by (*auto simp*: *C_def*)
moreover
have *ne*: $\bigwedge n.\ C\ n \neq \{\}$
using ‹*MCauchy* σ› **by** (*auto simp*: *C_def MCauchy_def disjnt_iff closure_of_eq_empty_gen*)
moreover
have *dec*: *decseq C*
unfolding *monotone_on_def*
proof (*intro strip*)
fix $m\ n$::*nat*
assume $m \leq n$
then have $\{n..\} \subseteq \{m..\}$
by *auto*
then show $C\ n \subseteq C\ m$
unfolding *C_def* **by** (*meson closure_of_mono image_mono*)
qed
moreover
have *C*: $\exists\, N\ u.\ C\ N \subseteq$ *mcball* $u\ \varepsilon$ **if** ε>*0* **for** ε
proof −
obtain N **where** $\bigwedge m\ n.\ N \leq m \wedge N \leq n \Longrightarrow d\ (\sigma\ m)\ (\sigma\ n) < \varepsilon$
by (*meson MCauchy_def* ‹*0* < ε› ‹*MCauchy* σ›)
then have σ ‘ $\{N..\} \subseteq$ *mcball* $(\sigma\ N)\ \varepsilon$
using *MCauchy_def* ‹*MCauchy* σ› **by** (*force simp*: *less_eq_real_def*)
then have $C\ N \subseteq$ *mcball* $(\sigma\ N)\ \varepsilon$
by (*simp add*: *C_def closure_of_minimal*)
then show *?thesis*
by *blast*
qed
ultimately obtain l **where** x: $l \in \bigcap$ (*range C*)
by (*metis R ex_in_conv*)
then have $*$: $\bigwedge \varepsilon\ N.\ 0 < \varepsilon \Longrightarrow \exists\, n'.\ N \leq n' \wedge l \in M \wedge \sigma\ n' \in M \wedge d\ l\ (\sigma\ n') < \varepsilon$
by (*force simp*: *C_def metric_closure_of*)
then have $l \in M$
using *gt_ex* **by** *blast*
show $\exists\, l.$ *limitin mtopology* σ l *sequentially*
unfolding *limitin_metric*
proof (*intro conjI strip exI*)
show $l \in M$
using ‹$\forall\, n.$ *closedin mtopology* $(C\ n)$› *closedin_subset x* **by** *fastforce*
fix ε::*real*
assume $\varepsilon > 0$
obtain N **where** N: $\bigwedge m\ n.\ N \leq m \wedge N \leq n \Longrightarrow d\ (\sigma\ m)\ (\sigma\ n) < \varepsilon/2$

```
      by (meson MCauchy_def ‹0 < ε› ‹MCauchy σ› half_gt_zero)
    with ∗ [of ε/2 N]
    have ∀ n≥N. σ n ∈ M ∧ d (σ n) l < ε
     by (smt (verit) ‹range σ ⊆ M› commute field_sum_of_halves range_subsetD
triangle)
    then show ∀F n in sequentially. σ n ∈ M ∧ d (σ n) l < ε
      using eventually_sequentially by blast
   qed
  qed
qed


lemma mcomplete_nest_sing:
  mcomplete ⟷
   (∀ C. (∀ n. closedin mtopology (C n)) ∧
         (∀ n. C n ≠ {}) ∧ decseq C ∧ (∀ e>0. ∃ n a. C n ⊆ mcball a e)
        ⟶ (∃ l. l ∈ M ∧ ⋂ (range C) = {l}))
proof −
  have ∗: False
   if clo: ∀ n. closedin mtopology (C n)
     and cover: ∀ ε>0. ∃ n a. C n ⊆ mcball a ε
     and no_sing: ⋀y. y ∈ M ⟹ ⋂ (range C) ≠ {y}
     and l: ∀ n. l ∈ C n
   for C :: nat ⇒ 'a set and l
  proof −
    have inM: ⋀x. x ∈ ⋂ (range C) ⟹ x ∈ M
      using closedin_metric clo by fastforce
    then have l ∈ M
      by (simp add: l)
    have False if l': l' ∈ ⋂ (range C) and l' ≠ l for l'
    proof −
      have l' ∈ M
        using inM l' by blast
      obtain n a where na: C n ⊆ mcball a (d l l' / 3)
        using inM ‹l ∈ M› l' ‹l' ≠ l› cover by force
      then have d a l ≤ (d l l' / 3) d a l' ≤ (d l l' / 3) a ∈ M
        using l l' na in_mcball by auto
      then have d l l' ≤ (d l l' / 3) + (d l l' / 3)
        using ‹l ∈ M› ‹l' ∈ M› mdist_reverse_triangle by fastforce
      then show False
        using nonneg [of l l'] ‹l' ≠ l› ‹l ∈ M› ‹l' ∈ M› zero by force
    qed
    then show False
       by (metis l ‹l ∈ M› no_sing INT_I empty_iff insertI1 is_singletonE
is_singletonI')
  qed
  show ?thesis
    unfolding mcomplete_nest imp_conjL
    apply (intro all_cong1 imp_cong refl)
```

```
    using ∗
    by (smt (verit) Inter_iff ex_in_conv range_constant range_eqI)
qed

lemma mcomplete_fip:
   mcomplete ⟷
    (∀𝒞. (∀ C ∈ 𝒞. closedin mtopology C) ∧
         (∀ e>0. ∃ C a. C ∈ 𝒞 ∧ C ⊆ mcball a e) ∧ (∀ℱ. finite ℱ ∧ ℱ ⊆ 𝒞 ⟶ ⋂
ℱ ≠ {})
         ⟶ ⋂ 𝒞 ≠ {})
   (is ?lhs = ?rhs)
proof
  assume L: ?lhs
  show ?rhs
    unfolding mcomplete_nest_sing imp_conjL
  proof (intro strip)
    fix 𝒞 :: 'a set set
    assume clo: ∀ C∈𝒞. closedin mtopology C
      and cover: ∀ e>0. ∃ C a. C ∈ 𝒞 ∧ C ⊆ mcball a e
      and fip: ∀ℱ. finite ℱ ⟶ ℱ ⊆ 𝒞 ⟶ ⋂ ℱ ≠ {}
    then have ∀ n. ∃ C. C ∈ 𝒞 ∧ (∃ a. C ⊆ mcball a (inverse (Suc n)))
      by simp
    then obtain C where C: ⋀n. C n ∈ 𝒞
        and coverC: ⋀n. ∃ a. C n ⊆ mcball a (inverse (Suc n))
      by metis
    define D where D ≡ λn. ⋂ (C ‘ {..n})
    have cloD: closedin mtopology (D n) for n
      unfolding D_def using clo C by blast
    have neD: D n ≠ {} for n
      using fip C by (simp add: D_def image_subset_iff)
    have decD: decseq D
      by (force simp: D_def decseq_def)
    have coverD: ∃ n a. D n ⊆ mcball a ε if  ε >0 for ε
    proof −
      obtain n where inverse (Suc n) < ε
        using ‹0 < ε› reals_Archimedean by blast
      then obtain a where C n ⊆ mcball a ε
       by (meson coverC less_eq_real_def mcball_subset_concentric order_trans)
      then show ?thesis
        unfolding D_def by blast
    qed
    have ∗: a ∈ ⋂𝒞 if a: ⋂ (range D) = {a} and a ∈ M for a
    proof −
      have aC: a ∈ C n for n
        using that by (auto simp: D_def)
      have eqa: ⋀u. (∀ n. u ∈ C n) ⟹ a = u
        using that by (auto simp: D_def)
      have a ∈ T if T ∈ 𝒞 for T
      proof −
```

```
      have cloT: closedin mtopology (T ∩ D n) for n
        using clo cloD that by blast
      have ⋂ (insert T (C ` {..n})) ≠ {} for n
        using that C by (intro fip [rule_format]) auto
      then have neT: T ∩ D n ≠ {} for n
        by (simp add: D_def)
      have decT: decseq (λn. T ∩ D n)
        by (force simp: D_def decseq_def)
      have coverT: ∃n a. T ∩ D n ⊆ mcball a ε if ε>0 for ε
        by (meson coverD le_infI2 that)
      show ?thesis
       using L [unfolded mcomplete_nest_sing, rule_format, of λn. T ∩ D n] a
        by (force simp: cloT neT decT coverT)
    qed
    then show ?thesis by auto
  qed
  show ⋂ C ≠ {}
    by (metis L cloD neD decD coverD * empty_iff mcomplete_nest_sing)
 qed
next
 assume R [rule_format]: ?rhs
 show ?lhs
  unfolding mcomplete_nest imp_conjL
 proof (intro strip)
  fix C :: nat ⇒ 'a set
  assume clo: ∀n. closedin mtopology (C n)
    and ne: ∀n. C n ≠ {}
    and dec: decseq C
    and cover: ∀ε>0. ∃n a. C n ⊆ mcball a ε
  have ⋂(C ` N) ≠ {} if finite N for N
  proof −
    obtain k where N ⊆ {..k}
      using ‹finite N› finite_nat_iff_bounded_le by auto
    with dec have C k ⊆ ⋂(C ` N) by (auto simp: decseq_def)
    then show ?thesis
      using ne by force
  qed
  with clo cover R [of range C] show ⋂ (range C) ≠ {}
   by (metis (no_types, opaque_lifting) finite_subset_image image_iff UNIV_I)
 qed
qed


lemma mcomplete_fip_sing:
  mcomplete ⟷
   (∀C. (∀ C∈C. closedin mtopology C) ∧
    (∀e>0. ∃c a. c ∈ C ∧ c ⊆ mcball a e) ∧
    (∀F. finite F ∧ F ⊆ C ⟶ ⋂ F ≠ {}) ⟶
    (∃l. l ∈ M ∧ ⋂ C = {l}))
```

```
  (is ?lhs = ?rhs)
proof
  have ∗: l ∈ M ⋂ 𝒞 = {l}
    if clo: Ball 𝒞 (closedin mtopology)
      and cover: ∀ e>0. ∃ C a. C ∈ 𝒞 ∧ C ⊆ mcball a e
      and fin: ∀ ℱ. finite ℱ ⟶ ℱ ⊆ 𝒞 ⟶ ⋂ ℱ ≠ {}
      and l: l ∈ ⋂ 𝒞
    for 𝒞 :: 'a set set and l
  proof −
    show l ∈ M
    by (meson Inf_lower2 clo cover gt_ex metric_closedin_iff_sequentially_closed
subsetD that(4))
    show ⋂ 𝒞 = {l}
    proof (cases 𝒞 = {})
      case True
      then show ?thesis
        using cover mbounded_pos by auto
    next
      case False
      have CM: ⋀a. a ∈ ⋂𝒞 ⟹ a ∈ M
        using False clo closedin_subset by fastforce
      have l' ∉ ⋂ 𝒞 if l' ≠ l for l'
      proof
        assume l': l' ∈ ⋂ 𝒞
        with CM have l' ∈ M by blast
        with that ‹l ∈ M› have gt0: 0 < d l l'
          by simp
        then obtain C a where C ∈ 𝒞 and C: C ⊆ mcball a (d l l' / 3)
          using cover [rule_format, of d l l' / 3] by auto
        then have d a l ≤ (d l l' / 3) d a l' ≤ (d l l' / 3) a ∈ M
          using l l' in_mcball by auto
        then have d l l' ≤ (d l l' / 3) + (d l l' / 3)
          using ‹l ∈ M› ‹l' ∈ M› mdist_reverse_triangle by fastforce
        with gt0 show False by auto
      qed
      then show ?thesis
        using l by fastforce
    qed
  qed
  assume L: ?lhs
  with ∗ show ?rhs
    unfolding mcomplete_fip imp_conjL ex_in_conv [symmetric]
    by (elim all_forward imp_forward2 asm_rl) (blast intro:  elim: )
next
  assume ?rhs then show ?lhs
    unfolding mcomplete_fip by (force elim!: all_forward)
qed

end
```

definition *mcomplete_of* :: *'a metric* ⇒ *bool*
where *mcomplete_of* ≡ λ*m*. *Metric_space.mcomplete* (*mspace m*) (*mdist m*)

lemma (**in** *Metric_space*) *mcomplete_of* [*simp*]: *mcomplete_of* (*metric* (*M*,*d*)) = *mcomplete*
by (*simp add*: *mcomplete_of_def*)

lemma *mcomplete_trivial*: *Metric_space.mcomplete* {} (λ*x y*. *0*)
using *Metric_space.intro Metric_space.mcomplete_empty_mspace* **by** *force*

lemma *mcomplete_trivial_singleton*: *Metric_space.mcomplete* {λ*x*. *a*} (λ*x y*. *0*)
proof −
interpret *Metric_space* {λ*x*. *a*} λ*x y*. *0*
by *unfold_locales auto*
show *?thesis*
unfolding *mcomplete_def MCauchy_def image_subset_iff* **by** (*metis UNIV_I limit_metric_sequentially*)
qed

lemma *MCauchy_iff_Cauchy* [*iff*]: *Met_TC.MCauchy* = *Cauchy*
by (*force simp*: *Cauchy_def Met_TC.MCauchy_def*)

lemma *mcomplete_iff_complete* [*iff*]:
Met_TC.mcomplete (*Pure.type* ::*'a*::*metric_space itself*) ⟷ *complete* (*UNIV*::*'a set*)
by (*auto simp*: *Met_TC.mcomplete_def complete_def*)

context *Submetric*
begin

lemma *MCauchy_submetric*:
sub.MCauchy σ ⟷ *range* σ ⊆ *A* ∧ *MCauchy* σ
using *MCauchy_def sub.MCauchy_def subset* **by** *force*

lemma *closedin_mcomplete_imp_mcomplete*:
assumes *clo*: *closedin mtopology A* **and** *mcomplete*
shows *sub.mcomplete*
unfolding *sub.mcomplete_def*
proof (*intro strip*)
fix σ
assume *sub.MCauchy* σ
then have σ: *MCauchy* σ *range* σ ⊆ *A*
using *MCauchy_submetric* **by** *blast*+
then obtain *x* **where** *x*: *limitin mtopology* σ *x sequentially*
using ‹*mcomplete*› **unfolding** *mcomplete_def* **by** *blast*
then have *x* ∈ *A*
using σ *clo metric_closedin_iff_sequentially_closed* **by** *force*
with σ *x* **show** ∃ *x*. *limitin sub.mtopology* σ *x sequentially*

using *limitin_submetric_iff range_subsetD* **by** *fastforce*
qed

lemma *sequentially_closedin_mcomplete_imp_mcomplete*:
assumes *mcomplete* **and** $\bigwedge\sigma$ *l*. *range* $\sigma \subseteq A \wedge$ *limitin mtopology* σ *l sequentially*
$\Longrightarrow l \in A$
shows *sub.mcomplete*
using *assms closedin_mcomplete_imp_mcomplete metric_closedin_iff_sequentially_closed subset* **by** *blast*

end

context *Metric_space*
begin

lemma *mcomplete_Un*:
assumes *A*: *Submetric M d A Metric_space.mcomplete A d*
and *B*: *Submetric M d B Metric_space.mcomplete B d*
shows *Submetric M d* $(A \cup B)$ *Metric_space.mcomplete* $(A \cup B)$ *d*
proof −
show *Submetric M d* $(A \cup B)$
by (*meson assms le_sup_iff Submetric_axioms_def Submetric_def*)
then interpret *MAB*: *Metric_space* $A \cup B$ *d*
by (*meson Submetric.subset subspace*)
interpret *MA*: *Metric_space A d*
by (*meson A Submetric.subset subspace*)
interpret *MB*: *Metric_space B d*
by (*meson B Submetric.subset subspace*)
show *Metric_space.mcomplete* $(A \cup B)$ *d*
unfolding *MAB.mcomplete_def*
proof (*intro strip*)
fix σ
assume *MAB.MCauchy* σ
then have *range* $\sigma \subseteq A \cup B$
using *MAB.MCauchy_def* **by** *blast*
then have *UNIV* $\subseteq \sigma -` A \cup \sigma -` B$
by *blast*
then consider *infinite* $(\sigma -` A) \mid$ *infinite* $(\sigma -` B)$
using *finite_subset* **by** *auto*
then show $\exists x$. *limitin MAB.mtopology* σ *x sequentially*
proof *cases*
case *1*
then obtain *r* **where** *strict_mono r* **and** *r*: $\bigwedge n$::*nat*. *r n* $\in \sigma -` A$
using *infinite_enumerate* **by** *blast*
then have *MA.MCauchy* $(\sigma \circ r)$
using *MA.MCauchy_def MAB.MCauchy_def MAB.MCauchy_subsequence* ‹*MAB.MCauchy* σ› **by** *auto*
with *A* **obtain** *x* **where** *limitin MA.mtopology* $(\sigma \circ r)$ *x sequentially*

```
      using MA.mcomplete_def by blast
    then have limitin MAB.mtopology (σ ∘ r) x sequentially
      by (metis MA.limit_metric_sequentially MAB.limit_metric_sequentially
UnCI)
    then show ?thesis
    using MAB.MCauchy_convergent_subsequence ‹MAB.MCauchy σ› ‹strict_mono
r› by blast
  next
    case 2
    then obtain r where strict_mono r and r: ⋀n::nat. r n ∈ σ −‘ B
      using infinite_enumerate by blast
    then have MB.MCauchy (σ ∘ r)
      using MB.MCauchy_def MAB.MCauchy_def MAB.MCauchy_subsequence
‹MAB.MCauchy σ› by auto
    with B obtain x where limitin MB.mtopology (σ ∘ r) x sequentially
      using MB.mcomplete_def by blast
    then have limitin MAB.mtopology (σ ∘ r) x sequentially
      by (metis MB.limit_metric_sequentially MAB.limit_metric_sequentially
UnCI)
    then show ?thesis
    using MAB.MCauchy_convergent_subsequence ‹MAB.MCauchy σ› ‹strict_mono
r› by blast
  qed
 qed
qed

lemma mcomplete_Union:
 assumes finite 𝒮
  and ⋀A. A ∈ 𝒮 ⟹ Submetric M d A ⋀A. A ∈ 𝒮 ⟹ Metric_space.mcomplete
A d
 shows  Submetric M d (⋃𝒮) Metric_space.mcomplete (⋃𝒮) d
 using assms
 by (induction rule: finite_induct) (auto simp: mcomplete_Un)

lemma mcomplete_Inter:
 assumes finite 𝒮 𝒮 ≠ {}
   and sub: ⋀A. A ∈ 𝒮 ⟹ Submetric M d A
   and comp: ⋀A. A ∈ 𝒮 ⟹ Metric_space.mcomplete A d
 shows Submetric M d (⋂𝒮) Metric_space.mcomplete (⋂𝒮) d
proof −
 show Submetric M d (⋂𝒮)
   using assms unfolding Submetric_def Submetric_axioms_def
   by (metis Inter_lower equals0I inf.orderE le_inf_iff)
 then interpret MS: Submetric M d ⋂𝒮
   by (meson Submetric.subset subspace)
 show Metric_space.mcomplete (⋂𝒮) d
   unfolding MS.sub.mcomplete_def
 proof (intro strip)
   fix σ
```

assume *MS.sub.MCauchy* σ
then have *range* $\sigma \subseteq \bigcap \mathcal{S}$
using *MS.MCauchy_submetric* **by** *blast*
obtain A **where** $A \in \mathcal{S}$ **and** A: *Metric_space.mcomplete A d*
using *assms* **by** *blast*
then have *range* $\sigma \subseteq A$
using ‹*range* $\sigma \subseteq \bigcap \mathcal{S}$› **by** *blast*
interpret *SA*: *Submetric M d A*
by (*meson* ‹$A \in \mathcal{S}$› *sub Submetric.subset subspace*)
have *MCauchy* σ
using *MS.MCauchy_submetric* ‹*MS.sub.MCauchy* σ› **by** *blast*
then obtain x **where** x: *limitin SA.sub.mtopology* σ x *sequentially*
by (*metis A SA.sub.MCauchy_def SA.sub.mcomplete_alt MCauchy_def* ‹*range* $\sigma \subseteq A$›)
show $\exists x$. *limitin MS.sub.mtopology* σ x *sequentially*
apply (*rule_tac x*=x **in** *exI*)
unfolding *MS.limitin_submetric_iff*
proof (*intro conjI*)
show $x \in \bigcap \mathcal{S}$
proof *clarsimp*
fix U
assume $U \in \mathcal{S}$
interpret *SU*: *Submetric M d U*
by (*meson* ‹$U \in \mathcal{S}$› *sub Submetric.subset subspace*)
have *range* $\sigma \subseteq U$
using ‹$U \in \mathcal{S}$› ‹*range* $\sigma \subseteq \bigcap \mathcal{S}$› **by** *blast*
moreover have *Metric_space.mcomplete U d*
by (*simp add*: ‹$U \in \mathcal{S}$› *comp*)
ultimately obtain x' **where** x': *limitin SU.sub.mtopology* σ x' *sequentially*
using *MCauchy_def SU.sub.MCauchy_def SU.sub.mcomplete_alt* ‹*MCauchy* σ› **by** *meson*
have $x' = x$
proof (*intro limitin_metric_unique*)
show *limitin mtopology* σ x' *sequentially*
by (*meson SU.Submetric_axioms Submetric.limitin_submetric_iff* x')
show *limitin mtopology* σ x *sequentially*
by (*meson SA.Submetric_axioms Submetric.limitin_submetric_iff* x)
qed *auto*
then show $x \in U$
using *SU.sub.limitin_mspace* x' **by** *blast*
qed
show $\forall_F$ n *in sequentially*. σ $n \in \bigcap \mathcal{S}$
by (*meson* ‹*range* $\sigma \subseteq \bigcap \mathcal{S}$› *always_eventually range_subsetD*)
show *limitin mtopology* σ x *sequentially*
by (*meson SA.Submetric_axioms Submetric.limitin_submetric_iff* x)
qed
qed
qed

lemma *mcomplete_Int*:
assumes *A*: *Submetric M d A Metric_space.mcomplete A d*
and *B*: *Submetric M d B Metric_space.mcomplete B d*
shows *Submetric M d* $(A \cap B)$ *Metric_space.mcomplete* $(A \cap B)$ *d*
using *mcomplete_Inter* [*of* $\{A,B\}$] *assms* **by** *force+*

5.7.11 Totally bounded subsets of metric spaces

definition *mtotally_bounded*
where *mtotally_bounded S* $\equiv \forall \varepsilon{>}0.\ \exists K.$ *finite K* $\wedge$ $K \subseteq S \wedge S \subseteq (\bigcup x{\in}K.$ *mball x* $\varepsilon)$

lemma *mtotally_bounded_empty* [*iff*]: *mtotally_bounded* {}
by (*simp add*: *mtotally_bounded_def*)

lemma *finite_imp_mtotally_bounded*:
$[\![$*finite S*; $S \subseteq M]\!] \Longrightarrow$ *mtotally_bounded S*
by (*auto simp*: *mtotally_bounded_def*)

lemma *mtotally_bounded_imp_subset*: *mtotally_bounded S* $\Longrightarrow S \subseteq M$
by (*force simp*: *mtotally_bounded_def intro*!: *zero_less_one*)

lemma *mtotally_bounded_sing* [*simp*]:
mtotally_bounded $\{x\} \longleftrightarrow x \in M$
by (*meson empty_subsetI finite.simps finite_imp_mtotally_bounded insert_subset mtotally_bounded_imp_subset*)

lemma *mtotally_bounded_Un*:
assumes *mtotally_bounded S mtotally_bounded T*
shows *mtotally_bounded* $(S \cup T)$
proof −
have $\exists K.$ *finite* $K \wedge K \subseteq S \cup T \wedge S \cup T \subseteq (\bigcup x{\in}K.$ *mball x e*)
if $e{>}0$ **and** *K*: *finite* $K \wedge K \subseteq S \wedge S \subseteq (\bigcup x{\in}K.$ *mball x e*)
and *L*: *finite* $L \wedge L \subseteq T \wedge T \subseteq (\bigcup x{\in}L.$ *mball x e*) **for** *K L e*
using *that* **by** (*rule_tac* $x{=}K \cup L$ **in** *exI*) *auto*
with *assms* **show** *?thesis*
unfolding *mtotally_bounded_def* **by** *presburger*
qed

lemma *mtotally_bounded_Union*:
assumes *finite f* $\bigwedge S.\ S \in f \Longrightarrow$ *mtotally_bounded S*
shows *mtotally_bounded* $(\bigcup f)$
using *assms* **by** (*induction f*) (*auto simp*: *mtotally_bounded_Un*)

lemma *mtotally_bounded_imp_mbounded*:
assumes *mtotally_bounded S*
shows *mbounded S*
proof −

```
  obtain K where finite K ∧ K ⊆ S ∧ S ⊆ (⋃x∈K. mball x 1)
    using assms by (force simp: mtotally_bounded_def)
  then show ?thesis
     by (smt (verit) finite_imageI image_iff mbounded_Union mbounded_mball
mbounded_subset)
qed


lemma mtotally_bounded_sequentially:
  mtotally_bounded S ⟷
    S ⊆ M ∧ (∀σ::nat ⇒ 'a. range σ ⊆ S ⟶ (∃r. strict_mono r ∧ MCauchy (σ
∘ r)))
  (is _ ⟷ _ ∧ ?rhs)
proof (cases S ⊆ M)
  case True
  show ?thesis
  proof −
    { fix σ :: nat ⇒ 'a
      assume L: mtotally_bounded S and σ: range σ ⊆ S
      have ∃j > i. d (σ i) (σ j) < 3∗ε/2 ∧ infinite (σ −‘ mball (σ j) (ε/2))
        if inf: infinite (σ −‘ mball (σ i) ε) and ε > 0 for i ε
      proof −
        obtain K where finite K K ⊆ S and K: S ⊆ (⋃x∈K. mball x (ε/4))
       by (metis L mtotally_bounded_def ‹ε > 0› zero_less_divide_iff zero_less_numeral)
        then have K_imp_ex: ⋀y. y ∈ S ⟹ ∃x∈K. d x y < ε/4
          by fastforce
        have False if ∀x∈K. d x (σ i) < ε + ε/4 ⟶ finite (σ −‘ mball x (ε/4))
        proof −
          have ∃w. w ∈ K ∧ d w (σ i) < 5 ∗ ε/4 ∧ d w (σ j) < ε/4
            if d (σ i) (σ j) < ε for j
          proof −
            obtain w where w: d w (σ j) < ε/4 w ∈ K
              using K_imp_ex σ by blast
            then have d w (σ i) < ε + ε/4
                by (smt (verit, ccfv_SIG) True ‹K ⊆ S› σ rangeI subset_eq that
triangle')
            with w show ?thesis
              using in_mball by auto
          qed
          then have (σ −‘ mball (σ i) ε) ⊆ (⋃x∈K. if d x (σ i) < ε + ε/4 then σ
−‘ mball x (ε/4) else {})
            using True ‹K ⊆ S› by force
          then show False
            using finite_subset inf ‹finite K› that by fastforce
        qed
        then obtain x where x ∈ K and dxi: d x (σ i) < ε + ε/4 and infx:
infinite (σ −‘ mball x (ε/4))
          by blast
        then obtain j where j ∈ (σ −‘ mball x (ε/4)) − {..i}
```

```
          using bounded_nat_set_is_finite by (meson Diff_infinite_finite fi-
nite_atMost)
      then have j > i and dxj: d x (σ j) < ε/4
        by auto
       have (σ −‘ mball x (ε/4)) ⊆ (σ −‘ mball y (ε/2)) if d x y < ε/4 y ∈ M
for y
        using that by (simp add: mball_subset vimage_mono)
      then have infj: infinite (σ −‘ mball (σ j) (ε/2))
        by (meson True ‹d x (σ j) < ε/4› σ in_mono infx rangeI finite_subset)
      have σ i ∈ M σ j ∈ M x ∈ M
        using True ‹K ⊆ S› ‹x ∈ K› σ by force+
      then have d (σ i) (σ j) ≤ d x (σ i) + d x (σ j)
        using triangle'' by blast
      also have ... < 3∗ε/2
        using dxi dxj by auto
      finally have d (σ i) (σ j) < 3∗ε/2 .
      with ‹i < j› infj show ?thesis by blast
    qed
    then obtain nxt where nxt: ⋀i ε. [[ε > 0; infinite (σ −‘ mball (σ i) ε)]] ⟹

          nxt i ε > i ∧ d (σ i) (σ (nxt i ε)) < 3∗ε/2 ∧ infinite (σ −‘ mball
(σ (nxt i ε)) (ε/2))
      by metis
    have mbounded S
      using L by (simp add: mtotally_bounded_imp_mbounded)
    then obtain B where B: ∀ y ∈ S. d (σ 0) y ≤ B and B > 0
      by (meson σ mbounded_alt_pos range_subsetD)
    define eps where eps ≡ λn. (B+1) / 2^n
    have [simp]: eps (Suc n) = eps n / 2 eps n > 0 for n
      using ‹B > 0› by (auto simp: eps_def)
    have UNIV ⊆ σ −‘ mball (σ 0) (B+1)
      using B True σ unfolding image_iff subset_iff
      by (smt (verit, best) UNIV_I in_mball vimageI)
    then have inf0: infinite (σ −‘ mball (σ 0) (eps 0))
      using finite_subset by (auto simp: eps_def)
    define r where r ≡ rec_nat 0 (λn rec. nxt rec (eps n))
    have [simp]: r 0 = 0 r (Suc n) = nxt (r n) (eps n) for n
      by (auto simp: r_def)
    have σrM[simp]: σ (r n) ∈ M for n
      using True σ by blast
    have inf: infinite (σ −‘ mball (σ (r n)) (eps n)) for n
    proof (induction n)
      case 0 then show ?case
        by (simp add: inf0)
    next
      case (Suc n) then show ?case
        using nxt [of eps n r n] by simp
    qed
    then have r (Suc n) > r n for n
```

```
      by (simp add: nxt)
    then have strict_mono r
      by (simp add: strict_mono_Suc_iff)
    have d_less: d (σ (r n)) (σ (r (Suc n))) < 3 * eps n / 2 for n
      using nxt [OF _ inf] by simp
    have eps_plus: eps (k + n) = eps n * (1/2)^k for k n
      by (simp add: eps_def power_add field_simps)
    have *: d (σ (r n)) (σ (r (k + n))) < 3 * eps n for n k
    proof −
      have d (σ (r n)) (σ (r (k+n))) ≤ 3/2 * eps n * (∑ i<k. (1/2)^i)
      proof (induction k)
        case 0 then show ?case
          by simp
      next
        case (Suc k)
        have d (σ (r n)) (σ (r (Suc k + n))) ≤ d (σ (r n)) (σ (r (k + n))) + d
(σ (r (k + n))) (σ (r (Suc (k + n))))
          by (metis σrM add.commute add_Suc_right triangle)
        with d_less[of k+n] Suc show ?case
          by (simp add: algebra_simps eps_plus)
      qed
      also have ... < 3/2 * eps n * 2
        using geometric_sum [of 1/2::real k] by simp
      finally show ?thesis by simp
    qed
    have ∃ N. ∀ n≥N. ∀ n'≥N. d (σ (r n)) (σ (r n')) < ε if ε > 0 for ε
    proof −
      define N where N ≡ nat ⌈(log 2 (6*(B+1) / ε))⌉
      have §: b ≤ 2 ^ nat ⌈log 2 b⌉ for b
        by (smt (verit) less_log_of_power real_nat_ceiling_ge)
      have N: 6 * eps N ≤ ε
      using § [of (6*(B+1) / ε)] that by (auto simp: N_def eps_def field_simps)
      have d (σ (r N)) (σ (r n)) < 3 * eps N if n ≥ N for n
        by (metis * add.commute nat_le_iff_add that)
      then have ∀ n≥N. ∀ n'≥N. d (σ (r n)) (σ (r n')) < 3 * eps N + 3 * eps N
        by (smt (verit, best) σrM triangle'')
      with N show ?thesis
        by fastforce
    qed
    then have MCauchy (σ ∘ r)
      unfolding MCauchy_def using True σ by auto
    then have ∃ r. strict_mono r ∧ MCauchy (σ ∘ r)
      using ‹strict_mono r› by blast
  }
  moreover
  { assume R: ?rhs
    have mtotally_bounded S
      unfolding mtotally_bounded_def
    proof (intro strip)
```

```
      fix ε :: real
      assume ε > 0
      have False if §: ⋀K. ⟦finite K; K ⊆ S⟧ ⟹ ∃s∈S. s ∉ (⋃x∈K. mball x ε)
      proof −
       obtain f where f: ⋀K. ⟦finite K; K ⊆ S⟧ ⟹ f K ∈ S ∧ f K ∉ (⋃x∈K.
mball x ε)
         using § by metis
        define σ where σ ≡ wfrec less_than (λseq n. f (seq ‘ {..<n}))
        have σ_eq: σ n = f (σ ‘ {..<n}) for n
         by (simp add: cut_apply def_wfrec [OF σ_def])
        have [simp]: σ n ∈ S for n
         using wf_less_than
        proof (induction n rule: wf_induct_rule)
         case (less n) with f show ?case
           by (auto simp: σ_eq [of n])
        qed
        then have range σ ⊆ S by blast
        have σ: p < n ⟹ ε ≤ d (σ p) (σ n) for n p
         using f[of σ ‘ {..<n}] True by (fastforce simp: σ_eq [of n] Ball_def)
        then obtain r where strict_mono r MCauchy (σ ∘ r)
         by (meson R ‹range σ ⊆ S›)
        with ‹0 < ε› obtain N
         where N: ⋀n n′. ⟦n≥N; n′≥N⟧ ⟹ d (σ (r n)) (σ (r n′)) < ε
         by (force simp: MCauchy_def)
        show ?thesis
         using N [of N Suc (r N)] ‹strict_mono r›
       by (smt (verit) Suc_le_eq σ le_SucI order_refl strict_mono_imp_increasing)
      qed
      then show ∃K. finite K ∧ K ⊆ S ∧ S ⊆ (⋃x∈K. mball x ε)
       by blast
     qed
    }
    ultimately show ?thesis
     using True by blast
  qed
qed (use mtotally_bounded_imp_subset in auto)


lemma mtotally_bounded_subset:
  ⟦mtotally_bounded S; T ⊆ S⟧ ⟹ mtotally_bounded T
  by (meson mtotally_bounded_sequentially order_trans)

lemma mtotally_bounded_submetric:
  assumes mtotally_bounded S S ⊆ T T ⊆ M
  shows Metric_space.mtotally_bounded T d S
proof −
  interpret Submetric M d T
   using ‹T ⊆ M› by unfold_locales
  show ?thesis
```

```
    using assms
    unfolding sub.mtotally_bounded_def mtotally_bounded_def
    by (force simp: subset_iff elim!: all_forward ex_forward)
qed

lemma mtotally_bounded_absolute:
   mtotally_bounded S ⟷ S ⊆ M ∧ Metric_space.mtotally_bounded S d S
proof −
  have mtotally_bounded S if S ⊆ M Metric_space.mtotally_bounded S d S
  proof −
    interpret Submetric M d S
      using ‹S ⊆ M› by unfold_locales
    show ?thesis
      using that
    by (meson MCauchy_submetric mtotally_bounded_sequentially sub.mtotally_bounded_sequentially)
  qed
  moreover have mtotally_bounded S ⟹ Metric_space.mtotally_bounded S d S
    by (simp add: mtotally_bounded_imp_subset mtotally_bounded_submetric)
  ultimately show ?thesis
    using mtotally_bounded_imp_subset by blast
qed

lemma mtotally_bounded_closure_of:
  assumes mtotally_bounded S
  shows mtotally_bounded (mtopology closure_of S)
proof −
  have S ⊆ M
    by (simp add: assms mtotally_bounded_imp_subset)
  have mtotally_bounded(mtopology closure_of S)
    unfolding mtotally_bounded_def
  proof (intro strip)
    fix ε::real
    assume ε > 0
    then obtain K where finite K K ⊆ S and K: S ⊆ (⋃x∈K. mball x (ε/2))
      by (metis assms mtotally_bounded_def half_gt_zero)
    have mtopology closure_of S ⊆ (⋃x∈K. mball x ε)
      unfolding metric_closure_of
    proof clarsimp
      fix x
      assume x ∈ M and x: ∀r>0. ∃y∈S. y ∈ M ∧ d x y < r
      then obtain y where y ∈ S and y: d x y < ε/2
        using ‹0 < ε› half_gt_zero by blast
      then obtain x' where x' ∈ K y ∈ mball x' (ε/2)
        using K by auto
      then have d x' x < ε/2 + ε/2
        using triangle y ‹x ∈ M› commute by fastforce
      then show ∃x'∈K. x' ∈ M ∧ d x' x < ε
        using ‹K ⊆ S› ‹S ⊆ M› ‹x' ∈ K› by force
    qed
```

then show $\exists K.$ *finite* $K \wedge K \subseteq$ *mtopology closure_of* $S \wedge$ *mtopology closure_of* $S \subseteq (\bigcup x{\in}K.$ *mball* $x\ \varepsilon)$
using *closure_of_subset_Int* ‹$K \subseteq S$› ‹*finite* K› K **by** *fastforce*
qed
then show *?thesis*
by (*simp add*: *assms inf.absorb2 mtotally_bounded_imp_subset*)
qed

lemma *mtotally_bounded_closure_of_eq*:
$S \subseteq M \Longrightarrow$ *mtotally_bounded* (*mtopology closure_of* S) $\longleftrightarrow$ *mtotally_bounded* S
by (*metis closure_of_subset mtotally_bounded_closure_of mtotally_bounded_subset topspace_mtopology*)

lemma *mtotally_bounded_cauchy_sequence*:
assumes *MCauchy* σ
shows *mtotally_bounded* (*range* σ)
unfolding *MCauchy_def mtotally_bounded_def*
proof (*intro strip*)
fix ε::*real*
assume $\varepsilon > 0$
then obtain N **where** $\bigwedge n.\ N \leq n \Longrightarrow d\ (\sigma\ N)\ (\sigma\ n) < \varepsilon$
using *assms* **by** (*force simp*: *MCauchy_def*)
then have $\bigwedge m.\ \exists n{\leq}N.\ \sigma\ n \in M \wedge \sigma\ m \in M \wedge d\ (\sigma\ n)\ (\sigma\ m) < \varepsilon$
by (*metis MCauchy_def assms mdist_zero nle_le range_subsetD*)
then
show $\exists K.$ *finite* $K \wedge K \subseteq$ *range* $\sigma \wedge$ *range* $\sigma \subseteq (\bigcup x{\in}K.$ *mball* $x\ \varepsilon)$
by (*rule_tac* $x{=}\sigma$ ‘ $\{0..N\}$ **in** *exI*) *force*
qed

lemma *MCauchy_imp_mbounded*:
MCauchy $\sigma \Longrightarrow$ *mbounded* (*range* σ)
by (*simp add*: *mtotally_bounded_cauchy_sequence mtotally_bounded_imp_mbounded*)

5.7.12 Compactness in metric spaces

lemma *Bolzano_Weierstrass_property*:
assumes $S \subseteq U\ S \subseteq M$
shows
(∀ σ::*nat*⇒'*a*. *range* $\sigma \subseteq S$
$\longrightarrow$ ($\exists\, l\ r.\ l \in U \wedge$ *strict_mono* $r \wedge$ *limitin mtopology* ($\sigma \circ r$) l *sequentially*))
$\longleftrightarrow$
(∀ $T.\ T \subseteq S \wedge$ *infinite* $T \longrightarrow U \cap$ *mtopology derived_set_of* $T \neq \{\}$) (**is** *?lhs=?rhs*)
proof
assume L: *?lhs*
show *?rhs*
proof *clarify*
fix T

```
    assume T ⊆ S and infinite T
      and T: U ∩ mtopology derived_set_of T = {}
    then obtain σ :: nat⇒'a where inj σ range σ ⊆ T
      by (meson infinite_countable_subset)
    with L obtain l r where l ∈ U strict_mono r
          and lr: limitin mtopology (σ ∘ r) l sequentially
      by (meson ‹T ⊆ S› subset_trans)
    then obtain ε where ε > 0 and ε: ⋀y. y ∈ T ⟹ y = l ∨ ¬ d l y < ε
      using T ‹T ⊆ S› ‹S ⊆ M›
      by (force simp: metric_derived_set_of limitin_metric disjoint_iff)
    with lr have ∀F n in sequentially. σ (r n) ∈ M ∧ d (σ (r n)) l < ε
      by (auto simp: limitin_metric)
    then obtain N where N: d (σ (r N)) l < ε d (σ (r (Suc N))) l < ε
      using less_le_not_le by (auto simp: eventually_sequentially)
    moreover have σ (r N) ≠ l ∨ σ (r (Suc N)) ≠ l
      by (meson ‹inj σ› ‹strict_mono r› injD n_not_Suc_n strict_mono_eq)
    ultimately
    show False
      using ε ‹range σ ⊆ T› commute by fastforce
  qed
next
  assume R: ?rhs
  show ?lhs
  proof (intro strip)
    fix σ :: nat ⇒ 'a
    assume range σ ⊆ S
    show ∃l r. l ∈ U ∧ strict_mono r ∧ limitin mtopology (σ ∘ r) l sequentially
    proof (cases finite (range σ))
      case True
      then obtain m where infinite (σ −‘ {σ m})
        by (metis image_iff inf_img_fin_dom nat_not_finite)
      then obtain r where [iff]: strict_mono r and r: ⋀n::nat. r n ∈ σ −‘ {σ
m}
        using infinite_enumerate by blast
      have [iff]: σ m ∈ U σ m ∈ M
        using ‹range σ ⊆ S› assms by blast+
      show ?thesis
      proof (intro conjI exI)
        show limitin mtopology (σ ∘ r) (σ m) sequentially
          using r by (simp add: limitin_metric)
      qed auto
    next
      case False
      then obtain l where l ∈ U and l: l ∈ mtopology derived_set_of (range σ)
        by (meson R ‹range σ ⊆ S› disjoint_iff)
      then obtain g where g: ⋀ε. ε>0 ⟹ σ (g ε) ≠ l ∧ d l (σ (g ε)) < ε
        by (simp add: metric_derived_set_of) metis
      have range σ ⊆ M
        using ‹range σ ⊆ S› assms by auto
```

```
    have l ∈ M
      using l metric_derived_set_of by auto
    define E where  — a construction to ensure monotonicity
      E ≡ λrec n. insert (inverse (Suc n)) ((λi. d l (σ i)) ‘ (⋃k<n. {0..rec k}))
− {0}
    define r where r ≡ wfrec less_than (λrec n. g (Min (E rec n)))
    have (⋃k<n. {0..cut r less_than n k}) = (⋃k<n. {0..r k}) for n
      by (auto simp: cut_apply)
    then have r_eq: r n = g (Min (E r n)) for n
      by (metis E_def def_wfrec [OF r_def] wf_less_than)
    have dl_pos[simp]: d l (σ (r n)) > 0 for n
      using wf_less_than
    proof (induction n rule: wf_induct_rule)
      case (less n)
      then have ∗: Min (E r n) > 0
        using ‹l ∈ M› ‹range σ ⊆ M› by (auto simp: E_def image_subset_iff)
      show ?case
        using g [OF ∗] r_eq [of n]
        by (metis ‹l ∈ M› ‹range σ ⊆ M› mdist_pos_less range_subsetD)
    qed
    then have non_l: σ (r n) ≠ l for n
      using ‹range σ ⊆ M› mdist_pos_eq by blast
    have Min_pos: Min (E r n) > 0 for n
    using dl_pos ‹l ∈ M› ‹range σ ⊆ M› by (auto simp: E_def image_subset_iff)
    have d_small: d (σ(r n)) l < inverse(Suc n) for n
    proof −
      have d (σ(r n)) l < Min (E r n)
        by (simp add: ‹0 < Min (E r n)› commute g r_eq)
      also have ... ≤ inverse(Suc n)
        by (simp add: E_def)
      finally show ?thesis .
    qed
    have d_lt_d: d l (σ (r n)) < d l (σ i) if §: p < n i ≤ r p σ i ≠ l for i p n
    proof −
      have 1: d l (σ i) ∈ E r n
        using § ‹l ∈ M› ‹range σ ⊆ M›
        by (force simp: E_def image_subset_iff image_iff)
      have d l (σ (g (Min (E r n)))) < Min (E r n)
        by (rule conjunct2 [OF g [OF Min_pos]])
      also have Min (E r n) ≤ d l (σ i)
        using 1 unfolding E_def by (force intro!: Min.coboundedI)
      finally show ?thesis
        by (simp add: r_eq)
    qed
    have r: r p < r n if p < n for p n
    using d_lt_d [OF that] non_l by (meson linorder_not_le order_less_irrefl)

    show ?thesis
    proof (intro exI conjI)
```

show *strict_mono r*
by (*simp add*: *r strict_monoI*)
show *limitin mtopology* ($\sigma \circ r$) *l sequentially*
unfolding *limitin_metric*
proof (*intro conjI strip* ‹$l \in M$›)
fix ε :: *real*
assume $\varepsilon > 0$
then have $\forall_F$ *n in sequentially. inverse*(*Suc n*) $< \varepsilon$
using *Archimedean_eventually_inverse* **by** *auto*
then show $\forall_F$ *n in sequentially.* ($\sigma \circ r$) *n* $\in M \wedge$ *d* (($\sigma \circ r$) *n*) *l* $< \varepsilon$
by (*smt* (*verit*) ‹*range* $\sigma \subseteq M$› *commute comp_apply d_small eventually_mono range_subsetD*)
qed
qed (*use* ‹$l \in U$› **in** *auto*)
qed
qed
qed

More on Bolzano Weierstrass

lemma *Bolzano_Weierstrass_A*:
assumes *compactin mtopology S T* $\subseteq S$ *infinite T*
shows $S \cap$ *mtopology derived_set_of T* $\neq \{\}$
by (*simp add*: *assms compactin_imp_Bolzano_Weierstrass*)

lemma *Bolzano_Weierstrass_B*:
fixes σ :: *nat* $\Rightarrow$ *'a*
assumes *S* $\subseteq$ *M range* $\sigma \subseteq S$
and $\bigwedge T$. $[\![T \subseteq S \wedge$ *infinite T*$]\!] \Longrightarrow S \cap$ *mtopology derived_set_of T* $\neq \{\}$
shows $\exists\, l\ r.\ l \in S \wedge$ *strict_mono r* $\wedge$ *limitin mtopology* ($\sigma \circ r$) *l sequentially*
using *Bolzano_Weierstrass_property assms* **by** *blast*

lemma *Bolzano_Weierstrass_C*:
assumes $S \subseteq M$
assumes $\bigwedge\sigma$:: *nat* $\Rightarrow$ *'a. range* $\sigma \subseteq S \Longrightarrow$
($\exists\, l\ r.\ l \in S \wedge$ *strict_mono r* $\wedge$ *limitin mtopology* ($\sigma \circ r$) *l sequentially*)
shows *mtotally_bounded S*
unfolding *mtotally_bounded_sequentially*
by (*metis convergent_imp_MCauchy assms image_comp image_mono subset_UNIV subset_trans*)

lemma *Bolzano_Weierstrass_D*:
assumes $S \subseteq M$ $S \subseteq \bigcup \mathcal{C}$ **and** *openU*: $\bigwedge U.\ U \in \mathcal{C} \Longrightarrow$ *openin mtopology U*
assumes §: ($\forall \sigma$::*nat*$\Rightarrow$*'a. range* $\sigma \subseteq S$
$\longrightarrow$ ($\exists\, l\ r.\ l \in S \wedge$ *strict_mono r* $\wedge$ *limitin mtopology* ($\sigma \circ r$) *l sequentially*))
shows $\exists \varepsilon{>}0.\ \forall x \in S.\ \exists\, U \in \mathcal{C}.$ *mball x* $\varepsilon \subseteq U$
proof (*rule ccontr*)
assume $\neg$ ($\exists \varepsilon{>}0.\ \forall x \in S.\ \exists\, U \in \mathcal{C}.$ *mball x* $\varepsilon \subseteq U$)
then have $\forall n.\ \exists x{\in}S.\ \forall\, U{\in}\mathcal{C}.\ \neg$ *mball x* (*inverse* (*Suc n*)) $\subseteq U$

by *simp*
then obtain σ **where** $\bigwedge n.\ \sigma\ n \in S$
and σ: $\bigwedge n\ U.\ U \in \mathcal{C} \Longrightarrow \neg\ mball\ (\sigma\ n)\ (inverse\ (Suc\ n)) \subseteq U$
by *metis*
then obtain $l\ r$ **where** $l \in S$ *strict_mono* r
and *lr*: *limitin mtopology* $(\sigma \circ r)\ l$ *sequentially*
by (*meson* § *image_subsetI*)
with ‹$S \subseteq \bigcup\mathcal{C}$› **obtain** B **where** $l \in B\ B \in \mathcal{C}$
by *auto*
then obtain ε **where** $\varepsilon > 0$ **and** ε: $\bigwedge z.\ [\![z \in M;\ d\ z\ l < \varepsilon]\!] \Longrightarrow z \in B$
by (*metis opeU* [*OF* ‹$B \in \mathcal{C}$›] *commute in_mball openin_mtopology subset_iff*)
then have $\forall_F\ n$ *in sequentially*. $\sigma\ (r\ n) \in M \wedge d\ (\sigma\ (r\ n))\ l < \varepsilon/2$
using *lr half_gt_zero* **unfolding** *limitin_metric o_def* **by** *blast*
moreover have $\forall_F\ n$ *in sequentially*. $inverse\ (real\ (Suc\ n)) < \varepsilon/2$
using *Archimedean_eventually_inverse* ‹$0 < \varepsilon$› *half_gt_zero* **by** *blast*
ultimately obtain n **where** n: $d\ (\sigma\ (r\ n))\ l < \varepsilon/2$ $inverse\ (real\ (Suc\ n)) < \varepsilon/2$
by (*smt* (*verit*, *del_insts*) *eventually_sequentially le_add1 le_add2*)
have $x \in B$ **if** $d\ (\sigma\ (r\ n))\ x < inverse\ (Suc(r\ n))\ x \in M$ **for** x
proof −
have *rle*: $inverse\ (real\ (Suc\ (r\ n))) \leq inverse\ (real\ (Suc\ n))$
using ‹*strict_mono* r› *strict_mono_imp_increasing* **by** *auto*
have $d\ x\ l \leq d\ (\sigma\ (r\ n))\ x + d\ (\sigma\ (r\ n))\ l$
using *that* **by** (*metis triangle* ‹$\bigwedge n.\ \sigma\ n \in S$› ‹$l \in S$› ‹$S \subseteq M$› *commute subsetD*)
also have $... < \varepsilon$
using *that n rle* **by** *linarith*
finally show *?thesis*
by (*simp add*: ε *that*)
qed
then show *False*
using σ [*of* $B\ r\ n$] **by** (*simp add*: ‹$B \in \mathcal{C}$› *subset_iff*)
qed

lemma *Bolzano_Weierstrass_E*:
assumes *mtotally_bounded* S $S \subseteq M$
and S: $\bigwedge\mathcal{C}.\ [\![\bigwedge U.\ U \in \mathcal{C} \Longrightarrow openin\ mtopology\ U;\ S \subseteq \bigcup\mathcal{C}]\!] \Longrightarrow \exists\varepsilon{>}0.\ \forall x \in S.\ \exists\ U \in \mathcal{C}.\ mball\ x\ \varepsilon \subseteq U$
shows *compactin mtopology* S
proof (*clarsimp simp*: *compactin_def assms*)
fix $\mathcal{U}$ :: *'a set set*
assume $\mathcal{U}$: $\forall x{\in}\mathcal{U}.\ openin\ mtopology\ x$ **and** $S \subseteq \bigcup\mathcal{U}$
then obtain ε **where** $\varepsilon{>}0$ **and** ε: $\bigwedge x.\ x \in S \Longrightarrow \exists\ U \in \mathcal{U}.\ mball\ x\ \varepsilon \subseteq U$
by (*metis* S)
then obtain f **where** f: $\bigwedge x.\ x \in S \Longrightarrow f\ x \in \mathcal{U} \wedge mball\ x\ \varepsilon \subseteq f\ x$
by *metis*
then obtain K **where** *finite* K $K \subseteq S$ **and** K: $S \subseteq (\bigcup x{\in}K.\ mball\ x\ \varepsilon)$
by (*metis* ‹$0 < \varepsilon$› ‹*mtotally_bounded* S› *mtotally_bounded_def*)
show $\exists\mathcal{F}.\ finite\ \mathcal{F} \wedge \mathcal{F} \subseteq \mathcal{U} \wedge S \subseteq \bigcup\mathcal{F}$

proof (*intro conjI exI*)
show *finite* (*f* ‘ *K*)
by (*simp add*: ‹*finite K*›)
show *f* ‘ *K* ⊆ *U*
using ‹*K* ⊆ *S*› *f* **by** *blast*
show *S* ⊆ ⋃(*f* ‘ *K*)
using *K* ‹*K* ⊆ *S*› **by** (*force dest*: *f*)
qed
qed

lemma *compactin_eq_Bolzano_Weierstrass*:
compactin mtopology S ⟷
S ⊆ *M* ∧ (∀ *T*. *T* ⊆ *S* ∧ *infinite T* ⟶ *S* ∩ *mtopology derived_set_of T* ≠ {})
using *Bolzano_Weierstrass_C Bolzano_Weierstrass_D Bolzano_Weierstrass_E*
by (*smt* (*verit*, *del_insts*) *Bolzano_Weierstrass_property compactin_imp_Bolzano_Weierstrass compactin_subspace subset_refl topspace_mtopology*)

lemma *compactin_sequentially*:
shows *compactin mtopology S* ⟷
S ⊆ *M* ∧
((∀ σ::*nat*⇒'*a*. *range* σ ⊆ *S*
⟶ (∃ *l r*. *l* ∈ *S* ∧ *strict_mono r* ∧ *limitin mtopology* (σ ∘ *r*) *l sequentially*)))
by (*metis Bolzano_Weierstrass_property compactin_eq_Bolzano_Weierstrass subset_refl*)

lemma *compactin_imp_mtotally_bounded*:
compactin mtopology S ⟹ *mtotally_bounded S*
by (*simp add*: *Bolzano_Weierstrass_C compactin_sequentially*)

lemma *lebesgue_number*:
⟦*compactin mtopology S*; *S* ⊆ ⋃*C*; ⋀*U*. *U* ∈ *C* ⟹ *openin mtopology U*⟧
⟹ ∃ε>0. ∀ *x* ∈ *S*. ∃ *U* ∈ *C*. *mball x* ε ⊆ *U*
by (*simp add*: *Bolzano_Weierstrass_D compactin_sequentially*)

lemma *compact_space_sequentially*:
compact_space mtopology ⟷
(∀ σ::*nat*⇒'*a*. *range* σ ⊆ *M*
⟶ (∃ *l r*. *l* ∈ *M* ∧ *strict_mono r* ∧ *limitin mtopology* (σ ∘ *r*) *l sequentially*))
by (*simp add*: *compact_space_def compactin_sequentially*)

lemma *compact_space_eq_Bolzano_Weierstrass*:
compact_space mtopology ⟷
(∀ *S*. *S* ⊆ *M* ∧ *infinite S* ⟶ *mtopology derived_set_of S* ≠ {})
using *Int_absorb1* [*OF derived_set_of_subset_topspace* [*of mtopology*]]
by (*force simp*: *compact_space_def compactin_eq_Bolzano_Weierstrass*)

lemma *compact_space_nest*:
compact_space mtopology ⟷

(∀ *C*. (∀ *n*::*nat*. *closedin mtopology* (*C n*)) ∧ (∀ *n*. *C n* ≠ {}) ∧ *decseq C* ⟶ ⋂(*range C*) ≠ {})
(**is** *?lhs*=*?rhs*)
proof
assume *L*: *?lhs*
show *?rhs*
proof *clarify*
fix *C* :: *nat* ⇒ *'a set*
assume ∀ *n*. *closedin mtopology* (*C n*)
and ∀ *n*. *C n* ≠ {}
and *decseq C*
and ⋂ (*range C*) = {}
then obtain *K* **where** *K*: *finite K* ⋂(*C* ‘ *K*) = {}
by (*metis L compact_space_imp_nest*)
then obtain *k* **where** *K* ⊆ {..*k*}
using *finite_nat_iff_bounded_le* **by** *auto*
then have *C k* ⊆ ⋂(*C* ‘ *K*)
using ‹*decseq C*› **by** (*auto simp*:*decseq_def*)
then show *False*
by (*simp add*: *K* ‹∀ *n*. *C n* ≠ {}›)
qed
next
assume *R* [*rule_format*]: *?rhs*
show *?lhs*
unfolding *compact_space_sequentially*
proof (*intro strip*)
fix σ :: *nat* ⇒ *'a*
assume σ: *range* σ ⊆ *M*
have *mtopology closure_of* σ ‘ {*n*..} ≠ {} **for** *n*
using ‹*range* σ ⊆ *M*› **by** (*auto simp*: *closure_of_eq_empty image_subset_iff*)
moreover have *decseq* (λ*n*. *mtopology closure_of* σ ‘ {*n*..})
using *closure_of_mono image_mono* **by** (*smt* (*verit*) *atLeast_subset_iff decseq_def*)
ultimately obtain *l* **where** *l*: ⋀*n*. *l* ∈ *mtopology closure_of* σ ‘ {*n*..}
using *R* [*of* λ*n*. *mtopology closure_of* (σ ‘ {*n*..})] **by** *auto*
then have *l* ∈ *M* **and** ⋀*n*. ∀ *r*>*0*. ∃ *k*≥*n*. σ *k* ∈ *M* ∧ *d l* (σ *k*) < *r*
using *metric_closure_of* **by** *fastforce*+
then obtain *f* **where** *f*: ⋀*n r*. *r*>*0* ⟹ *f n r* ≥ *n* ∧ σ (*f n r*) ∈ *M* ∧ *d l* (σ (*f n r*)) < *r*
by *metis*
define *r* **where** *r* = *rec_nat* (*f 0 1*) (λ*n rec*. (*f* (*Suc rec*) (*inverse* (*Suc* (*Suc n*)))))
have *r*: *d l* (σ(*r n*)) < *inverse*(*Suc n*) **for** *n*
by (*induction n*) (*auto simp*: *rec_nat_0_imp* [*OF r_def*] *rec_nat_Suc_imp* [*OF r_def*] *f*)
have *r n* < *r*(*Suc n*) **for** *n*
by (*simp add*: *Suc_le_lessD f r_def*)
then have *strict_mono r*
by (*simp add*: *strict_mono_Suc_iff*)

```
    moreover have limitin mtopology (σ ∘ r) l sequentially
      proof (clarsimp simp: limitin_metric ‹l ∈ M›)
        fix ε :: real
        assume ε > 0
        then have (∀F n in sequentially. inverse (real (Suc n)) < ε)
          using Archimedean_eventually_inverse by blast
        then show ∀F n in sequentially. σ (r n) ∈ M ∧ d (σ (r n)) l < ε
          by eventually_elim (metis commute ‹range σ ⊆ M›  order_less_trans r
range_subsetD)
      qed
    ultimately show ∃l r. l ∈ M ∧ strict_mono r ∧ limitin mtopology (σ ∘ r) l
sequentially
      using ‹l ∈ M› by blast
  qed
qed


lemma (in discrete_metric) mcomplete_discrete_metric: disc.mcomplete
proof (clarsimp simp: disc.mcomplete_def)
  fix σ :: nat ⇒ 'a
  assume disc.MCauchy σ
  then obtain N where ⋀n. N ≤ n ⟹ σ N = σ n
   unfolding disc.MCauchy_def by (metis dd_def dual_order.refl order_less_irrefl
zero_less_one)
  moreover have range σ ⊆ M
    using ‹disc.MCauchy σ› disc.MCauchy_def by blast
  ultimately have limitin disc.mtopology σ (σ N) sequentially
    by (metis disc.limit_metric_sequentially disc.zero range_subsetD)
  then show ∃x. limitin disc.mtopology σ x sequentially ..
qed

lemma compact_space_imp_mcomplete: compact_space mtopology ⟹ mcomplete
  by (simp add: compact_space_nest mcomplete_nest)

lemma (in Submetric) compactin_imp_mcomplete:
   compactin mtopology A ⟹ sub.mcomplete
 by (simp add: compactin_subspace mtopology_submetric sub.compact_space_imp_mcomplete)

lemma (in Submetric) mcomplete_imp_closedin:
  assumes sub.mcomplete
  shows closedin mtopology A
proof −
  have l ∈ A
    if range σ ⊆ A and l: limitin mtopology σ l sequentially
    for σ :: nat ⇒ 'a and l
  proof −
    have sub.MCauchy σ
     using convergent_imp_MCauchy subset that by (force simp: MCauchy_submetric)
    then have limitin sub.mtopology σ l sequentially
```

using *assms* **unfolding** *sub.mcomplete_def*
using *l limitin_metric_unique limitin_submetric_iff trivial_limit_sequentially*
by *blast*
then show *?thesis*
using *limitin_submetric_iff* **by** *blast*
qed
then show *?thesis*
using *metric_closedin_iff_sequentially_closed subset* **by** *auto*
qed

lemma (**in** *Submetric*) *closedin_eq_mcomplete*:
mcomplete $\Longrightarrow$ (*closedin mtopology A* $\longleftrightarrow$ *sub.mcomplete*)
using *closedin_mcomplete_imp_mcomplete mcomplete_imp_closedin* **by** *blast*

lemma *compact_space_eq_mcomplete_mtotally_bounded*:
compact_space mtopology $\longleftrightarrow$ *mcomplete* $\wedge$ *mtotally_bounded M*
by (*meson Bolzano_Weierstrass_C compact_space_imp_mcomplete compact_space_sequentially limitin_mspace*
mcomplete_alt mtotally_bounded_sequentially subset_refl)

lemma *compact_closure_of_imp_mtotally_bounded*:
$[\![$*compactin mtopology* (*mtopology closure_of S*); *S* $\subseteq$ *M*$]\!]$
$\Longrightarrow$ *mtotally_bounded S*
using *compactin_imp_mtotally_bounded mtotally_bounded_closure_of_eq* **by** *blast*

lemma *mtotally_bounded_eq_compact_closure_of*:
assumes *mcomplete*
shows *mtotally_bounded S* $\longleftrightarrow$ *S* $\subseteq$ *M* $\wedge$ *compactin mtopology* (*mtopology closure_of S*)
(**is** *?lhs=?rhs*)
proof
assume *L*: *?lhs*
show *?rhs*
unfolding *compactin_subspace*
proof (*intro conjI*)
show *S* $\subseteq$ *M*
using *L* **by** (*simp add*: *mtotally_bounded_imp_subset*)
show *mtopology closure_of S* $\subseteq$ *topspace mtopology*
by (*simp add*: ‹*S* $\subseteq$ *M*› *closure_of_minimal*)
then have *MSM*: *mtopology closure_of S* $\subseteq$ *M*
by *auto*
interpret *S*: *Submetric M d mtopology closure_of S*
proof qed (*use MSM* **in** *auto*)
have *S.sub.mtotally_bounded* (*mtopology closure_of S*)
using *L mtotally_bounded_absolute mtotally_bounded_closure_of* **by** *blast*
then
show *compact_space* (*subtopology mtopology* (*mtopology closure_of S*))

using *S.closedin_mcomplete_imp_mcomplete S.mtopology_submetric S.sub.compact_space_eq_mc*
assms **by** *force*
qed
qed (*auto simp*: *compact_closure_of_imp_mtotally_bounded*)

lemma *compact_closure_of_eq_Bolzano_Weierstrass*:
compactin mtopology (*mtopology closure_of S*) ⟷
(∀ *T*. *infinite T* ∧ *T* ⊆ *S* ∧ *T* ⊆ *M* ⟶ *mtopology derived_set_of T* ≠ {})
(**is** *?lhs*=*?rhs*)
proof
assume *L*: *?lhs*
show *?rhs*
proof (*intro strip*)
fix *T*
assume *T*: *infinite T* ∧ *T* ⊆ *S* ∧ *T* ⊆ *M*
show *mtopology derived_set_of T* ≠ {}
proof (*intro compact_closure_of_imp_Bolzano_Weierstrass*)
show *compactin mtopology* (*mtopology closure_of S*)
by (*simp add*: *L*)
qed (*use T* **in** *auto*)
qed
next
have *compactin mtopology* (*mtopology closure_of S*)
if §: ⋀*T*. ⟦*infinite T*; *T* ⊆ *S*⟧ ⟹ *mtopology derived_set_of T* ≠ {} **and** *S* ⊆ *M* **for** *S*
unfolding *compactin_sequentially*
proof (*intro conjI strip*)
show *MSM*: *mtopology closure_of S* ⊆ *M*
using *closure_of_subset_topspace* **by** *fastforce*
fix σ :: *nat* ⇒ *'a*
assume σ: *range* σ ⊆ *mtopology closure_of S*
then have ∃ *y* ∈ *S*. *d* (σ *n*) *y* < *inverse*(*Suc n*) **for** *n*
by (*simp add*: *metric_closure_of image_subset_iff*) (*metis inverse_Suc of_nat_Suc*)
then obtain τ **where** τ: ⋀*n*. τ *n* ∈ *S* ∧ *d* (σ *n*) (τ *n*) < *inverse*(*Suc n*)
by *metis*
then have *range* τ ⊆ *S*
by *blast*
moreover
have ∗: ∀ *T*. *T* ⊆ *S* ∧ *infinite T* ⟶ *mtopology closure_of S* ∩ *mtopology derived_set_of T* ≠ {}
using §(*1*) *derived_set_of_mono derived_set_of_subset_closure_of* **by** *fastforce*
moreover have *S* ⊆ *mtopology closure_of S*
by (*simp add*: ‹*S* ⊆ *M*› *closure_of_subset*)
ultimately obtain *l r* **where** *lr*:
l ∈ *mtopology closure_of S strict_mono r limitin mtopology* (τ ∘ *r*) *l sequen-*

```
tially
      using Bolzano_Weierstrass_property ‹S ⊆ M› by metis
    then have l ∈ M
      using limitin_mspace by blast
    have dr_less: d ((σ ∘ r) n) ((τ ∘ r) n) < inverse(Suc n) for n
    proof −
      have d ((σ ∘ r) n) ((τ ∘ r) n) < inverse(Suc (r n))
        using τ by auto
      also have ... ≤ inverse(Suc n)
        using lr strict_mono_imp_increasing by auto
      finally show ?thesis .
    qed
    have limitin mtopology (σ ∘ r) l sequentially
      unfolding limitin_metric
    proof (intro conjI strip)
      show l ∈ M
        using limitin_mspace lr by blast
      fix ε :: real
      assume ε > 0
      then have ∀F n in sequentially. (τ ∘ r) n ∈ M ∧ d ((τ ∘ r) n) l < ε/2
        using lr half_gt_zero limitin_metric by blast
      moreover have ∀F n in sequentially. inverse (real (Suc n)) < ε/2
        using Archimedean_eventually_inverse ‹0 < ε› half_gt_zero by blast
      then have ∀F n in sequentially. d ((σ ∘ r) n) ((τ ∘ r) n) < ε/2
        by eventually_elim (smt (verit, del_insts) dr_less)
      ultimately have ∀F n in sequentially. d ((σ ∘ r) n) l < ε/2 + ε/2
          by eventually_elim (smt (verit) triangle ‹l ∈ M› MSM σ comp_apply
order_trans range_subsetD)
      then show ∀F n in sequentially. (σ ∘ r) n ∈ M ∧ d ((σ ∘ r) n) l < ε
        apply eventually_elim
        using ‹mtopology closure_of S ⊆ M› σ by auto
    qed
     with lr show ∃l r. l ∈ mtopology closure_of S ∧ strict_mono r ∧ limitin
mtopology (σ ∘ r) l sequentially
      by blast
  qed
  then show ?rhs ⟹ ?lhs
    by (metis Int_subset_iff closure_of_restrict inf_le1 topspace_mtopology)
qed

end

lemma (in discrete_metric) mtotally_bounded_discrete_metric:
   disc.mtotally_bounded S ⟷ finite S ∧ S ⊆ M (is ?lhs=?rhs)
proof
  assume L: ?lhs
  show ?rhs
  proof
    show finite S
```

by (*metis* (*no_types*) *L closure_of_subset_Int compactin_discrete_topology disc.mtotally_bounded_eq_compact_closure_of*
disc.topspace_mtopology discrete_metric.mcomplete_discrete_metric inf.absorb_iff2 mtopology_discrete_metric finite_subset)
show $S \subseteq M$
by (*simp add*: *L disc.mtotally_bounded_imp_subset*)
qed
qed (*simp add*: *disc.finite_imp_mtotally_bounded*)

context *Metric_space*
begin

lemma *derived_set_of_infinite_openin_metric*:
mtopology derived_set_of S =
$\{x \in M.\ \forall U.\ x \in U \wedge$ *openin mtopology U* $\longrightarrow$ *infinite*$(S \cap U)\}$
by (*simp add*: *derived_set_of_infinite_openin Hausdorff_space_mtopology*)

lemma *derived_set_of_infinite_1*:
assumes *infinite* $(S \cap$ *mball x* $\varepsilon)$
shows *infinite* $(S \cap$ *mcball x* $\varepsilon)$
by (*meson Int_mono assms finite_subset mball_subset_mcball subset_refl*)

lemma *derived_set_of_infinite_2*:
assumes *openin mtopology U* $\bigwedge\varepsilon.\ 0 < \varepsilon \Longrightarrow$ *infinite* $(S \cap$ *mcball x* $\varepsilon)$ **and** $x \in U$
shows *infinite* $(S \cap U)$
by (*metis assms openin_mtopology_mcball finite_Int inf.absorb_iff2 inf_assoc*)

lemma *derived_set_of_infinite_mball*:
mtopology derived_set_of S $= \{x \in M.\ \forall e{>}0.$ *infinite*$(S \cap$ *mball x e*$)\}$
unfolding *derived_set_of_infinite_openin_metric*
by (*meson centre_in_mball_iff openin_mball derived_set_of_infinite_1 derived_set_of_infinite_2*)

lemma *derived_set_of_infinite_mcball*:
mtopology derived_set_of S $= \{x \in M.\ \forall e{>}0.$ *infinite*$(S \cap$ *mcball x e*$)\}$
unfolding *derived_set_of_infinite_openin_metric*
by (*meson centre_in_mball_iff openin_mball derived_set_of_infinite_1 derived_set_of_infinite_2*)

end

5.7.13 Continuous functions on metric spaces

context *Metric_space*
begin

lemma *continuous_map_to_metric*:

continuous_map X mtopology f ⟷
(∀ *x* ∈ *topspace X*. ∀ *ε*>*0*. ∃ *U*. *openin X U* ∧ *x* ∈ *U* ∧ (∀ *y*∈*U*. *f y* ∈ *mball* (*f x*) *ε*))
(**is** *?lhs*=*?rhs*)
proof
show *?lhs* ⟹ *?rhs*
unfolding *continuous_map_eq_topcontinuous_at topcontinuous_at_def*
by (*metis PiE centre_in_mball_iff openin_mball topspace_mtopology*)
next
assume *R*: *?rhs*
then have ∀ *x*∈*topspace X*. *f x* ∈ *M*
by (*meson gt_ex in_mball*)
moreover
have ⋀*x V*. ⟦*x* ∈ *topspace X*; *openin mtopology V*; *f x* ∈ *V*⟧ ⟹ ∃ *U*. *openin X U* ∧ *x* ∈ *U* ∧ (∀ *y*∈*U*. *f y* ∈ *V*)
unfolding *openin_mtopology* **by** (*metis Int_iff R inf.orderE*)
ultimately
show *?lhs*
by (*simp add*: *continuous_map_eq_topcontinuous_at topcontinuous_at_def*)
qed

lemma *continuous_map_from_metric*:
continuous_map mtopology X f ⟷
f ∈ *M* → *topspace X* ∧
(∀ *a* ∈ *M*. ∀ *U*. *openin X U* ∧ *f a* ∈ *U* ⟶ (∃ *r*>*0*. ∀ *x*. *x* ∈ *M* ∧ *d a x* < *r* ⟶ *f x* ∈ *U*))
proof (*cases f ' M* ⊆ *topspace X*)
case *True*
then show *?thesis*
by (*fastforce simp*: *continuous_map openin_mtopology subset_eq*)
next
case *False*
then show *?thesis*
by (*simp add*: *continuous_map_def image_subset_iff_funcset*)
qed

An abstract formulation, since the limits do not have to be sequential

lemma *continuous_map_uniform_limit*:
assumes *contf*: ∀ *F* *ξ* *in F*. *continuous_map X mtopology* (*f ξ*)
and *dfg*: ⋀*ε*. *0* < *ε* ⟹ ∀ *F* *ξ* *in F*. ∀ *x* ∈ *topspace X*. *g x* ∈ *M* ∧ *d* (*f ξ x*) (*g x*) < *ε*
and *nontriv*: ¬ *trivial_limit F*
shows *continuous_map X mtopology g*
unfolding *continuous_map_to_metric*
proof (*intro strip*)
fix *x* **and** *ε*::*real*
assume *x* ∈ *topspace X* **and** *ε* > *0*
then obtain *ξ* **where** *k*: *continuous_map X mtopology* (*f ξ*)
and *gM*: ∀ *x* ∈ *topspace X*. *g x* ∈ *M*

and *third*: $\forall x \in$ *topspace* X. $d\ (f\ \xi\ x)\ (g\ x) < \varepsilon/3$
using *eventually_conj* [*OF contf*] *contf dfg* [*of* $\varepsilon/3$] *eventually_happens'* [*OF nontriv*]
by (*smt* (*verit*, *ccfv_SIG*) *zero_less_divide_iff*)
then obtain U **where** U: *openin* $X\ U\ x \in U$ **and** *Uthird*: $\forall y{\in}U$. $d\ (f\ \xi\ y)\ (f\ \xi\ x) < \varepsilon/3$
unfolding *continuous_map_to_metric*
by (*metis* ‹$0 < \varepsilon$› ‹$x \in$ *topspace* X› *commute divide_pos_pos in_mball zero_less_numeral*)
have *f_inM*: $f\ \xi\ y \in M$ **if** $y{\in}U$ **for** y
using U *k openin_subset that* **by** (*fastforce simp*: *continuous_map_def*)
have $d\ (g\ y)\ (g\ x) < \varepsilon$ **if** $y{\in}U$ **for** y
proof −
have $g\ y \in M$
using U *gM openin_subset that* **by** *blast*
have $d\ (g\ y)\ (g\ x) \le d\ (g\ y)\ (f\ \xi\ x) + d\ (f\ \xi\ x)\ (g\ x)$
by (*simp add*: U ‹$g\ y \in M$› ‹$x \in$ *topspace* X› *f_inM gM triangle*)
also have $\ldots \le d\ (g\ y)\ (f\ \xi\ y) + d\ (f\ \xi\ y)\ (f\ \xi\ x) + d\ (f\ \xi\ x)\ (g\ x)$
by (*simp add*: U ‹$g\ y \in M$› *commute f_inM that triangle'*)
also have $\ldots < \varepsilon/3 + \varepsilon/3 + \varepsilon/3$
by (*smt* (*verit*) $U(1)$ *Uthird* ‹$x \in$ *topspace* X› *commute openin_subset subsetD that third*)
finally show *?thesis* **by** *simp*
qed
with U *gM* **show** $\exists U$. *openin* $X\ U \wedge x \in U \wedge (\forall y{\in}U.\ g\ y \in mball\ (g\ x)\ \varepsilon)$
by (*metis commute in_mball in_mono openin_subset*)
qed

lemma *continuous_map_uniform_limit_alt*:
assumes *contf*: $\forall_F\ \xi$ *in* F. *continuous_map* X *mtopology* $(f\ \xi)$
and *gim*: $g \in$ *topspace* $X \to M$
and *dfg*: $\bigwedge\varepsilon.\ 0 < \varepsilon \Longrightarrow \forall_F\ \xi$ *in* F. $\forall x \in$ *topspace* X. $d\ (f\ \xi\ x)\ (g\ x) < \varepsilon$
and *nontriv*: $\neg$ *trivial_limit* F
shows *continuous_map* X *mtopology* g
proof (*rule continuous_map_uniform_limit* [*OF contf*])
fix ε :: *real*
assume $\varepsilon > 0$
with *gim dfg* **show** $\forall_F\ \xi$ *in* F. $\forall x{\in}$*topspace* X. $g\ x \in M \wedge d\ (f\ \xi\ x)\ (g\ x) < \varepsilon$
by (*simp add*: *Pi_iff*)
qed (*use nontriv* **in** *auto*)

lemma *continuous_map_uniformly_Cauchy_limit*:
assumes *mcomplete*
assumes *contf*: $\forall_F\ n$ *in sequentially*. *continuous_map* X *mtopology* $(f\ n)$
and *Cauchy'*: $\bigwedge\varepsilon.\ \varepsilon > 0 \Longrightarrow \exists N.\ \forall m\ n\ x.\ N \le m \longrightarrow N \le n \longrightarrow x \in$ *topspace* $X \longrightarrow d\ (f\ m\ x)\ (f\ n\ x) < \varepsilon$
obtains g **where**
continuous_map X *mtopology* g

⋀ε. 0 < ε ⟹ ∀ F n in sequentially. ∀ x∈topspace X. d (f n x) (g x) < ε
proof −
have ⋀x. x ∈ topspace X ⟹ ∃ l. limitin mtopology (λn. f n x) l sequentially
using ‹mcomplete› [unfolded mcomplete, rule_format] assms
by (smt (verit) contf continuous_map_def eventually_mono topspace_mtopology Pi_iff)
then obtain g **where** g: ⋀x. x ∈ topspace X ⟹ limitin mtopology (λn. f n x) (g x) sequentially
by metis
show thesis
proof
show ∀ F n in sequentially. ∀ x∈topspace X. d (f n x) (g x) < ε
if ε > 0 **for** ε :: real
proof −
obtain N **where** N: ⋀m n x. ⟦N ≤ m; N ≤ n; x ∈ topspace X⟧ ⟹ d (f m x) (f n x) < ε/2
by (meson Cauchy' ‹0 < ε› half_gt_zero)
obtain P **where** P: ⋀n x. ⟦n ≥ P; x ∈ topspace X⟧ ⟹ f n x ∈ M
using contf **by** (auto simp: eventually_sequentially continuous_map_def)
show ?thesis
proof (intro eventually_sequentiallyI strip)
fix n x
assume max N P ≤ n **and** x: x ∈ topspace X
obtain L **where** g x ∈ M **and** L: ∀ n≥L. f n x ∈ M ∧ d (f n x) (g x) < ε/2
using g [OF x] ‹ε > 0› **unfolding** limitin_metric
by (metis (no_types, lifting) eventually_sequentially half_gt_zero)
define n' **where** n' ≡ Max{L,N,P}
have L': ∀ m ≥ n'. f m x ∈ M ∧ d (f m x) (g x) < ε/2
using L **by** (simp add: n'_def)
moreover
have d (f n x) (f n' x) < ε/2
using N [of n n' x] ‹max N P ≤ n› n'_def x **by** fastforce
ultimately have d (f n x) (g x) < ε/2 + ε/2
by (smt (verit, ccfv_SIG) P ‹g x ∈ M› ‹max N P ≤ n› le_refl max.bounded_iff mdist_zero triangle' x)
then show d (f n x) (g x) < ε **by** simp
qed
qed
then show continuous_map X mtopology g
by (smt (verit, del_insts) eventually_mono g limitin_mspace trivial_limit_sequentially continuous_map_uniform_limit [OF contf])
qed
qed

lemma metric_continuous_map:
assumes Metric_space M' d'
shows
continuous_map mtopology (Metric_space.mtopology M' d') f ⟷
f ' M ⊆ M' ∧ (∀ a ∈ M. ∀ε>0. ∃δ>0. (∀ x. x ∈ M ∧ d a x < δ ⟶ d' (f a)

```
(f x) < ε))
   (is ?lhs = ?rhs)
proof −
  interpret M': Metric_space M' d'
    by (simp add: assms)
  show ?thesis
  proof
    assume L: ?lhs
    show ?rhs
    proof (intro conjI strip)
      show f ‘ M ⊆ M'
        using L by (auto simp: continuous_map_def)
      fix a and ε :: real
      assume a ∈ M and ε > 0
      then have openin mtopology {x ∈ M. f x ∈ M'.mball (f a) ε} f a ∈ M'
        using L unfolding continuous_map_def by fastforce+
      then obtain δ where δ > 0 mball a δ ⊆ {x ∈ M. f x ∈ M' ∧ d' (f a) (f x)
< ε}
        using ‹0 < ε› ‹a ∈ M› openin_mtopology by auto
      then show ∃δ>0. ∀x. x ∈ M ∧ d a x < δ ⟶ d' (f a) (f x) < ε
        using ‹a ∈ M› in_mball by blast
    qed
  next
    assume R: ?rhs
    show ?lhs
      unfolding continuous_map_def
    proof (intro conjI strip)
      fix U
      assume openin M'.mtopology U
      then show openin mtopology {x ∈ topspace mtopology. f x ∈ U}
      apply (simp add: continuous_map_def openin_mtopology M'.openin_mtopology
subset_iff)
        by (metis R image_subset_iff)
    qed (use R in auto)
  qed
qed

end
```

5.7.14 Completely metrizable spaces

These spaces are topologically complete

```
definition completely_metrizable_space where
 completely_metrizable_space X ≡
      ∃M d. Metric_space M d ∧ Metric_space.mcomplete M d ∧ X = Met-
ric_space.mtopology M d

lemma empty_completely_metrizable_space:
  completely_metrizable_space trivial_topology
```

unfolding *completely_metrizable_space_def subtopology_eq_discrete_topology_empty* [*symmetric*]
by (*metis Metric_space.mcomplete_empty_mspace discrete_metric.mtopology_discrete_metric metric_M_dd*)

lemma *completely_metrizable_imp_metrizable_space*:
completely_metrizable_space X ⟹ *metrizable_space X*
using *completely_metrizable_space_def metrizable_space_def* **by** *auto*

lemma (**in** *Metric_space*) *completely_metrizable_space_mtopology*:
mcomplete ⟹ *completely_metrizable_space mtopology*
using *Metric_space_axioms completely_metrizable_space_def* **by** *blast*

lemma *completely_metrizable_space_discrete_topology*:
completely_metrizable_space (*discrete_topology U*)
unfolding *completely_metrizable_space_def*
by (*metis discrete_metric.mcomplete_discrete_metric discrete_metric.mtopology_discrete_metric metric_M_dd*)

lemma *completely_metrizable_space_euclidean*:
completely_metrizable_space (*euclidean*:: *'a*::*complete_space topology*)
using *Met_TC.completely_metrizable_space_mtopology complete_UNIV* **by** *auto*

lemma *completely_metrizable_space_closedin*:
assumes *X*: *completely_metrizable_space X* **and** *S*: *closedin X S*
shows *completely_metrizable_space*(*subtopology X S*)
proof −
obtain *M d* **where** *Metric_space M d* **and** *comp*: *Metric_space.mcomplete M d*

and *Xeq*: *X* = *Metric_space.mtopology M d*
using *assms completely_metrizable_space_def* **by** *blast*
then interpret *Metric_space M d*
by *blast*
show *?thesis*
unfolding *completely_metrizable_space_def*
proof (*intro conjI exI*)
show *Metric_space S d*
using *S Xeq closedin_subset subspace* **by** *force*
have *sub*: *Submetric_axioms M S*
by (*metis S Xeq closedin_metric Submetric_axioms_def*)
then show *Metric_space.mcomplete S d*
using *S Submetric.closedin_mcomplete_imp_mcomplete Submetric_def Xeq comp* **by** *blast*
show *subtopology X S* = *Metric_space.mtopology S d*
by (*metis Metric_space_axioms Xeq sub Submetric.intro Submetric.mtopology_submetric*)
qed
qed

lemma *completely_metrizable_space_cbox*: *completely_metrizable_space* (*top_of_set*

```
(cbox a b))
  using closed_closedin completely_metrizable_space_closedin completely_metrizable_space_euclidean
by blast


lemma homeomorphic_completely_metrizable_space_aux:
 assumes homXY: X homeomorphic_space Y and X: completely_metrizable_space
X
 shows completely_metrizable_space Y
proof −
 obtain f g where hmf: homeomorphic_map X Y f and hmg: homeomorphic_map
Y X g
   and fg: ⋀x. x ∈ topspace X ⟹ g(f x) = x ⋀y. y ∈ topspace Y ⟹ f(g y) = y
   and fim: f ∈ topspace X → topspace Y and gim: g ∈ topspace Y → topspace
X
   using homXY
   using homeomorphic_space_unfold by blast
 obtain M d where Md: Metric_space M d Metric_space.mcomplete M d and
Xeq: X = Metric_space.mtopology M d
   using X by (auto simp: completely_metrizable_space_def)
 then interpret MX: Metric_space M d by metis
 define D where D ≡ λx y. d (g x) (g y)
 have Metric_space (topspace Y) D
 proof
   show (D x y = 0) ⟷ (x = y) if x ∈ topspace Y y ∈ topspace Y for x y
     unfolding D_def
     by (metis that MX.topspace_mtopology MX.zero Xeq fg gim Pi_iff)
   show D x z ≤ D x y +D y z
     if x ∈ topspace Y y ∈ topspace Y z ∈ topspace Y for x y z
     using that MX.triangle Xeq gim by (auto simp: D_def)
 qed (auto simp: D_def MX.commute)
 then interpret MY: Metric_space topspace Y λx y. D x y by metis
 show ?thesis
   unfolding completely_metrizable_space_def
 proof (intro exI conjI)
   show Metric_space (topspace Y) D
     using MY.Metric_space_axioms by blast
   have gball: g ‘ MY.mball y r = MX.mball (g y) r if y ∈ topspace Y for y r
   using that MX.topspace_mtopology Xeq gim hmg homeomorphic_imp_surjective_map
     unfolding MX.mball_def MY.mball_def  by (fastforce simp: D_def)
   have ∃r>0. MY.mball y r ⊆ S if openin Y S and y ∈ S for S y
   proof −
     have openin X (g‘S)
       using hmg homeomorphic_map_openness_eq that by auto
     then obtain r where r>0 MX.mball (g y) r ⊆ g‘S
       using MX.openin_mtopology Xeq ‹y ∈ S› by auto
     then show ?thesis
         by (smt (verit, ccfv_SIG) MY.in_mball gball fg image_iff in_mono
openin_subset subsetI that(1))
```

```
    qed
    moreover have openin Y S
      if S ⊆ topspace Y and ⋀y. y ∈ S ⟹ ∃r>0. MY.mball y r ⊆ S for S
    proof −
      have ⋀x. x ∈ g‘S ⟹ ∃r>0. MX.mball x r ⊆ g‘S
        by (smt (verit) gball imageE image_mono subset_iff that)
      then have openin X (g‘S)
        using MX.openin_mtopology Xeq gim that(1) by auto
      then show ?thesis
        using hmg homeomorphic_map_openness_eq that(1) by blast
    qed
    ultimately show Yeq: Y = MY.mtopology
      unfolding topology_eq MY.openin_mtopology by (metis openin_subset)

    show MY.mcomplete
      unfolding MY.mcomplete_def
    proof (intro strip)
      fix σ
      assume σ: MY.MCauchy σ
      have MX.MCauchy (g ∘ σ)
        unfolding MX.MCauchy_def
      proof (intro conjI strip)
        show range (g ∘ σ) ⊆ M
          using MY.MCauchy_def Xeq σ gim by auto
        fix ε :: real
        assume ε > 0
        then obtain N where ∀n n'. N ≤ n ⟶ N ≤ n' ⟶ D (σ n) (σ n') < ε
          using MY.MCauchy_def σ by presburger
        then show ∃N. ∀n n'. N ≤ n ⟶ N ≤ n' ⟶ d ((g ∘ σ) n) ((g ∘ σ) n')
< ε
          by (auto simp: o_def D_def)
      qed
      then obtain x where x: limitin MX.mtopology (g ∘ σ) x sequentially x ∈
topspace X
        using MX.limitin_mspace MX.topspace_mtopology Md Xeq unfolding
MX.mcomplete_def
        by blast
      with x have limitin MY.mtopology (f ∘ (g ∘ σ)) (f x) sequentially
      by (metis Xeq Yeq continuous_map_limit hmf homeomorphic_imp_continuous_map)
      moreover have f ∘ (g ∘ σ) = σ
        using ‹MY.MCauchy σ› by (force simp add: fg MY.MCauchy_def sub-
set_iff)
      ultimately have limitin MY.mtopology σ (f x) sequentially by simp
      then show ∃y. limitin MY.mtopology σ y sequentially
        by blast
    qed
  qed
qed
```

lemma *homeomorphic_completely_metrizable_space*:
  X homeomorphic_space Y
    ⟹ *completely_metrizable_space X* ⟷ *completely_metrizable_space Y*
 by (*meson homeomorphic_completely_metrizable_space_aux homeomorphic_space_sym*)

lemma *completely_metrizable_space_retraction_map_image*:
  assumes *r*: *retraction_map X Y r* **and** *X*: *completely_metrizable_space X*
  shows *completely_metrizable_space Y*
proof −
  obtain *s* **where** *s*: *retraction_maps X Y r s*
    using *r retraction_map_def* **by** *blast*
  then have *subtopology X* (*s ' topspace Y*) *homeomorphic_space Y*
    using *retraction_maps_section_image2* **by** *blast*
  then show *?thesis*
    by (*metis X retract_of_space_imp_closedin retraction_maps_section_image1*

      homeomorphic_completely_metrizable_space completely_metrizable_space_closedin

      completely_metrizable_imp_metrizable_space metrizable_imp_Hausdorff_space
s)
qed

5.7.15 Product metric

For the nicest fit with the main Euclidean theories, we choose the Euclidean product, though other definitions of the product work.

definition *prod_dist* ≡ λ*d1 d2* (*x*,*y*) (*x'*,*y'*). *sqrt*(*d1 x x' ^ 2 + d2 y y' ^ 2*)

locale *Metric_space12 = M1*: *Metric_space M1 d1 + M2*: *Metric_space M2 d2*
for *M1 d1 M2 d2*

lemma (**in** *Metric_space12*) *prod_metric*: *Metric_space* (*M1 × M2*) (*prod_dist d1 d2*)
proof
  fix *x y z*
  assume *xyz*: *x* ∈ *M1 × M2 y* ∈ *M1 × M2 z* ∈ *M1 × M2*
  have *sqrt* ((*d1 x1 z1*)^2 + (*d2 x2 z2*)^2) ≤ *sqrt* ((*d1 x1 y1*)^2 + (*d2 x2 y2*)^2) + *sqrt* ((*d1 y1 z1*)^2 + (*d2 y2 z2*)^2)
      (**is** *sqrt ?L* ≤ *?R*)
    if *x* = (*x1*, *x2*) *y* = (*y1*, *y2*) *z* = (*z1*, *z2*)
    for *x1 x2 y1 y2 z1 z2*
  proof −
    have *tri*: *d1 x1 z1* ≤ *d1 x1 y1* + *d1 y1 z1 d2 x2 z2* ≤ *d2 x2 y2* + *d2 y2 z2*
      using *that xyz M1.triangle* [*of x1 y1 z1*] *M2.triangle* [*of x2 y2 z2*] **by** *auto*
    show *?thesis*
    proof (*rule real_le_lsqrt*)
      have *?L* ≤ (*d1 x1 y1* + *d1 y1 z1*)^2 + (*d2 x2 y2* + *d2 y2 z2*)^2
        using *tri* **by** (*smt* (*verit*) *M1.nonneg M2.nonneg power_mono*)
      also have ... ≤ *?R*^{2}

by (*metis real_sqrt_sum_squares_triangle_ineq sqrt_le_D*)
finally show *?L* ≤ *?R*^{2} **.**
qed *auto*
qed
then show *prod_dist d1 d2 x z* ≤ *prod_dist d1 d2 x y* + *prod_dist d1 d2 y z*
by (*simp add*: *prod_dist_def case_prod_unfold*)
qed (*auto simp*: *M1.commute M2.commute case_prod_unfold prod_dist_def*)

sublocale *Metric_space12* ⊆ *Prod_metric*: *Metric_space M1*×*M2 prod_dist d1 d2*
by (*simp add*: *prod_metric*)

For easy reference to theorems outside of the locale

lemma *Metric_space12_mspace_mdist*:
Metric_space12 (*mspace m1*) (*mdist m1*) (*mspace m2*) (*mdist m2*)
by (*simp add*: *Metric_space12_def Metric_space_mspace_mdist*)

definition *prod_metric* **where**
prod_metric ≡ λ*m1 m2*. *metric* (*mspace m1* × *mspace m2*, *prod_dist* (*mdist m1*) (*mdist m2*))

lemma *submetric_prod_metric*:
submetric (*prod_metric m1 m2*) (*S* × *T*) = *prod_metric* (*submetric m1 S*) (*submetric m2 T*)
apply (*simp add*: *prod_metric_def*)
by (*simp add*: *submetric_def Metric_space.mspace_metric Metric_space.mdist_metric Metric_space12.prod_metric Metric_space12_def Metric_space_mspace_mdist Times_Int_Times*)

lemma *mspace_prod_metric* [*simp*]:
mspace (*prod_metric m1 m2*) = *mspace m1* × *mspace m2*
by (*simp add*: *prod_metric_def Metric_space.mspace_metric Metric_space12.prod_metric Metric_space12_mspace_mdist*)

lemma *mdist_prod_metric* [*simp*]:
mdist (*prod_metric m1 m2*) = *prod_dist* (*mdist m1*) (*mdist m2*)
by (*metis Metric_space.mdist_metric Metric_space12.prod_metric Metric_space12_mspace_mdist prod_metric_def*)

lemma *prod_dist_dist* [*simp*]: *prod_dist dist dist* = *dist*
by (*simp add*: *prod_dist_def dist_prod_def fun_eq_iff*)

lemma *prod_metric_euclidean* [*simp*]:
prod_metric euclidean_metric euclidean_metric = *euclidean_metric*
by (*simp add*: *prod_metric_def euclidean_metric_def*)

context *Metric_space12*
begin

lemma *component_le_prod_metric*:

d1 x1 x2 ≤ *prod_dist d1 d2* (*x1*,*y1*) (*x2*,*y2*) *d2 y1 y2* ≤ *prod_dist d1 d2* (*x1*,*y1*) (*x2*,*y2*)
by (*auto simp*: *prod_dist_def*)

lemma *prod_metric_le_components*:
⟦*x1* ∈ *M1*; *y1* ∈ *M1*; *x2* ∈ *M2*; *y2* ∈ *M2*⟧
⟹ *prod_dist d1 d2* (*x1*,*x2*) (*y1*,*y2*) ≤ *d1 x1 y1* + *d2 x2 y2*
by (*auto simp*: *prod_dist_def sqrt_sum_squares_le_sum*)

lemma *mball_prod_metric_subset*:
Prod_metric.mball (*x*,*y*) *r* ⊆ *M1.mball x r* × *M2.mball y r*
by *clarsimp* (*smt* (*verit*, *best*) *component_le_prod_metric*)

lemma *mcball_prod_metric_subset*:
Prod_metric.mcball (*x*,*y*) *r* ⊆ *M1.mcball x r* × *M2.mcball y r*
by *clarsimp* (*smt* (*verit*, *best*) *component_le_prod_metric*)

lemma *mball_subset_prod_metric*:
M1.mball x1 r1 × *M2.mball x2 r2* ⊆ *Prod_metric.mball* (*x1*,*x2*) (*r1* + *r2*)
using *prod_metric_le_components* **by** *force*

lemma *mcball_subset_prod_metric*:
M1.mcball x1 r1 × *M2.mcball x2 r2* ⊆ *Prod_metric.mcball* (*x1*,*x2*) (*r1* + *r2*)
using *prod_metric_le_components* **by** *force*

lemma *mtopology_prod_metric*:
Prod_metric.mtopology = *prod_topology M1.mtopology M2.mtopology*
unfolding *prod_topology_def*
proof (*rule topology_base_unique* [*symmetric*])
fix *U*
assume *U* ∈ {*S* × *T* |*S T*. *openin M1.mtopology S* ∧ *openin M2.mtopology T*}
then obtain *S T* **where** *Ueq*: *U* = *S* × *T*
and *S*: *openin M1.mtopology S* **and** *T*: *openin M2.mtopology T*
by *auto*
have *S* ⊆ *M1*
using *M1.openin_mtopology S* **by** *auto*
have *T* ⊆ *M2*
using *M2.openin_mtopology T* **by** *auto*
show *openin Prod_metric.mtopology U*
unfolding *Prod_metric.openin_mtopology*
proof (*intro conjI strip*)
show *U* ⊆ *M1* × *M2*
using *Ueq* **by** (*simp add*: *Sigma_mono* ‹*S* ⊆ *M1*› ‹*T* ⊆ *M2*›)
fix *z*
assume *z* ∈ *U*
then obtain *x1 x2* **where** *x1* ∈ *S x2* ∈ *T* **and** *zeq*: *z* = (*x1*,*x2*)
using *Ueq* **by** *blast*
obtain *r1* **where** *r1*>*0* **and** *r1*: *M1.mball x1 r1* ⊆ *S*
by (*meson M1.openin_mtopology* ‹*openin M1.mtopology S*› ‹*x1* ∈ *S*›)

```
    obtain r2 where r2>0 and r2: M2.mball x2 r2 ⊆ T
      by (meson M2.openin_mtopology ‹openin M2.mtopology T› ‹x2 ∈ T›)
    have Prod_metric.mball (x1,x2) (min r1 r2) ⊆ U
    proof (rule order_trans [OF mball_prod_metric_subset])
      show M1.mball x1 (min r1 r2) × M2.mball x2 (min r1 r2) ⊆ U
        using Ueq r1 r2 by force
    qed
    then show ∃r>0. Prod_metric.mball z r ⊆ U
      by (smt (verit, del_insts) zeq ‹0 < r1› ‹0 < r2›)
  qed
next
  fix U z
  assume openin Prod_metric.mtopology U and z ∈ U
  then have U ⊆ M1 × M2
    by (simp add: Prod_metric.openin_mtopology)
  then obtain x y where x ∈ M1 y ∈ M2 and zeq: z = (x,y)
    using ‹z ∈ U› by blast
  obtain r where r>0 and r: Prod_metric.mball (x,y) r ⊆ U
    by (metis Prod_metric.openin_mtopology ‹openin Prod_metric.mtopology U›
‹z ∈ U› zeq)
  define B1 where B1 ≡ M1.mball x (r/2)
  define B2 where B2 ≡ M2.mball y (r/2)
  have openin M1.mtopology B1 openin M2.mtopology B2
    by (simp_all add: B1_def B2_def)
  moreover have (x,y) ∈ B1 × B2
    using ‹r > 0› by (simp add: ‹x ∈ M1› ‹y ∈ M2› B1_def B2_def)
  moreover have B1 × B2 ⊆ U
    using r prod_metric_le_components by (force simp add: B1_def B2_def)
  ultimately show ∃B. B ∈ {S × T |S T. openin M1.mtopology S ∧ openin
M2.mtopology T} ∧ z ∈ B ∧ B ⊆ U
    by (auto simp: zeq)
qed

lemma MCauchy_prod_metric:
  Prod_metric.MCauchy σ ⟷ M1.MCauchy (fst ∘ σ) ∧ M2.MCauchy (snd ∘ σ)
  (is ?lhs ⟷ ?rhs)
proof safe
  assume L: ?lhs
  then have range σ ⊆ M1 × M2
    using Prod_metric.MCauchy_def by blast
  then have 1: range (fst ∘ σ) ⊆ M1 and 2: range (snd ∘ σ) ⊆ M2
    by auto
  have N1: ∃N. ∀n≥N. ∀n'≥N. d1 (fst (σ n)) (fst (σ n')) < ε
    and N2: ∃N. ∀n≥N. ∀n'≥N. d2 (snd (σ n)) (snd (σ n')) < ε if ε>0 for ε
:: real
    using that L unfolding Prod_metric.MCauchy_def
  by (smt (verit, del_insts) add.commute add_less_imp_less_left add_right_mono

      component_le_prod_metric prod.collapse)+
```

show *M1.MCauchy* (*fst* ∘ *σ*)
using *1 N1 M1.MCauchy_def* **by** *auto*
have ∃ *N*. ∀ *n*≥*N*. ∀ *n′*≥*N*. *d2* (*snd* (*σ n*)) (*snd* (*σ n′*)) < *ε* **if** *ε*>*0* **for** *ε* :: *real*
using *that L* **unfolding** *Prod_metric.MCauchy_def*
by (*smt* (*verit*, *del_insts*) *add.commute add_less_imp_less_left add_right_mono*

component_le_prod_metric prod.collapse)
show *M2.MCauchy* (*snd* ∘ *σ*)
using *2 N2 M2.MCauchy_def* **by** *auto*
next
assume *M1*: *M1.MCauchy* (*fst* ∘ *σ*) **and** *M2*: *M2.MCauchy* (*snd* ∘ *σ*)
then have *subM12*: *range* (*fst* ∘ *σ*) ⊆ *M1 range* (*snd* ∘ *σ*) ⊆ *M2*
using *M1.MCauchy_def M2.MCauchy_def* **by** *blast*+
show *?lhs*
unfolding *Prod_metric.MCauchy_def*
proof (*intro conjI strip*)
show *range σ* ⊆ *M1* × *M2*
using *subM12* **by** (*smt* (*verit*, *best*) *SigmaI image_subset_iff o_apply prod.collapse*)

fix *ε* :: *real*
assume *ε* > *0*
obtain *N1* **where** *N1*: ⋀*n n′*. *N1* ≤ *n* ⟹ *N1* ≤ *n′* ⟹ *d1* ((*fst* ∘ *σ*) *n*) ((*fst* ∘ *σ*) *n′*) < *ε*/*2*
by (*meson M1.MCauchy_def* ‹*0* < *ε*› *M1 zero_less_divide_iff zero_less_numeral*)
obtain *N2* **where** *N2*: ⋀*n n′*. *N2* ≤ *n* ⟹ *N2* ≤ *n′* ⟹ *d2* ((*snd* ∘ *σ*) *n*) ((*snd* ∘ *σ*) *n′*) < *ε*/*2*
by (*meson M2.MCauchy_def* ‹*0* < *ε*› *M2 zero_less_divide_iff zero_less_numeral*)
have *prod_dist d1 d2* (*σ n*) (*σ n′*) < *ε*
if *N1* ≤ *n* **and** *N2* ≤ *n* **and** *N1* ≤ *n′* **and** *N2* ≤ *n′* **for** *n n′*
proof −
obtain *a b a′ b′* **where** *σ*: *σ n* = (*a*,*b*) *σ n′* = (*a′*,*b′*)
by *fastforce*+
have *prod_dist d1 d2* (*a*,*b*) (*a′*,*b′*) ≤ *d1 a a′* + *d2 b b′*
by (*metis* ‹*range σ* ⊆ *M1* × *M2*› *σ mem_Sigma_iff prod_metric_le_components range_subsetD*)
also have ... < *ε*/*2* + *ε*/*2*
using *N1 N2 σ that* **by** *fastforce*
finally show *?thesis*
by (*simp add*: *σ*)
qed
then show ∃ *N*. ∀ *n n′*. *N* ≤ *n* ⟶ *N* ≤ *n′* ⟶ *prod_dist d1 d2* (*σ n*) (*σ n′*) < *ε*
by (*metis order.trans linorder_le_cases*)
qed
qed

lemma *mcomplete_prod_metric*:
Prod_metric.mcomplete ⟷ *M1* = {} ∨ *M2* = {} ∨ *M1.mcomplete* ∧ *M2.mcomplete*

```
  (is ?lhs ⟷ ?rhs)
proof (cases M1 = {} ∨ M2 = {})
  case False
  then obtain x y where x ∈ M1 y ∈ M2
    by blast
  have M1.mcomplete ∧ M2.mcomplete ⟹ Prod_metric.mcomplete
   by (simp add: Prod_metric.mcomplete_def M1.mcomplete_def M2.mcomplete_def

       mtopology_prod_metric MCauchy_prod_metric limitin_pairwise)
  moreover
  { assume L: Prod_metric.mcomplete
    have M1.mcomplete
      unfolding M1.mcomplete_def
    proof (intro strip)
      fix σ
      assume M1.MCauchy σ
      then have Prod_metric.MCauchy (λn. (σ n, y))
          using ‹y ∈ M2› by (simp add: M1.MCauchy_def M2.MCauchy_def
MCauchy_prod_metric)
       then obtain z where limitin Prod_metric.mtopology (λn. (σ n, y)) z se-
quentially
        using L Prod_metric.mcomplete_def by blast
      then show ∃x. limitin M1.mtopology σ x sequentially
        by (auto simp: Prod_metric.mcomplete_def M1.mcomplete_def
            mtopology_prod_metric limitin_pairwise o_def)
    qed
  }
  moreover
  { assume L: Prod_metric.mcomplete
    have M2.mcomplete
      unfolding M2.mcomplete_def
    proof (intro strip)
      fix σ
      assume M2.MCauchy σ
      then have Prod_metric.MCauchy (λn. (x, σ n))
          using ‹x ∈ M1› by (simp add: M2.MCauchy_def M1.MCauchy_def
MCauchy_prod_metric)
       then obtain z where limitin Prod_metric.mtopology (λn. (x, σ n)) z se-
quentially
        using L Prod_metric.mcomplete_def by blast
      then show ∃x. limitin M2.mtopology σ x sequentially
        by (auto simp: Prod_metric.mcomplete_def M2.mcomplete_def
            mtopology_prod_metric limitin_pairwise o_def)
    qed
  }
  ultimately show ?thesis
    using False by blast
qed auto
```

lemma *mbounded_prod_metric*:
Prod_metric.mbounded U $\longleftrightarrow$ *M1.mbounded* (*fst* ‘ *U*) ∧ *M2.mbounded* (*snd* ‘ *U*)
proof −
have (∃ *B*. *U* ⊆ *Prod_metric.mcball* (*x*,*y*) *B*)
$\longleftrightarrow$ ((∃ *B*. (*fst* ‘ *U*) ⊆ *M1.mcball x B*) ∧ (∃ *B*. (*snd* ‘ *U*) ⊆ *M2.mcball y B*))
(**is** *?lhs* $\longleftrightarrow$ *?rhs*)
for *x y*
proof *safe*
fix *B*
assume *U* ⊆ *Prod_metric.mcball* (*x*, *y*) *B*
then have (*fst* ‘ *U*) ⊆ *M1.mcball x B* (*snd* ‘ *U*) ⊆ *M2.mcball y B*
using *mcball_prod_metric_subset* **by** *fastforce*+
then show ∃ *B*. (*fst* ‘ *U*) ⊆ *M1.mcball x B* ∃ *B*. (*snd* ‘ *U*) ⊆ *M2.mcball y B*
by *auto*
next
fix *B1 B2*
assume (*fst* ‘ *U*) ⊆ *M1.mcball x B1* (*snd* ‘ *U*) ⊆ *M2.mcball y B2*
then have *fst* ‘ *U* × *snd* ‘ *U* ⊆ *M1.mcball x B1* × *M2.mcball y B2*
by *blast*
also have ... ⊆ *Prod_metric.mcball* (*x*, *y*) (*B1*+*B2*)
by (*intro mcball_subset_prod_metric*)
finally show ∃ *B*. *U* ⊆ *Prod_metric.mcball* (*x*, *y*) *B*
by (*metis subsetD subsetI subset_fst_snd*)
qed
then show *?thesis*
by (*simp add*: *M1.mbounded_def M2.mbounded_def Prod_metric.mbounded_def*)
qed

lemma *mbounded_Times*:
Prod_metric.mbounded (*S* × *T*) $\longleftrightarrow$ *S* = {} ∨ *T* = {} ∨ *M1.mbounded S* ∧ *M2.mbounded T*
by (*auto simp*: *mbounded_prod_metric*)

lemma *mtotally_bounded_Times*:
Prod_metric.mtotally_bounded (*S* × *T*) $\longleftrightarrow$
S = {} ∨ *T* = {} ∨ *M1.mtotally_bounded S* ∧ *M2.mtotally_bounded T*
(**is** *?lhs* $\longleftrightarrow$ _)
proof (*cases S* = {} ∨ *T* = {})
case *False*
then obtain *x y* **where** *x* ∈ *S y* ∈ *T*
by *auto*
have *M1.mtotally_bounded S* **if** *L*: *?lhs*
unfolding *M1.mtotally_bounded_sequentially*
proof (*intro conjI strip*)
show *S* ⊆ *M1*
using *Prod_metric.mtotally_bounded_imp_subset* ‹*y* ∈ *T*› *that* **by** *blast*
fix *σ* :: *nat* ⇒ *'a*

```
    assume range σ ⊆ S
    with L obtain r where strict_mono r Prod_metric.MCauchy ((λn. (σ n,y))
∘ r)
      unfolding Prod_metric.mtotally_bounded_sequentially
      by (smt (verit) SigmaI ‹y ∈ T› image_subset_iff)
    then have M1.MCauchy (fst ∘ (λn. (σ n,y)) ∘ r)
      by (simp add: MCauchy_prod_metric o_def)
    with ‹strict_mono r› show ∃r. strict_mono r ∧ M1.MCauchy (σ ∘ r)
      by (auto simp add: o_def)
  qed
  moreover
  have M2.mtotally_bounded T if L: ?lhs
    unfolding M2.mtotally_bounded_sequentially
  proof (intro conjI strip)
    show T ⊆ M2
      using Prod_metric.mtotally_bounded_imp_subset ‹x ∈ S› that by blast
    fix σ :: nat ⇒ 'b
    assume range σ ⊆ T
    with L obtain r where strict_mono r Prod_metric.MCauchy ((λn. (x,σ n))
∘ r)
      unfolding Prod_metric.mtotally_bounded_sequentially
      by (smt (verit) SigmaI ‹x ∈ S› image_subset_iff)
    then have M2.MCauchy (snd ∘ (λn. (x,σ n)) ∘ r)
      by (simp add: MCauchy_prod_metric o_def)
    with ‹strict_mono r› show ∃r. strict_mono r ∧ M2.MCauchy (σ ∘ r)
      by (auto simp add: o_def)
  qed
  moreover have ?lhs if 1: M1.mtotally_bounded S and 2: M2.mtotally_bounded
T
    unfolding Prod_metric.mtotally_bounded_sequentially
  proof (intro conjI strip)
    show S × T ⊆ M1 × M2
      using that
    by (auto simp: M1.mtotally_bounded_sequentially M2.mtotally_bounded_sequentially)
    fix σ :: nat ⇒ 'a × 'b
    assume σ: range σ ⊆ S × T
    with 1 obtain r1 where r1: strict_mono r1 M1.MCauchy (fst ∘ σ ∘ r1)
      apply (clarsimp simp: M1.mtotally_bounded_sequentially image_subset_iff)
      by (metis SigmaE comp_eq_dest_lhs fst_conv)
    from σ 2 obtain r2 where r2: strict_mono r2 M2.MCauchy (snd ∘ σ ∘ r1 ∘
r2)
      apply (clarsimp simp: M2.mtotally_bounded_sequentially image_subset_iff)
      by (smt (verit, best) comp_apply mem_Sigma_iff prod.collapse)
    then have M1.MCauchy (fst ∘ σ ∘ r1 ∘ r2)
      by (simp add: M1.MCauchy_subsequence r1)
    with r2 have Prod_metric.MCauchy (σ ∘ (r1 ∘ r2))
      by (simp add: MCauchy_prod_metric o_def)
    then show ∃r. strict_mono r ∧ Prod_metric.MCauchy (σ ∘ r)
      using r1 r2 strict_mono_o by blast
```

```
  qed
  ultimately show ?thesis
    using False by blast
qed auto

lemma mtotally_bounded_prod_metric:
   Prod_metric.mtotally_bounded U ⟷
    M1.mtotally_bounded (fst ‘ U) ∧ M2.mtotally_bounded (snd ‘ U) (is ?lhs ⟷
?rhs)
proof
  assume L: ?lhs
  then have U ⊆ M1 × M2
   and ∗: ⋀σ. range σ ⊆ U ⟹ ∃ r::nat⇒nat. strict_mono r ∧ Prod_metric.MCauchy
(σ∘r)
    by (simp_all add: Prod_metric.mtotally_bounded_sequentially)
  show ?rhs
   unfolding M1.mtotally_bounded_sequentially M2.mtotally_bounded_sequentially
  proof (intro conjI strip)
    show fst ‘ U ⊆ M1 snd ‘ U ⊆ M2
      using ‹U ⊆ M1 × M2› by auto
  next
    fix σ :: nat ⇒ 'a
    assume range σ ⊆ fst ‘ U
    then obtain ζ where ζ: ⋀n. σ n = fst (ζ n) ∧ ζ n ∈ U
      unfolding image_subset_iff image_iff by (meson UNIV_I)
    then obtain r where strict_mono r ∧ Prod_metric.MCauchy (ζ∘r)
      by (metis ∗ image_subset_iff)
    with ζ show ∃ r. strict_mono r ∧ M1.MCauchy (σ ∘ r)
      by (auto simp: MCauchy_prod_metric o_def)
  next
    fix σ:: nat ⇒ 'b
    assume range σ ⊆ snd ‘ U
    then obtain ζ where ζ: ⋀n. σ n = snd (ζ n) ∧ ζ n ∈ U
      unfolding image_subset_iff image_iff by (meson UNIV_I)
    then obtain r where strict_mono r ∧ Prod_metric.MCauchy (ζ∘r)
      by (metis ∗ image_subset_iff)
    with ζ show ∃ r. strict_mono r ∧ M2.MCauchy (σ ∘ r)
      by (auto simp: MCauchy_prod_metric o_def)
  qed
next
  assume ?rhs
  then have Prod_metric.mtotally_bounded ((fst ‘ U) × (snd ‘ U))
    by (simp add: mtotally_bounded_Times)
  then show ?lhs
    by (metis Prod_metric.mtotally_bounded_subset subset_fst_snd)
qed

end
```

lemma *metrizable_space_prod_topology*:
metrizable_space (*prod_topology X Y*) ⟷
(*prod_topology X Y*) = *trivial_topology* ∨ *metrizable_space X* ∧ *metrizable_space Y*
(**is** *?lhs* ⟷ *?rhs*)
proof (*cases* (*prod_topology X Y*) = *trivial_topology*)
case *False*
then obtain *x y* **where** *x* ∈ *topspace X y* ∈ *topspace Y*
by *fastforce*
show *?thesis*
proof
show *?rhs* ⟹ *?lhs*
unfolding *metrizable_space_def*
using *Metric_space12.mtopology_prod_metric*
by (*metis False Metric_space12.prod_metric Metric_space12_def*)
next
assume *L*: *?lhs*
have *metrizable_space* (*subtopology* (*prod_topology X Y*) (*topspace X* × {*y*}))
metrizable_space (*subtopology* (*prod_topology X Y*) ({*x*} × *topspace Y*))
using *L metrizable_space_subtopology* **by** *auto*
moreover
have (*subtopology* (*prod_topology X Y*) (*topspace X* × {*y*})) *homeomorphic_space X*
by (*metis* ‹*y* ∈ *topspace Y*› *homeomorphic_space_prod_topology_sing1 homeomorphic_space_sym prod_topology_subtopology*(*2*))
moreover
have (*subtopology* (*prod_topology X Y*) ({*x*} × *topspace Y*)) *homeomorphic_space Y*
by (*metis* ‹*x* ∈ *topspace X*› *homeomorphic_space_prod_topology_sing2 homeomorphic_space_sym prod_topology_subtopology*(*1*))
ultimately show *?rhs*
by (*simp add*: *homeomorphic_metrizable_space*)
qed
qed *auto*

lemma *completely_metrizable_space_prod_topology*:
completely_metrizable_space (*prod_topology X Y*) ⟷
(*prod_topology X Y*) = *trivial_topology* ∨
completely_metrizable_space X ∧ *completely_metrizable_space Y*
(**is** *?lhs* ⟷ *?rhs*)
proof (*cases* (*prod_topology X Y*) = *trivial_topology*)
case *False*
then obtain *x y* **where** *x* ∈ *topspace X y* ∈ *topspace Y*
by *fastforce*
show *?thesis*
proof
show *?rhs* ⟹ *?lhs*

unfolding *completely_metrizable_space_def*
by (*metis False Metric_space12.mtopology_prod_metric Metric_space12.mcomplete_prod_metric Metric_space12.prod_metric Metric_space12_def*)
next
assume *L*: *?lhs*
then have *Hausdorff_space* (*prod_topology X Y*)
by (*simp add*: *completely_metrizable_imp_metrizable_space metrizable_imp_Hausdorff_space*)
then have *H*: *Hausdorff_space X* ∧ *Hausdorff_space Y*
using *False Hausdorff_space_prod_topology* **by** *blast*
then have *closedin* (*prod_topology X Y*) (*topspace X* × {*y*}) ∧ *closedin* (*prod_topology X Y*) ({*x*} × *topspace Y*)
using ‹*x* ∈ *topspace X*› ‹*y* ∈ *topspace Y*›
by (*auto simp*: *closedin_Hausdorff_sing_eq closedin_prod_Times_iff*)
with *L* **have** *completely_metrizable_space*(*subtopology* (*prod_topology X Y*) (*topspace X* × {*y*}))
∧ *completely_metrizable_space*(*subtopology* (*prod_topology X Y*) ({*x*} × *topspace Y*))
by (*simp add*: *completely_metrizable_space_closedin*)
moreover
have (*subtopology* (*prod_topology X Y*) (*topspace X* × {*y*})) *homeomorphic_space X*
by (*metis* ‹*y* ∈ *topspace Y*› *homeomorphic_space_prod_topology_sing1 homeomorphic_space_sym prod_topology_subtopology*(*2*))
moreover
have (*subtopology* (*prod_topology X Y*) ({*x*} × *topspace Y*)) *homeomorphic_space Y*
by (*metis* ‹*x* ∈ *topspace X*› *homeomorphic_space_prod_topology_sing2 homeomorphic_space_sym prod_topology_subtopology*(*1*))
ultimately show *?rhs*
by (*simp add*: *homeomorphic_completely_metrizable_space*)
qed
next
case *True* **then show** *?thesis*
using *empty_completely_metrizable_space* **by** *auto*
qed

5.7.16 The "atin-within" filter for topologies

definition *atin_within* :: ['*a topology*, '*a*, '*a set*] ⇒ '*a filter*
where *atin_within X a S* = *inf* (*nhdsin X a*) (*principal* (*topspace X* ∩ *S* − {*a*}))

lemma *atin_within_UNIV* [*simp*]:
shows *atin_within X a UNIV* = *atin X a*
by (*simp add*: *atin_def atin_within_def*)

lemma *eventually_atin_subtopology*:
assumes *a* ∈ *topspace X*
shows *eventually P* (*atin* (*subtopology X S*) *a*) ⟷
(*a* ∈ *S* ⟶ (∃ *U*. *openin* (*subtopology X S*) *U* ∧ *a* ∈ *U* ∧ (∀ *x*∈*U* − {*a*}. *P x*)))

using *assms* **by** (*simp add*: *eventually_atin*)

lemma *eventually_atin_within*:
eventually P (*atin_within X a S*) ⟷ *a* ∉ *topspace X* ∨ (∃ *T*. *openin X T* ∧ *a* ∈ *T* ∧ (∀ *x*∈*T*. *x* ∈ *S* ∧ *x* ≠ *a* ⟶ *P x*))
proof (*cases a* ∈ *topspace X*)
case *True*
hence *eventually P* (*atin_within X a S*) ⟷
(∃ *T*. *openin X T* ∧ *a* ∈ *T* ∧
(∀ *x*∈*T*. *x* ∈ *topspace X* ∧ *x* ∈ *S* ∧ *x* ≠ *a* ⟶ *P x*))
by (*simp add*: *atin_within_def eventually_inf_principal eventually_nhdsin*)
also have ... ⟷ (∃ *T*. *openin X T* ∧ *a* ∈ *T* ∧ (∀ *x*∈*T*. *x* ∈ *S* ∧ *x* ≠ *a* ⟶ *P x*))
using *openin_subset* **by** (*intro ex_cong*) *auto*
finally show *?thesis* **by** (*simp add*: *True*)
qed (*simp add*: *atin_within_def*)

lemma *eventually_within_imp*:
eventually P (*atin_within X a S*) ⟷ *eventually* (λ*x*. *x* ∈ *S* ⟶ *P x*) (*atin X a*)
by (*auto simp add*: *eventually_atin_within eventually_atin*)

lemma *limit_within_subset*:
⟦*limitin X f l* (*atin_within Y a S*); *T* ⊆ *S*⟧ ⟹ *limitin X f l* (*atin_within Y a T*)
by (*smt* (*verit*) *eventually_atin_within limitin_def subset_eq*)

lemma *atin_subtopology_within*:
assumes *a* ∈ *S*
shows *atin* (*subtopology X S*) *a* = *atin_within X a S*
proof −
have *eventually P* (*atin* (*subtopology X S*) *a*) ⟷ *eventually P* (*atin_within X a S*) **for** *P*
unfolding *eventually_atin eventually_atin_within openin_subtopology*
using *assms* **by** *auto*
then show *?thesis*
by (*meson le_filter_def order.eq_iff*)
qed

lemma *limit_continuous_map_within*:
⟦*continuous_map* (*subtopology X S*) *Y f*; *a* ∈ *S*; *a* ∈ *topspace X*⟧
⟹ *limitin Y f* (*f a*) (*atin_within X a S*)
by (*metis Int_iff atin_subtopology_within continuous_map_atin topspace_subtopology*)

lemma *atin_subtopology_within_if*:
shows *atin* (*subtopology X S*) *a* = (*if a* ∈ *S then atin_within X a S else bot*)
by (*simp add*: *atin_subtopology_within*)

lemma *trivial_limit_atpointof_within*:

trivial_limit(*atin_within X a S*) ⟷
(*a* ∉ *X derived_set_of S*)
by (*auto simp*: *trivial_limit_def eventually_atin_within in_derived_set_of*)

lemma *derived_set_of_trivial_limit*:
a ∈ *X derived_set_of S* ⟷ ¬ *trivial_limit*(*atin_within X a S*)
by (*simp add*: *trivial_limit_atpointof_within*)

5.7.17 More sequential characterizations in a metric space

context *Metric_space*
begin

definition *decreasing_dist* :: (*nat* ⇒ *'a*) ⇒ *'a* ⇒ *bool*
where *decreasing_dist σ x* ≡ (∀ *m n*. *m* < *n* ⟶ *d* (*σ n*) *x* < *d* (*σ m*) *x*)

lemma *decreasing_dist_imp_inj*: *decreasing_dist σ a* ⟹ *inj σ*
by (*metis decreasing_dist_def dual_order.irrefl linorder_inj_onI'*)

lemma *eventually_atin_within_metric*:
eventually P (*atin_within mtopology a S*) ⟷
(*a* ∈ *M* ⟶ (∃ *δ*>*0*. ∀ *x*. *x* ∈ *M* ∧ *x* ∈ *S* ∧ *0* < *d x a* ∧ *d x a* < *δ* ⟶ *P x*))
(**is** *?lhs*=*?rhs*)
proof
assume *?lhs* **then show** *?rhs*
unfolding *eventually_atin_within openin_mtopology subset_iff*
by (*metis commute in_mball mdist_zero order_less_irrefl topspace_mtopology*)
next
assume *R*: *?rhs*
show *?lhs*
proof (*cases a* ∈ *M*)
case *True*
then obtain *δ* **where** *δ* > *0* **and** *δ*: ⋀*x*. ⟦*x* ∈ *M*; *x* ∈ *S*; *0* < *d x a*; *d x a* < *δ*⟧ ⟹ *P x*
using *R* **by** *blast*
then have *openin mtopology* (*mball a δ*) ∧ (∀ *x* ∈ *mball a δ*. *x* ∈ *S* ∧ *x* ≠ *a* ⟶ *P x*)
by (*simp add*: *commute openin_mball*)
then show *?thesis*
by (*metis True* ‹*0* < *δ*› *centre_in_mball_iff eventually_atin_within*)
next
case *False*
with *R* **show** *?thesis*
by (*simp add*: *eventually_atin_within*)
qed
qed

lemma *eventually_atin_within_A*:

```
  assumes
    (⋀σ. ⟦range σ ⊆ (S ∩ M) − {a}; decreasing_dist σ a;
          inj σ; limitin mtopology σ a sequentially⟧
      ⟹ eventually (λn. P (σ n)) sequentially)
  shows eventually P (atin_within mtopology a S)
proof −
  have False if SP: ⋀δ. δ>0 ⟹ ∃x ∈ M−{a}. d x a < δ ∧ x ∈ S ∧ ¬ P x and
a ∈ M
  proof −
    define Φ where Φ ≡ λn x. x ∈ M−{a} ∧ d x a < inverse (Suc n) ∧ x ∈ S ∧
¬ P x
    obtain σ where σ: ⋀n. Φ n (σ n) and dless: ⋀n. d (σ(Suc n)) a < d (σ n) a
    proof −
      obtain x0 where x0: Φ 0 x0
        using SP [OF zero_less_one] by (force simp: Φ_def)
      have ∃y. Φ (Suc n) y ∧ d y a < d x a if Φ n x for n x
        using SP [of min (inverse (Suc (Suc n))) (d x a)] ‹a ∈ M› that
        by (auto simp: Φ_def)
      then obtain f where f: ⋀n x. Φ n x ⟹ Φ (Suc n) (f n x) ∧ d (f n x) a <
d x a
        by metis
      show thesis
        proof
          show Φ n (rec_nat x0 f n) for n
            by (induction n) (auto simp: x0 dest: f)
          with f show d (rec_nat x0 f (Suc n)) a < d (rec_nat x0 f n) a for n
            by auto
        qed
    qed
    have 1: range σ ⊆ (S ∩ M) − {a}
      using σ by (auto simp: Φ_def)
    have d (σ(Suc (m+n))) a < d (σ n) a for m n
      by (induction m) (auto intro: order_less_trans dless)
    then have 2: decreasing_dist σ a
      unfolding decreasing_dist_def by (metis add.commute less_imp_Suc_add)
    have ∀F xa in sequentially. d (σ xa) a < ε if ε > 0 for ε
    proof −
      obtain N where inverse (Suc N) < ε
        using ‹ε > 0› reals_Archimedean by blast
      with σ 2 show ?thesis
      unfolding decreasing_dist_def by (smt (verit, best) Φ_def eventually_at_top_dense)
    qed
    then have 4: limitin mtopology σ a sequentially
      using σ ‹a ∈ M› by (simp add: Φ_def limitin_metric)
    show False
     using 2 assms [OF 1 _ decreasing_dist_imp_inj 4] σ by (force simp: Φ_def)
  qed
  then show ?thesis
    by (fastforce simp: eventually_atin_within_metric)
```

qed

lemma *eventually_atin_within_B*:
assumes *ev*: *eventually P (atin_within mtopology a S)*
and *ran*: *range σ ⊆ (S ∩ M) − {a}*
and *lim*: *limitin mtopology σ a sequentially*
shows *eventually (λn. P (σ n)) sequentially*
proof −
have *a ∈ M*
using *lim limitin_mspace* **by** *auto*
with *ev* **obtain** *δ* **where** *0 < δ*
and *δ*: *⋀σ. ⟦σ ∈ M; σ ∈ S; 0 < d σ a; d σ a < δ⟧ ⟹ P σ*
by (*auto simp*: *eventually_atin_within_metric*)
then have ∗: *⋀n. σ n ∈ M ∧ d (σ n) a < δ ⟹ P (σ n)*
using ‹*a ∈ M*› *ran* **by** *auto*
have *∀ F n in sequentially. σ n ∈ M ∧ d (σ n) a < δ*
using *lim* ‹*0 < δ*› **by** (*auto simp*: *limitin_metric*)
then show *?thesis*
by (*simp add*: ∗ *eventually_mono*)
qed

lemma *eventually_atin_within_sequentially*:
eventually P (atin_within mtopology a S) ⟷
(∀ σ. range σ ⊆ (S ∩ M) − {a} ∧
limitin mtopology σ a sequentially
⟶ eventually (λn. P(σ n)) sequentially)
by (*metis eventually_atin_within_A eventually_atin_within_B*)

lemma *eventually_atin_within_sequentially_inj*:
eventually P (atin_within mtopology a S) ⟷
(∀ σ. range σ ⊆ (S ∩ M) − {a} ∧ inj σ ∧
limitin mtopology σ a sequentially
⟶ eventually (λn. P(σ n)) sequentially)
by (*metis eventually_atin_within_A eventually_atin_within_B*)

lemma *eventually_atin_within_sequentially_decreasing*:
eventually P (atin_within mtopology a S) ⟷
(∀ σ. range σ ⊆ (S ∩ M) − {a} ∧ decreasing_dist σ a ∧
limitin mtopology σ a sequentially
⟶ eventually (λn. P(σ n)) sequentially)
by (*metis eventually_atin_within_A eventually_atin_within_B*)

lemma *eventually_atin_sequentially*:
eventually P (atin mtopology a) ⟷
(∀ σ. range σ ⊆ M − {a} ∧ limitin mtopology σ a sequentially
⟶ eventually (λn. P(σ n)) sequentially)
using *eventually_atin_within_sequentially* [**where** *S=UNIV*] **by** *simp*

lemma *eventually_atin_sequentially_inj*:
eventually P (atin mtopology a) ⟷
(∀σ. range σ ⊆ M − {a} ∧ inj σ ∧
limitin mtopology σ a sequentially
⟶ eventually (λn. P(σ n)) sequentially)
using *eventually_atin_within_sequentially_inj* [**where** *S=UNIV*] **by** *simp*

lemma *eventually_atin_sequentially_decreasing*:
eventually P (atin mtopology a) ⟷
(∀σ. range σ ⊆ M − {a} ∧ decreasing_dist σ a ∧
limitin mtopology σ a sequentially
⟶ eventually (λn. P(σ n)) sequentially)
using *eventually_atin_within_sequentially_decreasing* [**where** *S=UNIV*] **by** *simp*

end

context *Metric_space12*
begin

lemma *limit_atin_sequentially_within*:
limitin M2.mtopology f l (atin_within M1.mtopology a S) ⟷
l ∈ M2 ∧
(∀σ. range σ ⊆ S ∩ M1 − {a} ∧
limitin M1.mtopology σ a sequentially
⟶ limitin M2.mtopology (f ∘ σ) l sequentially)
by (*auto simp*: *M1.eventually_atin_within_sequentially limitin_def*)

lemma *limit_atin_sequentially_within_inj*:
limitin M2.mtopology f l (atin_within M1.mtopology a S) ⟷
l ∈ M2 ∧
(∀σ. range σ ⊆ S ∩ M1 − {a} ∧ inj σ ∧
limitin M1.mtopology σ a sequentially
⟶ limitin M2.mtopology (f ∘ σ) l sequentially)
by (*auto simp*: *M1.eventually_atin_within_sequentially_inj limitin_def*)

lemma *limit_atin_sequentially_within_decreasing*:
limitin M2.mtopology f l (atin_within M1.mtopology a S) ⟷
l ∈ M2 ∧
(∀σ. range σ ⊆ S ∩ M1 − {a} ∧ M1.decreasing_dist σ a ∧
limitin M1.mtopology σ a sequentially
⟶ limitin M2.mtopology (f ∘ σ) l sequentially)
by (*auto simp*: *M1.eventually_atin_within_sequentially_decreasing limitin_def*)

lemma *limit_atin_sequentially*:
limitin M2.mtopology f l (atin M1.mtopology a) ⟷
l ∈ M2 ∧
(∀σ. range σ ⊆ M1 − {a} ∧

limitin M1.mtopology σ a sequentially
⟶ limitin M2.mtopology (f ∘ σ) l sequentially)
using *limit_atin_sequentially_within* [**where** *S=UNIV*] **by** *simp*

lemma *limit_atin_sequentially_inj*:
limitin M2.mtopology f l (atin M1.mtopology a) ⟷
l ∈ M2 ∧
(∀σ. range σ ⊆ M1 − {a} ∧ inj σ ∧
limitin M1.mtopology σ a sequentially
⟶ limitin M2.mtopology (f ∘ σ) l sequentially)
using *limit_atin_sequentially_within_inj* [**where** *S=UNIV*] **by** *simp*

lemma *limit_atin_sequentially_decreasing*:
limitin M2.mtopology f l (atin M1.mtopology a) ⟷
l ∈ M2 ∧
(∀σ. range σ ⊆ M1 − {a} ∧ M1.decreasing_dist σ a ∧
limitin M1.mtopology σ a sequentially
⟶ limitin M2.mtopology (f ∘ σ) l sequentially)
using *limit_atin_sequentially_within_decreasing* [**where** *S=UNIV*] **by** *simp*

end

An experiment: same result as within the locale, but using metric space variables

lemma *limit_atin_sequentially_within*:
limitin (mtopology_of m2) f l (atin_within (mtopology_of m1) a S) ⟷
l ∈ mspace m2 ∧
(∀σ. range σ ⊆ S ∩ mspace m1 − {a} ∧
limitin (mtopology_of m1) σ a sequentially
⟶ limitin (mtopology_of m2) (f ∘ σ) l sequentially)
using *Metric_space12.limit_atin_sequentially_within* [*OF Metric_space12_mspace_mdist*]
by (*metis mtopology_of_def*)

context *Metric_space*
begin

lemma *atin_within_imp_M*:
atin_within mtopology x S ≠ bot ⟹ x ∈ M
by (*metis derived_set_of_trivial_limit in_derived_set_of topspace_mtopology*)

lemma *atin_within_sequentially_sequence*:
assumes *atin_within mtopology x S ≠ bot*
obtains *σ* **where** *range σ ⊆ S ∩ M − {x}*
decreasing_dist σ x inj σ limitin mtopology σ x sequentially
by (*metis eventually_atin_within_A eventually_False assms*)

lemma *derived_set_of_sequentially*:
mtopology derived_set_of S =

$\{x \in M.\ \exists \sigma.\ range\ \sigma \subseteq S \cap M - \{x\} \wedge limitin\ mtopology\ \sigma\ x\ sequentially\}$
proof −
have *False*
if *range* $\sigma \subseteq S \cap M - \{x\}$
and *limitin mtopology* σ *x sequentially*
and *atin_within mtopology x S* = *bot*
for $x\ \sigma$
proof −
have $\forall_F\ n$ *in sequentially*. $P\ (\sigma\ n)$ **for** P
using *that* **by** (*metis eventually_atin_within_B eventually_bot*)
then show *False*
by (*meson eventually_False_sequentially eventually_mono*)
qed
then show *?thesis*
using *derived_set_of_trivial_limit*
by (*fastforce elim*!: *atin_within_sequentially_sequence intro*: *atin_within_imp_M*)
qed

lemma *derived_set_of_sequentially_alt*:
mtopology derived_set_of S =
$\{x.\ \exists \sigma.\ range\ \sigma \subseteq S - \{x\} \wedge limitin\ mtopology\ \sigma\ x\ sequentially\}$
proof −
have $*$: $\exists \sigma.\ range\ \sigma \subseteq S \cap M - \{x\} \wedge limitin\ mtopology\ \sigma\ x\ sequentially$
if σ: *range* $\sigma \subseteq S - \{x\}$ **and** *lim*: *limitin mtopology* σ *x sequentially* **for** $x\ \sigma$
proof −
obtain N **where** $\forall n \geq N.\ \sigma\ n \in M \wedge d\ (\sigma\ n)\ x < 1$
using *lim limit_metric_sequentially* **by** *fastforce*
with σ **obtain** a **where** a:$a \in S \cap M - \{x\}$ **by** *auto*
show *?thesis*
proof (*intro conjI exI*)
show *range* $(\lambda n.\ if\ \sigma\ n \in M\ then\ \sigma\ n\ else\ a) \subseteq S \cap M - \{x\}$
using $a\ \sigma$ **by** *fastforce*
show *limitin mtopology* $(\lambda n.\ if\ \sigma\ n \in M\ then\ \sigma\ n\ else\ a)\ x\ sequentially$
using *lim limit_metric_sequentially* **by** *fastforce*
qed
qed
show *?thesis*
by (*auto simp*: *limitin_mspace derived_set_of_sequentially intro*!: $*$)
qed

lemma *derived_set_of_sequentially_inj*:
mtopology derived_set_of S =
$\{x \in M.\ \exists \sigma.\ range\ \sigma \subseteq S \cap M - \{x\} \wedge inj\ \sigma \wedge limitin\ mtopology\ \sigma\ x\ sequentially\}$
proof −
have *False*
if $x \in M$ **and** *range* $\sigma \subseteq S \cap M - \{x\}$
and *limitin mtopology* σ *x sequentially*
and *atin_within mtopology x S* = *bot*

for x σ
proof −
have $\forall_F$ n *in sequentially*. P (σ n) **for** P
using *that derived_set_of_sequentially_alt derived_set_of_trivial_limit* **by** *fastforce*
then show *False*
by (*meson eventually_False_sequentially eventually_mono*)
qed
then show *?thesis*
using *derived_set_of_trivial_limit*
by (*fastforce elim*!: *atin_within_sequentially_sequence intro*: *atin_within_imp_M*)
qed

lemma *derived_set_of_sequentially_inj_alt*:
mtopology derived_set_of S =
{x. ∃σ. *range* σ ⊆ S − {x} ∧ *inj* σ ∧ *limitin mtopology* σ x *sequentially*}
proof −
have ∃σ. *range* σ ⊆ S − {x} ∧ *inj* σ ∧ *limitin mtopology* σ x *sequentially*
if *atin_within mtopology x S* ≠ *bot* **for** x
by (*metis Diff_empty Int_subset_iff atin_within_sequentially_sequence subset_Diff_insert that*)
moreover have *False*
if *range* (λx. σ (x::*nat*)) ⊆ S − {x}
and *limitin mtopology* σ x *sequentially*
and *atin_within mtopology x S* = *bot*
for x σ
proof −
have $\forall_F$ n *in sequentially*. P (σ n) **for** P
using *that derived_set_of_sequentially_alt derived_set_of_trivial_limit* **by** *fastforce*
then show *False*
by (*meson eventually_False_sequentially eventually_mono*)
qed
ultimately show *?thesis*
using *derived_set_of_trivial_limit* **by** (*fastforce intro*: *atin_within_imp_M*)
qed

lemma *derived_set_of_sequentially_decreasing*:
mtopology derived_set_of S =
{x ∈ M. ∃σ. *range* σ ⊆ S − {x} ∧ *decreasing_dist* σ x ∧ *limitin mtopology* σ x *sequentially*}
proof −
have ∃σ. *range* σ ⊆ S − {x} ∧ *decreasing_dist* σ x ∧ *limitin mtopology* σ x *sequentially*
if *atin_within mtopology x S* ≠ *bot* **for** x
by (*metis Diff_empty atin_within_sequentially_sequence le_infE subset_Diff_insert that*)
moreover have *False*

if $x \in M$ **and** *range* $\sigma \subseteq S - \{x\}$
and *limitin mtopology* σ x *sequentially*
and *atin_within mtopology* x S $=$ *bot*
for x σ
proof −
have $\forall_F$ n *in sequentially*. P $(\sigma$ $n)$ **for** P
using *that derived_set_of_sequentially_alt derived_set_of_trivial_limit* **by** *fastforce*
then show *False*
by (*meson eventually_False_sequentially eventually_mono*)
qed
ultimately show *?thesis*
using *derived_set_of_trivial_limit* **by** (*fastforce intro*: *atin_within_imp_M*)
qed

lemma *derived_set_of_sequentially_decreasing_alt*:
mtopology derived_set_of S $=$
$\{x.\ \exists \sigma.$ *range* $\sigma \subseteq S - \{x\} \wedge$ *decreasing_dist* σ $x \wedge$ *limitin mtopology* σ x *sequentially*$\}$
using *derived_set_of_sequentially_alt derived_set_of_sequentially_decreasing*
by *auto*

lemma *closure_of_sequentially*:
mtopology closure_of S $=$
$\{x \in M.\ \exists \sigma.$ *range* $\sigma \subseteq S \cap M \wedge$ *limitin mtopology* σ x *sequentially*$\}$
by (*auto simp*: *closure_of derived_set_of_sequentially*)

end

5.7.18 Three strong notions of continuity for metric spaces

Lipschitz continuity

definition *Lipschitz_continuous_map*
where *Lipschitz_continuous_map* $\equiv$
$\lambda m1\ m2\ f.\ f \in$ *mspace m1* $\to$ *mspace m2* $\wedge$
$(\exists B.\ \forall x \in$ *mspace m1*. $\forall y \in$ *mspace m1*. *mdist m2* $(f\ x)$ $(f\ y) \leq B *$ *mdist m1* x $y)$

lemma *Lipschitz_continuous_map_image*:
Lipschitz_continuous_map m1 m2 $f \Longrightarrow f \in$ *mspace m1* $\to$ *mspace m2*
by (*simp add*: *Lipschitz_continuous_map_def*)

lemma *Lipschitz_continuous_map_pos*:
Lipschitz_continuous_map m1 m2 $f \longleftrightarrow$
$f \in$ *mspace m1* $\to$ *mspace m2* $\wedge$
$(\exists B{>}0.\ \forall x \in$ *mspace m1*. $\forall y \in$ *mspace m1*. *mdist m2* $(f\ x)$ $(f\ y) \leq B *$ *mdist m1* x $y)$
proof −
have $B *$ *mdist m1* x $y \leq (|B| + 1) *$ *mdist m1* x y $|B| + 1 > 0$ **for** x y B

by (*auto simp add*: *mult_right_mono*)
then show *?thesis*
unfolding *Lipschitz_continuous_map_def* **by** (*meson dual_order.trans*)
qed

lemma *Lipschitz_continuous_map_eq*:
assumes *Lipschitz_continuous_map m1 m2 f* $\bigwedge x.\ x \in$ *mspace m1* $\Longrightarrow f\ x = g\ x$
shows *Lipschitz_continuous_map m1 m2 g*
using *Lipschitz_continuous_map_def assms* **by** (*simp add*: *Lipschitz_continuous_map_pos Pi_iff*)

lemma *Lipschitz_continuous_map_from_submetric*:
assumes *Lipschitz_continuous_map m1 m2 f*
shows *Lipschitz_continuous_map* (*submetric m1 S*) *m2 f*
unfolding *Lipschitz_continuous_map_def*
proof
show $f \in$ *mspace* (*submetric m1 S*) $\to$ *mspace m2*
using *Lipschitz_continuous_map_pos assms* **by** *fastforce*
qed (*use assms* **in** ‹*fastforce simp*: *Lipschitz_continuous_map_def*›)

lemma *Lipschitz_continuous_map_from_submetric_mono*:
$[\![$*Lipschitz_continuous_map* (*submetric m1 T*) *m2 f*; $S \subseteq T]\!]$
$\Longrightarrow$ *Lipschitz_continuous_map* (*submetric m1 S*) *m2 f*
by (*metis Lipschitz_continuous_map_from_submetric inf.absorb_iff2 submetric_submetric*)

lemma *Lipschitz_continuous_map_into_submetric*:
Lipschitz_continuous_map m1 (*submetric m2 S*) *f* $\longleftrightarrow$
$f \in$ *mspace m1* $\to S \wedge$ *Lipschitz_continuous_map m1 m2 f*
by (*auto simp*: *Lipschitz_continuous_map_def*)

lemma *Lipschitz_continuous_map_const*:
Lipschitz_continuous_map m1 m2 ($\lambda x.\ c$) $\longleftrightarrow$
mspace m1 $= \{\} \vee c \in$ *mspace m2*
unfolding *Lipschitz_continuous_map_def Pi_iff*
by (*metis all_not_in_conv mdist_nonneg mdist_zero mult_1*)

lemma *Lipschitz_continuous_map_id*:
Lipschitz_continuous_map m1 m1 ($\lambda x.\ x$)
unfolding *Lipschitz_continuous_map_def* **by** (*metis funcset_id mult_1 order_refl*)

lemma *Lipschitz_continuous_map_compose*:
assumes *f*: *Lipschitz_continuous_map m1 m2 f* **and** *g*: *Lipschitz_continuous_map m2 m3 g*
shows *Lipschitz_continuous_map m1 m3* ($g \circ f$)
unfolding *Lipschitz_continuous_map_def*

proof
show $g \circ f \in mspace\ m1 \to mspace\ m3$
by (*smt* (*verit*, *best*) *Lipschitz_continuous_map_image Pi_iff comp_apply f g*)
obtain B **where** B: $\forall x{\in}mspace\ m1.\ \forall y{\in}mspace\ m1.\ mdist\ m2\ (f\ x)\ (f\ y) \le B * mdist\ m1\ x\ y$
using *assms* **unfolding** *Lipschitz_continuous_map_def* **by** *presburger*
obtain C **where** $C{>}0$ **and** C: $\forall x{\in}mspace\ m2.\ \forall y{\in}mspace\ m2.\ mdist\ m3\ (g\ x)\ (g\ y) \le C * mdist\ m2\ x\ y$
using *assms* **unfolding** *Lipschitz_continuous_map_pos* **by** *metis*
show $\exists B.\ \forall x{\in}mspace\ m1.\ \forall y{\in}mspace\ m1.\ mdist\ m3\ ((g \circ f)\ x)\ ((g \circ f)\ y) \le B * mdist\ m1\ x\ y$
proof (*intro strip exI*)
fix $x\ y$
assume §: $x \in mspace\ m1\ y \in mspace\ m1$
then have $mdist\ m3\ ((g \circ f)\ x)\ ((g \circ f)\ y) \le C * mdist\ m2\ (f\ x)\ (f\ y)$
using C *Lipschitz_continuous_map_image f* **by** *fastforce*
also have $\ldots \le C * B * mdist\ m1\ x\ y$
by (*simp add*: § B ‹$0 < C$›)
finally show $mdist\ m3\ ((g \circ f)\ x)\ ((g \circ f)\ y) \le C * B * mdist\ m1\ x\ y$ **.**
qed
qed

Uniform continuity

definition *uniformly_continuous_map*
where *uniformly_continuous_map* ≡
$\lambda m1\ m2\ f.\ f \in mspace\ m1 \to mspace\ m2 \wedge$
$(\forall \varepsilon{>}0.\ \exists \delta{>}0.\ \forall x \in mspace\ m1.\ \forall y \in mspace\ m1.$
$mdist\ m1\ y\ x < \delta \longrightarrow mdist\ m2\ (f\ y)\ (f\ x) < \varepsilon)$

lemma *uniformly_continuous_map_funspace*:
uniformly_continuous_map m1 m2 f ⟹ $f \in mspace\ m1 \to mspace\ m2$
by (*simp add*: *uniformly_continuous_map_def*)

lemma *ucmap_A*:
assumes *uniformly_continuous_map m1 m2 f*
and $(\lambda n.\ mdist\ m1\ (\varrho\ n)\ (\sigma\ n)) \longrightarrow 0$
and *range* $\varrho \subseteq mspace\ m1$ *range* $\sigma \subseteq mspace\ m1$
shows $(\lambda n.\ mdist\ m2\ (f\ (\varrho\ n))\ (f\ (\sigma\ n))) \longrightarrow 0$
using *assms*
unfolding *uniformly_continuous_map_def image_subset_iff tendsto_iff*
apply *clarsimp*
by (*metis* (*mono_tags*, *lifting*) *eventually_sequentially*)

lemma *ucmap_B*:
assumes §: $\bigwedge \varrho\ \sigma.\ [\![range\ \varrho \subseteq mspace\ m1;\ range\ \sigma \subseteq mspace\ m1;$
$(\lambda n.\ mdist\ m1\ (\varrho\ n)\ (\sigma\ n)) \longrightarrow 0]\!]$
⟹ $(\lambda n.\ mdist\ m2\ (f\ (\varrho\ n))\ (f\ (\sigma\ n))) \longrightarrow 0$

```
    and 0 < ε
    and ϱ: range ϱ ⊆ mspace m1
    and σ: range σ ⊆ mspace m1
    and (λn. mdist m1 (ϱ n) (σ n)) ⟶ 0
  shows ∃n. mdist m2 (f (ϱ (n::nat))) (f (σ n)) < ε
  using § [OF ϱ σ] assms by (meson LIMSEQ_le_const linorder_not_less)

lemma ucmap_C:
  assumes §: ⋀ϱ σ ε. ⟦ε > 0; range ϱ ⊆ mspace m1; range σ ⊆ mspace m1;
                      ((λn. mdist m1 (ϱ n) (σ n)) ⟶ 0)⟧
                     ⟹ ∃n. mdist m2 (f (ϱ n)) (f (σ n)) < ε
    and fim: f ∈ mspace m1 → mspace m2
  shows uniformly_continuous_map m1 m2 f
proof −
  {assume ¬ (∀ε>0. ∃δ>0. ∀x∈mspace m1. ∀y∈mspace m1. mdist m1 y x < δ
⟶ mdist m2 (f y) (f x) < ε)
    then obtain ε where ε > 0
      and ⋀n. ∃x∈mspace m1. ∃y∈mspace m1. mdist m1 y x < inverse(Suc n) ∧
mdist m2 (f y) (f x) ≥ ε
      by (meson inverse_Suc linorder_not_le)
    then obtain ϱ σ where space: range ϱ ⊆ mspace m1 range σ ⊆ mspace m1
        and dist: ⋀n. mdist m1 (σ n) (ϱ n) < inverse(Suc n) ∧ mdist m2 (f(σ
n)) (f(ϱ n)) ≥ ε
      by (metis image_subset_iff)
    have False
      using § [OF ‹ε > 0› space] dist Lim_null_comparison
    by (smt (verit) LIMSEQ_norm_0 inverse_eq_divide mdist_commute mdist_nonneg
real_norm_def)
  }
  moreover
  have t ∈ mspace m2 if t ∈ f ‘ mspace m1 for t
    using fim that by blast
  ultimately show ?thesis
    by (fastforce simp: uniformly_continuous_map_def)
qed

lemma uniformly_continuous_map_sequentially:
  uniformly_continuous_map m1 m2 f ⟷
    f ∈ mspace m1 → mspace m2 ∧
    (∀ϱ σ. range ϱ ⊆ mspace m1 ∧ range σ ⊆ mspace m1 ∧ (λn. mdist m1 (ϱ n)
(σ n)) ⟶ 0
          ⟶ (λn. mdist m2 (f (ϱ n)) (f (σ n))) ⟶ 0)
   (is ?lhs ⟷ ?rhs)
proof
  show ?lhs ⟹ ?rhs
    by (simp add: ucmap_A uniformly_continuous_map_funspace)
  show ?rhs ⟹ ?lhs
    by (intro ucmap_B ucmap_C) auto
qed
```

lemma *uniformly_continuous_map_sequentially_alt*:
  uniformly_continuous_map m1 m2 f ⟷
   f ∈ *mspace m1* → *mspace m2* ∧
   (∀ *ε*>*0*. ∀ *ϱ σ*. *range ϱ* ⊆ *mspace m1* ∧ *range σ* ⊆ *mspace m1* ∧
      ((λ*n*. *mdist m1* (*ϱ n*) (*σ n*)) ⟶ *0*)
      ⟶ (∃ *n*. *mdist m2* (*f* (*ϱ n*)) (*f* (*σ n*)) < *ε*))
 (**is** *?lhs* ⟷ *?rhs*)
proof
 show *?lhs* ⟹ *?rhs*
  using *uniformly_continuous_map_funspace* **by** (*intro conjI strip ucmap_B* | *fastforce simp*: *ucmap_A*)+
 show *?rhs* ⟹ *?lhs*
  by (*intro ucmap_C*) *auto*
qed

lemma *uniformly_continuous_map_eq*:
 ⟦⋀*x*. *x* ∈ *mspace m1* ⟹ *f x* = *g x*; *uniformly_continuous_map m1 m2 f*⟧
  ⟹ *uniformly_continuous_map m1 m2 g*
 by (*simp add*: *uniformly_continuous_map_def Pi_iff*)

lemma *uniformly_continuous_map_from_submetric*:
 assumes *uniformly_continuous_map m1 m2 f*
 shows *uniformly_continuous_map* (*submetric m1 S*) *m2 f*
 unfolding *uniformly_continuous_map_def*
proof
 show *f* ∈ *mspace* (*submetric m1 S*) → *mspace m2*
  using *assms* **by** (*auto simp*: *uniformly_continuous_map_def*)
qed (*use assms* **in** ‹*force simp*: *uniformly_continuous_map_def*›)

lemma *uniformly_continuous_map_from_submetric_mono*:
 ⟦*uniformly_continuous_map* (*submetric m1 T*) *m2 f*; *S* ⊆ *T*⟧
  ⟹ *uniformly_continuous_map* (*submetric m1 S*) *m2 f*
 by (*metis uniformly_continuous_map_from_submetric inf.absorb_iff2 submetric_submetric*)

lemma *uniformly_continuous_map_into_submetric*:
  uniformly_continuous_map m1 (*submetric m2 S*) *f* ⟷
   f ∈ *mspace m1* → *S* ∧ *uniformly_continuous_map m1 m2 f*
 by (*auto simp*: *uniformly_continuous_map_def*)

lemma *uniformly_continuous_map_const*:
 uniformly_continuous_map m1 m2 (λ*x*. *c*) ⟷
   mspace m1 = {} ∨ *c* ∈ *mspace m2*
 unfolding *uniformly_continuous_map_def Pi_iff*
 by (*metis empty_iff equals0I mdist_zero*)

lemma *uniformly_continuous_map_id* [*simp*]:
uniformly_continuous_map m1 m1 (λ*x*. *x*)
by (*metis funcset_id uniformly_continuous_map_def*)

lemma *uniformly_continuous_map_compose*:
assumes *f*: *uniformly_continuous_map m1 m2 f* **and** *g*: *uniformly_continuous_map m2 m3 g*
shows *uniformly_continuous_map m1 m3* (*g* ∘ *f*)
by (*smt* (*verit*, *ccfv_threshold*) *comp_apply f g Pi_iff uniformly_continuous_map_def*)

lemma *uniformly_continuous_map_real_const* [*simp*]:
uniformly_continuous_map m euclidean_metric (λ*x*. *c*)
by (*simp add*: *euclidean_metric_def uniformly_continuous_map_const*)

Equivalence between "abstract" and "type class" notions

lemma *uniformly_continuous_map_euclidean* [*simp*]:
uniformly_continuous_map (*submetric euclidean_metric S*) *euclidean_metric f*
= *uniformly_continuous_on S f*
by (*auto simp add*: *uniformly_continuous_map_def uniformly_continuous_on_def*)

Cauchy continuity

definition *Cauchy_continuous_map* **where**
Cauchy_continuous_map ≡
λ*m1 m2 f*. ∀ σ. *Metric_space.MCauchy* (*mspace m1*) (*mdist m1*) σ
⟶ *Metric_space.MCauchy* (*mspace m2*) (*mdist m2*) (*f* ∘ σ)

lemma *Cauchy_continuous_map_euclidean* [*simp*]:
Cauchy_continuous_map (*submetric euclidean_metric S*) *euclidean_metric f*
= *Cauchy_continuous_on S f*
by (*auto simp add*: *Cauchy_continuous_map_def Cauchy_continuous_on_def Cauchy_def Met_TC.subspace Metric_space.MCauchy_def*)

lemma *Cauchy_continuous_map_funspace*:
assumes *Cauchy_continuous_map m1 m2 f*
shows *f* ∈ *mspace m1* → *mspace m2*
proof *clarsimp*
fix *x*
assume *x* ∈ *mspace m1*
then have *Metric_space.MCauchy* (*mspace m1*) (*mdist m1*) (λ*n*. *x*)
by (*simp add*: *Metric_space.MCauchy_const Metric_space_mspace_mdist*)
then have *Metric_space.MCauchy* (*mspace m2*) (*mdist m2*) (*f* ∘ (λ*n*. *x*))
by (*meson Cauchy_continuous_map_def assms*)
then show *f x* ∈ *mspace m2*
by (*simp add*: *Metric_space.MCauchy_def* [*OF Metric_space_mspace_mdist*])
qed

lemma *Cauchy_continuous_map_eq*:

⟦⋀x. x ∈ mspace m1 ⟹ f x = g x; Cauchy_continuous_map m1 m2 f⟧
⟹ Cauchy_continuous_map m1 m2 g
by (*simp add*: *image_subset_iff Cauchy_continuous_map_def Metric_space.MCauchy_def* [*OF Metric_space_mspace_mdist*])

lemma *Cauchy_continuous_map_from_submetric*:
assumes *Cauchy_continuous_map m1 m2 f*
shows *Cauchy_continuous_map* (*submetric m1 S*) *m2 f*
using *assms*
by (*simp add*: *image_subset_iff Cauchy_continuous_map_def Metric_space.MCauchy_def* [*OF Metric_space_mspace_mdist*])

lemma *Cauchy_continuous_map_from_submetric_mono*:
⟦*Cauchy_continuous_map* (*submetric m1 T*) *m2 f*; *S* ⊆ *T*⟧
⟹ *Cauchy_continuous_map* (*submetric m1 S*) *m2 f*
by (*metis Cauchy_continuous_map_from_submetric inf.absorb_iff2 submetric_submetric*)

lemma *Cauchy_continuous_map_into_submetric*:
Cauchy_continuous_map m1 (*submetric m2 S*) *f* ⟷
f ∈ *mspace m1* → *S* ∧ *Cauchy_continuous_map m1 m2 f* (**is** *?lhs* ⟷ *?rhs*)
proof −
have *?lhs* ⟹ *Cauchy_continuous_map m1 m2 f*
by (*simp add*: *Cauchy_continuous_map_def Metric_space.MCauchy_def* [*OF Metric_space_mspace_mdist*])
moreover have *?rhs* ⟹ *?lhs*
by (*auto simp*: *Cauchy_continuous_map_def Metric_space.MCauchy_def* [*OF Metric_space_mspace_mdist*])
ultimately show *?thesis*
by (*metis Cauchy_continuous_map_funspace Int_iff funcsetI funcset_mem mspace_submetric*)
qed

lemma *Cauchy_continuous_map_const* [*simp*]:
Cauchy_continuous_map m1 m2 (λx. *c*) ⟷ *mspace m1* = {} ∨ *c* ∈ *mspace m2*
proof −
have *mspace m1* = {} ⟹ *Cauchy_continuous_map m1 m2* (λx. *c*)
by (*simp add*: *Cauchy_continuous_map_def Metric_space.MCauchy_def Metric_space_mspace_mdist*)
moreover have *c* ∈ *mspace m2* ⟹ *Cauchy_continuous_map m1 m2* (λx. *c*)
by (*simp add*: *Cauchy_continuous_map_def o_def Metric_space.MCauchy_const* [*OF Metric_space_mspace_mdist*])
ultimately show *?thesis*
using *Cauchy_continuous_map_funspace* **by** *blast*
qed

lemma *Cauchy_continuous_map_id* [*simp*]:
Cauchy_continuous_map m1 m1 (λx. *x*)

by (*simp add*: *Cauchy_continuous_map_def o_def*)

lemma *Cauchy_continuous_map_compose*:
assumes *f*: *Cauchy_continuous_map m1 m2 f* **and** *g*: *Cauchy_continuous_map m2 m3 g*
shows *Cauchy_continuous_map m1 m3* (*g* ∘ *f*)
by (*metis* (*no_types*, *lifting*) *Cauchy_continuous_map_def f fun.map_comp g*)

lemma *Lipschitz_imp_uniformly_continuous_map*:
assumes *Lipschitz_continuous_map m1 m2 f*
shows *uniformly_continuous_map m1 m2 f*
proof −
have *f* ∈ *mspace m1* → *mspace m2*
by (*simp add*: *Lipschitz_continuous_map_image assms*)
moreover have ∃ *δ*>*0*. ∀ *x*∈*mspace m1*. ∀ *y*∈*mspace m1*. *mdist m1 y x* < *δ* ⟶ *mdist m2* (*f y*) (*f x*) < *ε*
if *ε* > *0* **for** *ε*
proof −
obtain *B* **where** ∀ *x*∈*mspace m1*. ∀ *y*∈*mspace m1*. *mdist m2* (*f x*) (*f y*) ≤ *B* ∗ *mdist m1 x y*
and *B*>*0*
using *that assms* **by** (*force simp add*: *Lipschitz_continuous_map_pos*)
then have ∀ *x*∈*mspace m1*. ∀ *y*∈*mspace m1*. *mdist m1 y x* < *ε*/*B* ⟶ *mdist m2* (*f y*) (*f x*) < *ε*
by (*smt* (*verit*, *ccfv_SIG*) *less_divide_eq mdist_nonneg mult.commute that zero_less_divide_iff*)
with ‹*B*>*0*› **show** *?thesis*
by (*metis divide_pos_pos that*)
qed
ultimately show *?thesis*
by (*auto simp*: *uniformly_continuous_map_def*)
qed

lemma *uniformly_imp_Cauchy_continuous_map*:
uniformly_continuous_map m1 m2 f ⟹ *Cauchy_continuous_map m1 m2 f*
unfolding *uniformly_continuous_map_def Cauchy_continuous_map_def*
apply (*simp add*: *image_subset_iff o_def Metric_space.MCauchy_def* [*OF Metric_space_mspace_mdist*])
by (*metis funcset_mem*)

lemma *locally_Cauchy_continuous_map*:
assumes *ε* > *0*
and §: ⋀*x*. *x* ∈ *mspace m1* ⟹ *Cauchy_continuous_map* (*submetric m1* (*mball_of m1 x ε*)) *m2 f*
shows *Cauchy_continuous_map m1 m2 f*
unfolding *Cauchy_continuous_map_def*
proof (*intro strip*)
interpret *M1*: *Metric_space mspace m1 mdist m1*
by (*simp add*: *Metric_space_mspace_mdist*)

interpret *M2*: *Metric_space mspace m2 mdist m2*
by (*simp add*: *Metric_space_mspace_mdist*)
fix σ
assume σ: *M1.MCauchy* σ
with ‹$\varepsilon > 0$›
obtain N **where** N: $\bigwedge n\ n'.\ [\![n \geq N;\ n' \geq N]\!] \Longrightarrow$ *mdist m1* (σ n) (σ n') $< \varepsilon$
using *M1.MCauchy_def* **by** *fastforce*
then have *M1.mball* (σ N) $\varepsilon \subseteq$ *mspace m1*
by (*auto simp*: *image_subset_iff M1.mball_def*)
then interpret *MS1*: *Metric_space mball_of m1* (σ N) $\varepsilon \cap$ *mspace m1 mdist m1*
by (*simp add*: *M1.subspace*)
show *M2.MCauchy* ($f \circ \sigma$)
proof (*rule M2.MCauchy_offset*)
have *M1.MCauchy* ($\sigma \circ (+)$ N)
by (*simp add*: *M1.MCauchy_imp_MCauchy_suffix* σ)
moreover have *range* ($\sigma \circ (+)$ N) $\subseteq$ *M1.mball* (σ N) ε
using N [*OF order_refl*] *M1.MCauchy_def* σ **by** *fastforce*
ultimately have *MS1.MCauchy* ($\sigma \circ (+)$ N)
unfolding *M1.MCauchy_def MS1.MCauchy_def* **by** (*simp add*: *mball_of_def*)
moreover have σ $N \in$ *mspace m1*
using *M1.MCauchy_def* σ **by** *auto*
ultimately show *M2.MCauchy* ($f \circ \sigma \circ (+)$ N)
unfolding *comp_assoc*
by (*metis* § *Cauchy_continuous_map_def mdist_submetric mspace_submetric*)
next
fix n
have σ $n \in$ *mspace m1*
by (*meson Metric_space.MCauchy_def Metric_space_mspace_mdist* σ *range_subsetD*)
then have σ $n \in$ *mball_of m1* (σ n) ε
by (*simp add*: *Metric_space.centre_in_mball_iff Metric_space_mspace_mdist assms*(*1*) *mball_of_def*)
then show ($f \circ \sigma$) $n \in$ *mspace m2*
using *Cauchy_continuous_map_funspace* [*OF* § [*of* σ n]] ‹σ $n \in$ *mspace m1*›
by *auto*
qed
qed

context *Metric_space12*
begin

lemma *Cauchy_continuous_imp_continuous_map*:
assumes *Cauchy_continuous_map* (*metric* (*M1*,*d1*)) (*metric* (*M2*,*d2*)) *f*
shows *continuous_map M1.mtopology M2.mtopology f*
proof (*clarsimp simp*: *continuous_map_atin*)
fix x
assume $x \in$ *M1*
show *limitin M2.mtopology f* (*f x*) (*atin M1.mtopology x*)
unfolding *limit_atin_sequentially*

```
  proof (intro conjI strip)
    show f x ∈ M2
      using Cauchy_continuous_map_funspace ‹x ∈ M1› assms by fastforce
    fix σ
    assume range σ ⊆ M1 − {x} ∧ limitin M1.mtopology σ x sequentially
    then have M1.MCauchy (λn. if even n then σ (n div 2) else x)
      by (force simp add: M1.MCauchy_interleaving)
    then have M2.MCauchy (f ∘ (λn. if even n then σ (n div 2) else x))
      using assms by (simp add: Cauchy_continuous_map_def)
    then show limitin M2.mtopology (f ∘ σ) (f x) sequentially
      using M2.MCauchy_interleaving [of f ∘ σ f x]
      by (simp add: o_def if_distrib cong: if_cong)
  qed
qed

lemma continuous_imp_Cauchy_continuous_map:
  assumes M1.mcomplete
    and f: continuous_map M1.mtopology M2.mtopology f
  shows Cauchy_continuous_map (metric (M1,d1)) (metric (M2,d2)) f
  unfolding Cauchy_continuous_map_def
proof clarsimp
  fix σ
  assume σ: M1.MCauchy σ
  then obtain y where y: limitin M1.mtopology σ y sequentially
    using M1.mcomplete_def assms by blast
  have range (f ∘ σ) ⊆ M2
   using σ f by (simp add: M2.subspace M1.MCauchy_def M1.metric_continuous_map
image_subset_iff)
  then show M2.MCauchy (f ∘ σ)
    using continuous_map_limit [OF f y] M2.convergent_imp_MCauchy
    by blast
qed

end
```

The same outside the locale

```
lemma Cauchy_continuous_imp_continuous_map:
  assumes Cauchy_continuous_map m1 m2 f
  shows continuous_map (mtopology_of m1) (mtopology_of m2) f
  using assms Metric_space12.Cauchy_continuous_imp_continuous_map [OF Met-
ric_space12_mspace_mdist]
  by (auto simp add: mtopology_of_def)

lemma continuous_imp_Cauchy_continuous_map:
  assumes Metric_space.mcomplete (mspace m1) (mdist m1)
    and continuous_map (mtopology_of m1) (mtopology_of m2) f
  shows Cauchy_continuous_map m1 m2 f
  using assms Metric_space12.continuous_imp_Cauchy_continuous_map [OF Met-
ric_space12_mspace_mdist]
```

by (*auto simp add*: *mtopology_of_def*)

lemma *uniformly_continuous_imp_continuous_map*:
uniformly_continuous_map m1 m2 f
⟹ *continuous_map* (*mtopology_of m1*) (*mtopology_of m2*) *f*
by (*simp add*: *Cauchy_continuous_imp_continuous_map uniformly_imp_Cauchy_continuous_map*)

lemma *Lipschitz_continuous_imp_continuous_map*:
Lipschitz_continuous_map m1 m2 f
⟹ *continuous_map* (*mtopology_of m1*) (*mtopology_of m2*) *f*
by (*simp add*: *Lipschitz_imp_uniformly_continuous_map uniformly_continuous_imp_continuous_map*)

lemma *Lipschitz_imp_Cauchy_continuous_map*:
Lipschitz_continuous_map m1 m2 f
⟹ *Cauchy_continuous_map m1 m2 f*
by (*simp add*: *Lipschitz_imp_uniformly_continuous_map uniformly_imp_Cauchy_continuous_map*)

lemma *Cauchy_imp_uniformly_continuous_map*:
assumes *f*: *Cauchy_continuous_map m1 m2 f*
and *tbo*: *Metric_space.mtotally_bounded* (*mspace m1*) (*mdist m1*) (*mspace m1*)
shows *uniformly_continuous_map m1 m2 f*
unfolding *uniformly_continuous_map_sequentially_alt imp_conjL*
proof (*intro conjI strip*)
show *f* ∈ *mspace m1* → *mspace m2*
by (*simp add*: *Cauchy_continuous_map_funspace f*)
interpret *M1*: *Metric_space mspace m1 mdist m1*
by (*simp add*: *Metric_space_mspace_mdist*)
interpret *M2*: *Metric_space mspace m2 mdist m2*
by (*simp add*: *Metric_space_mspace_mdist*)
fix ε :: *real* **and** ϱ σ
assume ε > *0*
and ϱ: *range* ϱ ⊆ *mspace m1*
and σ: *range* σ ⊆ *mspace m1*
and *0*: (λ*n*. *mdist m1* (ϱ *n*) (σ *n*)) ⟶ *0*
then obtain *r1* **where** *strict_mono r1* **and** *r1*: *M1.MCauchy* (ϱ ∘ *r1*)
using *M1.mtotally_bounded_sequentially tbo* **by** *meson*
then obtain *r2* **where** *strict_mono r2* **and** *r2*: *M1.MCauchy* (σ ∘ *r1* ∘ *r2*)
by (*metis M1.mtotally_bounded_sequentially tbo σ image_comp image_subset_iff range_subsetD*)
define *r* **where** *r* ≡ *r1* ∘ *r2*
have *r*: *strict_mono r*
by (*simp add*: *r_def* ‹*strict_mono r1*› ‹*strict_mono r2*› *strict_mono_o*)
define η **where** η ≡ λ*n*. *if even n then* (ϱ ∘ *r*) (*n div 2*) *else* (σ ∘ *r*) (*n div 2*)
have *M1.MCauchy* η
unfolding η*_def M1.MCauchy_interleaving_gen*
proof (*intro conjI*)
show *M1.MCauchy* (ϱ ∘ *r*)
by (*simp add*: *M1.MCauchy_subsequence* ‹*strict_mono r2*› *fun.map_comp*

```
r1 r_def)
    show M1.MCauchy (σ ∘ r)
      by (simp add: fun.map_comp r2 r_def)
    show (λn. mdist m1 ((ϱ ∘ r) n) ((σ ∘ r) n)) ⟶ 0
      using LIMSEQ_subseq_LIMSEQ [OF 0 r] by (simp add: o_def)
  qed
  then have Metric_space.MCauchy (mspace m2) (mdist m2) (f ∘ η)
    by (meson Cauchy_continuous_map_def f)
  then have (λn. mdist m2 (f (ϱ (r n))) (f (σ (r n)))) ⟶ 0
    using M2.MCauchy_interleaving_gen [of f ∘ ϱ ∘ r f ∘ σ ∘ r]
    by (simp add: η_def o_def if_distrib cong: if_cong)
  then show ∃n. mdist m2 (f (ϱ n)) (f (σ n)) < ε
    by (meson LIMSEQ_le_const ‹0 < ε› linorder_not_le)
qed

lemma continuous_imp_uniformly_continuous_map:
   compact_space (mtopology_of m1) ∧
        continuous_map (mtopology_of m1) (mtopology_of m2) f
        ⟹ uniformly_continuous_map m1 m2 f
 by (simp add: Cauchy_imp_uniformly_continuous_map continuous_imp_Cauchy_continuous_map
               Metric_space.compact_space_eq_mcomplete_mtotally_bounded
               Metric_space_mspace_mdist mtopology_of_def)

lemma continuous_eq_Cauchy_continuous_map:
   Metric_space.mcomplete (mspace m1) (mdist m1) ⟹
   continuous_map (mtopology_of m1) (mtopology_of m2) f ⟷ Cauchy_continuous_map
m1 m2 f
  using Cauchy_continuous_imp_continuous_map continuous_imp_Cauchy_continuous_map
by blast

lemma continuous_eq_uniformly_continuous_map:
   compact_space (mtopology_of m1)
    ⟹ continuous_map (mtopology_of m1) (mtopology_of m2) f ⟷
       uniformly_continuous_map m1 m2 f
  using continuous_imp_uniformly_continuous_map uniformly_continuous_imp_continuous_map
by blast

lemma Cauchy_eq_uniformly_continuous_map:
   Metric_space.mtotally_bounded (mspace m1) (mdist m1) (mspace m1)
     ⟹ Cauchy_continuous_map m1 m2 f ⟷ uniformly_continuous_map m1
m2 f
  using Cauchy_imp_uniformly_continuous_map uniformly_imp_Cauchy_continuous_map
by blast

lemma Lipschitz_continuous_map_projections:
  Lipschitz_continuous_map (prod_metric m1 m2) m1 fst
  Lipschitz_continuous_map (prod_metric m1 m2) m2 snd
  by (simp add: Lipschitz_continuous_map_def prod_dist_def fst_Pi snd_Pi;
    metis mult_numeral_1 real_sqrt_sum_squares_ge1 real_sqrt_sum_squares_ge2)+
```

lemma *Lipschitz_continuous_map_pairwise*:
Lipschitz_continuous_map m (prod_metric m1 m2) f ⟷
Lipschitz_continuous_map m m1 (fst ∘ f) ∧ Lipschitz_continuous_map m m2 (snd ∘ f)
(**is** *?lhs ⟷ ?rhs*)
proof
show *?lhs ⟹ ?rhs*
by (*simp add*: *Lipschitz_continuous_map_compose Lipschitz_continuous_map_projections*)
have *Lipschitz_continuous_map m (prod_metric m1 m2) (λx. (f1 x, f2 x))*
if *f1*: *Lipschitz_continuous_map m m1 f1* **and** *f2*: *Lipschitz_continuous_map m m2 f2* **for** *f1 f2*
proof −
obtain *B1* **where** *B1 > 0*
and *B1*: *⋀x y. ⟦x ∈ mspace m; y ∈ mspace m⟧ ⟹ mdist m1 (f1 x) (f1 y) ≤ B1 * mdist m x y*
by (*meson Lipschitz_continuous_map_pos f1*)
obtain *B2* **where** *B2 > 0*
and *B2*: *⋀x y. ⟦x ∈ mspace m; y ∈ mspace m⟧ ⟹ mdist m2 (f2 x) (f2 y) ≤ B2 * mdist m x y*
by (*meson Lipschitz_continuous_map_pos f2*)
show *?thesis*
unfolding *Lipschitz_continuous_map_pos*
proof (*intro exI conjI strip*)
have *f1im*: *f1 ∈ mspace m → mspace m1*
by (*simp add*: *Lipschitz_continuous_map_image f1*)
moreover have *f2im*: *f2 ∈ mspace m → mspace m2*
by (*simp add*: *Lipschitz_continuous_map_image f2*)
ultimately show *(λx. (f1 x, f2 x)) ∈ mspace m → mspace (prod_metric m1 m2)*
by *auto*
show *B1+B2 > 0*
using ‹*0 < B1*› ‹*0 < B2*› **by** *linarith*
fix *x y*
assume *xy*: *x ∈ mspace m y ∈ mspace m*
with *f1im f2im* **have** *mdist (prod_metric m1 m2) (f1 x, f2 x) (f1 y, f2 y) ≤ mdist m1 (f1 x) (f1 y) + mdist m2 (f2 x) (f2 y)*
unfolding *mdist_prod_metric*
by (*intro Metric_space12.prod_metric_le_components* [*OF Metric_space12_mspace_mdist*]) *auto*
also have *... ≤ (B1+B2) * mdist m x y*
using *B1* [*OF xy*] *B2* [*OF xy*] **by** (*simp add*: *vector_space_over_itself.scale_left_distrib*)

finally show *mdist (prod_metric m1 m2) (f1 x, f2 x) (f1 y, f2 y) ≤ (B1+B2) * mdist m x y* **.**
qed
qed
then show *?rhs ⟹ ?lhs*
by *force*

qed

lemma *uniformly_continuous_map_pairwise*:
uniformly_continuous_map m (*prod_metric m1 m2*) *f* ⟷
uniformly_continuous_map m m1 (*fst* ∘ *f*) ∧ *uniformly_continuous_map m m2* (*snd* ∘ *f*)
(**is** *?lhs* ⟷ *?rhs*)
proof
show *?lhs* ⟹ *?rhs*
by (*simp add*: *Lipschitz_continuous_map_projections Lipschitz_imp_uniformly_continuous_map uniformly_continuous_map_compose*)
have *uniformly_continuous_map m* (*prod_metric m1 m2*) (λ*x*. (*f1 x*, *f2 x*))
if *f1*: *uniformly_continuous_map m m1 f1* **and** *f2*: *uniformly_continuous_map m m2 f2* **for** *f1 f2*
proof −
show *?thesis*
unfolding *uniformly_continuous_map_def*
proof (*intro conjI strip*)
have *f1im*: *f1* ∈ *mspace m* → *mspace m1*
by (*simp add*: *uniformly_continuous_map_funspace f1*)
moreover have *f2im*: *f2* ∈ *mspace m* → *mspace m2*
by (*simp add*: *uniformly_continuous_map_funspace f2*)
ultimately show (λ*x*. (*f1 x*, *f2 x*)) ∈ *mspace m* → *mspace* (*prod_metric m1 m2*)
by *auto*
fix ε:: *real*
assume ε > *0*
obtain *δ1* **where** *δ1*>*0*
and *δ1*: ⋀*x y*. ⟦*x* ∈ *mspace m*; *y* ∈ *mspace m*; *mdist m y x* < *δ1*⟧ ⟹ *mdist m1* (*f1 y*) (*f1 x*) < ε/*2*
by (*metis* ‹*0* < ε› *f1 half_gt_zero uniformly_continuous_map_def*)
obtain *δ2* **where** *δ2*>*0*
and *δ2*: ⋀*x y*. ⟦*x* ∈ *mspace m*; *y* ∈ *mspace m*; *mdist m y x* < *δ2*⟧ ⟹ *mdist m2* (*f2 y*) (*f2 x*) < ε/*2*
by (*metis* ‹*0* < ε› *f2 half_gt_zero uniformly_continuous_map_def*)
show ∃*δ*>*0*. ∀*x*∈*mspace m*. ∀*y*∈*mspace m*. *mdist m y x* < *δ* ⟶ *mdist* (*prod_metric m1 m2*) (*f1 y*, *f2 y*) (*f1 x*, *f2 x*) < ε
proof (*intro exI conjI strip*)
show *min δ1 δ2*>*0*
using ‹*0* < *δ1*› ‹*0* < *δ2*› **by** *auto*
fix *x y*
assume *xy*: *x* ∈ *mspace m y* ∈ *mspace m* **and** *d*: *mdist m y x* < *min δ1 δ2*
have ∗: *mdist m1* (*f1 y*) (*f1 x*) < ε/*2 mdist m2* (*f2 y*) (*f2 x*) < ε/*2*
using *δ1 δ2 d xy* **by** *auto*
have *mdist* (*prod_metric m1 m2*) (*f1 y*, *f2 y*) (*f1 x*, *f2 x*) ≤ *mdist m1* (*f1 y*) (*f1 x*) + *mdist m2* (*f2 y*) (*f2 x*)
unfolding *mdist_prod_metric* **using** *f1im f2im xy*
by (*intro Metric_space12.prod_metric_le_components* [*OF Metric_space12_mspace_mdist*]) *auto*

```
      also have ... < ε/2 + ε/2
        using ∗ by simp
      finally show mdist (prod_metric m1 m2) (f1 y, f2 y) (f1 x, f2 x) < ε
        by simp
    qed
  qed
 qed
 then show ?rhs ⟹ ?lhs
  by force
qed

lemma Cauchy_continuous_map_pairwise:
  Cauchy_continuous_map m (prod_metric m1 m2) f ⟷ Cauchy_continuous_map
m m1 (fst ∘ f) ∧ Cauchy_continuous_map m m2 (snd ∘ f)
 by (auto simp: Cauchy_continuous_map_def Metric_space12.MCauchy_prod_metric[OF
Metric_space12_mspace_mdist] comp_assoc)

lemma Lipschitz_continuous_map_paired:
  Lipschitz_continuous_map m (prod_metric m1 m2) (λx. (f x, g x)) ⟷
      Lipschitz_continuous_map m m1 f ∧ Lipschitz_continuous_map m m2 g
 by (simp add: Lipschitz_continuous_map_pairwise o_def)

lemma uniformly_continuous_map_paired:
  uniformly_continuous_map m (prod_metric m1 m2) (λx. (f x, g x)) ⟷
      uniformly_continuous_map m m1 f ∧ uniformly_continuous_map m m2 g
 by (simp add: uniformly_continuous_map_pairwise o_def)

lemma Cauchy_continuous_map_paired:
  Cauchy_continuous_map m (prod_metric m1 m2) (λx. (f x, g x)) ⟷
      Cauchy_continuous_map m m1 f ∧ Cauchy_continuous_map m m2 g
 by (simp add: Cauchy_continuous_map_pairwise o_def)

lemma mbounded_Lipschitz_continuous_image:
 assumes f: Lipschitz_continuous_map m1 m2 f and S: Metric_space.mbounded
(mspace m1) (mdist m1) S
 shows Metric_space.mbounded (mspace m2) (mdist m2) (f‘S)
proof −
 obtain B where fim: f ∈ mspace m1 → mspace m2
  and B>0 and B: ⋀x y. ⟦x ∈ mspace m1; y ∈ mspace m1⟧ ⟹ mdist m2 (f x)
(f y) ≤ B ∗ mdist m1 x y
   by (metis Lipschitz_continuous_map_pos f)
 show ?thesis
  unfolding Metric_space.mbounded_alt_pos [OF Metric_space_mspace_mdist]
 proof
  obtain C where S ⊆ mspace m1 and C>0 and C: ⋀x y. ⟦x ∈ S; y ∈ S⟧ ⟹
mdist m1 x y ≤ C
   using S by (auto simp: Metric_space.mbounded_alt_pos [OF Metric_space_mspace_mdist])
   show f ‘ S ⊆ mspace m2
    using fim ‹S ⊆ mspace m1› by blast
```

```
    have ⋀x y. ⟦x ∈ S; y ∈ S⟧ ⟹ mdist m2 (f x) (f y) ≤ B * C
      by (smt (verit) B C ‹0 < B› ‹S ⊆ mspace m1› mdist_nonneg mult_mono
subsetD)
    moreover have B*C > 0
      by (simp add: ‹0 < B› ‹0 < C›)
    ultimately show ∃ B>0. ∀ x∈f ‘ S. ∀ y∈f ‘ S. mdist m2 x y ≤ B
      by auto
  qed
qed

lemma mtotally_bounded_Cauchy_continuous_image:
  assumes f: Cauchy_continuous_map m1 m2 f and S: Metric_space.mtotally_bounded
(mspace m1) (mdist m1) S
  shows Metric_space.mtotally_bounded (mspace m2) (mdist m2) (f ‘ S)
  unfolding Metric_space.mtotally_bounded_sequentially[OF Metric_space_mspace_mdist]
proof (intro conjI strip)
  have S ⊆ mspace m1
    using S by (simp add: Metric_space.mtotally_bounded_sequentially[OF Met-
ric_space_mspace_mdist])
  then show f ‘ S ⊆ mspace m2
    using Cauchy_continuous_map_funspace f by blast
  fix σ :: nat ⇒ 'b
  assume range σ ⊆ f ‘ S
  then have ∀ n. ∃ x. σ n = f x ∧ x ∈ S
    by (meson imageE range_subsetD)
  then obtain ϱ where ϱ: ⋀n. σ n = f (ϱ n) range ϱ ⊆ S
    by (metis image_subset_iff)
  then have σ = f ∘ ϱ
    by fastforce
  obtain r where strict_mono r Metric_space.MCauchy (mspace m1) (mdist m1)
(ϱ ∘ r)
   by (meson ϱ S Metric_space.mtotally_bounded_sequentially[OF Metric_space_mspace_mdist])
  then have Metric_space.MCauchy (mspace m2) (mdist m2) (f ∘ ϱ ∘ r)
    using f unfolding Cauchy_continuous_map_def by (metis fun.map_comp)
  then show ∃ r. strict_mono r ∧ Metric_space.MCauchy (mspace m2) (mdist
m2) (σ ∘ r)
    using ‹σ = f ∘ ϱ› ‹strict_mono r› by blast
qed

lemma Lipschitz_coefficient_pos:
  assumes x ∈ mspace m y ∈ mspace m f x ≠ f y
    and f ∈ mspace m → mspace m2
    and ⋀x y. ⟦x ∈ mspace m; y ∈ mspace m⟧
          ⟹ mdist m2 (f x) (f y) ≤ k * mdist m x y
  shows  0 < k
  using assms by (smt (verit, best) Pi_iff mdist_nonneg mdist_zero mult_nonpos_nonneg)

lemma Lipschitz_continuous_map_metric:
   Lipschitz_continuous_map (prod_metric m m) euclidean_metric (λ(x,y). mdist
```

m x y)
proof −
have $\bigwedge x\ y\ x'\ y'.\ [\![x \in mspace\ m;\ y \in mspace\ m;\ x' \in mspace\ m;\ y' \in mspace\ m]\!]$
$\Longrightarrow |mdist\ m\ x\ y - mdist\ m\ x'\ y'| \leq 2 * sqrt\ ((mdist\ m\ x\ x')^2 + (mdist\ m$
$y\ y')^2)$
by (*smt* (*verit*, *del_insts*) *mdist_commute mdist_triangle real_sqrt_sum_squares_ge2*)
then show *?thesis*
by (*fastforce simp add*: *Lipschitz_continuous_map_def prod_dist_def dist_real_def*)
qed

lemma *Lipschitz_continuous_map_mdist*:
assumes *f*: *Lipschitz_continuous_map m m' f*
and *g*: *Lipschitz_continuous_map m m' g*
shows *Lipschitz_continuous_map m euclidean_metric* (λ*x*. *mdist m'* (*f x*) (*g x*))
(**is** *Lipschitz_continuous_map m _ ?h*)
proof −
have *eq*: *?h* = ((λ(*x*,*y*). *mdist m' x y*) ∘ (λ*x*. (*f x*,*g x*)))
by *force*
show *?thesis*
unfolding *eq*
proof (*rule Lipschitz_continuous_map_compose*)
show *Lipschitz_continuous_map m* (*prod_metric m' m'*) (λ*x*. (*f x*, *g x*))
by (*simp add*: *Lipschitz_continuous_map_paired f g*)
show *Lipschitz_continuous_map* (*prod_metric m' m'*) *euclidean_metric* (λ(*x*,*y*).
mdist m' x y)
by (*simp add*: *Lipschitz_continuous_map_metric*)
qed
qed

lemma *uniformly_continuous_map_mdist*:
assumes *f*: *uniformly_continuous_map m m' f*
and *g*: *uniformly_continuous_map m m' g*
shows *uniformly_continuous_map m euclidean_metric* (λ*x*. *mdist m'* (*f x*) (*g*
x))
(**is** *uniformly_continuous_map m _ ?h*)
proof −
have *eq*: *?h* = ((λ(*x*,*y*). *mdist m' x y*) ∘ (λ*x*. (*f x*,*g x*)))
by *force*
show *?thesis*
unfolding *eq*
proof (*rule uniformly_continuous_map_compose*)
show *uniformly_continuous_map m* (*prod_metric m' m'*) (λ*x*. (*f x*, *g x*))
by (*simp add*: *uniformly_continuous_map_paired f g*)
show *uniformly_continuous_map* (*prod_metric m' m'*) *euclidean_metric* (λ(*x*,*y*).
mdist m' x y)
by (*simp add*: *Lipschitz_continuous_map_metric Lipschitz_imp_uniformly_continuous_map*)
qed
qed

lemma *Cauchy_continuous_map_mdist*:
assumes *f*: *Cauchy_continuous_map m m' f*
and *g*: *Cauchy_continuous_map m m' g*
shows *Cauchy_continuous_map m euclidean_metric* (λ*x*. *mdist m'* (*f x*) (*g x*))
(**is** *Cauchy_continuous_map m _ ?h*)
proof −
have *eq*: *?h* = ((λ(*x*,*y*). *mdist m' x y*) ∘ (λ*x*. (*f x*,*g x*)))
by *force*
show *?thesis*
unfolding *eq*
proof (*rule Cauchy_continuous_map_compose*)
show *Cauchy_continuous_map m* (*prod_metric m' m'*) (λ*x*. (*f x*, *g x*))
by (*simp add*: *Cauchy_continuous_map_paired f g*)
show *Cauchy_continuous_map* (*prod_metric m' m'*) *euclidean_metric* (λ(*x*,*y*).
mdist m' x y)
by (*simp add*: *Lipschitz_continuous_map_metric Lipschitz_imp_Cauchy_continuous_map*)
qed
qed

lemma *mtopology_of_prod_metric* [*simp*]:
mtopology_of (*prod_metric m1 m2*) = *prod_topology* (*mtopology_of m1*)
(*mtopology_of m2*)
by (*simp add*: *mtopology_of_def Metric_space12.mtopology_prod_metric*[*OF*
Metric_space12_mspace_mdist])

lemma *continuous_map_metric*:
continuous_map (*prod_topology* (*mtopology_of m*) (*mtopology_of m*)) *euclidean*
(λ(*x*,*y*). *mdist m x y*)
using *Lipschitz_continuous_imp_continuous_map* [*OF Lipschitz_continuous_map_metric*]
by *auto*

lemma *continuous_map_mdist_alt*:
assumes *continuous_map X* (*prod_topology* (*mtopology_of m*) (*mtopology_of*
m)) *f*
shows *continuous_map X euclidean* (λ*x*. *case_prod* (*mdist m*) (*f x*))
(**is** *continuous_map _ _ ?h*)
proof −
have *eq*: *?h* = *case_prod* (*mdist m*) ∘ *f*
by *force*
show *?thesis*
unfolding *eq*
using *assms continuous_map_compose continuous_map_metric* **by** *blast*
qed

lemma *continuous_map_mdist* [*continuous_intros*]:
assumes *f*: *continuous_map X* (*mtopology_of m*) *f*
and *g*: *continuous_map X* (*mtopology_of m*) *g*
shows *continuous_map X euclidean* (λ*x*. *mdist m* (*f x*) (*g x*))
(**is** *continuous_map X _ ?h*)

proof −
have *eq*: *?h* = ((λ(*x*,*y*). *mdist m x y*) ∘ (λ*x*. (*f x*,*g x*)))
by *force*
show *?thesis*
unfolding *eq*
proof (*rule continuous_map_compose*)
show *continuous_map X* (*prod_topology* (*mtopology_of m*) (*mtopology_of m*)) (λ*x*. (*f x*, *g x*))
by (*simp add*: *continuous_map_paired f g*)
qed (*simp add*: *continuous_map_metric*)
qed

lemma *continuous_on_mdist*:
a ∈ *mspace m* ⟹ *continuous_map* (*mtopology_of m*) *euclidean* (*mdist m a*)
by (*simp add*: *continuous_map_mdist*)

5.7.19 Isometries

lemma (**in** *Metric_space12*) *isometry_imp_embedding_map*:
assumes *fim*: *f* ∈ *M1* → *M2* **and** *d*: ⋀*x y*. ⟦*x* ∈ *M1*; *y* ∈ *M1*⟧ ⟹ *d2* (*f x*) (*f y*) = *d1 x y*
shows *embedding_map M1.mtopology M2.mtopology f*
proof −
have *inj_on f M1*
by (*metis M1.zero d inj_onI*)
then obtain *g* **where** *g*: ⋀*x*. *x* ∈ *M1* ⟹ *g* (*f x*) = *x*
by (*metis inv_into_f_f*)
have *homeomorphic_maps M1.mtopology* (*subtopology M2.mtopology* (*f* ‘ *topspace M1.mtopology*)) *f g*
unfolding *homeomorphic_maps_def*
proof (*intro conjI*; *clarsimp*)
show *continuous_map M1.mtopology* (*subtopology M2.mtopology* (*f* ‘ *M1*)) *f*
proof (*rule continuous_map_into_subtopology*)
show *continuous_map M1.mtopology M2.mtopology f*
by (*metis M1.metric_continuous_map M2.Metric_space_axioms d fim image_subset_iff_funcset*)
qed *simp*
have *Lipschitz_continuous_map* (*submetric* (*metric*(*M2*,*d2*)) (*f* ‘ *M1*)) (*metric*(*M1*,*d1*)) *g*
unfolding *Lipschitz_continuous_map_def*
proof (*intro conjI exI strip*; *simp*)
show *d1* (*g x*) (*g y*) ≤ *1* ∗ *d2 x y* **if** *x* ∈ *f* ‘ *M1* ∧ *x* ∈ *M2* **and** *y* ∈ *f* ‘ *M1* ∧ *y* ∈ *M2* **for** *x y*
using *that d g* **by** *force*
qed (*use g* **in** *auto*)
then have *continuous_map* (*mtopology_of* (*submetric* (*metric*(*M2*,*d2*)) (*f* ‘ *M1*))) *M1.mtopology g*
using *Lipschitz_continuous_imp_continuous_map* **by** *force*
moreover have *mtopology_of* (*submetric* (*metric*(*M2*,*d2*)) (*f* ‘ *M1*)) = *subtopol-*

ogy M2.mtopology (f ‘ M1)
by (*simp add*: *mtopology_of_submetric*)
ultimately show *continuous_map* (*subtopology M2.mtopology* (*f ‘ M1*)) *M1.mtopology g*
by *simp*
qed (*use g* **in** *auto*)
then show *?thesis*
by (*auto simp*: *embedding_map_def homeomorphic_map_maps*)
qed

lemma (**in** *Metric_space12*) *isometry_imp_homeomorphic_map*:
assumes *fim*: *f ‘ M1 = M2* **and** *d*: $\bigwedge x\ y.\ [\![x \in M1;\ y \in M1]\!] \Longrightarrow d2\ (f\ x)\ (f\ y) = d1\ x\ y$
shows *homeomorphic_map M1.mtopology M2.mtopology f*
by (*metis image_eqI M1.topspace_mtopology M2.subtopology_mspace d embedding_map_def fim isometry_imp_embedding_map Pi_iff*)

5.7.20 "Capped" equivalent bounded metrics and general product metrics

definition (**in** *Metric_space*) *capped_dist* **where**
capped_dist $\equiv \lambda\delta\ x\ y.$ *if* $\delta > 0$ *then min* δ (*d x y*) *else d x y*

lemma (**in** *Metric_space*) *capped_dist*: *Metric_space M* (*capped_dist* δ)
proof
fix *x y*
assume $x \in M$ $y \in M$
then show (*capped_dist* δ *x y* $= 0$) $= (x = y)$
by (*smt* (*verit*, *best*) *capped_dist_def zero*)
fix *z*
assume $z \in M$
show *capped_dist* δ *x z* $\leq$ *capped_dist* δ *x y* + *capped_dist* δ *y z*
unfolding *capped_dist_def* **using** ‹$x \in M$› ‹$y \in M$› ‹$z \in M$›
by (*smt* (*verit*, *del_insts*) *Metric_space.mdist_pos_eq Metric_space_axioms mdist_reverse_triangle*)
qed (*use capped_dist_def commute* **in** *auto*)

definition *capped_metric* **where**
capped_metric δ *m* $\equiv$ *metric*(*mspace m*, *Metric_space.capped_dist* (*mdist m*) δ)

lemma *capped_metric*:
capped_metric δ *m* = (*if* $\delta \leq 0$ *then m else metric*(*mspace m*, $\lambda x\ y.$ *min* δ (*mdist m x y*)))
proof −
interpret *Metric_space mspace m mdist m*
by (*simp add*: *Metric_space_mspace_mdist*)
show *?thesis*
by (*auto simp*: *capped_metric_def capped_dist_def*)

qed

lemma *capped_metric_mspace* [*simp*]:
mspace (*capped_metric* δ *m*) = *mspace m*
apply (*simp add*: *capped_metric not_le*)
by (*smt* (*verit*, *ccfv_threshold*) *Metric_space.mspace_metric Metric_space_def Metric_space_mspace_mdist*)

lemma *capped_metric_mdist*:
mdist (*capped_metric* δ *m*) = (λ*x y*. *if* $\delta \leq 0$ *then mdist m x y else min* δ (*mdist m x y*))
apply (*simp add*: *capped_metric not_le*)
by (*smt* (*verit*, *del_insts*) *Metric_space.mdist_metric Metric_space_def Metric_space_mspace_mdist*)

lemma *mdist_capped_le*: *mdist* (*capped_metric* δ *m*) *x y* $\leq$ *mdist m x y*
by (*simp add*: *capped_metric_mdist*)

lemma *mdist_capped*: $\delta > 0 \Longrightarrow$ *mdist* (*capped_metric* δ *m*) *x y* $\leq \delta$
by (*simp add*: *capped_metric_mdist*)

lemma *mball_of_capped_metric* [*simp*]:
assumes *x* $\in$ *mspace m r* $> \delta$ $\delta > 0$
shows *mball_of* (*capped_metric* δ *m*) *x r* = *mspace m*
proof −
interpret *Metric_space mspace m mdist m*
by (*simp add*: *Metric_space_mspace_mdist*)
have *Metric_space.mball* (*mspace m*) (*mdist* (*capped_metric* δ *m*)) *x r* $\subseteq$ *mspace m*
by (*metis Metric_space.mball_subset_mspace Metric_space_mspace_mdist capped_metric_mspace*)
moreover have *mspace m* $\subseteq$ *Metric_space.mball* (*mspace m*) (*mdist* (*capped_metric* δ *m*)) *x r*
by (*smt* (*verit*) *Metric_space.in_mball Metric_space_mspace_mdist assms capped_metric_mspace mdist_capped subset_eq*)
ultimately show *?thesis*
by (*simp add*: *mball_of_def*)
qed

lemma *Metric_space_capped_dist*[*simp*]:
Metric_space (*mspace m*) (*Metric_space.capped_dist* (*mdist m*) δ)
using *Metric_space.capped_dist Metric_space_mspace_mdist* **by** *blast*

lemma *mtopology_capped_metric*:
mtopology_of(*capped_metric* δ *m*) = *mtopology_of m*
proof (*cases* $\delta > 0$)
case *True*
interpret *Metric_space mspace m mdist m*
by (*simp add*: *Metric_space_mspace_mdist*)

```
  interpret Cap: Metric_space mspace m mdist (capped_metric δ m)
    by (metis Metric_space_mspace_mdist capped_metric_mspace)
  show ?thesis
    unfolding topology_eq
  proof
    fix S
    show openin (mtopology_of (capped_metric δ m)) S = openin (mtopology_of
m) S
    proof (cases S ⊆ mspace m)
      case True
      have mball x r ⊆ Cap.mball x r for x r
       by (smt (verit, ccfv_SIG) Cap.in_mball in_mball mdist_capped_le subsetI)
      moreover have ∃r>0. Cap.mball x r ⊆ S if r>0 x ∈ S and r: mball x r ⊆
S for r x
      proof (intro exI conjI)
        show min (δ/2) r > 0
          using ‹r>0› ‹δ>0› by force
        show Cap.mball x (min (δ/2) r) ⊆ S
          using that
              by clarsimp (smt (verit) capped_metric_mdist field_sum_of_halves
in_mball subsetD)
      qed
      ultimately have (∃r>0. Cap.mball x r ⊆ S) = (∃r>0. mball x r ⊆ S) if x
∈ S for x
        by (meson subset_trans that)
      then show ?thesis
       by (simp add: mtopology_of_def openin_mtopology Cap.openin_mtopology)
    qed (simp add: openin_closedin_eq)
  qed
qed (simp add: capped_metric)
```

Might have been easier to prove this within the locale to start with (using Self)

```
lemma (in Metric_space) mtopology_capped_metric:
  Metric_space.mtopology M (capped_dist δ) = mtopology
  using mtopology_capped_metric [of δ metric(M,d)]
  by (simp add: Metric_space.mtopology_of capped_dist capped_metric_def)

lemma (in Metric_space) MCauchy_capped_metric:
  Metric_space.MCauchy M (capped_dist δ) σ ⟷ MCauchy σ
proof (cases δ > 0)
  case True
  interpret Cap: Metric_space M capped_dist δ
    by (simp add: capped_dist)
  show ?thesis
  proof
    assume σ: Cap.MCauchy σ
    show MCauchy σ
      unfolding MCauchy_def
```

proof (*intro conjI strip*)
show *range* $\sigma \subseteq M$
using *Cap.MCauchy_def* σ **by** *presburger*
fix ε :: *real*
assume $\varepsilon > 0$
with *True* σ
obtain N **where** $\forall\, n\ n'.\ N \leq n \longrightarrow N \leq n' \longrightarrow$ *capped_dist* δ $(\sigma\ n)$ $(\sigma\ n')$ $< min\ \delta\ \varepsilon$
unfolding *Cap.MCauchy_def* **by** (*metis min_less_iff_conj*)
with *True* **show** $\exists N.\ \forall\, n\ n'.\ N \leq n \longrightarrow N \leq n' \longrightarrow d\ (\sigma\ n)\ (\sigma\ n') < \varepsilon$
by (*force simp*: *capped_dist_def*)
qed
next
assume *MCauchy* σ
then show *Cap.MCauchy* σ
unfolding *MCauchy_def Cap.MCauchy_def* **by** (*force simp*: *capped_dist_def*)
qed
qed (*simp add*: *capped_dist_def*)

lemma (**in** *Metric_space*) *mcomplete_capped_metric*:
Metric_space.mcomplete M (*capped_dist* δ) $\longleftrightarrow$ *mcomplete*
by (*simp add*: *MCauchy_capped_metric Metric_space.mcomplete_def capped_dist mtopology_capped_metric mcomplete_def*)

lemma *bounded_equivalent_metric*:
assumes $\delta > 0$
obtains m' **where** *mspace* m' = *mspace m mtopology_of* m' = *mtopology_of m* $\bigwedge x\ y.$ *mdist* $m'\ x\ y < \delta$
proof
let *?m* = *capped_metric* $(\delta/2)$ *m*
fix *x y*
show *mdist ?m x y* $< \delta$
by (*smt* (*verit*, *best*) *assms field_sum_of_halves mdist_capped*)
qed (*auto simp*: *mtopology_capped_metric*)

A technical lemma needed below

lemma *Sup_metric_cartesian_product*:
fixes *I m*
defines $S \equiv$ *PiE I* (*mspace* $\circ$ *m*)
defines $D \equiv \lambda x\ y.$ *if* $x \in S \wedge y \in S$ *then SUP* $i{\in}I.$ *mdist* $(m\ i)$ $(x\ i)$ $(y\ i)$ *else 0*
defines $m' \equiv$ *metric*(S,D)
assumes $I \neq \{\}$
and *c*: $\bigwedge i\ x\ y.\ [\![i \in I;\ x \in mspace(m\ i);\ y \in mspace(m\ i)]\!] \Longrightarrow$ *mdist* $(m\ i)\ x\ y \leq c$
shows *Metric_space S D*
and $\forall\, x \in S.\ \forall\, y \in S.\ \forall\, b.\ D\ x\ y \leq b \longleftrightarrow (\forall\, i \in I.$ *mdist* $(m\ i)\ (x\ i)\ (y\ i) \leq b)$ (**is** *?the2*)
proof −

```
  have bdd: bdd_above ((λi. mdist (m i) (x i) (y i)) ' I)
    if x ∈ S y ∈ S for x y
    using c that by (force simp: S_def bdd_above_def)
  have D_iff: D x y ≤ b ⟷ (∀ i ∈ I. mdist (m i) (x i) (y i) ≤ b)
    if x ∈ S y ∈ S for x y b
    using that ‹I ≠ {}› by (simp add: D_def PiE_iff cSup_le_iff bdd)
  show Metric_space S D
  proof
    fix x y
    show D0: 0 ≤ D x y
      using bdd
      apply (simp add: D_def)
     by (meson ‹I ≠ {}› cSUP_upper dual_order.trans ex_in_conv mdist_nonneg)
    show D x y = D y x
      by (simp add: D_def mdist_commute)
    assume x ∈ S and y ∈ S
    then
    have D x y = 0 ⟷ (∀ i∈I. mdist (m i) (x i) (y i) = 0)
      using D0 D_iff [of x y 0] nle_le by fastforce
    also have ... ⟷ x = y
      using ‹x ∈ S› ‹y ∈ S› by (fastforce simp: S_def PiE_iff extensional_def)
    finally show (D x y = 0) ⟷ (x = y) .
    fix z
    assume z ∈ S
    have mdist (m i) (x i) (z i) ≤ D x y + D y z if i ∈ I for i
    proof −
      have mdist (m i) (x i) (z i) ≤ mdist (m i) (x i) (y i) + mdist (m i) (y i) (z
i)
       by (metis PiE_E S_def ‹x ∈ S› ‹y ∈ S› ‹z ∈ S› comp_apply mdist_triangle
that)
      also have ... ≤ D x y + D y z
       using ‹x ∈ S› ‹y ∈ S› ‹z ∈ S› by (meson D_iff add_mono order_refl that)
      finally show ?thesis .
    qed
    then show D x z ≤ D x y + D y z
      by (simp add: D_iff ‹x ∈ S› ‹z ∈ S›)
  qed
  then interpret Metric_space S D .
  show ?the2
  proof (intro strip)
    show (D x y ≤ b) = (∀ i∈I. mdist (m i) (x i) (y i) ≤ b)
      if x ∈ S and y ∈ S for x y b
      using that by (simp add: D_iff m'_def)
  qed
qed

lemma metrizable_topology_A:
  assumes metrizable_space (product_topology X I)
  shows (product_topology X I) = trivial_topology ∨ (∀ i ∈ I. metrizable_space
```

(*X i*))
by (*meson assms metrizable_space_retraction_map_image retraction_map_product_projection*)

lemma *metrizable_topology_C*:
assumes *completely_metrizable_space* (*product_topology X I*)
shows (*product_topology X I*) = *trivial_topology* ∨ (∀ *i* ∈ *I*. *completely_metrizable_space* (*X i*))
by (*meson assms completely_metrizable_space_retraction_map_image retraction_map_product_projection*)

lemma *metrizable_topology_B*:
fixes *a X I*
defines *L* ≡ {*i* ∈ *I*. ∄ *a*. *topspace* (*X i*) ⊆ {*a*}}
assumes *topspace* (*product_topology X I*) ≠ {}
and *met*: *metrizable_space* (*product_topology X I*)
and ⋀*i*. *i* ∈ *I* ⟹ *metrizable_space* (*X i*)
shows *countable L*
proof −
have ⋀*i*. ∃ *p q*. *i* ∈ *L* ⟶ *p* ∈ *topspace*(*X i*) ∧ *q* ∈ *topspace*(*X i*) ∧ *p* ≠ *q*
unfolding *L_def* **by** *blast*
then obtain φ ψ **where** φ: ⋀*i*. *i* ∈ *L* ⟹ φ *i* ∈ *topspace*(*X i*) ∧ ψ *i* ∈ *topspace*(*X i*) ∧ φ *i* ≠ ψ *i*
by *metis*
obtain *z* **where** *z*: *z* ∈ (Π$_E$ *i*∈*I*. *topspace* (*X i*))
using *assms*(*2*) **by** *fastforce*
define *p* **where** *p* ≡ λ*i*. *if i* ∈ *L then* φ *i else z i*
define *q* **where** *q* ≡ λ*i j*. *if j* = *i then* ψ *i else p j*
have *p*: *p* ∈ *topspace*(*product_topology X I*)
using *z* φ **by** (*auto simp*: *p_def L_def*)
then have *q*: ⋀*i*. *i* ∈ *L* ⟹ *q i* ∈ *topspace* (*product_topology X I*)
by (*auto simp*: *L_def q_def* φ)
have *fin*: *finite* {*i* ∈ *L*. *q i* ∉ *U*} **if** *U*: *openin* (*product_topology X I*) *U p* ∈ *U* **for** *U*
proof −
obtain *V* **where** *V*: *finite* {*i* ∈ *I*. *V i* ≠ *topspace* (*X i*)} (∀ *i*∈*I*. *openin* (*X i*) (*V i*)) *p* ∈ *Pi*$_E$ *I V Pi*$_E$ *I V* ⊆ *U*
using *U* **by** (*force simp*: *openin_product_topology_alt*)
moreover
have *V x* ≠ *topspace* (*X x*) **if** *x* ∈ *L* **and** *q x* ∉ *U* **for** *x*
using *that V q*
by (*smt* (*verit*, *del_insts*) *PiE_iff q_def subset_eq topspace_product_topology*)
then have {*i* ∈ *L*. *q i* ∉ *U*} ⊆ {*i* ∈ *I*. *V i* ≠ *topspace* (*X i*)}
by (*force simp*: *L_def*)
ultimately show *?thesis*
by (*meson finite_subset*)
qed
obtain *M d* **where** *Metric_space M d* **and** *XI*: *product_topology X I* = *Metric_space.mtopology M d*
using *met metrizable_space_def* **by** *blast*

```
  then interpret Metric_space M d
    by blast
  define C where C ≡ ⋃ n::nat. {i ∈ L. q i ∉ mball p (inverse (Suc n))}
  have finite {i ∈ L. q i ∉ mball p (inverse (real (Suc n)))} for n
    using XI p  by (intro fin; force)
  then have countable C
    unfolding C_def
    by (meson countableI_type countable_UN countable_finite)
  moreover have L ⊆ C
  proof (clarsimp simp: C_def)
    fix i
    assume i ∈ L and q i ∈ M and p ∈ M
    then show ∃ n. ¬ d p (q i) < inverse (1 + real n)
      using reals_Archimedean [of d p (q i)]
     by (metis φ mdist_pos_eq not_less_iff_gr_or_eq of_nat_Suc p_def q_def)
  qed
  ultimately show ?thesis
    using countable_subset by blast
qed

lemma metrizable_topology_DD:
  assumes topspace (product_topology X I) ≠ {}
    and co: countable {i ∈ I. ∄ a. topspace (X i) ⊆ {a}}
    and m: ⋀i. i ∈ I ⟹ X i = mtopology_of (m i)
 obtains M d where Metric_space M d product_topology X I = Metric_space.mtopology
M d
                  (⋀i. i ∈ I ⟹ mcomplete_of (m i)) ⟹ Metric_space.mcomplete
M d
proof (cases I = {})
  case True
  then show ?thesis
   by (metis discrete_metric.mcomplete_discrete_metric discrete_metric.mtopology_discrete_metric
metric_M_dd product_topology_empty_discrete that)
next
  case False
  obtain nk and C:: nat set where nk: {i ∈ I. ∄ a. topspace (X i) ⊆ {a}} = nk
` C and inj_on nk C
    using co by (force simp: countable_as_injective_image_subset)
  then obtain kn where kn: ⋀w. w ∈ C ⟹ kn (nk w) = w
    by (metis inv_into_f_f)
  define cm where cm ≡ λi. capped_metric (inverse(Suc(kn i))) (m i)
  have mspace_cm: mspace (cm i) = mspace (m i) for i
    by (simp add: cm_def)
  have c1: ⋀i x y. mdist (cm i) x y ≤ 1
    by (simp add: cm_def capped_metric_mdist min_le_iff_disj divide_simps)
  then have bdd: bdd_above ((λi. mdist (cm i) (x i) (y i)) ` I) for x y
    by (meson bdd_above.I2)
  define M where M ≡ Pi_E I (mspace ∘ cm)
  define d where d ≡ λx y. if x ∈ M ∧ y ∈ M then SUP i∈I. mdist (cm i) (x i)
```

```
(y i) else 0

  have d_le1: d x y ≤ 1 for x y
    using ‹I ≠ {}› c1 by (simp add: d_def bdd cSup_le_iff)
  with ‹I ≠ {}› Sup_metric_cartesian_product [of I cm]
  have Metric_space M d
    and ∗: ∀ x∈M. ∀ y∈M. ∀ b. (d x y ≤ b) ⟷ (∀ i∈I. mdist (cm i) (x i) (y i) ≤
b)
    by (auto simp: False bdd M_def d_def cSUP_le_iff intro: c1)
  then interpret Metric_space M d
    by metis
  have le_d: mdist (cm i) (x i) (y i) ≤ d x y if i ∈ I x ∈ M y ∈ M for i x y
    using ∗ that by blast
  have product_m: PiE I (λi. mspace (m i)) = topspace(product_topology X I)
    using m by force

  define m' where m' = metric (M,d)
  define J where J ≡ λU. {i ∈ I. U i ≠ topspace (X i)}
  have 1: ∃ U. finite (J U) ∧ (∀ i∈I. openin (X i) (U i)) ∧ x ∈ Pi_E I U ∧ Pi_E I
U ⊆ mball z r
    if x ∈ M z ∈ M and r: 0 < r d z x < r for x z r
  proof −
    have x: ⋀i. i ∈ I ⟹ x i ∈ topspace(X i)
      using M_def m mspace_cm that(1) by auto
    have z: ⋀i. i ∈ I ⟹ z i ∈ topspace(X i)
      using M_def m mspace_cm that(2) by auto
    obtain R where 0 < R d z x < R R < r
      using r dense by (smt (verit, ccfv_threshold))
    define U where U ≡ λi. if R ≤ inverse(Suc(kn i)) then mball_of (m i) (z i)
R else topspace(X i)
    show ?thesis
    proof (intro exI conjI)
      obtain n where n: real n ∗ R > 1
        using ‹0 < R› ex_less_of_nat_mult by blast
      have finite (nk ‘ (C ∩ {..n}))
        by force
      moreover
      have ∃ m. m ∈ C ∧ m ≤ n ∧ i = nk m
        if R: R ≤ inverse (1 + real (kn i)) and i ∈ I
          and neq: mball_of (m i) (z i) R ≠ topspace (X i) for i
      proof −
        interpret MI: Metric_space mspace (m i) mdist (m i)
          by auto
        have MI.mball (z i) R ≠ topspace (X i)
          by (metis mball_of_def neq)
        then have ∄ a. topspace (X i) ⊆ {a}
          using ‹0 < R› m subset_antisym ‹i ∈ I› z by fastforce
        then have i ∈ nk ‘ C
          using nk ‹i ∈ I› by auto
```

```
      then show ?thesis
       by (smt (verit, ccfv_SIG) R ‹0 < R› image_iff kn lift_Suc_mono_less_iff
mult_mono n not_le_imp_less of_nat_0_le_iff of_nat_Suc right_inverse)
    qed
    then have J U ⊆ nk ` (C ∩ {..n})
      by (auto simp: image_iff Bex_def J_def U_def split: if_split_asm)
    ultimately show finite (J U)
      using finite_subset by blast
    show ∀ i∈I. openin (X i) (U i)
       by (simp add: Metric_space.openin_mball U_def mball_of_def mtopol-
ogy_of_def m)
    have xin: x ∈ Pi_E I (topspace ∘ X)
      using M_def ‹x ∈ M› x by auto
    moreover
    have ⋀i. ⟦i ∈ I; R ≤ inverse (1 + real (kn i))⟧ ⟹ mdist (m i) (z i) (x i)
< R
      by (smt (verit, ccfv_SIG) ‹d z x < R› capped_metric_mdist cm_def le_d
of_nat_Suc that)
    ultimately show x ∈ Pi_E I U
      using m z by (auto simp: U_def PiE_iff)
    show Pi_E I U ⊆ mball z r
    proof
      fix y
      assume y: y ∈ Pi_E I U
      then have y ∈ M
        by (force simp: PiE_iff M_def U_def m mspace_cm split: if_split_asm)
      moreover
      have ∀ i∈I. mdist (cm i) (z i) (y i) ≤ R
          by (smt (verit) PiE_mem U_def cm_def in_mball_of inverse_Suc
mdist_capped mdist_capped_le y)
      then have d z y ≤ R
        by (simp add: ‹y ∈ M› ‹z ∈ M› *)
      ultimately show y ∈ mball z r
        using ‹R < r› ‹z ∈ M› by force
    qed
  qed
  have 2: ∃ r>0. mball x r ⊆ S
    if finite (J U) and x: x ∈ Pi_E I U and S: Pi_E I U ⊆ S
      and U: ⋀i. i∈I ⟹ openin (X i) (U i)
      and x ∈ S for U S x
  proof −
    { fix i
      assume i ∈ J U
      then have i ∈ I
        by (auto simp: J_def)
      then have openin (mtopology_of (m i)) (U i)
        using U m by force
      then have openin (mtopology_of (cm i)) (U i)
```

by (*simp add*: *Abstract_Metric_Spaces.mtopology_capped_metric cm_def*)
then have $\exists r{>}0.$ *mball_of* (*cm i*) (*x i*) *r* $\subseteq$ *U i*
using *x*
by (*simp add*: *Metric_space.openin_mtopology PiE_mem* ‹*i* $\in$ *I*› *mball_of_def mtopology_of_def*)
}
then obtain *rf* **where** *rf*: $\bigwedge j.$ *j* $\in$ *J U* $\Longrightarrow$ *rf j* $>0 \wedge$ *mball_of* (*cm j*) (*x j*) (*rf j*) $\subseteq$ *U j*
by *metis*
define *r* **where** *r* $\equiv$ *Min* (*insert 1* (*rf* ‘ *J U*))
show *?thesis*
proof (*intro exI conjI*)
show *r* $>$ *0*
by (*simp add*: ‹*finite* (*J U*)› *r_def rf*)
have *r* [*simp*]: $\bigwedge j.$ *j* $\in$ *J U* $\Longrightarrow$ *r* $\leq$ *rf j r* $\leq$ *1*
by (*auto simp*: *r_def that*(*1*))
have $*$: *mball_of* (*cm i*) (*x i*) *r* $\subseteq$ *U i* **if** *i* $\in$ *I* **for** *i*
proof (*cases i* $\in$ *J U*)
case *True*
with *r* **show** *?thesis*
by (*smt* (*verit*) *Metric_space.in_mball Metric_space_mspace_mdist mball_of_def rf subset_eq*)
next
case *False*
then show *?thesis*
by (*simp add*: *J_def cm_def m subset_eq that*)
qed
show *mball x r* $\subseteq$ *S*
by (*smt* (*verit*) *x* $*$ *in_mball_of M_def Metric_space.in_mball Metric_space_axioms PiE_iff le_d o_apply subset_eq S*)
qed
qed
have *3*: *x* $\in$ *M*
if §: $\bigwedge x.$ *x*$\in$*S* $\Longrightarrow \exists$ *U*. *finite* (*J U*) $\wedge$ ($\forall$ *i*$\in$*I*. *openin* (*X i*) (*U i*)) $\wedge$ *x* $\in$ *Pi*$_E$ *I U* $\wedge$ *Pi*$_E$ *I U* $\subseteq$ *S*
and *x* $\in$ *S* **for** *S x*
using § [*OF* ‹*x* $\in$ *S*›] *m openin_subset* **by** (*fastforce simp*: *M_def PiE_iff cm_def*)
show *thesis*
proof
show *Metric_space M d*
using *Metric_space_axioms* **by** *blast*
show *eq*: *product_topology X I* = *Metric_space.mtopology M d*
unfolding *topology_eq openin_mtopology openin_product_topology_alt*
using *J_def 1 2 3 subset_iff zero* **by** (*smt* (*verit*, *ccfv_threshold*))
show *mcomplete* **if** $\bigwedge i.$ *i* $\in$ *I* $\Longrightarrow$ *mcomplete_of* (*m i*)
unfolding *mcomplete_def*
proof (*intro strip*)
fix σ

```
      assume σ: MCauchy σ
      have ∃ y. i ∈ I ⟶ limitin (X i) (λn. σ n i) y sequentially for i
      proof (cases i ∈ I)
        case True
        interpret MI: Metric_space mspace (m i) mdist (m i)
          by auto
        have ⋀σ. MI.MCauchy σ ⟶ (∃ x. limitin MI.mtopology σ x sequentially)
          by (meson MI.mcomplete_def True mcomplete_of_def that)
        moreover have MI.MCauchy (λn. σ n i)
          unfolding MI.MCauchy_def
        proof (intro conjI strip)
          show range (λn. σ n i) ⊆ mspace (m i)
              by (smt (verit, ccfv_threshold) MCauchy_def PiE_iff True σ eq im-
age_subset_iff m topspace_mtopology topspace_mtopology_of topspace_product_topology)
          fix ε::real
          define r where r ≡ min ε (inverse(Suc (kn i)))
          assume ε > 0
          then have r > 0
            by (simp add: r_def)
          then obtain N where N: ⋀n n'. N ≤ n ∧ N ≤ n' ⟹ d (σ n) (σ n') <
r
            using σ unfolding MCauchy_def by meson
          show ∃ N. ∀ n n'. N ≤ n ⟶ N ≤ n' ⟶ mdist (m i) (σ n i) (σ n' i) < ε
          proof (intro strip exI)
            fix n n'
            assume N ≤ n and N ≤ n'
            then have mdist (cm i) (σ n i) (σ n' i) < r
              using ∗
               by (smt (verit) Metric_space.MCauchy_def Metric_space_axioms N
True σ rangeI subsetD)
            then
            show mdist (m i) (σ n i) (σ n' i) < ε
              unfolding cm_def r_def
              by (smt (verit, ccfv_SIG) capped_metric_mdist)
          qed
        qed
        ultimately show ?thesis
          by (simp add: m mtopology_of_def)
      qed auto
    then obtain y where ⋀i. i ∈ I ⟹ limitin (X i) (λn. σ n i) (y i) sequentially
        by metis
      with σ show ∃ x. limitin mtopology σ x sequentially
        apply (rule_tac x=λi∈I. y i in exI)
        apply (simp add: MCauchy_def limitin_componentwise flip: eq)
       by (metis eq eventually_at_top_linorder range_subsetD topspace_mtopology
topspace_product_topology)
    qed
  qed
qed
```

lemma *metrizable_topology_D*:
assumes *topspace* (*product_topology X I*) ≠ {}
and *co*: *countable* {*i* ∈ *I*. ∄*a*. *topspace* (*X i*) ⊆ {*a*}}
and *met*: ⋀*i*. *i* ∈ *I* ⟹ *metrizable_space* (*X i*)
shows *metrizable_space* (*product_topology X I*)
proof −
have ⋀*i*. *i* ∈ *I* ⟹ ∃*m*. *X i* = *mtopology_of m*
by (*metis Metric_space.mtopology_of met metrizable_space_def*)
then obtain *m* **where** *m*: ⋀*i*. *i* ∈ *I* ⟹ *X i* = *mtopology_of* (*m i*)
by *metis*
then show *?thesis*
using *metrizable_topology_DD* [*of X I m*] *assms* **by** (*force simp*: *metrizable_space_def*)
qed

lemma *metrizable_topology_E*:
assumes *topspace* (*product_topology X I*) ≠ {}
and *countable* {*i* ∈ *I*. ∄*a*. *topspace* (*X i*) ⊆ {*a*}}
and *met*: ⋀*i*. *i* ∈ *I* ⟹ *completely_metrizable_space* (*X i*)
shows *completely_metrizable_space* (*product_topology X I*)
proof −
have ⋀*i*. *i* ∈ *I* ⟹ ∃*m*. *mcomplete_of m* ∧ *X i* = *mtopology_of m*
using *met Metric_space.mtopology_of Metric_space.mcomplete_of* **unfolding** *completely_metrizable_space_def*
by *metis*
then obtain *m* **where** ⋀*i*. *i* ∈ *I* ⟹ *mcomplete_of* (*m i*) ∧ *X i* = *mtopology_of* (*m i*)
by *metis*
then show *?thesis*
using *metrizable_topology_DD* [*of X I m*] *assms* **unfolding** *metrizable_space_def*
by (*metis* (*full_types*) *completely_metrizable_space_def*)
qed

proposition *metrizable_space_product_topology*:
metrizable_space (*product_topology X I*) ⟷
(*product_topology X I*) = *trivial_topology* ∨
countable {*i* ∈ *I*. ¬ (∃*a*. *topspace*(*X i*) ⊆ {*a*})} ∧
(∀*i* ∈ *I*. *metrizable_space* (*X i*))
by (*metis* (*mono_tags*, *lifting*) *empty_metrizable_space metrizable_topology_A metrizable_topology_B metrizable_topology_D subtopology_eq_discrete_topology_empty*)

proposition *completely_metrizable_space_product_topology*:
completely_metrizable_space (*product_topology X I*) ⟷
(*product_topology X I*) = *trivial_topology* ∨
countable {*i* ∈ *I*. ¬ (∃*a*. *topspace*(*X i*) ⊆ {*a*})} ∧
(∀*i* ∈ *I*. *completely_metrizable_space* (*X i*))

by (*smt* (*verit*, *del_insts*) *Collect_cong completely_metrizable_imp_metrizable_space empty_completely_metrizable_space metrizable_topology_B metrizable_topology_C metrizable_topology_E subtopology_eq_discrete_topology_empty*)

lemma *completely_metrizable_Euclidean_space*:
completely_metrizable_space(*Euclidean_space n*)
unfolding *Euclidean_space_def*
proof (*rule completely_metrizable_space_closedin*)
show *completely_metrizable_space* (*powertop_real* (*UNIV*::*nat set*))
by (*simp add*: *completely_metrizable_space_product_topology completely_metrizable_space_euclidean*
show *closedin* (*powertop_real UNIV*) {*x*. ∀ *i*≥*n*. *x i* = *0*}
using *closedin_Euclidean_space topspace_Euclidean_space* **by** *auto*
qed

lemma *metrizable_Euclidean_space*:
metrizable_space(*Euclidean_space n*)
by (*simp add*: *completely_metrizable_Euclidean_space completely_metrizable_imp_metrizable_space*)

lemma *locally_connected_Euclidean_space*:
locally_connected_space(*Euclidean_space n*)
by (*simp add*: *locally_path_connected_Euclidean_space locally_path_connected_imp_locally_connecte*

end

5.8 Infinite sums

In this theory, we introduce the definition of infinite sums, i.e., sums ranging over an infinite, potentially uncountable index set with no particular ordering. (This is different from series. Those are sums indexed by natural numbers, and the order of the index set matters.)

Our definition is quite standard: $s := \sum_{x\in A} f(x)$ is the limit of finite sums $s_F := \sum_{x\in F} f(x)$ for increasing F. That is, s is the limit of the net s_F where F are finite subsets of A ordered by inclusion. We believe that this is the standard definition for such sums. See, e.g., Definition 4.11 in [2]. This definition is quite general: it is well-defined whenever f takes values in some commutative monoid endowed with a Hausdorff topology. (Examples are reals, complex numbers, normed vector spaces, and more.)

theory *Infinite_Sum*
imports
Elementary_Topology
HOL−Library.Extended_Nonnegative_Real
HOL−Library.Complex_Order
begin

5.8.1 Definition and syntax

definition *HAS_SUM* :: ‹('a ⇒ 'b :: {*comm_monoid_add*, *topological_space*}) ⇒ 'a *set* ⇒ 'b ⇒ *bool*›
  where *has_sum_def*: ‹*HAS_SUM f A x* ≡ (*sum f* ⟶ *x*) (*finite_subsets_at_top A*)›

abbreviation *has_sum* (**infixr** *has'_sum 46*) **where**
  (*f has_sum S*) *A* ≡ *HAS_SUM f A S*

definition *summable_on* :: ('a ⇒ 'b::{*comm_monoid_add*, *topological_space*}) ⇒ 'a *set* ⇒ *bool* (**infixr** *summable'_on 46*) **where**
  f summable_on A ≡ (∃ *x*. (*f has_sum x*) *A*)

definition *infsum* :: ('a ⇒ 'b::{*comm_monoid_add*,*t2_space*}) ⇒ 'a *set* ⇒ 'b **where**
  infsum f A = (*if f summable_on A then Lim* (*finite_subsets_at_top A*) (*sum f*) *else 0*)

abbreviation *abs_summable_on* :: ('a ⇒ 'b::*real_normed_vector*) ⇒ 'a *set* ⇒ *bool* (**infixr** *abs'_summable'_on 46*) **where**
  f abs_summable_on A ≡ (λ*x*. *norm* (*f x*)) *summable_on A*

syntax (*ASCII*)
  _infsum :: *pttrn* ⇒ 'a *set* ⇒ 'b ⇒ 'b::*topological_comm_monoid_add* ((*3INFSUM* (_/:_)./ _) [*0*, *51*, *10*] *10*)
syntax
  _infsum :: *pttrn* ⇒ 'a *set* ⇒ 'b ⇒ 'b::*topological_comm_monoid_add* ((*2*∑ ∞(_/∈_)./ _) [*0*, *51*, *10*] *10*)
translations — Beware of argument permutation!
  ∑ ∞*i*∈*A*. *b* ⇌ *CONST infsum* (λ*i*. *b*) *A*

syntax (*ASCII*)
  _univinfsum :: *pttrn* ⇒ 'a ⇒ 'a ((*3INFSUM* _./ _) [*0*, *10*] *10*)
syntax
  univinfsum :: *pttrn* ⇒ 'a ⇒ 'a ((*2*∑ ∞./ _) [*0*, *10*] *10*)
translations
  ∑ ∞*x*. *t* ⇌ *CONST infsum* (λ*x*. *t*) (*CONST UNIV*)

syntax (*ASCII*)
  _qinfsum :: *pttrn* ⇒ *bool* ⇒ 'a ⇒ 'a ((*3INFSUM* _ |/ _./ _) [*0*, *0*, *10*] *10*)
syntax
  qinfsum :: *pttrn* ⇒ *bool* ⇒ 'a ⇒ 'a ((*2*∑ ∞ | (_)./ _) [*0*, *0*, *10*] *10*)
translations
  ∑ ∞*x*|*P*. *t* => *CONST infsum* (λ*x*. *t*) {*x*. *P*}

print_translation ‹
let
  fun sum_tr' [*Abs* (*x*, *Tx*, *t*), *Const* (@{*const_syntax Collect*}, _) $ *Abs* (*y*, *Ty*, *P*)] =

```
        if x <> y then raise Match
        else
          let
            val x' = Syntax_Trans.mark_bound_body (x, Tx);
            val t' = subst_bound (x', t);
            val P' = subst_bound (x', P);
          in
         Syntax.const @{syntax_const _qinfsum} $ Syntax_Trans.mark_bound_abs
(x, Tx) $ P' $ t'
          end
    | sum_tr' _ = raise Match;
in [(@{const_syntax infsum}, K sum_tr')] end
›
```

5.8.2 General properties

lemma *infsumI*:
fixes *f g* :: ‹'*a* ⇒ '*b*::{*comm_monoid_add*, *t2_space*}›
assumes ‹(*f has_sum x*) *A*›
shows ‹*infsum f A* = *x*›
by (*metis assms finite_subsets_at_top_neq_bot infsum_def summable_on_def has_sum_def tendsto_Lim*)

lemma *infsum_eqI*:
fixes *f g* :: ‹'*a* ⇒ '*b*::{*comm_monoid_add*, *t2_space*}›
assumes ‹*x* = *y*›
assumes ‹(*f has_sum x*) *A*›
assumes ‹(*g has_sum y*) *B*›
shows ‹*infsum f A* = *infsum g B*›
using *assms infsumI* **by** *blast*

lemma *infsum_eqI'*:
fixes *f g* :: ‹'*a* ⇒ '*b*::{*comm_monoid_add*, *t2_space*}›
assumes ‹⋀*x*. (*f has_sum x*) *A* ⟷ (*g has_sum x*) *B*›
shows ‹*infsum f A* = *infsum g B*›
by (*metis assms infsum_def infsum_eqI summable_on_def*)

lemma *infsum_not_exists*:
fixes *f* :: ‹'*a* ⇒ '*b*::{*comm_monoid_add*, *t2_space*}›
assumes ‹¬ *f summable_on A*›
shows ‹*infsum f A* = *0*›
by (*simp add*: *assms infsum_def*)

lemma *summable_iff_has_sum_infsum*: *f summable_on A* ⟷ (*f has_sum* (*infsum f A*)) *A*
using *infsumI summable_on_def* **by** *blast*

lemma *has_sum_infsum*[*simp*]:
assumes ‹*f summable_on S*›

shows ‹(*f has_sum* (*infsum f S*)) *S*›
using *assms summable_iff_has_sum_infsum* **by** *blast*

lemma *has_sum_cong_neutral*:
fixes *f g* :: ‹'*a* ⇒ '*b*::{*comm_monoid_add*, *topological_space*}›
assumes ‹⋀*x*. *x*∈*T*−*S* ⟹ *g x* = *0*›
assumes ‹⋀*x*. *x*∈*S*−*T* ⟹ *f x* = *0*›
assumes ‹⋀*x*. *x*∈*S*∩*T* ⟹ *f x* = *g x*›
shows (*f has_sum x*) *S* ⟷ (*g has_sum x*) *T*
proof −
have ‹*eventually P* (*filtermap* (*sum f*) (*finite_subsets_at_top S*))
= *eventually P* (*filtermap* (*sum g*) (*finite_subsets_at_top T*))› **for** *P*
proof
assume ‹*eventually P* (*filtermap* (*sum f*) (*finite_subsets_at_top S*))›
then obtain *F0* **where** ‹*finite F0*› **and** ‹*F0* ⊆ *S*› **and** *F0_P*: ‹⋀*F*. *finite F*
⟹ *F* ⊆ *S* ⟹ *F* ⊇ *F0* ⟹ *P* (*sum f F*)›
by (*metis* (*no_types*, *lifting*) *eventually_filtermap eventually_finite_subsets_at_top*)
define *F0'* **where** ‹*F0'* = *F0* ∩ *T*›
have [*simp*]: ‹*finite F0'*› ‹*F0'* ⊆ *T*›
by (*simp_all add*: *F0'_def* ‹*finite F0*›)
have ‹*P* (*sum g F*)› **if** ‹*finite F*› ‹*F* ⊆ *T*› ‹*F* ⊇ *F0'*› **for** *F*
proof −
have ‹*P* (*sum f* ((*F*∩*S*) ∪ (*F0*∩*S*)))›
by (*intro F0_P*) (*use* ‹*F0* ⊆ *S*› ‹*finite F0*› *that* **in** *auto*)
also have ‹*sum f* ((*F*∩*S*) ∪ (*F0*∩*S*)) = *sum g F*›
by (*intro sum.mono_neutral_cong*) (*use that* ‹*finite F0*› *F0'_def assms* **in**
auto)
finally show *?thesis* **.**
qed
with ‹*F0'* ⊆ *T*› ‹*finite F0'*› **show** ‹*eventually P* (*filtermap* (*sum g*) (*finite_subsets_at_top*
T))›
by (*metis* (*no_types*, *lifting*) *eventually_filtermap eventually_finite_subsets_at_top*)
next
assume ‹*eventually P* (*filtermap* (*sum g*) (*finite_subsets_at_top T*))›
then obtain *F0* **where** ‹*finite F0*› **and** ‹*F0* ⊆ *T*› **and** *F0_P*: ‹⋀*F*. *finite F*
⟹ *F* ⊆ *T* ⟹ *F* ⊇ *F0* ⟹ *P* (*sum g F*)›
by (*metis* (*no_types*, *lifting*) *eventually_filtermap eventually_finite_subsets_at_top*)
define *F0'* **where** ‹*F0'* = *F0* ∩ *S*›
have [*simp*]: ‹*finite F0'*› ‹*F0'* ⊆ *S*›
by (*simp_all add*: *F0'_def* ‹*finite F0*›)
have ‹*P* (*sum f F*)› **if** ‹*finite F*› ‹*F* ⊆ *S*› ‹*F* ⊇ *F0'*› **for** *F*
proof −
have ‹*P* (*sum g* ((*F*∩*T*) ∪ (*F0*∩*T*)))›
by (*intro F0_P*) (*use* ‹*F0* ⊆ *T*› ‹*finite F0*› *that* **in** *auto*)
also have ‹*sum g* ((*F*∩*T*) ∪ (*F0*∩*T*)) = *sum f F*›
by (*intro sum.mono_neutral_cong*) (*use that* ‹*finite F0*› *F0'_def assms* **in**
auto)
finally show *?thesis* **.**
qed

with ‹*F0'* ⊆ *S*› ‹*finite F0'*› **show** ‹*eventually P* (*filtermap* (*sum f*) (*finite_subsets_at_top S*))›
by (*metis* (*no_types*, *lifting*) *eventually_filtermap eventually_finite_subsets_at_top*)
qed

then have *tendsto_x*: (*sum f* ⟶ *x*) (*finite_subsets_at_top S*) ⟷ (*sum g* ⟶ *x*) (*finite_subsets_at_top T*) **for** *x*
by (*simp add*: *le_filter_def filterlim_def*)

then show *?thesis*
by (*simp add*: *has_sum_def*)
qed

lemma *summable_on_cong_neutral*:
fixes *f g* :: ‹'*a* ⇒ '*b*::{*comm_monoid_add*, *topological_space*}›
assumes ‹⋀*x*. *x*∈*T*−*S* ⟹ *g x* = *0*›
assumes ‹⋀*x*. *x*∈*S*−*T* ⟹ *f x* = *0*›
assumes ‹⋀*x*. *x*∈*S*∩*T* ⟹ *f x* = *g x*›
shows *f summable_on S* ⟷ *g summable_on T*
using *has_sum_cong_neutral*[*of T S g f*, *OF assms*]
by (*simp add*: *summable_on_def*)

lemma *infsum_cong_neutral*:
fixes *f g* :: ‹'*a* ⇒ '*b*::{*comm_monoid_add*, *t2_space*}›
assumes ‹⋀*x*. *x*∈*T*−*S* ⟹ *g x* = *0*›
assumes ‹⋀*x*. *x*∈*S*−*T* ⟹ *f x* = *0*›
assumes ‹⋀*x*. *x*∈*S*∩*T* ⟹ *f x* = *g x*›
shows ‹*infsum f S* = *infsum g T*›
by (*smt* (*verit*, *best*) *assms has_sum_cong_neutral infsum_eqI'*)

lemma *has_sum_cong*:
assumes ⋀*x*. *x*∈*A* ⟹ *f x* = *g x*
shows (*f has_sum x*) *A* ⟷ (*g has_sum x*) *A*
using *assms* **by** (*intro has_sum_cong_neutral*) *auto*

lemma *summable_on_cong*:
assumes ⋀*x*. *x*∈*A* ⟹ *f x* = *g x*
shows *f summable_on A* ⟷ *g summable_on A*
by (*metis assms summable_on_def has_sum_cong*)

lemma *infsum_cong*:
assumes ⋀*x*. *x*∈*A* ⟹ *f x* = *g x*
shows *infsum f A* = *infsum g A*
using *assms infsum_eqI' has_sum_cong* **by** *blast*

lemma *summable_on_cofin_subset*:
fixes *f* :: '*a* ⇒ '*b*::*topological_ab_group_add*
assumes *f summable_on A* **and** [*simp*]: *finite F*
shows *f summable_on* (*A* − *F*)

proof −
from *assms*(*1*) **obtain** *x* **where** *lim_f*: (*sum f* ⟶ *x*) (*finite_subsets_at_top A*)
unfolding *summable_on_def has_sum_def* **by** *auto*
define *F'* **where** *F'* = *F*∩*A*
with *assms* **have** *finite F'* **and** *A*−*F* = *A*−*F'*
by *auto*
have *filtermap* ((∪)*F'*) (*finite_subsets_at_top* (*A*−*F*))
≤ *finite_subsets_at_top A*
proof (*rule filter_leI*)
fix *P* **assume** *eventually P* (*finite_subsets_at_top A*)
then obtain *X* **where** [*simp*]: *finite X* **and** *XA*: *X* ⊆ *A*
and *P*: ∀ *Y*. *finite Y* ∧ *X* ⊆ *Y* ∧ *Y* ⊆ *A* ⟶ *P Y*
unfolding *eventually_finite_subsets_at_top* **by** *auto*
define *X'* **where** *X'* = *X*−*F*
hence [*simp*]: *finite X'* **and** [*simp*]: *X'* ⊆ *A*−*F*
using *XA* **by** *auto*
hence *finite Y* ∧ *X'* ⊆ *Y* ∧ *Y* ⊆ *A* − *F* ⟶ *P* (*F'* ∪ *Y*) **for** *Y*
using *P XA* **unfolding** *X'_def* **using** *F'_def* ‹*finite F'*› **by** *blast*
thus *eventually P* (*filtermap* ((∪) *F'*) (*finite_subsets_at_top* (*A* − *F*)))
unfolding *eventually_filtermap eventually_finite_subsets_at_top*
by (*rule_tac x*=*X'* **in** *exI*, *simp*)
qed
with *lim_f* **have** (*sum f* ⟶ *x*) (*filtermap* ((∪)*F'*) (*finite_subsets_at_top* (*A*−*F*)))
using *tendsto_mono* **by** *blast*
have ((λ*G*. *sum f* (*F'* ∪ *G*)) ⟶ *x*) (*finite_subsets_at_top* (*A* − *F*))
if ((*sum f* ∘ (∪) *F'*) ⟶ *x*) (*finite_subsets_at_top* (*A* − *F*))
using *that* **unfolding** *o_def* **by** *auto*
hence ((λ*G*. *sum f* (*F'* ∪ *G*)) ⟶ *x*) (*finite_subsets_at_top* (*A*−*F*))
using *tendsto_compose_filtermap* [*symmetric*]
by (*simp add*: ‹(*sum f* ⟶ *x*) (*filtermap* ((∪) *F'*) (*finite_subsets_at_top* (*A* − *F*)))›
tendsto_compose_filtermap)
have ∀ *Y*. *finite Y* ∧ *Y* ⊆ *A* − *F* ⟶ *sum f* (*F'* ∪ *Y*) = *sum f F'* + *sum f Y*
by (*metis Diff_disjoint Int_Diff* ‹*A* − *F* = *A* − *F'*› ‹*finite F'*› *inf.orderE sum.union_disjoint*)
hence ∀ *F* *x* *in* *finite_subsets_at_top* (*A* − *F*). *sum f* (*F'* ∪ *x*) = *sum f F'* + *sum f x*
unfolding *eventually_finite_subsets_at_top*
using *exI* [**where** *x* = {}]
by (*simp add*: ‹⋀*P*. *P* {} ⟹ ∃ *x*. *P x*›)
hence ((λ*G*. *sum f F'* + *sum f G*) ⟶ *x*) (*finite_subsets_at_top* (*A*−*F*))
using *tendsto_cong* [*THEN iffD1* , *rotated*]
‹((λ*G*. *sum f* (*F'* ∪ *G*)) ⟶ *x*) (*finite_subsets_at_top* (*A* − *F*))› **by** *fastforce*
hence ((λ*G*. *sum f F'* + *sum f G*) ⟶ *sum f F'* + (*x*−*sum f F'*)) (*finite_subsets_at_top* (*A*−*F*))
by *simp*

```
  hence (sum f ⟶ x − sum f F') (finite_subsets_at_top (A−F))
    using tendsto_add_const_iff by blast
  thus f summable_on (A − F)
    unfolding summable_on_def has_sum_def by auto
qed

lemma
  fixes f :: 'a ⇒ 'b::{topological_ab_group_add}
  assumes ‹(f has_sum b) B› and ‹(f has_sum a) A› and AB: A ⊆ B
  shows has_sum_Diff: (f has_sum (b − a)) (B − A)
proof −
  have finite_subsets1:
   finite_subsets_at_top (B − A) ≤ filtermap (λF. F − A) (finite_subsets_at_top
B)
  proof (rule filter_leI)
    fix P assume eventually P (filtermap (λF. F − A) (finite_subsets_at_top B))
    then obtain X where finite X and X ⊆ B
      and P: finite Y ∧ X ⊆ Y ∧ Y ⊆ B ⟶ P (Y − A) for Y
      unfolding eventually_filtermap eventually_finite_subsets_at_top by auto

    hence finite (X−A) and X−A ⊆ B − A
      by auto
    moreover have finite Y ∧ X−A ⊆ Y ∧ Y ⊆ B − A ⟶ P Y for Y
      using P[where Y=Y∪X] ‹finite X› ‹X ⊆ B›
        by (metis Diff_subset Int_Diff Un_Diff finite_Un inf.orderE le_sup_iff
sup.orderE sup_ge2)
    ultimately show eventually P (finite_subsets_at_top (B − A))
      unfolding eventually_finite_subsets_at_top by meson
  qed
  have finite_subsets2:
    filtermap (λF. F ∩ A) (finite_subsets_at_top B) ≤ finite_subsets_at_top A
    apply (rule filter_leI)
    using assms unfolding eventually_filtermap eventually_finite_subsets_at_top
      by (metis Int_subset_iff finite_Int inf_le2 subset_trans)

  from assms(1) have limB: (sum f ⟶ b) (finite_subsets_at_top B)
    using has_sum_def by auto
  from assms(2) have limA: (sum f ⟶ a) (finite_subsets_at_top A)
    using has_sum_def by blast
  have ((λF. sum f (F∩A)) ⟶ a) (finite_subsets_at_top B)
  proof (subst asm_rl [of (λF. sum f (F∩A)) = sum f ∘ (λF. F∩A)])
    show (λF. sum f (F ∩ A)) = sum f ∘ (λF. F ∩ A)
      unfolding o_def by auto
    show ((sum f ∘ (λF. F ∩ A)) ⟶ a) (finite_subsets_at_top B)
      unfolding o_def
      using tendsto_compose_filtermap finite_subsets2 limA tendsto_mono
        ‹(λF. sum f (F ∩ A)) = sum f ∘ (λF. F ∩ A)› by fastforce
  qed
```

with *limB* **have** $((\lambda F.\ sum\ f\ F - sum\ f\ (F \cap A)) \longrightarrow b - a)$ (*finite_subsets_at_top* B)
using *tendsto_diff* **by** *blast*
have $sum\ f\ X - sum\ f\ (X \cap A) = sum\ f\ (X - A)$ **if** *finite* X **and** $X \subseteq B$ **for** $X :: 'a\ set$
using *that* **by** (*metis add_diff_cancel_left' sum.Int_Diff*)
hence $\forall_F\ x\ in\ finite\_subsets\_at\_top\ B.\ sum\ f\ x - sum\ f\ (x \cap A) = sum\ f\ (x - A)$
by (*rule eventually_finite_subsets_at_top_weakI*)
hence $((\lambda F.\ sum\ f\ (F - A)) \longrightarrow b - a)$ (*finite_subsets_at_top* B)
using *tendsto_cong* [*THEN iffD1* , *rotated*]
‹$((\lambda F.\ sum\ f\ F - sum\ f\ (F \cap A)) \longrightarrow b - a)$ (*finite_subsets_at_top* B)›
by *fastforce*
hence $(sum\ f \longrightarrow b - a)$ (*filtermap* $(\lambda F.\ F - A)$ (*finite_subsets_at_top* B))
by (*subst tendsto_compose_filtermap*[*symmetric*], *simp add*: *o_def*)
thus *?thesis*
using *finite_subsets1 has_sum_def tendsto_mono* **by** *blast*
qed

lemma
fixes $f :: 'a \Rightarrow 'b$::{*topological_ab_group_add*}
assumes f *summable_on* B **and** f *summable_on* A **and** $A \subseteq B$
shows *summable_on_Diff*: f *summable_on* $(B - A)$
by (*meson assms summable_on_def has_sum_Diff*)

lemma
fixes $f :: 'a \Rightarrow 'b$::{*topological_ab_group_add*,*t2_space*}
assumes f *summable_on* B **and** f *summable_on* A **and** AB: $A \subseteq B$
shows *infsum_Diff*: $infsum\ f\ (B - A) = infsum\ f\ B - infsum\ f\ A$
by (*metis AB assms has_sum_Diff infsumI summable_on_def*)

lemma *has_sum_mono_neutral*:
fixes $f :: 'a \Rightarrow 'b$::{*ordered_comm_monoid_add*,*linorder_topology*}

assumes ‹$(f\ has\_sum\ a)\ A$› **and** $(g\ has\_sum\ b)\ B$
assumes ‹$\bigwedge x.\ x \in A \cap B \Longrightarrow f\ x \leq g\ x$›
assumes ‹$\bigwedge x.\ x \in A - B \Longrightarrow f\ x \leq 0$›
assumes ‹$\bigwedge x.\ x \in B - A \Longrightarrow g\ x \geq 0$›
shows $a \leq b$
proof −
define $f'\ g'$ **where** ‹$f'\ x = (if\ x \in A\ then\ f\ x\ else\ 0)$› **and** ‹$g'\ x = (if\ x \in B\ then\ g\ x\ else\ 0)$› **for** x
have [*simp*]: ‹f *summable_on* A› ‹g *summable_on* B›
using *assms*(*1*,*2*) *summable_on_def* **by** *auto*
have ‹$(f'\ has\_sum\ a)\ (A \cup B)$›
by (*smt* (*verit*, *best*) *DiffE IntE Un_iff f'_def assms*(*1*) *has_sum_cong_neutral*)
then have *f'_lim*: ‹$(sum\ f' \longrightarrow a)$ (*finite_subsets_at_top* $(A \cup B)$)›
by (*meson has_sum_def*)

```
  have ‹(g' has_sum b) (A∪B)›
 by (smt (verit, best) DiffD1 DiffD2 IntE UnCI g'_def assms(2) has_sum_cong_neutral)
  then have g'_lim: ‹(sum g' ⟶ b) (finite_subsets_at_top (A∪B))›
    using has_sum_def by blast

  have ⋀X i. ⟦X ⊆ A ∪ B; i ∈ X⟧ ⟹ f' i ≤ g' i
    using assms by (auto simp: f'_def g'_def)
  then have ‹∀⇩F x in finite_subsets_at_top (A ∪ B). sum f' x ≤ sum g' x›
    by (intro eventually_finite_subsets_at_top_weakI sum_mono)
  then show ?thesis
    using f'_lim finite_subsets_at_top_neq_bot g'_lim tendsto_le by blast
qed

lemma infsum_mono_neutral:
  fixes f :: 'a⇒'b::{ordered_comm_monoid_add,linorder_topology}
  assumes f summable_on A and g summable_on B
  assumes ‹⋀x. x ∈ A∩B ⟹ f x ≤ g x›
  assumes ‹⋀x. x ∈ A−B ⟹ f x ≤ 0›
  assumes ‹⋀x. x ∈ B−A ⟹ g x ≥ 0›
  shows infsum f A ≤ infsum g B
  by (smt (verit, best) assms has_sum_infsum has_sum_mono_neutral)

lemma has_sum_mono:
  fixes f :: 'a⇒'b::{ordered_comm_monoid_add,linorder_topology}
  assumes (f has_sum x) A and (g has_sum y) A
  assumes ‹⋀x. x ∈ A ⟹ f x ≤ g x›
  shows x ≤ y
  using assms has_sum_mono_neutral by force

lemma infsum_mono:
  fixes f :: 'a⇒'b::{ordered_comm_monoid_add,linorder_topology}
  assumes f summable_on A and g summable_on A
  assumes ‹⋀x. x ∈ A ⟹ f x ≤ g x›
  shows infsum f A ≤ infsum g A
  by (meson assms has_sum_infsum has_sum_mono)

lemma has_sum_finite[simp]:
  assumes finite F
  shows (f has_sum (sum f F)) F
  using assms
 by (auto intro: tendsto_Lim simp: finite_subsets_at_top_finite infsum_def has_sum_def
principal_eq_bot_iff)

lemma summable_on_finite[simp]:
  fixes f :: ‹'a ⇒ 'b::{comm_monoid_add,topological_space}›
  assumes finite F
  shows f summable_on F
  using assms summable_on_def has_sum_finite by blast
```

lemma *infsum_finite*[*simp*]:
assumes *finite F*
shows *infsum f F = sum f F*
by (*simp add*: *assms infsumI*)

lemma *has_sum_finite_approximation*:
fixes *f* :: *'a* ⇒ *'b*::{*comm_monoid_add*,*metric_space*}
assumes (*f has_sum x*) *A* **and** *ε > 0*
shows ∃ *F*. *finite F* ∧ *F* ⊆ *A* ∧ *dist* (*sum f F*) *x* ≤ *ε*
proof −
have (*sum f* ⟶ *x*) (*finite_subsets_at_top A*)
by (*meson assms*(*1*) *has_sum_def*)
hence ∗: ∀ *F* *F in* (*finite_subsets_at_top A*). *dist* (*sum f F*) *x* < *ε*
using *assms*(*2*) **by** (*rule tendstoD*)
thus *?thesis*
unfolding *eventually_finite_subsets_at_top* **by** *fastforce*
qed

lemma *infsum_finite_approximation*:
fixes *f* :: *'a* ⇒ *'b*::{*comm_monoid_add*,*metric_space*}
assumes *f summable_on A* **and** *ε > 0*
shows ∃ *F*. *finite F* ∧ *F* ⊆ *A* ∧ *dist* (*sum f F*) (*infsum f A*) ≤ *ε*
proof −
from *assms* **have** (*f has_sum* (*infsum f A*)) *A*
by (*simp add*: *summable_iff_has_sum_infsum*)
from *this* **and** ‹*ε > 0*› **show** *?thesis*
by (*rule has_sum_finite_approximation*)
qed

lemma *abs_summable_summable*:
fixes *f* :: ‹*'a* ⇒ *'b* :: *banach*›
assumes ‹*f abs_summable_on A*›
shows ‹*f summable_on A*›
proof −
from *assms* **obtain** *L* **where** *lim*: ‹(*sum* (λ*x*. *norm* (*f x*)) ⟶ *L*) (*finite_subsets_at_top A*)›
unfolding *has_sum_def summable_on_def* **by** *blast*
then have ∗: ‹*cauchy_filter* (*filtermap* (*sum* (λ*x*. *norm* (*f x*))) (*finite_subsets_at_top A*))›
by (*auto intro*!: *nhds_imp_cauchy_filter simp*: *filterlim_def*)
have ‹∃ *P*. *eventually P* (*finite_subsets_at_top A*) ∧
(∀ *F F'*. *P F* ∧ *P F'* ⟶ *dist* (*sum f F*) (*sum f F'*) < *e*)› **if** ‹*e*>*0*› **for** *e*
proof −
define *d P* **where** ‹*d* = *e*/*4*› **and** ‹*P F* ⟷ *finite F* ∧ *F* ⊆ *A* ∧ *dist* (*sum* (λ*x*. *norm* (*f x*)) *F*) *L* < *d*› **for** *F*
then have ‹*d > 0*›
by (*simp add*: *d_def that*)
have *ev_P*: ‹*eventually P* (*finite_subsets_at_top A*)›
using *lim*

by (*auto simp add*: *P_def*[*abs_def*] ‹*0 < d*› *eventually_conj_iff eventually_finite_subsets_at_top_weakI tendsto_iff*)

moreover have ‹*dist* (*sum f F1*) (*sum f F2*) < *e*› **if** ‹*P F1*› **and** ‹*P F2*› **for** *F1 F2*
proof −
from *ev_P*
obtain *F'* **where** ‹*finite F'*› **and** ‹*F' ⊆ A*› **and** *P_sup_F'*: ‹*finite F* ∧ *F ⊇ F'* ∧ *F ⊆ A* ⟹ *P F*› **for** *F*
by *atomize_elim* (*simp add*: *eventually_finite_subsets_at_top*)
define *F* **where** ‹*F* = *F'* ∪ *F1* ∪ *F2*›
have ‹*finite F*› **and** ‹*F ⊆ A*›
using *F_def P_def*[*abs_def*] *that* ‹*finite F'*› ‹*F' ⊆ A*› **by** *auto*
have *dist_F*: ‹*dist* (*sum* (λ*x*. *norm* (*f x*)) *F*) *L* < *d*›
by (*metis F_def* ‹*F ⊆ A*› *P_def P_sup_F'* ‹*finite F*› *le_supE order_refl*)

have *dist_F_subset*: ‹*dist* (*sum f F*) (*sum f F'*) < *2*d*› **if** *F'*: ‹*F' ⊆ F*› ‹*P F'*› **for** *F'*
proof −
have ‹*dist* (*sum f F*) (*sum f F'*) = *norm* (*sum f* (*F*−*F'*))›
unfolding *dist_norm* **using** ‹*finite F*› *F'* **by** (*subst sum_diff*) *auto*
also have ‹... ≤ *norm* (∑ *x*∈*F*−*F'*. *norm* (*f x*))›
by (*rule order.trans*[*OF sum_norm_le*[*OF order.refl*]]) *auto*
also have ‹... = *dist* (∑ *x*∈*F*. *norm* (*f x*)) (∑ *x*∈*F'*. *norm* (*f x*))›
unfolding *dist_norm* **using** ‹*finite F*› *F'* **by** (*subst sum_diff*) *auto*
also have ‹... < *2* ∗ *d*›
using *dist_F F'* **unfolding** *P_def dist_norm real_norm_def* **by** *linarith*
finally show ‹*dist* (*sum f F*) (*sum f F'*) < *2*d*› **.**
qed

have ‹*dist* (*sum f F1*) (*sum f F2*) ≤ *dist* (*sum f F*) (*sum f F1*) + *dist* (*sum f F*) (*sum f F2*)›
by (*rule dist_triangle3*)
also have ‹... < *2* ∗ *d* + *2* ∗ *d*›
by (*intro add_strict_mono dist_F_subset that*) (*auto simp*: *F_def*)
also have ‹... ≤ *e*›
by (*auto simp*: *d_def*)
finally show ‹*dist* (*sum f F1*) (*sum f F2*) < *e*› **.**
qed
then show *?thesis*
using *ev_P* **by** *blast*
qed
then have ‹*cauchy_filter* (*filtermap* (*sum f*) (*finite_subsets_at_top A*))›
by (*simp add*: *cauchy_filter_metric_filtermap*)
moreover have *complete* (*UNIV*::'*b set*)
by (*meson Cauchy_convergent UNIV_I complete_def convergent_def*)
ultimately obtain *L'* **where** ‹(*sum f* ⟶ *L'*) (*finite_subsets_at_top A*)›
using *complete_uniform*[**where** *S*=*UNIV*] **by** (*force simp add*: *filterlim_def*)
then show *?thesis*

using *summable_on_def has_sum_def* **by** *blast*
qed

The converse of *abs_summable_summable* does not hold: Consider the Hilbert space of square-summable sequences. Let e_i denote the sequence with 1 in the ith position and 0 elsewhere. Let $f(i) := e_i/i$ for $i \geq 1$. We have $\neg$ *f abs_summable_on UNIV* because $\|f(i)\| = 1/i$ and thus the sum over $\|f(i)\|$ diverges. On the other hand, we have *f summable_on UNIV*; the limit is the sequence with $1/i$ in the ith position.

(We have not formalized this separating example here because to the best of our knowledge, this Hilbert space has not been formalized in Isabelle/HOL yet.)

lemma *norm_has_sum_bound*:
fixes *f* :: *'b* ⇒ *'a*::*real_normed_vector*
and *A* :: *'b set*
assumes ((λ*x*. *norm* (*f x*)) *has_sum n*) *A*
assumes (*f has_sum a*) *A*
shows *norm a* ≤ *n*
proof −
have *norm a* ≤ *n* + ε **if** ε>*0* **for** ε
proof−
have ∃ *F*. *norm* (*a* − *sum f F*) ≤ ε ∧ *finite F* ∧ *F* ⊆ *A*
using *has_sum_finite_approximation*[**where** *A*=*A* **and** *f*=*f* **and** ε=ε] *assms* ‹*0* < ε›
by (*metis dist_commute dist_norm*)
then obtain *F* **where** *norm* (*a* − *sum f F*) ≤ ε
and *finite F* **and** *F* ⊆ *A*
by (*simp add*: *atomize_elim*)
hence *norm a* ≤ *norm* (*sum f F*) + ε
by (*metis add.commute diff_add_cancel dual_order.refl norm_triangle_mono*)
also have ... ≤ *sum* (λ*x*. *norm* (*f x*)) *F* + ε
using *norm_sum* **by** *auto*
also have ... ≤ *n* + ε
proof (*intro add_right_mono* [*OF has_sum_mono_neutral*])
show ((λ*x*. *norm* (*f x*)) *has_sum* (∑ *x*∈*F*. *norm* (*f x*))) *F*
by (*simp add*: ‹*finite F*›)
qed (*use* ‹*F* ⊆ *A*› *assms* **in** *auto*)
finally show *?thesis*
by *assumption*
qed
thus *?thesis*
using *linordered_field_class.field_le_epsilon* **by** *blast*
qed

lemma *norm_infsum_bound*:
fixes *f* :: *'b* ⇒ *'a*::*real_normed_vector*
and *A* :: *'b set*
assumes *f abs_summable_on A*

```
  shows norm (infsum f A) ≤ infsum (λx. norm (f x)) A
proof (cases f summable_on A)
  case True
  have ((λx. norm (f x)) has_sum (∑∞x∈A. norm (f x))) A
    by (simp add: assms)
  then show ?thesis
    by (metis True has_sum_infsum norm_has_sum_bound)
next
  case False
  obtain t where t_def: (sum (λx. norm (f x)) ⟶ t) (finite_subsets_at_top
A)
    using assms unfolding summable_on_def has_sum_def by blast
  have sumpos: sum (λx. norm (f x)) X ≥ 0
    for X
    by (simp add: sum_nonneg)
  have tgeq0:t ≥ 0
  proof(rule ccontr)
    define S::real set where S = {s. s < 0}
    assume ¬ 0 ≤ t
    hence t < 0 by simp
    hence t ∈ S
      unfolding S_def by blast
    moreover have open S
      by (metis S_def lessThan_def open_real_lessThan)
    ultimately have ∀F X in finite_subsets_at_top A. (∑ x∈X. norm (f x)) ∈ S
      using t_def unfolding tendsto_def by blast
    hence ∃ X. (∑ x∈X. norm (f x)) ∈ S
    by (metis (no_types, lifting) eventually_mono filterlim_iff finite_subsets_at_top_neq_bot
tendsto_Lim)
    then obtain X where (∑ x∈X. norm (f x)) ∈ S
      by blast
    hence (∑ x∈X. norm (f x)) < 0
      unfolding S_def by auto
    thus False by (simp add: leD sumpos)
  qed
  have ∃!h. (sum (λx. norm (f x)) ⟶ h) (finite_subsets_at_top A)
    using t_def finite_subsets_at_top_neq_bot tendsto_unique by blast
  hence t = (Topological_Spaces.Lim (finite_subsets_at_top A) (sum (λx. norm
(f x))))
    using t_def unfolding Topological_Spaces.Lim_def
    by (metis the_equality)
  hence Lim (finite_subsets_at_top A) (sum (λx. norm (f x))) ≥ 0
    using tgeq0 by blast
  thus ?thesis unfolding infsum_def
    using False by auto
qed

lemma infsum_tendsto:
  assumes ‹f summable_on S›
```

shows ‹((λ*F*. *sum f F*) ⟶ *infsum f S*) (*finite_subsets_at_top S*)›
using *assms has_sum_def has_sum_infsum* **by** *blast*

lemma *has_sum_0*:
assumes ‹⋀*x*. *x*∈*M* ⟹ *f x* = *0*›
shows ‹(*f has_sum 0*) *M*›
by (*metis assms finite.intros*(*1*) *has_sum_cong has_sum_cong_neutral has_sum_finite sum.neutral_const*)

lemma *summable_on_0*:
assumes ‹⋀*x*. *x*∈*M* ⟹ *f x* = *0*›
shows ‹*f summable_on M*›
using *assms summable_on_def has_sum_0* **by** *blast*

lemma *infsum_0*:
assumes ‹⋀*x*. *x*∈*M* ⟹ *f x* = *0*›
shows ‹*infsum f M* = *0*›
by (*metis assms finite_subsets_at_top_neq_bot infsum_def has_sum_0 has_sum_def tendsto_Lim*)

Variants of *infsum_0* etc. suitable as simp-rules

lemma *infsum_0_simp*[*simp*]: ‹*infsum* (λ_. *0*) *M* = *0*›
by (*simp_all add*: *infsum_0*)

lemma *summable_on_0_simp*[*simp*]: ‹(λ_. *0*) *summable_on M*›
by (*simp_all add*: *summable_on_0*)

lemma *has_sum_0_simp*[*simp*]: ‹((λ_. *0*) *has_sum 0*) *M*›
by (*simp_all add*: *has_sum_0*)

lemma *has_sum_add*:
fixes *f g* :: *'a* ⇒ *'b*::{*topological_comm_monoid_add*}
assumes ‹(*f has_sum a*) *A*›
assumes ‹(*g has_sum b*) *A*›
shows ‹((λ*x*. *f x* + *g x*) *has_sum* (*a* + *b*)) *A*›
proof −
from *assms* **have** *lim_f*: ‹(*sum f* ⟶ *a*) (*finite_subsets_at_top A*)›
and *lim_g*: ‹(*sum g* ⟶ *b*) (*finite_subsets_at_top A*)›
by (*simp_all add*: *has_sum_def*)
then have *lim*: ‹(*sum* (λ*x*. *f x* + *g x*) ⟶ *a* + *b*) (*finite_subsets_at_top A*)›
unfolding *sum.distrib* **by** (*rule tendsto_add*)
then show *?thesis*
by (*simp_all add*: *has_sum_def*)
qed

lemma *summable_on_add*:
fixes *f g* :: *'a* ⇒ *'b*::{*topological_comm_monoid_add*}
assumes ‹*f summable_on A*›

assumes ‹*g summable_on A*›
shows ‹(λ*x*. *f x* + *g x*) *summable_on A*›
by (*metis* (*full_types*) *assms summable_on_def has_sum_add*)

lemma *infsum_add*:
fixes *f g* :: *'a* ⇒ *'b*::{*topological_comm_monoid_add*, *t2_space*}
assumes ‹*f summable_on A*›
assumes ‹*g summable_on A*›
shows ‹*infsum* (λ*x*. *f x* + *g x*) *A* = *infsum f A* + *infsum g A*›
proof −
have ‹((λ*x*. *f x* + *g x*) *has_sum* (*infsum f A* + *infsum g A*)) *A*›
by (*simp add*: *assms has_sum_add*)
then show *?thesis*
using *infsumI* **by** *blast*
qed

lemma *has_sum_Un_disjoint*:
fixes *f* :: *'a* ⇒ *'b*::*topological_comm_monoid_add*
assumes (*f has_sum a*) *A*
assumes (*f has_sum b*) *B*
assumes *disj*: *A* ∩ *B* = {}
shows ‹(*f has_sum* (*a* + *b*)) (*A* ∪ *B*)›
proof −
define *fA fB* **where** ‹*fA x* = (*if x* ∈ *A then f x else* 0)›
and ‹*fB x* = (*if x* ∉ *A then f x else* 0)› **for** *x*
have *fA*: ‹(*fA has_sum a*) (*A* ∪ *B*)›
by (*smt* (*verit*, *ccfv_SIG*) *DiffD1 DiffD2 UnCI fA_def assms*(1) *has_sum_cong_neutral inf_sup_absorb*)
have *fB*: ‹(*fB has_sum b*) (*A* ∪ *B*)›
by (*smt* (*verit*, *best*) *DiffD1 DiffD2 IntE Un_iff fB_def assms*(2) *disj disjoint_iff has_sum_cong_neutral*)
have *fAB*: ‹*f x* = *fA x* + *fB x*› **for** *x*
unfolding *fA_def fB_def* **by** *simp*
show *?thesis*
unfolding *fAB*
using *fA fB* **by** (*rule has_sum_add*)
qed

lemma *summable_on_Un_disjoint*:
fixes *f* :: *'a* ⇒ *'b*::*topological_comm_monoid_add*
assumes *f summable_on A*
assumes *f summable_on B*
assumes *disj*: *A* ∩ *B* = {}
shows ‹*f summable_on* (*A* ∪ *B*)›
by (*meson assms disj summable_on_def has_sum_Un_disjoint*)

lemma *infsum_Un_disjoint*:
fixes *f* :: *'a* ⇒ *'b*::{*topological_comm_monoid_add*, *t2_space*}

assumes *f summable_on A*
assumes *f summable_on B*
assumes *disj*: $A \cap B = \{\}$
shows ‹*infsum f* $(A \cup B) =$ *infsum f A* + *infsum f B*›
by (*intro infsumI has_sum_Un_disjoint has_sum_infsum assms*)

lemma *norm_summable_imp_has_sum*:
fixes *f* :: *nat* ⇒ *'a* :: *banach*
assumes *summable* (λ*n*. *norm* (*f n*)) **and** *f sums S*
shows (*f has_sum S*) (*UNIV* :: *nat set*)
unfolding *has_sum_def tendsto_iff eventually_finite_subsets_at_top*
proof *clarsimp*
fix *ε*::*real*
assume *ε* > *0*
from *assms* **obtain** *S'* **where** *S'*: (λ*n*. *norm* (*f n*)) *sums S'*
by (*auto simp*: *summable_def*)
with ‹*ε* > *0*› **obtain** *N* **where** *N*: ⋀*n*. *n* ≥ *N* ⟹ |*S'* − (∑ *i*<*n*. *norm* (*f i*))| < *ε*
by (*auto simp*: *tendsto_iff eventually_at_top_linorder sums_def dist_norm abs_minus_commute*)
have *dist* (*sum f Y*) *S* < *ε* **if** *finite Y* {..<*N*} ⊆ *Y* **for** *Y*
proof −
from *that* **have** (λ*n*. *if n* ∈ *Y then 0 else f n*) *sums* (*S* − *sum f Y*)
by (*intro sums_If_finite_set'*[*OF* ‹*f sums S*›]) (*auto simp*: *sum_negf*)
hence *S* − *sum f Y* = (∑ *n*. *if n* ∈ *Y then 0 else f n*)
by (*simp add*: *sums_iff*)
also have *norm* ... ≤ (∑ *n*. *norm* (*if n* ∈ *Y then 0 else f n*))
by (*rule summable_norm*[*OF summable_comparison_test'*[*OF assms*(*1*)]]) *auto*
also have ... ≤ (∑ *n*. *if n* < *N then 0 else norm* (*f n*))
using *that* **by** (*intro suminf_le summable_comparison_test'*[*OF assms*(*1*)]) *auto*
also have (λ*n*. *if n* ∈ {..<*N*} *then 0 else norm* (*f n*)) *sums* (*S'* − (∑ *i*<*N*. *norm* (*f i*)))
by (*intro sums_If_finite_set'*[*OF S'*]) (*auto simp*: *sum_negf*)
hence (∑ *n*. *if n* < *N then 0 else norm* (*f n*)) = *S'* − (∑ *i*<*N*. *norm* (*f i*))
by (*simp add*: *sums_iff*)
also have *S'* − (∑ *i*<*N*. *norm* (*f i*)) ≤ |*S'* − (∑ *i*<*N*. *norm* (*f i*))| **by** *simp*
also have ... < *ε* **by** (*rule N*) *auto*
finally show *?thesis* **by** (*simp add*: *dist_norm norm_minus_commute*)
qed
then show ∃ *X*. *finite X* ∧ (∀ *Y*. *finite Y* ∧ *X* ⊆ *Y* ⟶ *dist* (*sum f Y*) *S* < *ε*)
by (*meson finite_lessThan subset_UNIV*)
qed

lemma *norm_summable_imp_summable_on*:
fixes *f* :: *nat* ⇒ *'a* :: *banach*
assumes *summable* (λ*n*. *norm* (*f n*))
shows *f summable_on UNIV*

using *norm_summable_imp_has_sum*[*OF assms*, *of suminf f*] *assms*
by (*auto simp*: *sums_iff summable_on_def dest*: *summable_norm_cancel*)

The following lemma indeed needs a complete space (as formalized by the premise *complete UNIV*). The following two counterexamples show this:

- Consider the real vector space V of sequences with finite support, and with the ℓ_2-norm (sum of squares). Let e_i denote the sequence with a 1 at position i. Let $f : \mathbb{Z} \to V$ be defined as $f(n) := e_{|n|}/n$ (with $f(0) := 0$). We have that $\sum_{n\in\mathbb{Z}} f(n) = 0$ (it even converges absolutely). But $\sum_{n\in\mathbb{N}} f(n)$ does not exist (it would converge against a sequence with infinite support).
- Let f be a positive rational valued function such that $\sum_{x\in B} f(x)$ is $\sqrt{2}$ and $\sum_{x\in A} f(x)$ is 1 (over the reals, with $A \subseteq B$). Then $\sum_{x\in B} f(x)$ does not exist over the rationals. But $\sum_{x\in A} f(x)$ exists.

The lemma also requires uniform continuity of the addition. And example of a topological group with continuous but not uniformly continuous addition would be the positive reals with the usual multiplication as the addition. We do not know whether the lemma would also hold for such topological groups.

lemma *summable_on_subset_aux*:
fixes *A B* **and** *f* :: ‹*'a* ⇒ *'b*::{*ab_group_add*, *uniform_space*}›
assumes ‹*complete* (*UNIV* :: *'b set*)›
assumes *plus_cont*: ‹*uniformly_continuous_on UNIV* (λ(*x*::*'b*,*y*). *x*+*y*)›
assumes ‹*f summable_on A*›
assumes ‹*B* ⊆ *A*›
shows ‹*f summable_on B*›
proof −
let *?filter_fB* = ‹*filtermap* (*sum f*) (*finite_subsets_at_top B*)›
from ‹*f summable_on A*›
obtain *S* **where** ‹(*sum f* ⟶ *S*) (*finite_subsets_at_top A*)› (**is** ‹(*sum f* ⟶ *S*) *?filter_A*›)
using *summable_on_def has_sum_def* **by** *blast*
then have *cauchy_fA*: ‹*cauchy_filter* (*filtermap* (*sum f*) (*finite_subsets_at_top A*))› (**is** ‹*cauchy_filter ?filter_fA*›)
by (*auto intro*!: *nhds_imp_cauchy_filter simp*: *filterlim_def*)

have ‹*cauchy_filter* (*filtermap* (*sum f*) (*finite_subsets_at_top B*))›
proof (*unfold cauchy_filter_def*, *rule filter_leI*)
fix *E* :: ‹(*'b*×*'b*) ⇒ *bool*› **assume** ‹*eventually E uniformity*›
then obtain *E'* **where** ‹*eventually E' uniformity*› **and** *E'E'E*: ‹*E'* (*x*, *y*) ⟶ *E'* (*y*, *z*) ⟶ *E* (*x*, *z*)› **for** *x y z*
using *uniformity_trans* **by** *blast*
obtain *D* **where** ‹*eventually D uniformity*› **and** *DE*: ‹*D* (*x*, *y*) ⟹ *E'* (*x*+*c*, *y*+*c*)› **for** *x y c*
using *plus_cont* ‹*eventually E' uniformity*›

unfolding *uniformly_continuous_on_uniformity filterlim_def le_filter_def uniformity_prod_def*
by (*auto simp*: *case_prod_beta eventually_filtermap eventually_prod_same uniformity_refl*)
have *DE'*: $E'\ (x,\ y)$ **if** $D\ (x + c,\ y + c)$ **for** *x y c*
using *DE*[*of x + c y + c* $-c$] *that* **by** *simp*

from ‹*eventually D uniformity*› **and** *cauchy_fA* **have** ‹*eventually D* (*?filter_fA* $\times_F$ *?filter_fA*)›
unfolding *cauchy_filter_def le_filter_def* **by** *simp*
then obtain *P1 P2*
where *ev_P1*: ‹*eventually* ($\lambda F.$ *P1* (*sum f F*)) *?filter_A*›
and *ev_P2*: ‹*eventually* ($\lambda F.$ *P2* (*sum f F*)) *?filter_A*›
and *P1P2E*: ‹*P1 x* $\Longrightarrow$ *P2 y* $\Longrightarrow$ *D* $(x,\ y)$› **for** *x y*
unfolding *eventually_prod_filter eventually_filtermap*
by *auto*
from *ev_P1* **obtain** *F1* **where** *F1*: ‹*finite F1*› ‹*F1* $\subseteq$ *A*› ‹$\bigwedge F.$ *F*$\supseteq$*F1* $\Longrightarrow$ *finite F* $\Longrightarrow$ *F*$\subseteq$*A* $\Longrightarrow$ *P1* (*sum f F*)›
by (*metis eventually_finite_subsets_at_top*)
from *ev_P2* **obtain** *F2* **where** *F2*: ‹*finite F2*› ‹*F2* $\subseteq$ *A*› ‹$\bigwedge F.$ *F*$\supseteq$*F2* $\Longrightarrow$ *finite F* $\Longrightarrow$ *F*$\subseteq$*A* $\Longrightarrow$ *P2* (*sum f F*)›
by (*metis eventually_finite_subsets_at_top*)
define *F0 F0A F0B* **where** ‹*F0* $\equiv$ *F1* $\cup$ *F2*› **and** ‹*F0A* $\equiv$ *F0* $-$ *B*› **and** ‹*F0B* $\equiv$ *F0* $\cap$ *B*›
have [*simp*]: ‹*finite F0*› ‹*F0* $\subseteq$ *A*›
using ‹*F1* $\subseteq$ *A*› ‹*F2* $\subseteq$ *A*› ‹*finite F1*› ‹*finite F2*› **unfolding** *F0_def* **by** *blast*+

have $*$: $E'\ (sum\ f\ F1',\ sum\ f\ F2')$
if *F1'*$\supseteq$*F0B F2'*$\supseteq$*F0B finite F1' finite F2' F1'*$\subseteq$*B F2'*$\subseteq$*B* **for** *F1' F2'*
proof (*intro DE'*[**where** $c = sum\ f\ F0A$] *P1P2E*)
have *P1* (*sum f* (*F1'* $\cup$ *F0A*))
using *that assms F1*(*1*,*2*) *F2*(*1*,*2*) **by** (*intro F1*) (*auto simp*: *F0A_def F0B_def F0_def*)
thus *P1* (*sum f F1'* + *sum f F0A*)
by (*subst* (*asm*) *sum.union_disjoint*) (*use that* **in** ‹*auto simp*: *F0A_def*›)
next
have *P2* (*sum f* (*F2'* $\cup$ *F0A*))
using *that assms F1*(*1*,*2*) *F2*(*1*,*2*) **by** (*intro F2*) (*auto simp*: *F0A_def F0B_def F0_def*)
thus *P2* (*sum f F2'* + *sum f F0A*)
by (*subst* (*asm*) *sum.union_disjoint*) (*use that* **in** ‹*auto simp*: *F0A_def*›)

qed

have *eventually* ($\lambda x.$ $E'\ (x,\ sum\ f\ F0B)$) (*filtermap* (*sum f*) (*finite_subsets_at_top B*))
and *eventually* ($\lambda x.$ $E'\ (sum\ f\ F0B,\ x)$) (*filtermap* (*sum f*) (*finite_subsets_at_top B*))

unfolding *eventually_filtermap eventually_finite_subsets_at_top*
by (*rule exI*[*of* _ *F0B*]; *use* ∗ **in** ‹*force simp*: *F0B_def*›)+
then
show ‹*eventually E* (*?filter_fB* ×$_F$ *?filter_fB*)›
unfolding *eventually_prod_filter*
using *E'E'E* **by** *blast*
qed

then obtain *x* **where** ‹*?filter_fB* ≤ *nhds x*›
using *cauchy_filter_complete_converges*[*of ?filter_fB UNIV*] ‹*complete* (*UNIV* :: _)›
by (*auto simp*: *filtermap_bot_iff*)
then have ‹(*sum f* ⟶ *x*) (*finite_subsets_at_top B*)›
by (*auto simp*: *filterlim_def*)
then show *?thesis*
by (*auto simp*: *summable_on_def has_sum_def*)
qed

A special case of *summable_on_subset_aux* for Banach spaces with fewer premises.

lemma *summable_on_subset_banach*:
fixes *A B* **and** *f* :: ‹'*a* ⇒ '*b*::*banach*›
assumes ‹*f summable_on A*›
assumes ‹*B* ⊆ *A*›
shows ‹*f summable_on B*›
by (*meson Cauchy_convergent UNIV_I assms complete_def convergent_def isUCont_plus summable_on_subset_aux*)

lemma *has_sum_empty*[*simp*]: ‹(*f has_sum 0*) {}›
by (*meson ex_in_conv has_sum_0*)

lemma *summable_on_empty*[*simp*]: ‹*f summable_on* {}›
by *auto*

lemma *infsum_empty*[*simp*]: ‹*infsum f* {} = *0*›
by *simp*

lemma *sum_has_sum*:
fixes *f* :: '*a* ⇒ '*b*::*topological_comm_monoid_add*
assumes ‹*finite A*›
assumes ‹⋀*a*. *a* ∈ *A* ⟹ (*f has_sum* (*s a*)) (*B a*)›
assumes ‹⋀*a a'*. *a*∈*A* ⟹ *a'*∈*A* ⟹ *a*≠*a'* ⟹ *B a* ∩ *B a'* = {}›
shows ‹(*f has_sum* (*sum s A*)) (⋃*a*∈*A*. *B a*)›
using *assms*
proof (*induction*)
case *empty*
then show *?case*
by *simp*
next

case (*insert x A*)
have ‹(*f has_sum* (*s x*)) (*B x*)›
by (*simp add*: *insert.prems*)
moreover have *IH*: ‹(*f has_sum* (*sum s A*)) (⋃*a*∈*A*. *B a*)›
using *insert* **by** *simp*
ultimately have ‹(*f has_sum* (*s x* + *sum s A*)) (*B x* ∪ (⋃*a*∈*A*. *B a*))›
using *insert* **by** (*intro has_sum_Un_disjoint*) *auto*
then show *?case*
using *insert.hyps* **by** *auto*
qed

lemma *summable_on_finite_union_disjoint*:
fixes *f* :: *'a* ⇒ *'b*::*topological_comm_monoid_add*
assumes *finite*: ‹*finite A*›
assumes *conv*: ‹⋀*a*. *a* ∈ *A* ⟹ *f summable_on* (*B a*)›
assumes *disj*: ‹⋀*a a'*. *a*∈*A* ⟹ *a'*∈*A* ⟹ *a*≠*a'* ⟹ *B a* ∩ *B a'* = {}›
shows ‹*f summable_on* (⋃*a*∈*A*. *B a*)›
using *sum_has_sum* [*of A f B*] *assms* **unfolding** *summable_on_def* **by** *metis*

lemma *sum_infsum*:
fixes *f* :: *'a* ⇒ *'b*::{*topological_comm_monoid_add*, *t2_space*}
assumes *finite*: ‹*finite A*›
assumes *conv*: ‹⋀*a*. *a* ∈ *A* ⟹ *f summable_on* (*B a*)›
assumes *disj*: ‹⋀*a a'*. *a*∈*A* ⟹ *a'*∈*A* ⟹ *a*≠*a'* ⟹ *B a* ∩ *B a'* = {}›
shows ‹*sum* (λ*a*. *infsum f* (*B a*)) *A* = *infsum f* (⋃*a*∈*A*. *B a*)›
by (*metis* (*no_types*, *lifting*) *assms has_sum_infsum infsumI sum_has_sum*)

The lemmas *infsum_comm_additive_general* and *infsum_comm_additive* (and variants) below both state that the infinite sum commutes with a continuous additive function. *infsum_comm_additive_general* is stated more for more general type classes at the expense of a somewhat less compact formulation of the premises. E.g., by avoiding the constant *additive* which introduces an additional sort constraint (group instead of monoid). For example, extended reals (*ereal*, *ennreal*) are not covered by *infsum_comm_additive*.

lemma *has_sum_comm_additive_general*:
fixes *f* :: ‹*'b* :: {*comm_monoid_add*,*topological_space*} ⇒ *'c* :: {*comm_monoid_add*,*topological_space*}›
assumes *f_sum*: ‹⋀*F*. *finite F* ⟹ *F* ⊆ *S* ⟹ *sum* (*f* ∘ *g*) *F* = *f* (*sum g F*)›
— Not using *additive* because it would add sort constraint *ab_group_add*
assumes *cont*: ‹*f* −*x*→ *f x*›
— For *t2_space*, this is equivalent to *isCont f x* by *isCont_def*.
assumes *infsum*: ‹(*g has_sum x*) *S*›
shows ‹((*f* ∘ *g*) *has_sum* (*f x*)) *S*›
proof −
have ‹(*sum g* ⟶ *x*) (*finite_subsets_at_top S*)›
using *infsum has_sum_def* **by** *blast*
then have ‹((*f* ∘ *sum g*) ⟶ *f x*) (*finite_subsets_at_top S*)›
by (*meson cont filterlim_def tendsto_at_iff_tendsto_nhds tendsto_compose_filtermap*

tendsto_mono)
then have ‹(*sum* (*f* ∘ *g*) ⟶ *f x*) (*finite_subsets_at_top S*)›
using *tendsto_cong f_sum*
by (*simp add*: *Lim_transform_eventually eventually_finite_subsets_at_top_weakI*)
then show ‹((*f* ∘ *g*) *has_sum* (*f x*)) *S*›
using *has_sum_def* **by** *blast*
qed

lemma *summable_on_comm_additive_general*:
fixes *f* :: ‹'*b* :: {*comm_monoid_add*,*topological_space*} ⇒ '*c* :: {*comm_monoid_add*,*topological_space*}
assumes ‹⋀*F*. *finite F* ⟹ *F* ⊆ *S* ⟹ *sum* (*f* ∘ *g*) *F* = *f* (*sum g F*)›
— Not using *additive* because it would add sort constraint *ab_group_add*
assumes ‹⋀*x*. (*g has_sum x*) *S* ⟹ *f* −*x*→ *f x*›
— For *t2_space*, this is equivalent to *isCont f x* by *isCont_def*.
assumes ‹*g summable_on S*›
shows ‹(*f* ∘ *g*) *summable_on S*›
by (*meson assms summable_on_def has_sum_comm_additive_general has_sum_def infsum_tendsto*)

lemma *infsum_comm_additive_general*:
fixes *f* :: ‹'*b* :: {*comm_monoid_add*,*t2_space*} ⇒ '*c* :: {*comm_monoid_add*,*t2_space*}›
assumes *f_sum*: ‹⋀*F*. *finite F* ⟹ *F* ⊆ *S* ⟹ *sum* (*f* ∘ *g*) *F* = *f* (*sum g F*)›
— Not using *additive* because it would add sort constraint *ab_group_add*
assumes ‹*isCont f* (*infsum g S*)›
assumes ‹*g summable_on S*›
shows ‹*infsum* (*f* ∘ *g*) *S* = *f* (*infsum g S*)›
using *assms*
by (*intro infsumI has_sum_comm_additive_general has_sum_infsum*) (*auto simp*: *isCont_def*)

lemma *has_sum_comm_additive*:
fixes *f* :: ‹'*b* :: {*ab_group_add*,*topological_space*} ⇒ '*c* :: {*ab_group_add*,*topological_space*}›
assumes ‹*additive f*›
assumes ‹*f* −*x*→ *f x*›
— For *t2_space*, this is equivalent to *isCont f x* by *isCont_def*.
assumes *infsum*: ‹(*g has_sum x*) *S*›
shows ‹((*f* ∘ *g*) *has_sum* (*f x*)) *S*›
using *assms*
by (*intro has_sum_comm_additive_general has_sum_infsum*) (*auto simp*: *isCont_def additive.sum*)

lemma *summable_on_comm_additive*:
fixes *f* :: ‹'*b* :: {*ab_group_add*,*t2_space*} ⇒ '*c* :: {*ab_group_add*,*topological_space*}›
assumes ‹*additive f*›
assumes ‹*isCont f* (*infsum g S*)›
assumes ‹*g summable_on S*›
shows ‹(*f* ∘ *g*) *summable_on S*›
by (*meson assms summable_on_def has_sum_comm_additive has_sum_infsum isContD*)

lemma *infsum_comm_additive*:
fixes $f :: \langle 'b :: \{ab\_group\_add,t2\_space\} \Rightarrow 'c :: \{ab\_group\_add,t2\_space\}\rangle$
assumes ‹*additive f*›
assumes ‹*isCont f* (*infsum g S*)›
assumes ‹*g summable_on S*›
shows ‹*infsum* (*f* ∘ *g*) *S* = *f* (*infsum g S*)›
by (*rule infsum_comm_additive_general*; *auto simp*: *assms additive.sum*)

lemma *nonneg_bdd_above_has_sum*:
fixes *f* :: ‹*'a* ⇒ *'b* :: {*conditionally_complete_linorder*, *ordered_comm_monoid_add*, *linorder_topology*}›
assumes ‹⋀*x*. *x*∈*A* ⟹ *f x* ≥ *0*›
assumes ‹*bdd_above* (*sum f* ‘ {*F*. *F*⊆*A* ∧ *finite F*})›
shows ‹(*f has_sum* (*SUP F*∈{*F*. *finite F* ∧ *F*⊆*A*}. *sum f F*)) *A*›
proof −
have ‹(*sum f* ⟶⟶ (*SUP F*∈{*F*. *finite F* ∧ *F*⊆*A*}. *sum f F*)) (*finite_subsets_at_top A*)›
proof (*rule order_tendstoI*)
fix *a* **assume** ‹*a* < (*SUP F*∈{*F*. *finite F* ∧ *F*⊆*A*}. *sum f F*)›
then obtain *F* **where** ‹*a* < *sum f F*› **and** ‹*finite F*› **and** ‹*F* ⊆ *A*›
by (*metis* (*mono_tags*, *lifting*) *Collect_cong Collect_empty_eq assms*(*2*) *empty_subsetI finite.emptyI less_cSUP_iff mem_Collect_eq*)
have ⋀*Y*. ⟦*finite Y*; *F* ⊆ *Y*; *Y* ⊆ *A*⟧ ⟹ *a* < *sum f Y*
by (*meson DiffE* ‹*a* < *sum f F*› *assms*(*1*) *less_le_trans subset_iff sum_mono2*)
then show ‹∀*F x in finite_subsets_at_top A*. *a* < *sum f x*›
by (*metis* ‹*F* ⊆ *A*› ‹*finite F*› *eventually_finite_subsets_at_top*)
next
fix *a* **assume** ∗: ‹(*SUP F*∈{*F*. *finite F* ∧ *F*⊆*A*}. *sum f F*) < *a*›
have *sum f F* ≤ (*SUP F*∈{*F*. *finite F* ∧ *F*⊆*A*}. *sum f F*) **if** ‹*F*⊆*A*› **and** ‹*finite F*› **for** *F*
by (*rule cSUP_upper*) (*use that assms*(*2*) **in** ‹*auto simp*: *conj_commute*›)
then show ‹∀*F x in finite_subsets_at_top A*. *sum f x* < *a*›
by (*metis* (*no_types*, *lifting*) ∗ *eventually_finite_subsets_at_top_weakI order_le_less_trans*)
qed
then show *?thesis*
using *has_sum_def* **by** *blast*
qed

lemma *nonneg_bdd_above_summable_on*:
fixes *f* :: ‹*'a* ⇒ *'b* :: {*conditionally_complete_linorder*, *ordered_comm_monoid_add*, *linorder_topology*}›
assumes ‹⋀*x*. *x*∈*A* ⟹ *f x* ≥ *0*›
assumes ‹*bdd_above* (*sum f* ‘ {*F*. *F*⊆*A* ∧ *finite F*})›
shows ‹*f summable_on A*›
using *assms summable_on_def nonneg_bdd_above_has_sum* **by** *blast*

lemma *nonneg_bdd_above_infsum*:

fixes $f :: \langle 'a \Rightarrow 'b :: \{conditionally\_complete\_linorder, ordered\_comm\_monoid\_add, linorder\_topology\}\rangle$
assumes $\langle \bigwedge x.\ x \in A \Longrightarrow f\ x \geq 0\rangle$
assumes $\langle bdd\_above\ (sum\ f\ `\ \{F.\ F \subseteq A \wedge finite\ F\})\rangle$
shows $\langle infsum\ f\ A = (SUP\ F \in \{F.\ finite\ F \wedge F \subseteq A\}.\ sum\ f\ F)\rangle$
using *assms* **by** (*auto intro*!: *infsumI nonneg_bdd_above_has_sum*)

lemma *nonneg_has_sum_complete*:
fixes $f :: \langle 'a \Rightarrow 'b :: \{complete\_linorder, ordered\_comm\_monoid\_add, linorder\_topology\}\rangle$
assumes $\langle \bigwedge x.\ x \in A \Longrightarrow f\ x \geq 0\rangle$
shows $\langle (f\ has\_sum\ (SUP\ F \in \{F.\ finite\ F \wedge F \subseteq A\}.\ sum\ f\ F))\ A\rangle$
using *assms nonneg_bdd_above_has_sum* **by** *blast*

lemma *nonneg_summable_on_complete*:
fixes $f :: \langle 'a \Rightarrow 'b :: \{complete\_linorder, ordered\_comm\_monoid\_add, linorder\_topology\}\rangle$
assumes $\langle \bigwedge x.\ x \in A \Longrightarrow f\ x \geq 0\rangle$
shows $\langle f\ summable\_on\ A\rangle$
using *assms nonneg_bdd_above_summable_on* **by** *blast*

lemma *nonneg_infsum_complete*:
fixes $f :: \langle 'a \Rightarrow 'b :: \{complete\_linorder, ordered\_comm\_monoid\_add, linorder\_topology\}\rangle$
assumes $\langle \bigwedge x.\ x \in A \Longrightarrow f\ x \geq 0\rangle$
shows $\langle infsum\ f\ A = (SUP\ F \in \{F.\ finite\ F \wedge F \subseteq A\}.\ sum\ f\ F)\rangle$
using *assms nonneg_bdd_above_infsum* **by** *blast*

lemma *has_sum_nonneg*:
fixes $f :: 'a \Rightarrow 'b :: \{ordered\_comm\_monoid\_add, linorder\_topology\}$
assumes $(f\ has\_sum\ a)\ M$
and $\bigwedge x.\ x \in M \Longrightarrow 0 \leq f\ x$
shows $a \geq 0$
by (*metis* (*no_types*, *lifting*) *DiffD1 assms empty_iff has_sum_0 has_sum_mono_neutral order_refl*)

lemma *infsum_nonneg*:
fixes $f :: 'a \Rightarrow 'b :: \{ordered\_comm\_monoid\_add, linorder\_topology\}$
assumes $\bigwedge x.\ x \in M \Longrightarrow 0 \leq f\ x$
shows $infsum\ f\ M \geq 0$ (**is** *?lhs* $\geq$ _)
by (*metis assms has_sum_infsum has_sum_nonneg infsum_not_exists linorder_linear*)

lemma *has_sum_mono2*:
fixes $f :: 'a \Rightarrow 'b :: \{topological\_ab\_group\_add, ordered\_comm\_monoid\_add, linorder\_topology\}$
assumes $(f\ has\_sum\ S)\ A$ $(f\ has\_sum\ S')\ B$ $A \subseteq B$
assumes $\bigwedge x.\ x \in B - A \Longrightarrow f\ x \geq 0$
shows $S \leq S'$
by (*metis add_0 add_right_mono assms diff_add_cancel has_sum_Diff has_sum_nonneg*)

lemma *infsum_mono2*:
fixes $f :: 'a \Rightarrow 'b :: \{topological\_ab\_group\_add, ordered\_comm\_monoid\_add, linorder\_topology\}$
assumes $f\ summable\_on\ A$ $f\ summable\_on\ B$ $A \subseteq B$

assumes $\bigwedge x.\ x \in B - A \Longrightarrow f\ x \geq 0$
shows *infsum f A* $\leq$ *infsum f B*
by (*rule has_sum_mono2*[*OF has_sum_infsum has_sum_infsum*]) (*use assms* **in** *auto*)

lemma *finite_sum_le_has_sum*:
fixes $f :: 'a \Rightarrow 'b$::{*topological_ab_group_add*, *ordered_comm_monoid_add*,*linorder_topology*}
assumes (*f has_sum S*) *A finite B* $B \subseteq A$
assumes $\bigwedge x.\ x \in A - B \Longrightarrow f\ x \geq 0$
shows *sum f B* $\leq S$
by (*meson assms has_sum_finite has_sum_mono2*)

lemma *finite_sum_le_infsum*:
fixes $f :: 'a \Rightarrow 'b$::{*topological_ab_group_add*, *ordered_comm_monoid_add*,*linorder_topology*}
assumes *f summable_on A finite B* $B \subseteq A$
assumes $\bigwedge x.\ x \in A - B \Longrightarrow f\ x \geq 0$
shows *sum f B* $\leq$ *infsum f A*
by (*rule finite_sum_le_has_sum*[*OF has_sum_infsum*]) (*use assms* **in** *auto*)

lemma *has_sum_reindex*:
assumes ‹*inj_on h A*›
shows ‹(*g has_sum x*) (*h* ‘ *A*) $\longleftrightarrow$ ((*g* $\circ$ *h*) *has_sum x*) *A*›
proof −
have ‹(*g has_sum x*) (*h* ‘ *A*) $\longleftrightarrow$ (*sum g* $\longrightarrow$ *x*) (*finite_subsets_at_top* (*h* ‘ *A*))›
by (*simp add*: *has_sum_def*)
also have ‹... $\longleftrightarrow$ ((λF. *sum g* (*h* ‘ *F*)) $\longrightarrow$ *x*) (*finite_subsets_at_top A*)›
by (*metis assms filterlim_filtermap filtermap_image_finite_subsets_at_top*)
also have ‹... $\longleftrightarrow$ (*sum* (*g* $\circ$ *h*) $\longrightarrow$ *x*) (*finite_subsets_at_top A*)›
proof (*intro tendsto_cong eventually_finite_subsets_at_top_weakI sum.reindex*)
show $\bigwedge X.\ [\![\mathit{finite}\ X;\ X \subseteq A]\!] \Longrightarrow$ *inj_on h X*
using *assms subset_inj_on* **by** *blast*
qed
also have ‹... $\longleftrightarrow$ ((*g* $\circ$ *h*) *has_sum x*) *A*›
by (*simp add*: *has_sum_def*)
finally show *?thesis* **.**
qed

lemma *summable_on_reindex*:
assumes ‹*inj_on h A*›
shows ‹*g summable_on* (*h* ‘ *A*) $\longleftrightarrow$ (*g* $\circ$ *h*) *summable_on A*›
by (*simp add*: *assms summable_on_def has_sum_reindex*)

lemma *infsum_reindex*:
assumes ‹*inj_on h A*›
shows ‹*infsum g* (*h* ‘ *A*) = *infsum* (*g* $\circ$ *h*) *A*›
by (*metis assms has_sum_infsum has_sum_reindex infsumI infsum_def*)

lemma *summable_on_reindex_bij_betw*:

assumes *bij_betw g A B*
shows $(\lambda x.\ f\ (g\ x))$ *summable_on A* $\longleftrightarrow$ *f summable_on B*
by (*smt* (*verit*) *assms bij_betw_def o_apply summable_on_cong summable_on_reindex*)

lemma *infsum_reindex_bij_betw*:
assumes *bij_betw g A B*
shows *infsum* $(\lambda x.\ f\ (g\ x))\ A =$ *infsum f B*
by (*metis* (*mono_tags*, *lifting*) *assms bij_betw_def infsum_cong infsum_reindex o_def*)

lemma *sum_uniformity*:
assumes *plus_cont*: ‹*uniformly_continuous_on UNIV* (λ(*x*::'*b*::{*uniform_space*,*comm_monoid_add*}, *x*+*y*)›
assumes *EE*: ‹*eventually E uniformity*›
obtains *D* **where** ‹*eventually D uniformity*›
and ‹$\bigwedge M$::'*a set*. $\bigwedge f\ f'$:: '*a* ⇒ '*b*. *card M* ≤ *n* ∧ (∀ *m*∈*M*. *D* (*f m*, *f' m*)) ⟹ *E* (*sum f M*, *sum f' M*)›
proof (*atomize_elim*, *insert EE*, *induction n arbitrary*: *E rule*:*nat_induct*)
case *0*
then show *?case*
by (*metis card_eq_0_iff equals0D le_zero_eq sum.infinite sum.not_neutral_contains_not_neutral uniformity_refl*)
next
case (*Suc n*)
from *plus_cont*[*unfolded uniformly_continuous_on_uniformity filterlim_def le_filter_def*, *rule_format*, *OF Suc.prems*]
obtain *D1 D2* **where** ‹*eventually D1 uniformity*› **and** ‹*eventually D2 uniformity*›

and *D1D2E*: ‹*D1* (*x*, *y*) ⟹ *D2* (*x'*, *y'*) ⟹ *E* (*x* + *x'*, *y* + *y'*)› **for** *x y x' y'*
apply *atomize_elim*
by (*auto simp*: *eventually_prod_filter case_prod_beta uniformity_prod_def eventually_filtermap*)

from *Suc.IH*[*OF* ‹*eventually D2 uniformity*›]
obtain *D3* **where** ‹*eventually D3 uniformity*› **and** *D3*: ‹*card M* ≤ *n* ⟹ (∀ *m*∈*M*. *D3* (*f m*, *f' m*)) ⟹ *D2* (*sum f M*, *sum f' M*)›
for *M* :: ‹'*a set*› **and** *f f'*
by *metis*

define *D* **where** ‹*D x* ≡ *D1 x* ∧ *D3 x*› **for** *x*
have ‹*eventually D uniformity*›
using *D_def* ‹*eventually D1 uniformity*› ‹*eventually D3 uniformity*› *eventually_elim2* **by** *blast*

have ‹*E* (*sum f M*, *sum f' M*)›
if ‹*card M* ≤ *Suc n*› **and** *DM*: ‹∀ *m*∈*M*. *D* (*f m*, *f' m*)›
for *M* :: ‹'*a set*› **and** *f f'*
proof (*cases* ‹*card M* = *0*›)

```
    case True
    then show ?thesis
     by (metis Suc.prems card_eq_0_iff sum.empty sum.infinite uniformity_refl)
  next
    case False
    with ‹card M ≤ Suc n› obtain N x where ‹card N ≤ n› and ‹x ∉ N› and
‹M = insert x N›
      by (metis card_Suc_eq less_Suc_eq_0_disj less_Suc_eq_le)

    from DM have ‹⋀m. m∈N ⟹ D (f m, f' m)›
      using ‹M = insert x N› by blast
    with D3[OF ‹card N ≤ n›]
    have D2_N: ‹D2 (sum f N, sum f' N)›
      using D_def by blast

    from DM
    have ‹D (f x, f' x)›
      using ‹M = insert x N› by blast
    then have ‹D1 (f x, f' x)›
      by (simp add: D_def)

    with D2_N
    have ‹E (f x + sum f N, f' x + sum f' N)›
      using D1D2E by presburger

    then show ‹E (sum f M, sum f' M)›
        by (metis False ‹M = insert x N› ‹x ∉ N› card.infinite finite_insert
sum.insert)
  qed
  with ‹eventually D uniformity› show ?case
    by auto
qed

lemma has_sum_Sigma:
  fixes A :: 'a set and B :: 'a ⇒ 'b set
    and f :: ‹'a × 'b ⇒ 'c::{comm_monoid_add,uniform_space}›
  assumes plus_cont: ‹uniformly_continuous_on UNIV (λ(x::'c,y). x+y)›
  assumes summableAB: (f has_sum a) (Sigma A B)
  assumes summableB: ‹⋀x. x∈A ⟹ ((λy. f (x, y)) has_sum b x) (B x)›
  shows (b has_sum a) A
proof −
  define F FB FA where ‹F = finite_subsets_at_top (Sigma A B)› and ‹FB x
= finite_subsets_at_top (B x)›
    and ‹FA = finite_subsets_at_top A› for x

  from summableB
  have sum_b: ‹(sum (λy. f (x, y)) ⟶ b x) (FB x)› if ‹x ∈ A› for x
    using FB_def[abs_def] has_sum_def that by auto
  from summableAB
```

have *sum_S*: ‹(*sum f* ⟶ *a*) *F*›
using *F_def has_sum_def* **by** *blast*

have *finite_proj*: ‹*finite* {*b*| *b*. (*a*,*b*) ∈ *H*}› **if** ‹*finite H*› **for** *H* :: ‹('*a*×'*b*) *set*› **and** *a*
by (*metis* (*no_types*, *lifting*) *finite_imageI finite_subset image_eqI mem_Collect_eq snd_conv subsetI that*)

have ‹(*sum b* ⟶ *a*) *FA*›
proof (*rule tendsto_iff_uniformity*[*THEN iffD2*, *rule_format*])
fix *E* :: ‹('*c* × '*c*) ⇒ *bool*›
assume ‹*eventually E uniformity*›
then obtain *D* **where** *D_uni*: ‹*eventually D uniformity*› **and** *DDE'*: ‹⋀*x y z*. *D* (*x*, *y*) ⟹ *D* (*y*, *z*) ⟹ *E* (*x*, *z*)›
by (*metis* (*no_types*, *lifting*) ‹*eventually E uniformity*› *uniformity_transE*)
from *sum_S* **obtain** *G* **where** ‹*finite G*› **and** ‹*G* ⊆ *Sigma A B*›
and *G_sum*: ‹*G* ⊆ *H* ⟹ *H* ⊆ *Sigma A B* ⟹ *finite H* ⟹ *D* (*sum f H*, *a*)› **for** *H*
unfolding *tendsto_iff_uniformity*
by (*metis* (*mono_tags*, *lifting*) *D_uni F_def eventually_finite_subsets_at_top*)
have ‹*finite* (*fst* ‘ *G*)› **and** ‹*fst* ‘ *G* ⊆ *A*›
using ‹*finite G*› ‹*G* ⊆ *Sigma A B*› **by** *auto*
thm *uniformity_prod_def*
define *Ga* **where** ‹*Ga a* = {*b*. (*a*,*b*) ∈ *G*}› **for** *a*
have *Ga_fin*: ‹*finite* (*Ga a*)› **and** *Ga_B*: ‹*Ga a* ⊆ *B a*› **for** *a*
using ‹*finite G*› ‹*G* ⊆ *Sigma A B*› *finite_proj* **by** (*auto simp*: *Ga_def finite_proj*)

have ‹*E* (*sum b M*, *a*)› **if** ‹*M* ⊇ *fst* ‘ *G*› **and** ‹*finite M*› **and** ‹*M* ⊆ *A*› **for** *M*
proof −
define *FMB* **where** ‹*FMB* = *finite_subsets_at_top* (*Sigma M B*)›
have ‹*eventually* (λ*H*. *D* (∑ *a*∈*M*. *b a*, ∑ (*a*,*b*)∈*H*. *f* (*a*,*b*))) *FMB*›
proof −
obtain *D'* **where** *D'_uni*: ‹*eventually D' uniformity*›
and ‹*card M'* ≤ *card M* ∧ (∀ *m*∈*M'*. *D'* (*g m*, *g' m*)) ⟹ *D* (*sum g M'*, *sum g' M'*)›
for *M'* :: ‹'*a set*› **and** *g g'*
using *sum_uniformity*[*OF plus_cont* ‹*eventually D uniformity*›] **by** *blast*
then have *D'_sum_D*: ‹(∀ *m*∈*M*. *D'* (*g m*, *g' m*)) ⟹ *D* (*sum g M*, *sum g' M*)› **for** *g g'*
by *auto*

obtain *Ha* **where** ‹*Ha a* ⊇ *Ga a*› **and** *Ha_fin*: ‹*finite* (*Ha a*)› **and** *Ha_B*: ‹*Ha a* ⊆ *B a*›
and *D'_sum_Ha*: ‹*Ha a* ⊆ *L* ⟹ *L* ⊆ *B a* ⟹ *finite L* ⟹ *D'* (*b a*, *sum* (λ*b*. *f* (*a*,*b*)) *L*)› **if** ‹*a* ∈ *A*› **for** *a L*
proof −
from *sum_b*[*unfolded tendsto_iff_uniformity*, *rule_format*, *OF_ D'_uni*[*THEN uniformity_sym*]]

```
        obtain Ha0 where ‹finite (Ha0 a)› and ‹Ha0 a ⊆ B a›
         and ‹Ha0 a ⊆ L ⟹ L ⊆ B a ⟹ finite L ⟹ D' (b a, sum (λb. f (a,b)) L)› if ‹a ∈ A› for a L
        unfolding FB_def eventually_finite_subsets_at_top unfolding prod.case
by metis
        moreover define Ha where ‹Ha a = Ha0 a ∪ Ga a› for a
        ultimately show ?thesis
          using that[where Ha=Ha]
          using Ga_fin Ga_B by auto
       qed

      have ‹D (∑a∈M. b a, ∑(a,b)∈H. f (a,b))› if ‹finite H› and ‹H ⊆ Sigma M B› and ‹H ⊇ Sigma M Ha› for H
       proof −
        define Ha' where ‹Ha' a = {b| b. (a,b) ∈ H}› for a
        have [simp]: ‹finite (Ha' a)› and [simp]: ‹Ha' a ⊇ Ha a› and [simp]: ‹Ha' a ⊆ B a› if ‹a ∈ M› for a
          unfolding Ha'_def using ‹finite H› ‹H ⊆ Sigma M B› ‹Sigma M Ha ⊆ H› that finite_proj by auto
        have ‹Sigma M Ha' = H›
          using that by (auto simp: Ha'_def)
        then have ∗: ‹(∑(a,b)∈H. f (a,b)) = (∑a∈M. ∑b∈Ha' a. f (a,b))›
          by (simp add: ‹finite M› sum.Sigma)
        have ‹D' (b a, sum (λb. f (a,b)) (Ha' a))› if ‹a ∈ M› for a
          using D'_sum_Ha ‹M ⊆ A› that by auto
        then have ‹D (∑a∈M. b a, ∑a∈M. sum (λb. f (a,b)) (Ha' a))›
          by (rule_tac D'_sum_D, auto)
        with ∗ show ?thesis
          by auto
       qed
       moreover have ‹Sigma M Ha ⊆ Sigma M B›
        using Ha_B ‹M ⊆ A› by auto
       ultimately show ?thesis
        unfolding FMB_def eventually_finite_subsets_at_top
       by (metis (no_types, lifting) Ha_fin finite_SigmaI subsetD that(2) that(3))
      qed
      moreover have ‹eventually (λH. D (∑(a,b)∈H. f (a,b), a)) FMB›
       unfolding FMB_def eventually_finite_subsets_at_top
      proof (rule exI[of _ G], safe)
       fix Y assume Y: finite Y G ⊆ Y Y ⊆ Sigma M B
       thus D (∑(a,b)∈Y. f (a, b), a)
        using G_sum[of Y] Y using that(3) by fastforce
      qed (use ‹finite G› ‹G ⊆ Sigma A B› that in auto)
      ultimately have ‹∀F x in FMB. E (sum b M, a)›
       by eventually_elim (use DDE' in auto)
      then show ‹E (sum b M, a)›
       using FMB_def by force
     qed
     then show ‹∀F x in FA. E (sum b x, a)›
```

```
    using ‹finite (fst ‘ G)› and ‹fst ‘ G ⊆ A›
    by (metis (mono_tags, lifting) FA_def eventually_finite_subsets_at_top)
  qed
  then show ?thesis
    by (simp add: FA_def has_sum_def)
qed

lemma summable_on_Sigma:
  fixes A :: 'a set and B :: 'a ⇒ 'b set
    and f :: ‹'a ⇒ 'b ⇒ 'c::{comm_monoid_add, t2_space, uniform_space}›
  assumes plus_cont: ‹uniformly_continuous_on UNIV (λ(x::'c,y). x+y)›
  assumes summableAB: (λ(x,y). f x y) summable_on (Sigma A B)
  assumes summableB: ‹⋀x. x∈A ⟹ (f x) summable_on (B x)›
  shows ‹(λx. infsum (f x) (B x)) summable_on A›
proof −
  from summableAB obtain a where a: ‹((λ(x,y). f x y) has_sum a) (Sigma A
B)›
    using has_sum_infsum by blast
  from summableB have b: ‹⋀x. x∈A ⟹ (f x has_sum infsum (f x) (B x)) (B
x)›
    by (auto intro!: has_sum_infsum)
  show ?thesis
    using plus_cont a b
    by (smt (verit) has_sum_Sigma[where f=‹λ(x,y). f x y›] has_sum_cong
old.prod.case summable_on_def)
qed

lemma infsum_Sigma:
  fixes A :: 'a set and B :: 'a ⇒ 'b set
    and f :: ‹'a × 'b ⇒ 'c::{comm_monoid_add, t2_space, uniform_space}›
  assumes plus_cont: ‹uniformly_continuous_on UNIV (λ(x::'c,y). x+y)›
  assumes summableAB: f summable_on (Sigma A B)
  assumes summableB: ‹⋀x. x∈A ⟹ (λy. f (x, y)) summable_on (B x)›
  shows infsum f (Sigma A B) = infsum (λx. infsum (λy. f (x, y)) (B x)) A
proof −
  from summableAB have a: ‹(f has_sum infsum f (Sigma A B)) (Sigma A B)›
    using has_sum_infsum by blast
  from summableB have b: ‹⋀x. x∈A ⟹ ((λy. f (x, y)) has_sum infsum (λy. f
(x, y)) (B x)) (B x)›
    by (auto intro!: has_sum_infsum)
  show ?thesis
    using plus_cont a b by (auto intro: infsumI[symmetric] has_sum_Sigma simp:
summable_on_def)
qed

lemma infsum_Sigma':
  fixes A :: 'a set and B :: 'a ⇒ 'b set
    and f :: ‹'a ⇒ 'b ⇒ 'c::{comm_monoid_add, t2_space, uniform_space}›
  assumes plus_cont: ‹uniformly_continuous_on UNIV (λ(x::'c,y). x+y)›
```

assumes *summableAB*: $(\lambda(x,y).\ f\ x\ y)$ *summable_on* (*Sigma A B*)
assumes *summableB*: ‹$\bigwedge x.\ x{\in}A \Longrightarrow (f\ x)$ *summable_on* $(B\ x)$›
shows ‹*infsum* $(\lambda x.$ *infsum* $(f\ x)\ (B\ x))\ A =$ *infsum* $(\lambda(x,y).\ f\ x\ y)$ (*Sigma A B*)›
using *infsum_Sigma*[*of* ‹$\lambda(x,y).\ f\ x\ y$› *A B*]
using *assms* **by** *auto*

A special case of *infsum_Sigma* etc. for Banach spaces. It has less premises.

lemma
fixes $A :: 'a$ *set* **and** $B :: 'a \Rightarrow 'b$ *set*
and $f ::$ ‹$'a \Rightarrow 'b \Rightarrow 'c$::*banach*›
assumes [*simp*]: $(\lambda(x,y).\ f\ x\ y)$ *summable_on* (*Sigma A B*)
shows *infsum_Sigma'_banach*: ‹*infsum* $(\lambda x.$ *infsum* $(f\ x)\ (B\ x))\ A =$ *infsum* $(\lambda(x,y).\ f\ x\ y)$ (*Sigma A B*)› (**is** *?thesis1*)
and *summable_on_Sigma_banach*: ‹$(\lambda x.$ *infsum* $(f\ x)\ (B\ x))$ *summable_on A*› (**is** *?thesis2*)
proof −
have *fsum*: ‹$(f\ x)$ *summable_on* $(B\ x)$› **if** ‹$x \in A$› **for** x
proof −
from *assms*
have ‹$(\lambda(x,y).\ f\ x\ y)$ *summable_on* (*Pair x* ‘ $B\ x$)›
by (*meson image_subset_iff summable_on_subset_banach mem_Sigma_iff that*)
then have ‹$((\lambda(x,y).\ f\ x\ y) \circ$ *Pair* $x)$ *summable_on* $(B\ x)$›
by (*metis summable_on_reindex inj_on_def prod.inject*)
then show *?thesis*
by (*auto simp*: *o_def*)
qed
show *?thesis1*
using *fsum assms infsum_Sigma' isUCont_plus* **by** *blast*
show *?thesis2*
using *fsum assms isUCont_plus summable_on_Sigma* **by** *blast*
qed

lemma *infsum_Sigma_banach*:
fixes $A :: 'a$ *set* **and** $B :: 'a \Rightarrow 'b$ *set*
and $f ::$ ‹$'a \times 'b \Rightarrow 'c$::*banach*›
assumes [*simp*]: *f summable_on* (*Sigma A B*)
shows ‹*infsum* $(\lambda x.$ *infsum* $(\lambda y.\ f\ (x,y))\ (B\ x))\ A =$ *infsum f* (*Sigma A B*)›
using *assms* **by** (*simp add*: *infsum_Sigma'_banach*)

lemma *infsum_swap*:
fixes $A :: 'a$ *set* **and** $B :: 'b$ *set*
fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c$::{*comm_monoid_add,t2_space,uniform_space*}
assumes *plus_cont*: ‹*uniformly_continuous_on UNIV* $(\lambda(x{::}'c,y).\ x{+}y)$›
assumes ‹$(\lambda(x,\ y).\ f\ x\ y)$ *summable_on* $(A \times B)$›
assumes ‹$\bigwedge a.\ a{\in}A \Longrightarrow (f\ a)$ *summable_on B*›
assumes ‹$\bigwedge b.\ b{\in}B \Longrightarrow (\lambda a.\ f\ a\ b)$ *summable_on A*›
shows ‹*infsum* $(\lambda x.$ *infsum* $(\lambda y.\ f\ x\ y)\ B)\ A =$ *infsum* $(\lambda y.$ *infsum* $(\lambda x.\ f\ x\ y)$

```
A) B›
proof −
  have (λ(x, y). f y x) ∘ prod.swap summable_on A × B
    by (simp add: assms(2) summable_on_cong)
  then have fyx: ‹(λ(x, y). f y x) summable_on (B × A)›
   by (metis has_sum_reindex infsum_reindex inj_swap product_swap summable_iff_has_sum_infsum
  have ‹infsum (λx. infsum (λy. f x y) B) A = infsum (λ(x,y). f x y) (A × B)›
    using assms infsum_Sigma' by blast
  also have ‹... = infsum (λ(x,y). f y x) (B × A)›
    apply (subst product_swap[symmetric])
    apply (subst infsum_reindex)
    using assms by (auto simp: o_def)
  also have ‹... = infsum (λy. infsum (λx. f x y) A) B›
    by (smt (verit) fyx assms(1) assms(4) infsum_Sigma' infsum_cong)
  finally show ?thesis .
qed

lemma infsum_swap_banach:
  fixes A :: 'a set and B :: 'b set
  fixes f :: 'a ⇒ 'b ⇒ 'c::banach
  assumes ‹(λ(x, y). f x y) summable_on (A × B)›
  shows infsum (λx. infsum (λy. f x y) B) A = infsum (λy. infsum (λx. f x y) A)
B
proof −
  have §: ‹(λ(x, y). f y x) summable_on (B × A)›
     by (metis (mono_tags, lifting) assms case_swap inj_swap o_apply prod-
uct_swap summable_on_cong summable_on_reindex)
  have ‹infsum (λx. infsum (λy. f x y) B) A = infsum (λ(x,y). f x y) (A × B)›
    using assms infsum_Sigma'_banach by blast
  also have ‹... = infsum (λ(x,y). f y x) (B × A)›
    apply (subst product_swap[symmetric])
    apply (subst infsum_reindex)
    using assms by (auto simp: o_def)
  also have ‹... = infsum (λy. infsum (λx. f x y) A) B›
    by (metis (mono_tags, lifting) § infsum_Sigma'_banach infsum_cong)
  finally show ?thesis .
qed

lemma nonneg_infsum_le_0D:
 fixes f :: 'a ⇒ 'b::{topological_ab_group_add,ordered_ab_group_add,linorder_topology}
  assumes infsum f A ≤ 0
    and abs_sum: f summable_on A
    and nneg: ⋀x. x ∈ A ⟹ f x ≥ 0
    and x ∈ A
  shows f x = 0
proof (rule ccontr)
  assume ‹f x ≠ 0›
  have ex: ‹f summable_on (A−{x})›
    by (rule summable_on_cofin_subset) (use assms in auto)
```

have *pos*: ‹*infsum f* (*A* − {*x*}) ≥ *0*›
by (*rule infsum_nonneg*) (*use nneg* **in** *auto*)

have [*trans*]: ‹*x* ≥ *y* ⟹ *y* > *z* ⟹ *x* > *z*› **for** *x y z* :: *'b* **by** *auto*

have ‹*infsum f A* = *infsum f* (*A*−{*x*}) + *infsum f* {*x*}›
by (*subst infsum_Un_disjoint*[*symmetric*]) (*use assms ex* **in** ‹*auto simp*: *insert_absorb*›)
also have ‹... ≥ *infsum f* {*x*}› (**is** ‹_ ≥ ...›)
using *pos* **by** (*rule add_increasing*) *simp*
also have ‹... = *f x*› (**is** ‹_ = ...›)
by (*subst infsum_finite*) *auto*
also have ‹... > *0*›
using ‹*f x* ≠ *0*› *assms*(*4*) *nneg* **by** *fastforce*
finally show *False*
using *assms* **by** *auto*
qed

lemma *nonneg_has_sum_le_0D*:
fixes *f* :: *'a* ⇒ *'b*::{*topological_ab_group_add*,*ordered_ab_group_add*,*linorder_topology*}
assumes (*f has_sum a*) *A* ‹*a* ≤ *0*›
and ⋀*x*. *x* ∈ *A* ⟹ *f x* ≥ *0*
and *x* ∈ *A*
shows *f x* = *0*
by (*metis assms infsumI nonneg_infsum_le_0D summable_on_def*)

lemma *has_sum_cmult_left*:
fixes *f* :: *'a* ⇒ *'b* :: {*topological_semigroup_mult*, *semiring_0*}
assumes ‹(*f has_sum a*) *A*›
shows ((λ*x*. *f x* ∗ *c*) *has_sum* (*a* ∗ *c*)) *A*
using *assms tendsto_mult_right*
by (*force simp add*: *has_sum_def sum_distrib_right*)

lemma *infsum_cmult_left*:
fixes *f* :: *'a* ⇒ *'b* :: {*t2_space*, *topological_semigroup_mult*, *semiring_0*}
assumes ‹*c* ≠ *0* ⟹ *f summable_on A*›
shows *infsum* (λ*x*. *f x* ∗ *c*) *A* = *infsum f A* ∗ *c*
using *assms has_sum_cmult_left infsumI summable_iff_has_sum_infsum* **by** *fastforce*

lemma *summable_on_cmult_left*:
fixes *f* :: *'a* ⇒ *'b* :: {*t2_space*, *topological_semigroup_mult*, *semiring_0*}
assumes ‹*f summable_on A*›
shows (λ*x*. *f x* ∗ *c*) *summable_on A*
using *assms summable_on_def has_sum_cmult_left* **by** *blast*

lemma *has_sum_cmult_right*:
fixes *f* :: *'a* ⇒ *'b* :: {*topological_semigroup_mult*, *semiring_0*}
assumes ‹(*f has_sum a*) *A*›

shows $((\lambda x.\ c * f\ x)\ has\_sum\ (c * a))\ A$
using *assms tendsto_mult_left*
by (*force simp add*: *has_sum_def sum_distrib_left*)

lemma *infsum_cmult_right*:
fixes $f :: 'a \Rightarrow 'b :: \{t2\_space, topological\_semigroup\_mult, semiring\_0\}$
assumes ‹$c \neq 0 \Longrightarrow f\ summable\_on\ A$›
shows ‹$infsum\ (\lambda x.\ c * f\ x)\ A = c * infsum\ f\ A$›
using *assms has_sum_cmult_right infsumI summable_iff_has_sum_infsum* **by** *fastforce*

lemma *summable_on_cmult_right*:
fixes $f :: 'a \Rightarrow 'b :: \{t2\_space, topological\_semigroup\_mult, semiring\_0\}$
assumes ‹$f\ summable\_on\ A$›
shows $(\lambda x.\ c * f\ x)\ summable\_on\ A$
using *assms summable_on_def has_sum_cmult_right* **by** *blast*

lemma *summable_on_cmult_left'*:
fixes $f :: 'a \Rightarrow 'b :: \{t2\_space, topological\_semigroup\_mult, division\_ring\}$
assumes ‹$c \neq 0$›
shows $(\lambda x.\ f\ x * c)\ summable\_on\ A \longleftrightarrow f\ summable\_on\ A$
proof
assume ‹$f\ summable\_on\ A$›
then show ‹$(\lambda x.\ f\ x * c)\ summable\_on\ A$›
by (*rule summable_on_cmult_left*)
next
assume ‹$(\lambda x.\ f\ x * c)\ summable\_on\ A$›
then have ‹$(\lambda x.\ f\ x * c * inverse\ c)\ summable\_on\ A$›
by (*rule summable_on_cmult_left*)
then show ‹$f\ summable\_on\ A$›
by (*smt* (*verit*, *del_insts*) *assms divide_inverse nonzero_divide_eq_eq summable_on_cong*)
qed

lemma *summable_on_cmult_right'*:
fixes $f :: 'a \Rightarrow 'b :: \{t2\_space, topological\_semigroup\_mult, division\_ring\}$
assumes ‹$c \neq 0$›
shows $(\lambda x.\ c * f\ x)\ summable\_on\ A \longleftrightarrow f\ summable\_on\ A$
by (*metis* (*no_types*, *lifting*) *assms left_inverse mult.assoc mult_1 summable_on_cmult_right summable_on_cong*)

lemma *infsum_cmult_left'*:
fixes $f :: 'a \Rightarrow 'b :: \{t2\_space, topological\_semigroup\_mult, division\_ring\}$
shows $infsum\ (\lambda x.\ f\ x * c)\ A = infsum\ f\ A * c$
by (*metis* (*full_types*) *infsum_cmult_left infsum_not_exists mult_eq_0_iff summable_on_cmult_left*

lemma *infsum_cmult_right'*:
fixes $f :: 'a \Rightarrow 'b :: \{t2\_space, topological\_semigroup\_mult, division\_ring\}$
shows $infsum\ (\lambda x.\ c * f\ x)\ A = c * infsum\ f\ A$
by (*metis* (*full_types*) *infsum_cmult_right infsum_not_exists mult_eq_0_iff*

summable_on_cmult_right')

lemma *has_sum_constant*[*simp*]:
assumes ‹*finite F*›
shows ‹((λ_. *c*) *has_sum of_nat* (*card F*) ∗ *c*) *F*›
by (*metis assms has_sum_finite sum_constant*)

lemma *infsum_constant*[*simp*]:
assumes ‹*finite F*›
shows ‹*infsum* (λ_. *c*) *F* = *of_nat* (*card F*) ∗ *c*›
by (*simp add*: *assms*)

lemma *infsum_diverge_constant*:
— This probably does not really need all of *archimedean_field* but Isabelle/HOL has no type class such as, e.g., "archimedean ring".
fixes *c* :: ‹'*a*::{*archimedean_field*, *comm_monoid_add*, *linorder_topology*, *topological_semigroup_mult*}›
assumes ‹*infinite A*› **and** ‹*c* ≠ *0*›
shows ‹¬ (λ_. *c*) *summable_on A*›
proof (*rule notI*)
assume ‹(λ_. *c*) *summable_on A*›
then have ‹(λ_. *inverse c* ∗ *c*) *summable_on A*›
by (*rule summable_on_cmult_right*)
then have [*simp*]: ‹(λ_. *1*::'*a*) *summable_on A*›
using *assms* **by** *auto*
have ‹*infsum* (λ_. *1*) *A* ≥ *d*› **for** *d* :: '*a*
proof −
obtain *n* :: *nat* **where** ‹*of_nat n* ≥ *d*›
by (*meson real_arch_simple*)
from *assms*
obtain *F* **where** ‹*F* ⊆ *A*› **and** ‹*finite F*› **and** ‹*card F* = *n*›
by (*meson infinite_arbitrarily_large*)
note ‹*d* ≤ *of_nat n*›
also have ‹*of_nat n* = *infsum* (λ_. *1*::'*a*) *F*›
by (*simp add*: ‹*card F* = *n*› ‹*finite F*›)
also have ‹... ≤ *infsum* (λ_. *1*::'*a*) *A*›
apply (*rule infsum_mono_neutral*)
using ‹*finite F*› ‹*F* ⊆ *A*› **by** *auto*
finally show *?thesis* **.**
qed
then show *False*
by (*meson linordered_field_no_ub not_less*)
qed

lemma *has_sum_constant_archimedean*[*simp*]:
— This probably does not really need all of *archimedean_field* but Isabelle/HOL has no type class such as, e.g., "archimedean ring".
fixes *c* :: ‹'*a*::{*archimedean_field*, *comm_monoid_add*, *linorder_topology*, *topological_semigroup_mult*}›

shows ‹*infsum* (λ_. *c*) *A* = *of_nat* (*card A*) * *c*›
by (*metis infsum_0 infsum_constant infsum_diverge_constant infsum_not_exists sum.infinite sum_constant*)

lemma *has_sum_uminus*:
fixes *f* :: ‹'*a* ⇒ '*b*::*topological_ab_group_add*›
shows ‹((λ*x*. − *f x*) *has_sum a*) *A* ⟷ (*f has_sum* (− *a*)) *A*›
by (*auto simp add*: *sum_negf*[*abs_def*] *tendsto_minus_cancel_left has_sum_def*)

lemma *summable_on_uminus*:
fixes *f* :: ‹'*a* ⇒ '*b*::*topological_ab_group_add*›
shows‹(λ*x*. − *f x*) *summable_on A* ⟷ *f summable_on A*›
by (*metis summable_on_def has_sum_uminus verit_minus_simplify*(*4*))

lemma *infsum_uminus*:
fixes *f* :: ‹'*a* ⇒ '*b*::{*topological_ab_group_add*, *t2_space*}›
shows ‹*infsum* (λ*x*. − *f x*) *A* = − *infsum f A*›
by (*metis* (*full_types*) *add.inverse_inverse add.inverse_neutral infsumI infsum_def has_sum_infsum has_sum_uminus*)

lemma *has_sum_le_finite_sums*:
fixes *a* :: ‹'*a*::{*comm_monoid_add*,*topological_space*,*linorder_topology*}›
assumes ‹(*f has_sum a*) *A*›
assumes ‹⋀*F*. *finite F* ⟹ *F* ⊆ *A* ⟹ *sum f F* ≤ *b*›
shows ‹*a* ≤ *b*›
by (*metis assms eventually_finite_subsets_at_top_weakI finite_subsets_at_top_neq_bot has_sum_def tendsto_upperbound*)

lemma *infsum_le_finite_sums*:
fixes *b* :: ‹'*a*::{*comm_monoid_add*,*topological_space*,*linorder_topology*}›
assumes ‹*f summable_on A*›
assumes ‹⋀*F*. *finite F* ⟹ *F* ⊆ *A* ⟹ *sum f F* ≤ *b*›
shows ‹*infsum f A* ≤ *b*›
by (*meson assms has_sum_infsum has_sum_le_finite_sums*)

lemma *summable_on_scaleR_left* [*intro*]:
fixes *c* :: ‹'*a* :: *real_normed_vector*›
assumes *c* ≠ *0* ⟹ *f summable_on A*
shows (λ*x*. *f x* *$_R$ *c*) *summable_on A*
proof (*cases* ‹*c* = *0*›)
case *False*
then have (λ*y*. *y* *$_R$ *c*) ∘ *f summable_on A*
using *assms* **by** (*auto simp add*: *scaleR_left.additive_axioms summable_on_comm_additive*)
then show *?thesis*
by (*metis* (*mono_tags*, *lifting*) *comp_apply summable_on_cong*)
qed *auto*

lemma *summable_on_scaleR_right* [*intro*]:
fixes $f :: \langle 'a \Rightarrow 'b :: real\_normed\_vector \rangle$
assumes $c \neq 0 \Longrightarrow f$ *summable_on* A
shows $(\lambda x.\ c *_R f\ x)$ *summable_on* A
proof (*cases* $\langle c = 0 \rangle$)
case *False*
then have $(*_R)\ c \circ f$ *summable_on* A
using *assms* **by** (*auto simp add*: *scaleR_right.additive_axioms summable_on_comm_additive*)
then show *?thesis*
by (*metis* (*mono_tags*, *lifting*) *comp_apply summable_on_cong*)
qed *auto*

lemma *infsum_scaleR_left*:
fixes $c :: \langle 'a :: real\_normed\_vector \rangle$
assumes $c \neq 0 \Longrightarrow f$ *summable_on* A
shows *infsum* $(\lambda x.\ f\ x *_R c)\ A =$ *infsum* $f\ A *_R c$
proof (*cases* $\langle c = 0 \rangle$)
case *False*
then have *infsum* $((\lambda y.\ y *_R c) \circ f)\ A =$ *infsum* $f\ A *_R c$
using *assms* **by** (*auto simp add*: *scaleR_left.additive_axioms infsum_comm_additive*)
then show *?thesis*
by (*metis* (*mono_tags*, *lifting*) *comp_apply infsum_cong*)
qed *auto*

lemma *infsum_scaleR_right*:
fixes $f :: \langle 'a \Rightarrow 'b :: real\_normed\_vector \rangle$
shows *infsum* $(\lambda x.\ c *_R f\ x)\ A = c *_R$ *infsum* $f\ A$
proof −
consider (*summable*) $\langle f$ *summable_on* $A \rangle$ | (*c0*) $\langle c = 0 \rangle$ | (*not_summable*) $\langle \neg f$ *summable_on* $A \rangle\ \langle c \neq 0 \rangle$
by *auto*
then show *?thesis*
proof *cases*
case *summable*
then have *infsum* $((*_R)\ c \circ f)\ A = c *_R$ *infsum* $f\ A$
by (*auto simp add*: *scaleR_right.additive_axioms infsum_comm_additive*)
then show *?thesis*
by (*metis* (*mono_tags*, *lifting*) *comp_apply infsum_cong*)
next
case *c0*
then show *?thesis* **by** *auto*
next
case *not_summable*
have $\langle \neg\ (\lambda x.\ c *_R f\ x)$ *summable_on* $A \rangle$
proof (*rule notI*)
assume $\langle (\lambda x.\ c *_R f\ x)$ *summable_on* $A \rangle$
then have $\langle (\lambda x.\ inverse\ c *_R c *_R f\ x)$ *summable_on* $A \rangle$
using *summable_on_scaleR_right* **by** *blast*
with *not_summable* **show** *False*

```
      by simp
   qed
   then show ?thesis
     by (simp add: infsum_not_exists not_summable(1))
 qed
qed


lemma infsum_Un_Int:
 fixes f :: 'a ⇒ 'b::{topological_ab_group_add, t2_space}
 assumes f summable_on A − B f summable_on B − A ‹f summable_on A ∩
B›
 shows   infsum f (A ∪ B) = infsum f A + infsum f B − infsum f (A ∩ B)
proof −
 obtain ‹f summable_on A› ‹f summable_on B›
  using assms by (metis Int_Diff_Un Int_Diff_disjoint inf_commute summable_on_Un_disjoint)
 then have ‹infsum f (A ∪ B) = infsum f A + infsum f (B − A)›
   using assms(2) infsum_Un_disjoint by fastforce
 moreover have ‹infsum f (B − A) = infsum f B − infsum f (A ∩ B)›
   using assms by (metis Diff_Int2 Un_Int_eq(2) ‹f summable_on B› inf_le2
infsum_Diff)
 ultimately show ?thesis
   by auto
qed

lemma inj_combinator':
 assumes x ∉ F
 shows ‹inj_on (λ(g, y). g(x := y)) (Pi_E F B × B x)›
proof −
 have inj_on ((λ(y, g). g(x := y)) ∘ prod.swap) (Pi_E F B × B x)
   using inj_combinator[of x F B] assms by (intro comp_inj_on) (auto simp:
product_swap)
 thus ?thesis
   by (simp add: o_def)
qed

lemma infsum_prod_PiE:
 — See also infsum_prod_PiE_abs below with incomparable premises.
 fixes f :: 'a ⇒ 'b ⇒ 'c :: {comm_monoid_mult, topological_semigroup_mult,
division_ring, banach}
 assumes finite: finite A
 assumes ⋀x. x ∈ A ⟹ f x summable_on B x
 assumes (λg. ∏ x∈A. f x (g x)) summable_on (PiE A B)
 shows   infsum (λg. ∏ x∈A. f x (g x)) (PiE A B) = (∏ x∈A. infsum (f x) (B
x))
proof (use finite assms(2−) in induction)
 case empty
 then show ?case
   by auto
```

next
case (*insert x F*)
have *pi*: ‹Pi_E (*insert x F*) *B* = (λ(*g*,*y*). *g*(*x*:=*y*)) ' (Pi_E *F B* × *B x*)›
unfolding *PiE_insert_eq*
by (*subst swap_product* [*symmetric*]) (*simp add*: *image_image case_prod_unfold*)
have *prod*: ‹(∏ *x*′∈*F*. *f x*′ ((*p*(*x*:=*y*)) *x*′)) = (∏ *x*′∈*F*. *f x*′ (*p x*′))› **for** *p y*
by (*rule prod.cong*) (*use insert.hyps* **in** *auto*)
have *inj*: ‹*inj_on* (λ(*g*, *y*). *g*(*x* := *y*)) (Pi_E *F B* × *B x*)›
using ‹*x* ∉ *F*› **by** (*rule inj_combinator*′)

have *summable1*: ‹(λ*g*. ∏ *x*∈*insert x F*. *f x* (*g x*)) *summable_on* Pi_E (*insert x F*) *B*›
using *insert.prems*(*2*) **.**
also have ‹Pi_E (*insert x F*) *B* = (λ(*g*,*y*). *g*(*x*:=*y*)) ' (Pi_E *F B* × *B x*)›
by (*simp only*: *pi*)
also have (λ*g*. ∏ *x*∈*insert x F*. *f x* (*g x*)) *summable_on* ... ⟷
((λ*g*. ∏ *x*∈*insert x F*. *f x* (*g x*)) ∘ (λ(*g*,*y*). *g*(*x*:=*y*))) *summable_on* (Pi_E *F B* × *B x*)
using *inj* **by** (*rule summable_on_reindex*)
also have (∏ *z*∈*F*. *f z* ((*g*(*x* := *y*)) *z*)) = (∏ *z*∈*F*. *f z* (*g z*)) **for** *g y*
using *insert.hyps* **by** (*intro prod.cong*) *auto*
hence ((λ*g*. ∏ *x*∈*insert x F*. *f x* (*g x*)) ∘ (λ(*g*,*y*). *g*(*x*:=*y*))) =
(λ(*p*, *y*). *f x y* ∗ (∏ *x*′∈*F*. *f x*′ (*p x*′)))
using *insert.hyps* **by** (*auto simp*: *fun_eq_iff cong*: *prod.cong_simp*)
finally have *summable2*: ‹(λ(*p*, *y*). *f x y* ∗ (∏ *x*′∈*F*. *f x*′ (*p x*′))) *summable_on* Pi_E *F B* × *B x*› **.**

then have ‹(λ*p*. ∑∞*y*∈*B x*. *f x y* ∗ (∏ *x*′∈*F*. *f x*′ (*p x*′))) *summable_on* Pi_E *F B*›
by (*rule summable_on_Sigma_banach*)
then have ‹(λ*p*. (∑∞*y*∈*B x*. *f x y*) ∗ (∏ *x*′∈*F*. *f x*′ (*p x*′))) *summable_on* Pi_E *F B*›
by (*metis* (*mono_tags*, *lifting*) *infsum_cmult_left*′ *infsum_cong summable_on_cong*)
then have *summable3*: ‹(λ*p*. (∏ *x*′∈*F*. *f x*′ (*p x*′))) *summable_on* Pi_E *F B*› **if** ‹(∑∞*y*∈*B x*. *f x y*) ≠ *0*›
using *summable_on_cmult_right*′ *that* **by** *blast*

have ‹(∑∞*g*∈Pi_E (*insert x F*) *B*. ∏ *x*∈*insert x F*. *f x* (*g x*))
= (∑∞(*p*,*y*)∈Pi_E *F B* × *B x*. ∏ *x*′∈*insert x F*. *f x*′ ((*p*(*x*:=*y*)) *x*′))›
by (*smt* (*verit*, *ccfv_SIG*) *comp_apply infsum_cong infsum_reindex inj pi prod.cong split_def*)
also have ‹... = (∑∞(*p*, *y*)∈Pi_E *F B* × *B x*. *f x y* ∗ (∏ *x*′∈*F*. *f x*′ ((*p*(*x*:=*y*)) *x*′)))›
using *insert.hyps* **by** *auto*
also have ‹... = (∑∞(*p*, *y*)∈Pi_E *F B* × *B x*. *f x y* ∗ (∏ *x*′∈*F*. *f x*′ (*p x*′)))›
using *prod* **by** *presburger*
also have ‹... = (∑∞*p*∈Pi_E *F B*. ∑∞*y*∈*B x*. *f x y* ∗ (∏ *x*′∈*F*. *f x*′ (*p x*′)))›
using *infsum_Sigma*′*_banach summable2* **by** *force*
also have ‹... = (∑∞*y*∈*B x*. *f x y*) ∗ (∑∞*p*∈Pi_E *F B*. ∏ *x*′∈*F*. *f x*′ (*p x*′))›

by (*smt* (*verit*) *infsum_cmult_left'* *infsum_cmult_right'* *infsum_cong*)
also have ‹... = (∏ x∈*insert x F*. *infsum* (*f x*) (*B x*))›
using *insert summable3* **by** *auto*
finally show *?case*
by *simp*
qed

lemma *infsum_prod_PiE_abs*:
— See also *infsum_prod_PiE* above with incomparable premises.
fixes *f* :: *'a* ⇒ *'b* ⇒ *'c* :: {*banach*, *real_normed_div_algebra*, *comm_semiring_1*}
assumes *finite*: *finite A*
assumes ⋀*x*. *x* ∈ *A* ⟹ *f x abs_summable_on B x*
shows *infsum* (λ*g*. ∏ *x*∈*A*. *f x* (*g x*)) (*PiE A B*) = (∏ *x*∈*A*. *infsum* (*f x*) (*B x*))
proof (*use finite assms*(*2*) **in** *induction*)
case *empty*
then show *?case*
by *auto*
next
case (*insert x A*)

have *pi*: ‹*Pi*$_E$ (*insert x F*) *B* = (λ(*g*,*y*). *g*(*x*:=*y*)) ‘ (*Pi*$_E$ *F B* × *B x*)› **for** *x F* **and** *B* :: *'a* ⇒ *'b set*
unfolding *PiE_insert_eq*
by (*subst swap_product* [*symmetric*]) (*simp add*: *image_image case_prod_unfold*)
have *prod*: ‹(∏ *x'*∈*A*. *f x'* ((*p*(*x*:=*y*)) *x'*)) = (∏ *x'*∈*A*. *f x'* (*p x'*))› **for** *p y*
by (*rule prod.cong*) (*use insert.hyps* **in** *auto*)
have *inj*: ‹*inj_on* (λ(*g*, *y*). *g*(*x* := *y*)) (*Pi*$_E$ *A B* × *B x*)›
using ‹*x* ∉ *A*› **by** (*rule inj_combinator'*)

define *s* **where** ‹*s x* = *infsum* (λ*y*. *norm* (*f x y*)) (*B x*)› **for** *x*

have ‹(∑ *p*∈*P*. *norm* (∏ *x*∈*F*. *f x* (*p x*))) ≤ *prod s F*›
if *P*: ‹*P* ⊆ *Pi*$_E$ *F B*› **and** [*simp*]: ‹*finite P*› ‹*finite F*›
and *sum*: ‹⋀*x*. *x* ∈ *F* ⟹ *f x abs_summable_on B x*› **for** *P F*
proof −
define *B'* **where** ‹*B' x* = {*p x*| *p*. *p*∈*P*}› **for** *x*
have *fin_B'*[*simp*]: ‹*finite* (*B' x*)› **for** *x*
using *that* **by** (*auto simp*: *B'_def*)
have [*simp*]: ‹*finite* (*Pi*$_E$ *F B'*)›
by (*simp add*: *finite_PiE*)
have [*simp*]: ‹*P* ⊆ *Pi*$_E$ *F B'*›
using *that* **by** (*auto simp*: *B'_def*)
have *B'B*: ‹*B' x* ⊆ *B x*› **if** ‹*x* ∈ *F*› **for** *x*
unfolding *B'_def* **using** *P that*
by *auto*
have *s_bound*: ‹(∑ *y*∈*B' x*. *norm* (*f x y*)) ≤ *s x*› **if** ‹*x* ∈ *F*› **for** *x*
by (*metis B'B fin_B' finite_sum_le_has_sum has_sum_infsum norm_ge_zero s_def sum that*)

```
    have ‹(∑ p∈P. norm (∏ x∈F. f x (p x))) ≤ (∑ p∈Pi_E F B'. norm (∏ x∈F. f
x (p x)))›
      by (simp add: sum_mono2)
    also have ‹... = (∑ p∈Pi_E F B'. ∏ x∈F. norm (f x (p x)))›
      by (simp add: prod_norm)
    also have ‹... = (∏ x∈F. ∑ y∈B' x. norm (f x y))›
    proof (use ‹finite F› in induction)
      case empty
      then show ?case by simp
    next
      case (insert x F)
      have inj: ‹inj_on (λ(g, y). g(x := y)) (Pi_E F B' × B' x)›
        by (simp add: inj_combinator' insert.hyps)
      then have ‹(∑ p∈Pi_E (insert x F) B'. ∏ x∈insert x F. norm (f x (p x)))
          = (∑ (p,y)∈Pi_E F B' × B' x. ∏ x'∈insert x F. norm (f x' ((p(x := y))
x')))›
        by (simp add: pi sum.reindex case_prod_unfold)
      also have ‹... = (∑ (p, y)∈Pi_E F B' × B' x. norm (f x y) * (∏ x'∈F. norm
(f x' (p x'))))›
        by (smt (verit, del_insts) fun_upd_apply insert.hyps prod.cong prod.insert
split_def sum.cong)
      also have ‹... = (∑ y∈B' x. norm (f x y)) * (∑ p∈Pi_E F B'. ∏ x'∈F. norm
(f x' (p x')))›
      by (simp add: sum_product sum.swap [of_ Pi_E F B'] sum.cartesian_product)
      also have ‹... = (∏ x∈insert x F. ∑ y∈B' x. norm (f x y))›
        using insert by force
      finally show ?case .
    qed
    also have ‹... ≤ (∏ x∈F. s x)›
      using s_bound by (simp add: prod_mono sum_nonneg)
    finally show ?thesis .
  qed
  then have bdd_above
     (sum (λg. norm (∏ x∈insert x A. f x (g x))) ` {F. F ⊆ Pi_E (insert x A) B ∧
finite F})
    using insert.hyps insert.prems by (intro bdd_aboveI) blast
  then have ‹(λg. ∏ x∈insert x A. f x (g x)) abs_summable_on Pi_E (insert x A)
B›
    using nonneg_bdd_above_summable_on
    by (metis (mono_tags, lifting) Collect_cong norm_ge_zero)
  also have ‹Pi_E (insert x A) B = (λ(g,y). g(x:=y)) ` (Pi_E A B × B x)›
    by (simp only: pi)
  also have (λg. ∏ x∈insert x A. f x (g x)) abs_summable_on ... ⟷
            ((λg. ∏ x∈insert x A. f x (g x)) ∘ (λ(g,y). g(x:=y))) abs_summable_on
(Pi_E A B × B x)
    using inj by (subst summable_on_reindex) (auto simp: o_def)
  also have (∏ z∈A. f z ((g(x := y)) z)) = (∏ z∈A. f z (g z)) for g y
    using insert.hyps by (intro prod.cong) auto
  hence ((λg. ∏ x∈insert x A. f x (g x)) ∘ (λ(g,y). g(x:=y))) =
```

(λ(p, y). f x y ∗ (∏ x′∈A. f x′ (p x′)))
using *insert.hyps* **by** (*auto simp*: *fun_eq_iff cong*: *prod.cong_simp*)
finally have *summable2*: ‹(λ(p, y). f x y ∗ (∏ x′∈A. f x′ (p x′))) *abs_summable_on* Pi_E A B × B x› **.**

have ‹(∑∞g∈Pi_E (*insert* x A) B. ∏ x∈*insert* x A. f x (g x))
= (∑∞(p,y)∈Pi_E A B × B x. ∏ x′∈*insert* x A. f x′ ((p(x:=y)) x′))›
using *inj* **by** (*simp add*: *pi infsum_reindex o_def case_prod_unfold*)
also have ‹... = (∑∞(p,y) ∈ Pi_E A B × B x. f x y ∗ (∏ x′∈A. f x′ (p x′)))›
using *prod insert.hyps* **by** *auto*
also have ‹... = (∑∞p∈Pi_E A B. ∑∞y∈B x. f x y ∗ (∏ x′∈A. f x′ (p x′)))›
using *abs_summable_summable infsum_Sigma′_banach summable2* **by** *fastforce*
also have ‹... = (∑∞y∈B x. f x y) ∗ (∑∞p∈Pi_E A B. ∏ x′∈A. f x′ (p x′))›
by (*smt* (*verit*, *best*) *infsum_cmult_left′ infsum_cmult_right′ infsum_cong*)
finally show *?case*
by (*simp add*: *insert*)
qed

5.8.3 Absolute convergence

lemma *abs_summable_countable*:
assumes ‹f *abs_summable_on* A›
shows ‹*countable* {x∈A. f x ≠ 0}›
proof −
have *fin*: ‹*finite* {x∈A. *norm* (f x) ≥ t}› **if** ‹t > 0› **for** t
proof (*rule ccontr*)
assume ∗: ‹*infinite* {x ∈ A. t ≤ *norm* (f x)}›
have ‹*infsum* (λx. *norm* (f x)) A ≥ b› **for** b
proof −
obtain b′ **where** b′: ‹*of_nat* b′ ≥ b / t›
by (*meson real_arch_simple*)
from ∗
obtain F **where** *cardF*: ‹*card* F ≥ b′› **and** ‹*finite* F› **and** F: ‹F ⊆ {x ∈ A. t ≤ *norm* (f x)}›
by (*meson finite_if_finite_subsets_card_bdd nle_le*)
have ‹b ≤ *of_nat* b′ ∗ t›
using b′ ‹t > 0› **by** (*simp add*: *field_simps split*: *if_splits*)
also have ‹... ≤ *of_nat* (*card* F) ∗ t›
by (*simp add*: *cardF that*)
also have ‹... = *sum* (λx. t) F›
by *simp*
also have ‹... ≤ *sum* (λx. *norm* (f x)) F›
by (*metis* (*mono_tags*, *lifting*) F *in_mono mem_Collect_eq sum_mono*)
also have ‹... = *infsum* (λx. *norm* (f x)) F›
using ‹*finite* F› **by** (*rule infsum_finite*[*symmetric*])
also have ‹... ≤ *infsum* (λx. *norm* (f x)) A›
by (*rule infsum_mono_neutral*) (*use* ‹*finite* F› *assms* F **in** *auto*)
finally show *?thesis* **.**

```
    qed
    then show False
      by (meson gt_ex linorder_not_less)
  qed
  have ‹countable (⋃i∈{1..}. {x∈A. norm (f x) ≥ 1/of_nat i})›
    by (rule countable_UN) (use fin in ‹auto intro!: countable_finite›)
  also have ‹... = {x∈A. f x ≠ 0}›
  proof safe
    fix x assume x: x ∈ A f x ≠ 0
    define i where i = max 1 (nat (ceiling (1 / norm (f x))))
    have i ≥ 1
      by (simp add: i_def)
    moreover have real i ≥ 1 / norm (f x)
      unfolding i_def by linarith
    hence 1 / real i ≤ norm (f x) using ‹f x ≠ 0›
      by (auto simp: divide_simps mult_ac)
    ultimately show x ∈ (⋃i∈{1..}. {x ∈ A. 1 / real i ≤ norm (f x)})
      using ‹x ∈ A› by auto
  qed auto
  finally show ?thesis .
qed


lemma summable_on_iff_abs_summable_on_real:
  fixes f :: ‹'a ⇒ real›
  shows ‹f summable_on A ⟷ f abs_summable_on A›
proof (rule iffI)
  assume ‹f summable_on A›
  define n A_p A_n
    where ‹n ≡ λx. norm (f x)› and ‹A_p = {x∈A. f x ≥ 0}› and ‹A_n = {x∈A. f
x < 0}› for x
  have A: ‹A_p ∪ A_n = A› ‹A_p ∩ A_n = {}›
    by (auto simp: A_p_def A_n_def)
  from ‹f summable_on A› have ‹f summable_on A_p› ‹f summable_on A_n›
    using A_p_def A_n_def summable_on_subset_banach by fastforce+
  then have ‹n summable_on A_p›
   by (smt (verit) A_p_def n_def mem_Collect_eq real_norm_def summable_on_cong)
  moreover have ‹n summable_on A_n›
     by (smt (verit, best) ‹f summable_on A_n› summable_on_uminus A_n_def
n_def summable_on_cong mem_Collect_eq real_norm_def)
  ultimately show ‹n summable_on A›
    using A summable_on_Un_disjoint by blast
next
  show ‹f abs_summable_on A ⟹ f summable_on A›
    using abs_summable_summable by blast
qed

lemma abs_summable_on_Sigma_iff:
  shows   f abs_summable_on Sigma A B ⟷
```

```
          (∀ x∈A. (λy. f (x, y)) abs_summable_on B x) ∧
          ((λx. infsum (λy. norm (f (x, y))) (B x)) abs_summable_on A)
proof (intro iffI conjI ballI)
  assume asm: ‹f abs_summable_on Sigma A B›
  then have ‹(λx. infsum (λy. norm (f (x,y))) (B x)) summable_on A›
    by (simp add: cond_case_prod_eta summable_on_Sigma_banach)
  then show ‹(λx. ∑∞y∈B x. norm (f (x, y))) abs_summable_on A›
    using summable_on_iff_abs_summable_on_real by force

  show ‹(λy. f (x, y)) abs_summable_on B x› if ‹x ∈ A› for x
  proof −
    from asm have ‹f abs_summable_on Pair x ‘ B x›
      by (simp add: image_subset_iff summable_on_subset_banach that)
    then show ?thesis
      by (metis (mono_tags, lifting) o_def inj_on_def summable_on_reindex
prod.inject summable_on_cong)
  qed
next
  assume asm: ‹(∀ x∈A. (λxa. f (x, xa)) abs_summable_on B x) ∧
    (λx. ∑∞y∈B x. norm (f (x, y))) abs_summable_on A›
  have ‹(∑ xy∈F. norm (f xy)) ≤ (∑∞x∈A. ∑∞y∈B x. norm (f (x, y)))›
    if ‹F ⊆ Sigma A B› and [simp]: ‹finite F› for F
  proof −
    have [simp]: ‹(SIGMA x:fst ‘ F. {y. (x, y) ∈ F}) = F›
      by (auto intro!: set_eqI simp add: Domain.DomainI fst_eq_Domain)
    have [simp]: ‹finite {y. (x, y) ∈ F}› for x
     by (metis ‹finite F› Range.intros finite_Range finite_subset mem_Collect_eq
subsetI)
     have ‹(∑ xy∈F. norm (f xy)) = (∑ x∈fst ‘ F. ∑ y∈{y. (x,y)∈F}. norm (f
(x,y)))›
      by (simp add: sum.Sigma)
    also have ‹... = (∑∞x∈fst ‘ F. ∑∞y∈{y. (x,y)∈F}. norm (f (x,y)))›
      by auto
    also have ‹... ≤ (∑∞x∈fst ‘ F. ∑∞y∈B x. norm (f (x,y)))›
      using asm that(1) by (intro infsum_mono infsum_mono_neutral) auto
    also have ‹... ≤ (∑∞x∈A. ∑∞y∈B x. norm (f (x,y)))›
      by (rule infsum_mono_neutral) (use asm that(1) in ‹auto simp add: inf-
sum_nonneg›)
    finally show ?thesis .
  qed
  then show ‹f abs_summable_on Sigma A B›
    by (intro nonneg_bdd_above_summable_on) (auto simp: bdd_above_def)
qed

lemma abs_summable_on_comparison_test:
  assumes g abs_summable_on A
  assumes ⋀x. x ∈ A ⟹ norm (f x) ≤ norm (g x)
  shows   f abs_summable_on A
proof (rule nonneg_bdd_above_summable_on)
```

show *bdd_above* (*sum* (λ*x*. *norm* (*f x*)) ‘ {*F*. *F* ⊆ *A* ∧ *finite F*})
proof (*rule bdd_aboveI2*)
fix *F* **assume** *F*: *F* ∈ {*F*. *F* ⊆ *A* ∧ *finite F*}
have ‹*sum* (λ*x*. *norm* (*f x*)) *F* ≤ *sum* (λ*x*. *norm* (*g x*)) *F*›
using *assms F* **by** (*intro sum_mono*) *auto*
also have ‹... = *infsum* (λ*x*. *norm* (*g x*)) *F*›
using *F* **by** *simp*
also have ‹... ≤ *infsum* (λ*x*. *norm* (*g x*)) *A*›
by (*smt* (*verit*) *F assms*(*1*) *infsum_mono2 mem_Collect_eq norm_ge_zero summable_on_subset_banach*)
finally show (∑ *x*∈*F*. *norm* (*f x*)) ≤ (∑ ∞*x*∈*A*. *norm* (*g x*)) **.**
qed
qed *auto*

lemma *abs_summable_iff_bdd_above*:
fixes *f* :: ‹*'a* ⇒ *'b*::*real_normed_vector*›
shows ‹*f abs_summable_on A* ⟷ *bdd_above* (*sum* (λ*x*. *norm* (*f x*)) ‘ {*F*. *F*⊆*A* ∧ *finite F*})›
proof (*rule iffI*)
assume ‹*f abs_summable_on A*›
show ‹*bdd_above* (*sum* (λ*x*. *norm* (*f x*)) ‘ {*F*. *F* ⊆ *A* ∧ *finite F*})›
proof (*rule bdd_aboveI2*)
fix *F* **assume** *F*: *F* ∈ {*F*. *F* ⊆ *A* ∧ *finite F*}
show (∑ *x*∈*F*. *norm* (*f x*)) ≤ (∑ ∞*x*∈*A*. *norm* (*f x*))
by (*rule finite_sum_le_infsum*) (*use* ‹*f abs_summable_on A*› *F* **in** *auto*)
qed
next
assume ‹*bdd_above* (*sum* (λ*x*. *norm* (*f x*)) ‘ {*F*. *F*⊆*A* ∧ *finite F*})›
then show ‹*f abs_summable_on A*›
by (*simp add*: *nonneg_bdd_above_summable_on*)
qed

lemma *abs_summable_product*:
fixes *x* :: *'a* ⇒ *'b*::{*real_normed_div_algebra*,*banach*,*second_countable_topology*}
assumes *x2_sum*: (λ*i*. (*x i*) ∗ (*x i*)) *abs_summable_on A*
and *y2_sum*: (λ*i*. (*y i*) ∗ (*y i*)) *abs_summable_on A*
shows (λ*i*. *x i* ∗ *y i*) *abs_summable_on A*
proof (*rule nonneg_bdd_above_summable_on*)
show *bdd_above* (*sum* (λ*xa*. *norm* (*x xa* ∗ *y xa*)) ‘ {*F*. *F* ⊆ *A* ∧ *finite F*})
proof (*rule bdd_aboveI2*)
fix *F* **assume** *F*: ‹*F* ∈ {*F*. *F* ⊆ *A* ∧ *finite F*}›
then have *r1*: *finite F* **and** *b4*: *F* ⊆ *A*
by *auto*

have *a1*: (∑ ∞*i*∈*F*. *norm* (*x i* ∗ *x i*)) ≤ (∑ ∞*i*∈*A*. *norm* (*x i* ∗ *x i*))
by (*metis* (*no_types*, *lifting*) *b4 infsum_mono2 norm_ge_zero summable_on_subset_banach x2_sum*)

have *norm* (*x i* ∗ *y i*) ≤ *norm* (*x i* ∗ *x i*) + *norm* (*y i* ∗ *y i*) **for** *i*

unfolding *norm_mult* **by** (*smt mult_left_mono mult_nonneg_nonneg mult_right_mono norm_ge_zero*)
hence $(\sum i{\in}F.\ norm\ (x\ i * y\ i)) \leq (\sum i{\in}F.\ norm\ (x\ i * x\ i) + norm\ (y\ i * y\ i))$
by (*simp add*: *sum_mono*)
also have $\ldots = (\sum i{\in}F.\ norm\ (x\ i * x\ i)) + (\sum i{\in}F.\ norm\ (y\ i * y\ i))$
by (*simp add*: *sum.distrib*)
also have $\ldots = (\sum_\infty i{\in}F.\ norm\ (x\ i * x\ i)) + (\sum_\infty i{\in}F.\ norm\ (y\ i * y\ i))$
by (*simp add*: ‹*finite F*›)
also have $\ldots \leq (\sum_\infty i{\in}A.\ norm\ (x\ i * x\ i)) + (\sum_\infty i{\in}A.\ norm\ (y\ i * y\ i))$
using *F assms*
by (*intro add_mono infsum_mono2*) *auto*
finally show ‹$(\sum xa{\in}F.\ norm\ (x\ xa * y\ xa)) \leq (\sum_\infty i{\in}A.\ norm\ (x\ i * x\ i)) + (\sum_\infty i{\in}A.\ norm\ (y\ i * y\ i))$›
by *simp*
qed
qed *auto*

5.8.4 Extended reals and nats

lemma *summable_on_ennreal*[*simp*]: ‹(*f*::_ ⇒ *ennreal*) *summable_on S*› **and** *summable_on_enat*[*simp*]: ‹(*f*::_ ⇒ *enat*) *summable_on S*›
by (*simp_all add*: *nonneg_summable_on_complete*)

lemma *has_sum_superconst_infinite_ennreal*:
fixes *f* :: ‹'*a* ⇒ *ennreal*›
assumes *geqb*: ‹$\bigwedge x.\ x \in S \Longrightarrow f\ x \geq b$›
assumes *b*: ‹$b > 0$›
assumes ‹*infinite S*›
shows (*f has_sum* ∞) *S*
proof −
have ‹(*sum f* ⟶ ∞) (*finite_subsets_at_top S*)›
proof (*rule order_tendstoI*)
fix *y* :: *ennreal* **assume** ‹$y < \infty$›
then have ‹$y\ /\ b < \infty$› ‹$y < top$›
using *b ennreal_divide_eq_top_iff top.not_eq_extremum* **by** *force+*
then obtain *F* **where** ‹*finite F*› **and** ‹$F \subseteq S$› **and** *cardF*: ‹$card\ F > y\ /\ b$›
using ‹*infinite S*›
by (*metis ennreal_Ex_less_of_nat infinite_arbitrarily_large infinity_ennreal_def*)
moreover have ‹$sum\ f\ Y > y$› **if** ‹*finite Y*› **and** ‹$F \subseteq Y$› **and** ‹$Y \subseteq S$› **for** *Y*
proof −
have ‹$y < b * card\ F$›
by (*metis b* ‹$y < top$› *cardF divide_less_ennreal ennreal_mult_eq_top_iff gr_implies_not_zero mult.commute top.not_eq_extremum*)
also have ‹$\ldots \leq b * card\ Y$›
by (*meson b card_mono less_imp_le mult_left_mono of_nat_le_iff that*)
also have ‹$\ldots = sum\ (\lambda\_.\ b)\ Y$›
by (*simp add*: *mult.commute*)

```
    also have ‹... ≤ sum f Y›
      using geqb by (meson subset_eq sum_mono that(3))
    finally show ?thesis .
  qed
  ultimately show ‹∀F x in finite_subsets_at_top S. y < sum f x›
    unfolding eventually_finite_subsets_at_top by auto
 qed auto
 then show ?thesis
  by (simp add: has_sum_def)
qed

lemma infsum_superconst_infinite_ennreal:
 fixes f :: ‹'a ⇒ ennreal›
 assumes ‹⋀x. x ∈ S ⟹ f x ≥ b›
 assumes ‹b > 0›
 assumes ‹infinite S›
 shows infsum f S = ∞
 using assms infsumI has_sum_superconst_infinite_ennreal by blast

lemma infsum_superconst_infinite_ereal:
 fixes f :: ‹'a ⇒ ereal›
 assumes geqb: ‹⋀x. x ∈ S ⟹ f x ≥ b›
 assumes b: ‹b > 0›
 assumes ‹infinite S›
 shows infsum f S = ∞
proof −
 obtain b' where b': ‹e2ennreal b' = b› and ‹b' > 0›
  using b by blast
 have 0 < e2ennreal b
  using b' b
  by (metis dual_order.refl enn2ereal_e2ennreal gr_zeroI order_less_le zero_ennreal.abs_eq)
 hence ∗: ‹infsum (e2ennreal ∘ f) S = ∞›
  using assms b'
  by (intro infsum_superconst_infinite_ennreal[where b=b']) (auto intro!: e2en-
nreal_mono)
 have ‹infsum f S = infsum (enn2ereal ∘ (e2ennreal ∘ f)) S›
  using geqb b by (intro infsum_cong) (fastforce simp: enn2ereal_e2ennreal)
 also have ‹... = enn2ereal ∞›
  using ∗ by (simp add: infsum_comm_additive_general continuous_at_enn2ereal
nonneg_summable_on_complete)
 also have ‹... = ∞›
  by simp
 finally show ?thesis .
qed

lemma has_sum_superconst_infinite_ereal:
 fixes f :: ‹'a ⇒ ereal›
 assumes ‹⋀x. x ∈ S ⟹ f x ≥ b›
 assumes ‹b > 0›
```

assumes ‹*infinite S*›
shows (*f has_sum* ∞) *S*
by (*metis Infty_neq_0*(*1*) *assms infsum_def has_sum_infsum infsum_superconst_infinite_ereal*)

lemma *infsum_superconst_infinite_enat*:
fixes *f* :: ‹*'a* ⇒ *enat*›
assumes *geqb*: ‹⋀*x*. *x* ∈ *S* ⟹ *f x* ≥ *b*›
assumes *b*: ‹*b* > *0*›
assumes ‹*infinite S*›
shows *infsum f S* = ∞
proof −
have ‹*ennreal_of_enat* (*infsum f S*) = *infsum* (*ennreal_of_enat* ∘ *f*) *S*›
by (*simp flip*: *infsum_comm_additive_general*)
also have ‹... = ∞›
by (*metis assms*(*3*) *b comp_def ennreal_of_enat_0 ennreal_of_enat_le_iff geqb infsum_superconst_infinite_ennreal leD leI*)
also have ‹... = *ennreal_of_enat* ∞›
by *simp*
finally show *?thesis*
by (*rule ennreal_of_enat_inj*[*THEN iffD1*])
qed

lemma *has_sum_superconst_infinite_enat*:
fixes *f* :: ‹*'a* ⇒ *enat*›
assumes ‹⋀*x*. *x* ∈ *S* ⟹ *f x* ≥ *b*›
assumes ‹*b* > *0*›
assumes ‹*infinite S*›
shows (*f has_sum* ∞) *S*
by (*metis assms i0_lb has_sum_infsum infsum_superconst_infinite_enat nonneg_summable_on_complete*)

This lemma helps to relate a real-valued infsum to a supremum over extended nonnegative reals.

lemma *infsum_nonneg_is_SUPREMUM_ennreal*:
fixes *f* :: *'a* ⇒ *real*
assumes *summable*: *f summable_on A*
and *fnn*: ⋀*x*. *x*∈*A* ⟹ *f x* ≥ *0*
shows *ennreal* (*infsum f A*) = (*SUP F*∈{*F*. *finite F* ∧ *F* ⊆ *A*}. (*ennreal* (*sum f F*)))
proof −
have §: ⋀*F*. ⟦*finite F*; *F* ⊆ *A*⟧ ⟹ *sum* (*ennreal* ∘ *f*) *F* = *ennreal* (*sum f F*)
by (*metis* (*mono_tags*, *lifting*) *comp_def fnn subsetD sum.cong sum_ennreal*)
then have ‹*ennreal* (*infsum f A*) = *infsum* (*ennreal* ∘ *f*) *A*›
by (*simp add*: *infsum_comm_additive_general local.summable*)
also have ‹... = (*SUP F*∈{*F*. *finite F* ∧ *F* ⊆ *A*}. (*ennreal* (*sum f F*)))›
by (*metis* (*mono_tags*, *lifting*) § *image_cong mem_Collect_eq nonneg_infsum_complete zero_le*)
finally show *?thesis* **.**
qed

This lemma helps to related a real-valued infsum to a supremum over extended reals.

lemma *infsum_nonneg_is_SUPREMUM_ereal*:
fixes $f :: 'a \Rightarrow real$
assumes *summable*: f *summable_on* A
and *fnn*: $\bigwedge x.\ x{\in}A \Longrightarrow f\ x \geq 0$
shows $ereal\ (infsum\ f\ A) = (SUP\ F{\in}\{F.\ finite\ F \wedge F \subseteq A\}.\ (ereal\ (sum\ f\ F)))$
proof −
have $\bigwedge F.\ [\![finite\ F;\ F \subseteq A]\!] \Longrightarrow sum\ (ereal \circ f)\ F = ereal\ (sum\ f\ F)$
by *auto*
then have ‹$ereal\ (infsum\ f\ A) = infsum\ (ereal \circ f)\ A$›
by (*simp add*: *infsum_comm_additive_general local.summable*)
also have ‹$\ldots = (SUP\ F{\in}\{F.\ finite\ F \wedge F \subseteq A\}.\ (ereal\ (sum\ f\ F)))$›
by (*subst nonneg_infsum_complete*) (*simp_all add*: *assms*)
finally show *?thesis* **.**
qed

5.8.5 Real numbers

Most lemmas in the general property section already apply to real numbers. A few ones that are specific to reals are given here.

lemma *infsum_nonneg_is_SUPREMUM_real*:
fixes $f :: 'a \Rightarrow real$
assumes *summable*: f *summable_on* A
and *fnn*: $\bigwedge x.\ x{\in}A \Longrightarrow f\ x \geq 0$
shows $infsum\ f\ A = (SUP\ F{\in}\{F.\ finite\ F \wedge F \subseteq A\}.\ (sum\ f\ F))$
proof −
have ∗: $ereal\ (infsum\ f\ A) = (SUP\ F{\in}\{F.\ finite\ F \wedge F \subseteq A\}.\ (ereal\ (sum\ f\ F)))$
using *assms* **by** (*rule infsum_nonneg_is_SUPREMUM_ereal*)
also have $\ldots = ereal\ (SUP\ F{\in}\{F.\ finite\ F \wedge F \subseteq A\}.\ (sum\ f\ F))$
by (*metis* (*no_types*, *lifting*) ∗ *MInfty_neq_ereal*(*2*) *PInfty_neq_ereal*(*2*)
SUP_cong abs_eq_infinity_cases ereal_SUP)
finally show *?thesis* **by** *simp*
qed

lemma *has_sum_nonneg_SUPREMUM_real*:
fixes $f :: 'a \Rightarrow real$
assumes f *summable_on* A **and** $\bigwedge x.\ x{\in}A \Longrightarrow f\ x \geq 0$
shows $(f\ has\_sum\ (SUP\ F{\in}\{F.\ finite\ F \wedge F \subseteq A\}.\ (sum\ f\ F)))\ A$
by (*metis* (*mono_tags*, *lifting*) *assms has_sum_infsum infsum_nonneg_is_SUPREMUM_real*)

lemma *summable_countable_real*:
fixes $f :: ‹'a \Rightarrow real›$
assumes ‹f *summable_on* A›
shows ‹*countable* $\{x{\in}A.\ f\ x \neq 0\}$›
using *abs_summable_countable assms summable_on_iff_abs_summable_on_real*
by *blast*

5.8.6 Complex numbers

lemma *has_sum_cnj_iff*[*simp*]:
fixes *f* :: ‹'*a* ⇒ *complex*›
shows ‹((λ*x*. *cnj* (*f x*)) *has_sum cnj a*) *M* ⟷ (*f has_sum a*) *M*›
by (*simp add*: *has_sum_def lim_cnj del*: *cnj_sum add*: *cnj_sum*[*symmetric*, *abs_def*, *of f*])

lemma *summable_on_cnj_iff*[*simp*]:
(λ*i*. *cnj* (*f i*)) *summable_on A* ⟷ *f summable_on A*
by (*metis complex_cnj_cnj summable_on_def has_sum_cnj_iff*)

lemma *infsum_cnj*[*simp*]: ‹*infsum* (λ*x*. *cnj* (*f x*)) *M* = *cnj* (*infsum f M*)›
by (*metis complex_cnj_zero infsumI has_sum_cnj_iff infsum_def summable_on_cnj_iff has_sum_infsum*)

lemma *has_sum_Re*:
assumes (*f has_sum a*) *M*
shows ((λ*x*. *Re* (*f x*)) *has_sum Re a*) *M*
using *has_sum_comm_additive*[**where** *f*=*Re*]
using *assms tendsto_Re* **by** (*fastforce simp add*: *o_def additive_def*)

lemma *infsum_Re*:
assumes *f summable_on M*
shows *infsum* (λ*x*. *Re* (*f x*)) *M* = *Re* (*infsum f M*)
by (*simp add*: *assms has_sum_Re infsumI*)

lemma *summable_on_Re*:
assumes *f summable_on M*
shows (λ*x*. *Re* (*f x*)) *summable_on M*
by (*metis assms has_sum_Re summable_on_def*)

lemma *has_sum_Im*:
assumes (*f has_sum a*) *M*
shows ((λ*x*. *Im* (*f x*)) *has_sum Im a*) *M*
using *has_sum_comm_additive*[**where** *f*=*Im*]
using *assms tendsto_Im* **by** (*fastforce simp add*: *o_def additive_def*)

lemma *infsum_Im*:
assumes *f summable_on M*
shows *infsum* (λ*x*. *Im* (*f x*)) *M* = *Im* (*infsum f M*)
by (*simp add*: *assms has_sum_Im infsumI*)

lemma *summable_on_Im*:
assumes *f summable_on M*
shows (λ*x*. *Im* (*f x*)) *summable_on M*
by (*metis assms has_sum_Im summable_on_def*)

lemma *nonneg_infsum_le_0D_complex*:
fixes *f* :: '*a* ⇒ *complex*

assumes *infsum f A* ≤ *0*
and *abs_sum*: *f summable_on A*
and *nneg*: ⋀*x*. *x* ∈ *A* ⟹ *f x* ≥ *0*
and *x* ∈ *A*
shows *f x* = *0*
proof −
have ‹*Im* (*f x*) = *0*›
using *assms*(*4*) *less_eq_complex_def nneg* **by** *auto*
moreover have ‹*Re* (*f x*) = *0*›
using *assms* **by** (*auto simp add*: *summable_on_Re infsum_Re less_eq_complex_def intro*: *nonneg_infsum_le_0D*[**where** *A*=*A*])
ultimately show *?thesis*
by (*simp add*: *complex_eqI*)
qed

lemma *nonneg_has_sum_le_0D_complex*:
fixes *f* :: *'a* ⇒ *complex*
assumes (*f has_sum a*) *A* **and** ‹*a* ≤ *0*›
and ⋀*x*. *x* ∈ *A* ⟹ *f x* ≥ *0* **and** *x* ∈ *A*
shows *f x* = *0*
by (*metis assms infsumI nonneg_infsum_le_0D_complex summable_on_def*)

The lemma *infsum_mono_neutral* above applies to various linear ordered monoids such as the reals but not to the complex numbers. Thus we have a separate corollary for those:

lemma *infsum_mono_neutral_complex*:
fixes *f* :: *'a* ⇒ *complex*
assumes [*simp*]: *f summable_on A*
and [*simp*]: *g summable_on B*
assumes ‹⋀*x*. *x* ∈ *A*∩*B* ⟹ *f x* ≤ *g x*›
assumes ‹⋀*x*. *x* ∈ *A*−*B* ⟹ *f x* ≤ *0*›
assumes ‹⋀*x*. *x* ∈ *B*−*A* ⟹ *g x* ≥ *0*›
shows ‹*infsum f A* ≤ *infsum g B*›
proof −
have ‹*infsum* (λ*x*. *Re* (*f x*)) *A* ≤ *infsum* (λ*x*. *Re* (*g x*)) *B*›
by (*smt* (*verit*) *assms infsum_cong infsum_mono_neutral less_eq_complex_def summable_on_Re zero_complex.simps*(*1*))
then have *Re*: ‹*Re* (*infsum f A*) ≤ *Re* (*infsum g B*)›
by (*metis assms*(*1*−*2*) *infsum_Re*)
have ‹*infsum* (λ*x*. *Im* (*f x*)) *A* = *infsum* (λ*x*. *Im* (*g x*)) *B*›
by (*smt* (*verit*, *best*) *assms*(*3*−*5*) *infsum_cong_neutral less_eq_complex_def zero_complex.simps*(*2*))
then have *Im*: ‹*Im* (*infsum f A*) = *Im* (*infsum g B*)›
by (*metis assms*(*1*−*2*) *infsum_Im*)
from *Re Im* **show** *?thesis*
by (*auto simp*: *less_eq_complex_def*)
qed

lemma *infsum_mono_complex*:

— For *real*, *infsum_mono* can be used. But *complex* does not have the right typeclass.
fixes *f g* :: *'a* ⇒ *complex*
assumes *f_sum*: *f summable_on A* **and** *g_sum*: *g summable_on A*
assumes *leq*: ⋀*x*. *x* ∈ *A* ⟹ *f x* ≤ *g x*
shows *infsum f A* ≤ *infsum g A*
by (*metis DiffE IntD1 f_sum g_sum infsum_mono_neutral_complex leq*)

lemma *infsum_nonneg_complex*:
fixes *f* :: *'a* ⇒ *complex*
assumes *f summable_on M*
and ⋀*x*. *x* ∈ *M* ⟹ *0* ≤ *f x*
shows *infsum f M* ≥ *0* (**is** *?lhs* ≥ _)
by (*metis assms infsum_0_simp summable_on_0_simp infsum_mono_complex*)

lemma *infsum_cmod*:
assumes *f summable_on M*
and *fnn*: ⋀*x*. *x* ∈ *M* ⟹ *0* ≤ *f x*
shows *infsum* (λ*x*. *cmod* (*f x*)) *M* = *cmod* (*infsum f M*)
proof −
have ‹*complex_of_real* (*infsum* (λ*x*. *cmod* (*f x*)) *M*) = *infsum* (λ*x*. *complex_of_real* (*cmod* (*f x*))) *M*›
proof (*rule infsum_comm_additive*[*symmetric*, *unfolded o_def*])
have (λ*z*. *Re* (*f z*)) *summable_on M*
using *assms summable_on_Re* **by** *blast*
also have *?this* ⟷ *f abs_summable_on M*
using *fnn* **by** (*intro summable_on_cong*) (*auto simp*: *less_eq_complex_def cmod_def*)
finally show ... **.**
qed (*auto simp*: *additive_def*)
also have ‹... = *infsum f M*›
using *fnn cmod_eq_Re complex_is_Real_iff less_eq_complex_def* **by** (*force cong*: *infsum_cong*)
finally show *?thesis*
by (*metis abs_of_nonneg infsum_def le_less_trans norm_ge_zero norm_infsum_bound norm_of_real not_le order_refl*)
qed

lemma *summable_on_iff_abs_summable_on_complex*:
fixes *f* :: ‹*'a* ⇒ *complex*›
shows ‹*f summable_on A* ⟷ *f abs_summable_on A*›
proof (*rule iffI*)
assume ‹*f summable_on A*›
define *i r ni nr n* **where** ‹*i x* = *Im* (*f x*)› **and** ‹*r x* = *Re* (*f x*)›
and ‹*ni x* = *norm* (*i x*)› **and** ‹*nr x* = *norm* (*r x*)› **and** ‹*n x* = *norm* (*f x*)› **for** *x*
from ‹*f summable_on A*› **have** ‹*i summable_on A*›

```
    by (simp add: i_def[abs_def] summable_on_Im)
  then have [simp]: ‹ni summable_on A›
    using ni_def[abs_def] summable_on_iff_abs_summable_on_real by force

  from ‹f summable_on A› have ‹r summable_on A›
    by (simp add: r_def[abs_def] summable_on_Re)
  then have [simp]: ‹nr summable_on A›
   by (metis nr_def summable_on_cong summable_on_iff_abs_summable_on_real)

  have n_sum: ‹n x ≤ nr x + ni x› for x
    by (simp add: n_def nr_def ni_def r_def i_def cmod_le)

  have ∗: ‹(λx. nr x + ni x) summable_on A›
    by (simp add: summable_on_add)
  have bdd_above (sum n ` {F. F ⊆ A ∧ finite F})
    apply (rule bdd_aboveI[where M=‹infsum (λx. nr x + ni x) A›])
    using ∗ n_sum by (auto simp flip: infsum_finite simp: ni_def nr_def intro!:
infsum_mono_neutral)
  then show ‹n summable_on A›
    by (simp add: n_def nonneg_bdd_above_summable_on)
next
  show ‹f abs_summable_on A ⟹ f summable_on A›
    using abs_summable_summable by blast
qed

lemma summable_countable_complex:
  fixes f :: ‹'a ⇒ complex›
  assumes ‹f summable_on A›
  shows ‹countable {x∈A. f x ≠ 0}›
 using abs_summable_countable assms summable_on_iff_abs_summable_on_complex
by blast



inductive (in topological_space) convergent_filter :: 'a filter ⇒ bool where
  F ≤ nhds x ⟹ convergent_filter F

lemma (in topological_space) convergent_filter_iff: convergent_filter F ⟷ (∃ x.
F ≤ nhds x)
  by (auto simp: convergent_filter.simps)

lemma (in uniform_space) cauchy_filter_mono:
  cauchy_filter F ⟹ F' ≤ F ⟹ cauchy_filter F'
  unfolding cauchy_filter_def by (meson dual_order.trans prod_filter_mono)

lemma (in uniform_space) convergent_filter_cauchy:
  assumes convergent_filter F
  shows   cauchy_filter F
  using assms cauchy_filter_mono nhds_imp_cauchy_filter[OF order.refl]
```

by (*auto simp*: *convergent_filter_iff*)

lemma (**in** *topological_space*) *convergent_filter_bot* [*simp*, *intro*]: *convergent_filter bot*
by (*simp add*: *convergent_filter_iff*)

class *complete_uniform_space* = *uniform_space* +
assumes *cauchy_filter_convergent'*: *cauchy_filter* (*F* :: 'a *filter*) ⟹ *F* ≠ *bot* ⟹ *convergent_filter F*

lemma (**in** *complete_uniform_space*)
cauchy_filter_convergent: *cauchy_filter* (*F* :: 'a *filter*) ⟹ *convergent_filter F*
using *cauchy_filter_convergent'*[*of F*] **by** (*cases F* = *bot*) *auto*

lemma (**in** *complete_uniform_space*) *convergent_filter_iff_cauchy*:
convergent_filter F ⟷ *cauchy_filter F*
using *convergent_filter_cauchy cauchy_filter_convergent* **by** *blast*

definition *countably_generated_filter* :: 'a *filter* ⇒ *bool* **where**
countably_generated_filter F ⟷ (∃ *U* :: *nat* ⇒ 'a *set*. *F* = (*INF* (*n*::*nat*). *principal* (*U n*)))

lemma *countably_generated_filter_has_antimono_basis*:
assumes *countably_generated_filter F*
obtains *B* :: *nat* ⇒ 'a *set*
where *antimono B* **and** *F* = (*INF n*. *principal* (*B n*)) **and**
⋀*P*. *eventually P F* ⟷ (∃ *i*. ∀ *x*∈*B i*. *P x*)
proof −
from *assms* **obtain** *B* **where** *B*: *F* = (*INF* (*n*::*nat*). *principal* (*B n*))
unfolding *countably_generated_filter_def* **by** *blast*

define *B'* **where** *B'* = (λ*n*. ⋂ *k*≤*n*. *B k*)
have *antimono B'*
unfolding *decseq_def B'_def* **by** *force*

have (*INF n*. *principal* (*B'* *n*)) = (*INF n*. *INF k*∈{..*n*}. *principal* (*B k*))
unfolding *B'_def* **by** (*intro INF_cong refl INF_principal_finite* [*symmetric*])
auto
also have ... = (*INF* (*n*::*nat*). *principal* (*B n*))
apply (*intro antisym*)
apply (*meson INF_lower INF_mono atMost_iff order_refl*)
apply (*meson INF_greatest INF_lower UNIV_I*)
done
also have ... = *F*
by (*simp add*: *B*)
finally have *F*: *F* = (*INF n*. *principal* (*B'* *n*)) **..**

moreover have *eventually P F* ⟷ (∃ *i*. *eventually P* (*principal* (*B′ i*))) **for** *P*
unfolding *F* **using** ‹*antimono B′*›
apply (*subst eventually_INF_base*)
apply (*auto simp*: *decseq_def*)
by (*meson nat_le_linear*)
ultimately show *?thesis*
using ‹*antimono B′*› *that*[*of B′*] **unfolding** *eventually_principal* **by** *blast*
qed

lemma (**in** *uniform_space*) *cauchy_filter_iff*:
cauchy_filter F ⟷ (∀ *P*. *eventually P uniformity* ⟶ (∃ *X*. *eventually* (λ*x*. *x* ∈ *X*) *F* ∧ (∀ *z*∈*X*×*X*. *P z*)))
unfolding *cauchy_filter_def le_filter_def*
apply *auto*
apply (*smt* (*z3*) *eventually_mono eventually_prod_same mem_Collect_eq*)
using *eventually_prod_same* **by** *blast*

lemma (**in** *uniform_space*) *controlled_sequences_convergent_imp_complete_aux_sequence*:
fixes *U* :: *nat* ⇒ (′*a* × ′*a*) *set*
fixes *F* :: ′*a filter*
assumes *cauchy_filter F F* ≠ *bot*
assumes ⋀*n*. *eventually* (λ*z*. *z* ∈ *U n*) *uniformity*
obtains *g G* **where**
antimono G ⋀*n*. *g n* ∈ *G n*
⋀*n*. *eventually* (λ*x*. *x* ∈ *G n*) *F* ⋀*n*. *G n* × *G n* ⊆ *U n*
proof −
have ∃ *C*. *eventually* (λ*x*. *x* ∈ *C*) *F* ∧ *C* × *C* ⊆ *U n* **for** *n*
using *assms*(*1*) *assms*(*3*)[*of n*] **unfolding** *cauchy_filter_iff* **by** *blast*
then obtain *G* **where** *G*: ⋀*n*. *eventually* (λ*x*. *x* ∈ *G n*) *F* ⋀*n*. *G n* × *G n* ⊆ *U n*
by *metis*
define *G′* **where** *G′* = (λ*n*. ⋂*k*≤*n*. *G k*)
have *1*: *eventually* (λ*x*. *x* ∈ *G′ n*) *F* **for** *n*
using *G* **by** (*auto simp*: *G′_def intro*: *eventually_ball_finite*)
have *2*: *G′ n* × *G′ n* ⊆ *U n* **for** *n*
using *G* **unfolding** *G′_def* **by** *fast*
have *3*: *antimono G′*
unfolding *G′_def decseq_def* **by** *force*

have ∃ *g*. *g* ∈ *G′ n* **for** *n*
using *1 assms*(*2*) *eventually_happens′* **by** *auto*
then obtain *g* **where** *g*: ⋀*n*. *g n* ∈ *G′ n*
by *metis*
from *g 1 2 3 that*[*of G′ g*] **show** *?thesis*
by *metis*
qed

definition *lift_filter* :: (′*a set* ⇒ ′*b filter*) ⇒ ′*a filter* ⇒ ′*b filter* **where**
lift_filter f F = (*INF X*∈{*X*. *eventually* (λ*x*. *x* ∈ *X*) *F*}. *f X*)

lemma *lift_filter_top* [*simp*]: *lift_filter g top = g UNIV*
proof −
have {*X*. ∀ *x*::'*b*. *x* ∈ *X*} = {*UNIV*}
by *auto*
thus *?thesis*
by (*simp add*: *lift_filter_def*)
qed

lemma *eventually_lift_filter_iff*:
assumes *mono g*
shows *eventually P* (*lift_filter g F*) ⟷ (∃ *X*. *eventually* (λ*x*. *x* ∈ *X*) *F* ∧ *eventually P* (*g X*))
unfolding *lift_filter_def*
proof (*subst eventually_INF_base*, *goal_cases*)
case *1*
thus *?case* **by** (*auto intro*: *exI*[*of* _ *UNIV*])
next
case (*2 X Y*)
thus *?case*
by (*auto intro*!: *exI*[*of* _ *X* ∩ *Y*] *eventually_conj monoD*[*OF assms*])
qed *auto*

lemma *lift_filter_le*:
assumes *eventually* (λ*x*. *x* ∈ *X*) *F g X* ≤ *F*′
shows *lift_filter g F* ≤ *F*′
unfolding *lift_filter_def*
by (*metis INF_lower2 assms mem_Collect_eq*)

definition *lift_filter*′ :: ('*a set* ⇒ '*b set*) ⇒ '*a filter* ⇒ '*b filter* **where**
lift_filter′ *f F* = *lift_filter* (*principal* ∘ *f*) *F*

lemma *lift_filter*′_*top* [*simp*]: *lift_filter*′ *g top* = *principal* (*g UNIV*)
by (*simp add*: *lift_filter*′_*def*)

lemma *eventually_lift_filter*′_*iff*:
assumes *mono g*
shows *eventually P* (*lift_filter*′ *g F*) ⟷ (∃ *X*. *eventually* (λ*x*. *x* ∈ *X*) *F* ∧ (∀ *x*∈*g X*. *P x*))
unfolding *lift_filter*′_*def* **using** *assms*
by (*subst eventually_lift_filter_iff*) (*auto simp*: *mono_def eventually_principal*)

lemma *lift_filter*′_*le*:
assumes *eventually* (λ*x*. *x* ∈ *X*) *F principal* (*g X*) ≤ *F*′
shows *lift_filter*′ *g F* ≤ *F*′
unfolding *lift_filter*′_*def* **using** *assms*
by (*intro lift_filter_le*[**where** *X* = *X*]) *auto*

lemma (**in** *uniform_space*) *comp_uniformity_le_uniformity*:
lift_filter' (λ*X*. *X O X*) *uniformity* ≤ *uniformity*
unfolding *le_filter_def*
proof *safe*
fix *P* **assume** *P*: *eventually P uniformity*
have [*simp*]: *mono* (λ*X*::('*a* × '*a*) *set*. *X O X*)
by (*intro monoI*) *auto*
from *P* **obtain** *P'* **where** *P'*: *eventually P' uniformity* (⋀*x y z*. *P'* (*x*, *y*) ⟹ *P'* (*y*, *z*) ⟹ *P* (*x*, *z*))
using *uniformity_transE* **by** *blast*
show *eventually P* (*lift_filter'* (λ*X*. *X O X*) *uniformity*)
by (*auto simp*: *eventually_lift_filter'_iff intro*!: *exI*[*of* _ {*x*. *P' x*}] *P'*)
qed

lemma (**in** *uniform_space*) *comp_mem_uniformity_sets*:
assumes *eventually* (λ*z*. *z* ∈ *X*) *uniformity*
obtains *Y* **where** *eventually* (λ*z*. *z* ∈ *Y*) *uniformity Y O Y* ⊆ *X*
proof −
have [*simp*]: *mono* (λ*X*::('*a* × '*a*) *set*. *X O X*)
by (*intro monoI*) *auto*
have *eventually* (λ*z*. *z* ∈ *X*) (*lift_filter'* (λ*X*. *X O X*) *uniformity*)
using *assms comp_uniformity_le_uniformity* **using** *filter_leD* **by** *blast*
thus *?thesis* **using** *that*
by (*auto simp*: *eventually_lift_filter'_iff*)
qed

lemma (**in** *uniform_space*) *le_nhds_of_cauchy_adhp_aux*:
assumes ⋀*P*. *eventually P uniformity* ⟹ (∃ *X*. *eventually* (λ*y*. *y* ∈ *X*) *F* ∧ (∀ *z*∈*X* × *X*. *P z*) ∧ (∃ *y*. *P* (*x*, *y*) ∧ *y* ∈ *X*))
shows *F* ≤ *nhds x*
unfolding *le_filter_def*
proof *safe*
fix *P* **assume** *eventually P* (*nhds x*)
hence ∀ *F z in uniformity*. *z* ∈ {*z*. *fst z* = *x* ⟶ *P* (*snd z*)}
by (*simp add*: *eventually_nhds_uniformity case_prod_unfold*)
then obtain *Y* **where** *Y*: ∀ *F z in uniformity*. *z* ∈ *Y Y O Y* ⊆ {*z*. *fst z* = *x* ⟶ *P* (*snd z*)}
using *comp_mem_uniformity_sets* **by** *blast*
obtain *X y* **where** *Xy*: *eventually* (λ*y*. *y* ∈ *X*) *F X*×*X* ⊆ *Y* (*x*, *y*) ∈ *Y y* ∈ *X*
using *assms*[*OF Y*(*1*)] **by** *blast*
have ∗: *P x* **if** *x* ∈ *X* **for** *x*
using *Y*(*2*) *Xy*(*2*−*4*) *that* **unfolding** *relcomp_unfold* **by** *force*
show *eventually P F*
by (*rule eventually_mono*[*OF Xy*(*1*)]) (*use* ∗ **in** *auto*)
qed

lemma (**in** *uniform_space*) *eventually_uniformity_imp_nhds*:
assumes *eventually P uniformity*
shows *eventually* (λ*y*. *P* (*x*, *y*)) (*nhds x*)

using *assms* **unfolding** *eventually_nhds_uniformity* **by** (*elim eventually_mono*) *auto*

lemma (**in** *uniform_space*) *controlled_sequences_convergent_imp_complete_aux*:
fixes $U :: nat \Rightarrow ('a \times 'a)\ set$
assumes *gen*: *countably_generated_filter* (*uniformity* :: $('a \times 'a)$ *filter*)
assumes *U*: $\bigwedge n.$ *eventually* $(\lambda z.\ z \in U\ n)$ *uniformity*
assumes *conv*: $\bigwedge(u :: nat \Rightarrow 'a).\ (\bigwedge N\ m\ n.\ N \leq m \Longrightarrow N \leq n \Longrightarrow (u\ m,\ u\ n) \in U\ N) \Longrightarrow$ *convergent u*
assumes *cauchy_filter F*
shows *convergent_filter F*
proof (*cases* $F = bot$)
case *False*
note $F =$ ‹*cauchy_filter F*› ‹$F \neq bot$›
from *gen* **obtain** $B :: nat \Rightarrow ('a \times 'a)\ set$ **where** *B*:
antimono B uniformity $= (INF\ n.\ principal\ (B\ n))$
$\bigwedge P.$ *eventually P uniformity* $\longleftrightarrow (\exists i.\ \forall x{\in}B\ i.\ P\ x)$
using *countably_generated_filter_has_antimono_basis* **by** *blast*

have *ev_B*: *eventually* $(\lambda z.\ z \in B\ n)$ *uniformity* **for** *n*
by (*subst B*(*3*)) *auto*
hence *ev_B'*: *eventually* $(\lambda z.\ z \in B\ n \cap U\ n)$ *uniformity* **for** *n*
using *U* **by** (*auto intro*: *eventually_conj*)

obtain *g G* **where** *gG*: *antimono G* $\bigwedge n.\ g\ n \in G\ n$
$\bigwedge n.$ *eventually* $(\lambda x.\ x \in G\ n)\ F\ \bigwedge n.\ G\ n \times G\ n \subseteq B\ n \cap U\ n$
using *controlled_sequences_convergent_imp_complete_aux_sequence*[*of F* $\lambda n.\ B\ n \cap U\ n$, *OF F ev_B'*]
by *metis*

have *convergent g*
proof (*rule conv*)
fix *N m n* :: *nat*
assume *mn*: $N \leq m\ N \leq n$
have $(g\ m,\ g\ n) \in G\ m \times G\ n$
using *gG* **by** *auto*
also from *mn* **have** $\ldots \subseteq G\ N \times G\ N$
by (*intro Sigma_mono gG antimonoD*[*OF gG*(*1*)])
also have $\ldots \subseteq U\ N$
using *gG* **by** *blast*
finally show $(g\ m,\ g\ n) \in U\ N$ **.**
qed
then obtain *L* **where** *G*: $g \longrightarrow L$
unfolding *convergent_def* **by** *blast*

have $F \leq nhds\ L$
proof (*rule le_nhds_of_cauchy_adhp_aux*)
fix $P :: 'a \times 'a \Rightarrow bool$
assume *P*: *eventually P uniformity*

hence *eventually* (λ*n*. ∀ *x*∈*B n*. *P x*) *sequentially*
using ‹*antimono B*› **unfolding** *B*(*3*) *eventually_sequentially decseq_def* **by** *blast*
moreover have *eventually* (λ*n*. *P* (*L*, *g n*)) *sequentially*
using *P eventually_compose_filterlim eventually_uniformity_imp_nhds G* **by** *blast*
ultimately have *eventually* (λ*n*. (∀ *x*∈*B n*. *P x*) ∧ *P* (*L*, *g n*)) *sequentially*
by *eventually_elim auto*
then obtain *n* **where** ∀ *x*∈*B n*. *P x P* (*L*, *g n*)
unfolding *eventually_at_top_linorder* **by** *blast*
then show ∃ *X*. (∀ *F y in F*. *y* ∈ *X*) ∧ (∀ *z*∈*X* × *X*. *P z*) ∧ (∃ *y*. *P* (*L*, *y*) ∧ *y* ∈ *X*)
using *gG* **by** *blast*+
qed
thus *convergent_filter F*
by (*auto simp*: *convergent_filter_iff*)
qed *auto*

theorem (**in** *uniform_space*) *controlled_sequences_convergent_imp_complete*:
fixes *U* :: *nat* ⇒ (*'a* × *'a*) *set*
assumes *gen*: *countably_generated_filter* (*uniformity* :: (*'a* × *'a*) *filter*)
assumes *U*: ⋀*n*. *eventually* (λ*z*. *z* ∈ *U n*) *uniformity*
assumes *conv*: ⋀(*u* :: *nat* ⇒ *'a*). (⋀*N m n*. *N* ≤ *m* ⟹ *N* ≤ *n* ⟹ (*u m*, *u n*) ∈ *U N*) ⟹ *convergent u*
shows *class.complete_uniform_space open uniformity*
by *unfold_locales* (*use assms controlled_sequences_convergent_imp_complete_aux* **in** *blast*)

lemma *filtermap_prod_filter*: *filtermap* (*map_prod f g*) (*F* ×$_F$ *G*) = *filtermap f F* ×$_F$ *filtermap g G*
proof (*intro antisym*)
show *filtermap* (*map_prod f g*) (*F* ×$_F$ *G*) ≤ *filtermap f F* ×$_F$ *filtermap g G*
by (*auto simp*: *le_filter_def eventually_filtermap eventually_prod_filter*)
next
show *filtermap f F* ×$_F$ *filtermap g G* ≤ *filtermap* (*map_prod f g*) (*F* ×$_F$ *G*)
unfolding *le_filter_def*
proof *safe*
fix *P* **assume** *P*: *eventually P* (*filtermap* (*map_prod f g*) (*F* ×$_F$ *G*))
then obtain *Pf Pg* **where** ∗: *eventually Pf F eventually Pg G* ∀ *x*. *Pf x* ⟶ (∀ *y*. *Pg y* ⟶ *P* (*f x*, *g y*))
by (*auto simp*: *eventually_filtermap eventually_prod_filter*)

define *Pf'* **where** *Pf'* = (λ*x*. ∃ *y*. *x* = *f y* ∧ *Pf y*)
define *Pg'* **where** *Pg'* = (λ*x*. ∃ *y*. *x* = *g y* ∧ *Pg y*)

from ∗(*1*) **have** ∀ *x in F*. *Pf'* (*f x*)
by *eventually_elim* (*auto simp*: *Pf'_def*)
moreover from ∗(*2*) **have** ∀ *x in G*. *Pg'* (*g x*)
by *eventually_elim* (*auto simp*: *Pg'_def*)

```
    moreover have (∀ x y. Pf' x ⟶ Pg' y ⟶ P (x, y))
      using ∗(3) by (auto simp: Pf'_def Pg'_def)
    ultimately show eventually P (filtermap f F ×_F filtermap g G)
      unfolding eventually_prod_filter eventually_filtermap
      by blast
  qed
qed


lemma (in uniform_space) Cauchy_seq_iff_tendsto:
  Cauchy f ⟷ filterlim (map_prod f f) uniformity (at_top ×_F at_top)
 unfolding Cauchy_uniform cauchy_filter_def filterlim_def filtermap_prod_filter
..

theorem (in uniform_space) controlled_seq_imp_Cauchy_seq:
  fixes U :: nat ⇒ ('a × 'a) set
  assumes U: ⋀P. eventually P uniformity ⟹ (∃ n. ∀ x∈U n. P x)
  assumes controlled: ⋀N m n. N ≤ m ⟹ N ≤ n ⟹ (f m, f n) ∈ U N
  shows   Cauchy f
  unfolding Cauchy_seq_iff_tendsto
proof −
  show filterlim (map_prod f f) uniformity (sequentially ×_F sequentially)
    unfolding filterlim_def le_filter_def
  proof safe
    fix P :: 'a × 'a ⇒ bool
    assume P: eventually P uniformity
    from U[OF this] obtain N where ∀ x∈U N. P x
      by blast
    then show eventually P (filtermap (map_prod f f) (sequentially ×_F sequentially))
      unfolding eventually_filtermap eventually_prod_sequentially
      by (metis controlled map_prod_simp)
  qed
qed

lemma (in uniform_space) Cauchy_seq_convergent_imp_complete_aux:
  fixes U :: nat ⇒ ('a × 'a) set
  assumes gen: countably_generated_filter (uniformity :: ('a × 'a) filter)
  assumes conv: ⋀(u :: nat ⇒ 'a). Cauchy u ⟹ convergent u
  assumes cauchy_filter F
  shows   convergent_filter F
proof −
  from gen obtain B :: nat ⇒ ('a × 'a) set where B:
    antimono B uniformity = (INF n. principal (B n))
    ⋀P. eventually P uniformity ⟷ (∃ i. ∀ x∈B i. P x)
    using countably_generated_filter_has_antimono_basis by blast

  show ?thesis
   proof (rule controlled_sequences_convergent_imp_complete_aux[where U =
```

```
B])
    show ∀F z in uniformity. z ∈ B n for n
      unfolding B(3) by blast
  next
    fix f :: nat ⇒ 'a
    assume f: ⋀N m n. N ≤ m ⟹ N ≤ n ⟹ (f m, f n) ∈ B N
    have Cauchy f using f B
      by (intro controlled_seq_imp_Cauchy_seq[where U = B]) auto
    with conv show convergent f
      by simp
  qed fact+
qed

theorem (in uniform_space) Cauchy_seq_convergent_imp_complete:
  fixes U :: nat ⇒ ('a × 'a) set
  assumes gen: countably_generated_filter (uniformity :: ('a × 'a) filter)
  assumes conv: ⋀(u :: nat ⇒ 'a). Cauchy u ⟹ convergent u
  shows   class.complete_uniform_space open uniformity
   by unfold_locales (use assms Cauchy_seq_convergent_imp_complete_aux in
blast)

lemma (in metric_space) countably_generated_uniformity:
  countably_generated_filter uniformity
proof −
  have (INF e∈{0<..}. principal {(x, y). dist (x::'a) y < e}) =
        (INF n∈UNIV. principal {(x, y). dist x y < 1 / real (Suc n)}) (is ?F =
?G)
    unfolding uniformity_dist
  proof (intro antisym)
    have ?G = (INF e∈(λn. 1 / real (Suc n)) ‘ UNIV. principal {(x, y). dist x y
< e})
      by (simp add: image_image)
    also have ... ≥ ?F
      by (intro INF_superset_mono) auto
    finally show ?F ≤ ?G .
  next
    show ?G ≤ ?F
      unfolding le_filter_def
    proof safe
      fix P assume eventually P ?F
     then obtain ε where ε: ε > 0 eventually P (principal {(x, y). dist x y < ε})
      proof (subst (asm) eventually_INF_base, goal_cases)
        case (2 ε1 ε2)
        thus ?case
          by (intro bexI[of _ min ε1 ε2]) auto
      qed auto
      from ‹ε > 0› obtain n where 1 / real (Suc n) < ε
        using nat_approx_posE by blast
      then have eventually P (principal {(x, y). dist x y < 1 / real (Suc n)})
```

```
      using ε(2) by (auto simp: eventually_principal)
    thus eventually P ?G
      by (intro eventually_INF1) auto
  qed
 qed
 thus countably_generated_filter uniformity
  unfolding countably_generated_filter_def uniformity_dist by fast
qed

subclass (in complete_space) complete_uniform_space
proof (rule Cauchy_seq_convergent_imp_complete)
 show convergent f if Cauchy f for f
  using Cauchy_convergent that by blast
qed (fact countably_generated_uniformity)

lemma (in complete_uniform_space) complete_UNIV_cuspace [intro]: complete
UNIV
 unfolding complete_uniform using cauchy_filter_convergent
 by (auto simp: convergent_filter.simps)



lemma norm_infsum_le:
 assumes (f has_sum S) X
 assumes (g has_sum T) X
 assumes ⋀x. x ∈ X ⟹ norm (f x) ≤ g x
 shows   norm S ≤ T
proof (rule tendsto_le)
 show ((λY. norm (∑x∈Y. f x)) ⟶ norm S) (finite_subsets_at_top X)
  using assms(1) unfolding has_sum_def by (intro tendsto_norm)
 show ((λY. ∑x∈Y. g x) ⟶ T) (finite_subsets_at_top X)
  using assms(2) unfolding has_sum_def .
 show ∀F x in finite_subsets_at_top X. norm (sum f x) ≤ (∑x∈x. g x)
  by (simp add: assms(3) eventually_finite_subsets_at_top_weakI subsetD sum_norm_le)
qed auto



lemma has_sum_imp_summable: (f has_sum S) A ⟹ f summable_on A
 by (auto simp: summable_on_def)

lemma has_sum_reindex_bij_betw:
 assumes bij_betw g A B
 shows   ((λx. f (g x)) has_sum S) A = (f has_sum S) B
proof -
 have ((λx. f (g x)) has_sum S) A ⟷ (f has_sum S) (g ‘ A)
  by (subst has_sum_reindex) (use assms in ‹auto dest: bij_betw_imp_inj_on
simp: o_def›)
 then show ?thesis
```

using *assms bij_betw_imp_surj_on* **by** *blast*
qed

lemma *has_sum_reindex_bij_witness*:
assumes $\bigwedge a.\ a \in S \Longrightarrow i\ (j\ a) = a$
assumes $\bigwedge a.\ a \in S \Longrightarrow j\ a \in T$
assumes $\bigwedge b.\ b \in T \Longrightarrow j\ (i\ b) = b$
assumes $\bigwedge b.\ b \in T \Longrightarrow i\ b \in S$
assumes $\bigwedge a.\ a \in S \Longrightarrow h\ (j\ a) = g\ a$
assumes $s = s'$
shows $(g\ has\_sum\ s)\ S = (h\ has\_sum\ s')\ T$
by (*smt* (*verit*, *del_insts*) *assms bij_betwI' has_sum_cong has_sum_reindex_bij_betw*)

lemma *has_sum_homomorphism*:
assumes $(f\ has\_sum\ S)\ A\ h\ 0 = 0\ \bigwedge a\ b.\ h\ (a + b) = h\ a + h\ b\ continuous\_on$
UNIV h
shows $((\lambda x.\ h\ (f\ x))\ has\_sum\ (h\ S))\ A$
proof −
have $sum\ (h \circ f)\ X = h\ (sum\ f\ X)$ **for** X
by (*induction X rule*: *infinite_finite_induct*) (*simp_all add*: *assms*)
hence *sum_h*: $sum\ (h \circ f) = h \circ sum\ f$
by (*intro ext*) *auto*
then have $((h \circ f)\ has\_sum\ h\ S)\ A$
using *assms*
by (*metis UNIV_I continuous_on_def has_sum_comm_additive_general o_apply*)
thus *?thesis*
by (*simp add*: *o_def*)
qed

lemma *summable_on_homomorphism*:
assumes *f summable_on A h 0 = 0* $\bigwedge a\ b.\ h\ (a + b) = h\ a + h\ b\ continuous\_on$
UNIV h
shows $(\lambda x.\ h\ (f\ x))\ summable\_on\ A$
proof −
from *assms*(*1*) **obtain** S **where** $(f\ has\_sum\ S)\ A$
by (*auto simp*: *summable_on_def*)
hence $((\lambda x.\ h\ (f\ x))\ has\_sum\ h\ S)\ A$
by (*rule has_sum_homomorphism*) (*use assms* **in** *auto*)
thus *?thesis*
by (*auto simp*: *summable_on_def*)
qed

lemma *infsum_homomorphism_strong*:
fixes $h :: 'a :: \{t2\_space, topological\_comm\_monoid\_add\} \Rightarrow$
$'b :: \{t2\_space, topological\_comm\_monoid\_add\}$
assumes $(\lambda x.\ h\ (f\ x))\ summable\_on\ A \longleftrightarrow f\ summable\_on\ A$
assumes $h\ 0 = 0$
assumes $\bigwedge S.\ (f\ has\_sum\ S)\ A \Longrightarrow ((\lambda x.\ h\ (f\ x))\ has\_sum\ (h\ S))\ A$

shows *infsum* (λx. *h* (*f x*)) *A* = *h* (*infsum f A*)
by (*metis assms has_sum_infsum infsumI infsum_not_exists*)

lemma *has_sum_bounded_linear*:
assumes *bounded_linear h* **and** (*f has_sum S*) *A*
shows ((λx. *h* (*f x*)) *has_sum h S*) *A*
proof −
interpret *bounded_linear h* **by** *fact*
from *assms*(*2*) **show** *?thesis*
by (*rule has_sum_homomorphism*) (*auto simp*: *add intro*!: *continuous_on*)
qed

lemma *summable_on_bounded_linear*:
assumes *bounded_linear h* **and** *f summable_on A*
shows (λx. *h* (*f x*)) *summable_on A*
by (*metis assms has_sum_bounded_linear summable_on_def*)

lemma *summable_on_bounded_linear_iff*:
assumes *bounded_linear h* **and** *bounded_linear h′* **and** ⋀x. *h′* (*h x*) = *x*
shows (λx. *h* (*f x*)) *summable_on A* ⟷ *f summable_on A*
by (*metis* (*full_types*) *assms summable_on_bounded_linear summable_on_cong*)

lemma *infsum_bounded_linear_strong*:
fixes *h* :: *′a* :: *real_normed_vector* ⇒ *′b* :: *real_normed_vector*
assumes (λx. *h* (*f x*)) *summable_on A* ⟷ *f summable_on A*
assumes *bounded_linear h*
shows *infsum* (λx. *h* (*f x*)) *A* = *h* (*infsum f A*)
proof −
interpret *bounded_linear h* **by** *fact*
show *?thesis*
by (*rule infsum_homomorphism_strong*)
(*insert assms*, *auto intro*: *add continuous_on has_sum_bounded_linear*)
qed

lemma *infsum_bounded_linear_strong′*:
fixes *mult* :: *′c* :: *zero* ⇒ *′a* :: *real_normed_vector* ⇒ *′b* :: *real_normed_vector*
assumes *c* ≠ *0* ⟹ (λx. *mult c* (*f x*)) *summable_on A* ⟷ *f summable_on A*
assumes *bounded_linear* (*mult c*)
assumes [*simp*]: ⋀x. *mult 0 x* = *0*
shows *infsum* (λx. *mult c* (*f x*)) *A* = *mult c* (*infsum f A*)
by (*metis assms infsum_0 infsum_bounded_linear_strong*)

lemma *has_sum_of_nat*: (*f has_sum S*) *A* ⟹ ((λx. *of_nat* (*f x*)) *has_sum of_nat S*) *A*
by (*erule has_sum_homomorphism*) (*auto intro*!: *continuous_intros*)

lemma *has_sum_of_int*: (*f has_sum S*) *A* ⟹ ((λx. *of_int* (*f x*)) *has_sum of_int S*) *A*
by (*erule has_sum_homomorphism*) (*auto intro*!: *continuous_intros*)

lemma *summable_on_of_nat*: *f summable_on A* $\Longrightarrow$ $(\lambda x.\ of\_nat\ (f\ x))$ *summable_on A*
by (*erule summable_on_homomorphism*) (*auto intro*!: *continuous_intros*)

lemma *summable_on_of_int*: *f summable_on A* $\Longrightarrow$ $(\lambda x.\ of\_int\ (f\ x))$ *summable_on A*
by (*erule summable_on_homomorphism*) (*auto intro*!: *continuous_intros*)

lemma *summable_on_discrete_iff*:
fixes *f* :: *'a* $\Rightarrow$ *'b* :: {*discrete_topology*, *topological_comm_monoid_add*, *cancel_comm_monoid_add*}
shows *f summable_on A* $\longleftrightarrow$ *finite* $\{x{\in}A.\ f\ x \neq 0\}$
proof
assume $*$: *finite* $\{x{\in}A.\ f\ x \neq 0\}$
hence *f summable_on* $\{x{\in}A.\ f\ x \neq 0\}$
by (*rule summable_on_finite*)
then show *f summable_on A*
by (*smt* (*verit*) *DiffE mem_Collect_eq summable_on_cong_neutral*)
next
assume *f summable_on A*
then obtain *S* **where** (*f has_sum S*) *A*
by (*auto simp*: *summable_on_def*)
hence $\forall_F\ x$ *in finite_subsets_at_top A*. *sum f x* = *S*
unfolding *has_sum_def tendsto_discrete* **.**
then obtain *X* **where** *X*: *finite X X* $\subseteq$ *A* $\bigwedge Y$. *finite Y* $\Longrightarrow$ *X* $\subseteq$ *Y* $\Longrightarrow$ *Y* $\subseteq$ *A* $\Longrightarrow$ *sum f Y* = *S*
unfolding *eventually_finite_subsets_at_top* **by** *metis*
have $\{x{\in}A.\ f\ x \neq 0\} \subseteq X$
proof
fix *x* **assume** *x*: *x* $\in$ $\{x{\in}A.\ f\ x \neq 0\}$
show *x* $\in$ *X*
proof (*rule ccontr*)
assume [*simp*]: *x* $\notin$ *X*
have *sum f* (*insert x X*) = *S*
using *X x* **by** (*intro X*) *auto*
then have *f x* = *0*
using *X* **by** *auto*
with *x* **show** *False*
by *auto*
qed
qed
thus *finite* $\{x{\in}A.\ f\ x \neq 0\}$
using *X*(*1*) *finite_subset* **by** *blast*
qed

lemma *has_sum_imp_sums*: (*f has_sum S*) (*UNIV* :: *nat set*) $\Longrightarrow$ *f sums S*
unfolding *sums_def has_sum_def* **by** (*rule filterlim_compose*[*OF* _ *filterlim_lessThan_at_top*])

lemma *summable_on_imp_summable*: *f summable_on* (*UNIV* :: *nat set*) ⟹ *summable f*
unfolding *summable_on_def summable_def* **by** (*auto dest*: *has_sum_imp_sums*)

lemma *summable_on_UNIV_nonneg_real_iff*:
assumes ⋀*n*. *f n* ≥ (*0* :: *real*)
shows *f summable_on UNIV* ⟷ *summable f*
using *assms* **by** (*auto intro*: *norm_summable_imp_summable_on summable_on_imp_summable*)

lemma *summable_on_imp_bounded_partial_sums*:
fixes *f* :: _ ⇒ *'a* :: {*topological_comm_monoid_add*, *linorder_topology*}
assumes *f*: *f summable_on A*
shows ∃ *C*. *eventually* (λ*X*. *sum f X* ≤ *C*) (*finite_subsets_at_top A*)
proof −
from *assms* **obtain** *S* **where** *S*: (*sum f* ⟶ *S*) (*finite_subsets_at_top A*)
unfolding *summable_on_def has_sum_def* **by** *blast*
show *?thesis*
proof (*cases* ∃ *C*. *C* > *S*)
case *True*
then obtain *C* **where** *C*: *C* > *S*
by *blast*
have ∀ *F X in finite_subsets_at_top A*. *sum f X* < *C*
using *S C* **by** (*rule order_tendstoD*(*2*))
thus *?thesis*
by (*meson eventually_mono nless_le*)
next
case *False* **thus** *?thesis*
by (*meson not_eventuallyD not_le_imp_less*)
qed
qed

lemma *has_sum_mono'*:
fixes *S S'* :: *'a* :: {*linorder_topology*, *ordered_comm_monoid_add*, *topological_comm_monoid_add*}
assumes *f*: (*f has_sum S*) *A* (*f has_sum S'*) *B*
and *AB*: *A* ⊆ *B* ⋀*x*. *x* ∈ *B* − *A* ⟹ *f x* ≥ *0*
shows *S* ≤ *S'*
using *AB has_sum_mono_neutral*[*OF f*] **by** *fastforce*

context
assumes *SORT_CONSTRAINT*(*'a* :: {*topological_comm_monoid_add*, *order_topology*, *ordered_comm_monoid_add*, *conditionally_complete_linorder*})
begin

Any family of non-negative numbers with bounded partial sums is summable, and the sum is simply the supremum of the partial sums.

lemma *nonneg_bounded_partial_sums_imp_has_sum_SUP*:
assumes *nonneg*: ⋀*x*. *x* ∈ *A* ⟹ *f x* ≥ (*0*::*'a*)

```
    and bound: eventually (λX. sum f X ≤ C) (finite_subsets_at_top A)
  shows   (f has_sum (SUP X∈{X. X ⊆ A ∧ finite X}. sum f X)) A
proof −
  from bound obtain X0
    where X0: X0 ⊆ A finite X0 ⋀X. X0 ⊆ X ⟹ X ⊆ A ⟹ finite X ⟹ sum
f X ≤ C
    by (force simp: eventually_finite_subsets_at_top)
  have bound': sum f X ≤ C if X ⊆ A finite X for X
  proof −
    have sum f X ≤ sum f (X ∪ X0)
      using that X0 assms(1) by (intro sum_mono2) auto
    also have ... ≤ C
      by (simp add: X0 that)
    finally show ?thesis .
  qed
  hence bdd: bdd_above (sum f ` {X. X ⊆ A ∧ finite X})
    by (auto simp: bdd_above_def)

  show ?thesis unfolding has_sum_def
  proof (rule increasing_tendsto)
    show ∀⇩F X in finite_subsets_at_top A. sum f X ≤ Sup (sum f ` {X. X ⊆ A
∧ finite X})
      by (intro eventually_finite_subsets_at_top_weakI cSUP_upper[OF _ bdd])
auto
  next
    fix y assume y < Sup (sum f ` {X. X ⊆ A ∧ finite X})
    then obtain X where X: X ⊆ A finite X y < sum f X
      by (subst (asm) less_cSUP_iff[OF _ bdd]) auto
   from X have eventually (λX'. X ⊆ X' ∧ X' ⊆ A ∧ finite X') (finite_subsets_at_top
A)
      by (auto simp: eventually_finite_subsets_at_top)
    thus eventually (λX'. y < sum f X') (finite_subsets_at_top A)
    proof eventually_elim
      case (elim X')
      note ‹y < sum f X›
      also have sum f X ≤ sum f X'
        using nonneg elim by (intro sum_mono2) auto
      finally show ?case .
    qed
  qed
qed

lemma nonneg_bounded_partial_sums_imp_summable_on:
  assumes nonneg: ⋀x. x ∈ A ⟹ f x ≥ (0::'a)
    and bound: eventually (λX. sum f X ≤ C) (finite_subsets_at_top A)
  shows   f summable_on A
  using nonneg_bounded_partial_sums_imp_has_sum_SUP[OF assms] by (auto
simp: summable_on_def)
```

end

context
assumes *SORT_CONSTRAINT*('*a* :: {*topological_comm_monoid_add*, *linorder_topology*, *ordered_comm_monoid_add*, *conditionally_complete_linorder*})
begin

lemma *summable_on_comparison_test*:
assumes *f summable_on A* **and** $\bigwedge x.\ x \in A \Longrightarrow g\ x \leq f\ x$ **and** $\bigwedge x.\ x \in A \Longrightarrow (0::'a) \leq g\ x$
shows *g summable_on A*
proof −
obtain *C* **where** *C*: $\forall_F\ X$ *in finite_subsets_at_top A. sum f X* $\leq C$
using *assms*(*1*) *summable_on_imp_bounded_partial_sums* **by** *blast*
show *?thesis*
proof (*rule nonneg_bounded_partial_sums_imp_summable_on*)
show $\forall_F\ X$ *in finite_subsets_at_top A. sum g X* $\leq C$
using *C assms*
unfolding *eventually_finite_subsets_at_top*
by (*smt* (*verit*, *ccfv_SIG*) *order_trans subsetD sum_mono*)
qed (*use assms* **in** *auto*)
qed

end

lemma *summable_on_subset*:
fixes *f* :: _ ⇒ '*a* :: {*uniform_topological_group_add*, *topological_comm_monoid_add*, *ab_group_add*, *complete_uniform_space*}
assumes *f summable_on A B* ⊆ *A*
shows *f summable_on B*
by (*rule summable_on_subset_aux*[*OF* _ _ *assms*]) (*auto simp*: *uniformly_continuous_add*)

lemma *summable_on_union*:
fixes *f* :: _ ⇒ '*a* :: {*uniform_topological_group_add*, *topological_comm_monoid_add*, *ab_group_add*, *complete_uniform_space*}
assumes *f summable_on A f summable_on B*
shows *f summable_on* (*A* ∪ *B*)
proof −
have *f summable_on* (*A* ∪ (*B* − *A*))
by (*meson Diff_disjoint Diff_subset assms summable_on_Un_disjoint summable_on_subset*)
also have *A* ∪ (*B* − *A*) = *A* ∪ *B*
by *blast*
finally show *?thesis* **.**
qed

lemma *summable_on_insert_iff*:
fixes *f* :: _ ⇒ '*a* :: {*uniform_topological_group_add*, *topological_comm_monoid_add*,

ab_group_add, *complete_uniform_space*}
shows *f summable_on insert x A* ⟷ *f summable_on A*
using *summable_on_union*[*of f A* {*x*}] **by** (*auto intro*: *summable_on_subset*)

lemma *has_sum_finiteI*: *finite A* ⟹ *S* = *sum f A* ⟹ (*f has_sum S*) *A*
by *simp*

lemma *has_sum_insert*:
fixes *f* :: *'a* ⇒ *'b* :: *topological_comm_monoid_add*
assumes *x* ∉ *A* **and** (*f has_sum S*) *A*
shows (*f has_sum* (*f x* + *S*)) (*insert x A*)
proof −
have (*f has_sum* (*f x* + *S*)) ({*x*} ∪ *A*)
using *assms* **by** (*intro has_sum_Un_disjoint*) (*auto intro*: *has_sum_finiteI*)
thus *?thesis* **by** *simp*
qed

lemma *infsum_insert*:
fixes *f* :: _ ⇒ *'a* :: {*topological_comm_monoid_add*, *t2_space*}
assumes *f summable_on A a* ∉ *A*
shows *infsum f* (*insert a A*) = *f a* + *infsum f A*
by (*meson assms has_sum_insert infsumI summable_iff_has_sum_infsum*)

lemma *has_sum_SigmaD*:
fixes *f* :: *'b* × *'c* ⇒ *'a* :: {*topological_comm_monoid_add*,*t3_space*}
assumes *sum1*: (*f has_sum S*) (*Sigma A B*)
assumes *sum2*: ⋀*x*. *x* ∈ *A* ⟹ ((λ*y*. *f* (*x*, *y*)) *has_sum g x*) (*B x*)
shows (*g has_sum S*) *A*
unfolding *has_sum_def tendsto_def eventually_finite_subsets_at_top*
proof (*safe*, *goal_cases*)
case (*1 X*)
with *nhds_closed*[*of S X*] **obtain** *X'*
where *X'*: *S* ∈ *X'* *closed X'* *X'* ⊆ *X* *eventually* (λ*y*. *y* ∈ *X'*) (*nhds S*) **by** *blast*
from *X'*(*4*) **obtain** *X''* **where** *X''*: *S* ∈ *X''* *open X''* *X''* ⊆ *X'*
by (*auto simp*: *eventually_nhds*)
with *sum1* **obtain** *Y* :: (*'b* × *'c*) *set*
where *Y*: *Y* ⊆ *Sigma A B* *finite Y*
⋀*Z*. *Y* ⊆ *Z* ⟹ *Z* ⊆ *Sigma A B* ⟹ *finite Z* ⟹ *sum f Z* ∈ *X''*
unfolding *has_sum_def tendsto_def eventually_finite_subsets_at_top* **by** *force*
define *Y1* :: *'b set* **where** *Y1* = *fst* ‘ *Y*
from *Y* **have** *Y1*: *Y1* ⊆ *A* **by** (*auto simp*: *Y1_def*)
define *Y2* :: *'b* ⇒ *'c set* **where** *Y2* = (λ*x*. {*y*. (*x*, *y*) ∈ *Y*})
have *Y2*: *finite* (*Y2 x*) *Y2 x* ⊆ *B x* **if** *x* ∈ *A* **for** *x*
using *that Y*(*1*,*2*) **unfolding** *Y2_def*
by (*force simp*: *image_iff intro*: *finite_subset*[*of* _ *snd* ‘ *Y*])+

show *?case*
proof (*rule exI*[*of* _ *Y1*], *safe*, *goal_cases*)

case (*3 Z*)
define *H* **where** *H* = (*INF x∈Z. filtercomap* (λ*p. p x*) (*finite_subsets_at_top* (*B x*)))

have *sum g Z* ∈ *X′*
proof (*rule Lim_in_closed_set*)
show *closed X′* **by** *fact*
next
show ((λ*B′. sum* (λ*x. sum* (λ*y. f* (*x*, *y*)) (*B′ x*)) *Z*) ⟶ *sum g Z*) *H*
unfolding *H_def*
proof (*intro tendsto_sum filterlim_INF′*)
fix *x* **assume** *x*: *x* ∈ *Z*
with *3* **have** *x* ∈ *A* **by** *auto*
from *sum2*[*OF this*] **have** (*sum* (λ*y. f* (*x*, *y*)) ⟶ *g x*) (*finite_subsets_at_top* (*B x*))
by (*simp add*: *has_sum_def*)
thus ((λ*B′. sum* (λ*y. f* (*x*, *y*)) (*B′ x*)) ⟶ *g x*)
(*filtercomap* (λ*p. p x*) (*finite_subsets_at_top* (*B x*)))
by (*rule filterlim_compose*[*OF _ filterlim_filtercomap*])
qed *auto*
next
show ∀ *h in H. sum* (λ*x. sum* (λ*y. f* (*x*, *y*)) (*h x*)) *Z* ∈ *X′*
unfolding *H_def*
proof (*subst eventually_INF_finite*[*OF* ‹*finite Z*›], *rule exI*, *safe*)
fix *x* **assume** *x*: *x* ∈ *Z*
hence *x′*: *x* ∈ *A* **using** *3* **by** *auto*
show *eventually* (λ*h. finite* (*h x*) ∧ *Y2 x* ⊆ *h x* ∧ *h x* ⊆ *B x*)
(*filtercomap* (λ*p. p x*) (*finite_subsets_at_top* (*B x*))) **using** *3 Y2*[*OF x′*]
by (*intro eventually_filtercomapI*)
(*auto simp*: *eventually_finite_subsets_at_top intro*: *exI*[*of _ Y2 x*])
next
fix *h*
assume ∗: ∀ *x*∈*Z. finite* (*h x*) ∧ *Y2 x* ⊆ *h x* ∧ *h x* ⊆ *B x*
hence *sum* (λ*x. sum* (λ*y. f* (*x*, *y*)) (*h x*)) *Z* = *sum f* (*Sigma Z h*)
using ‹*finite Z*› **by** (*subst sum.Sigma*) *auto*
also have ... ∈ *X″*
using ∗ *3 Y*(*1*,*2*) **by** (*intro Y*; *force simp*: *Y1_def Y2_def*)
also have *X″* ⊆ *X′* **by** *fact*
finally show *sum* (λ*x. sum* (λ*y. f* (*x*, *y*)) (*h x*)) *Z* ∈ *X′* **.**
qed
next
have *H* = (*INF x*∈*SIGMA x*:*Z*. {*X. finite X* ∧ *X* ⊆ *B x*}.
principal {*y. finite* (*y* (*fst x*)) ∧ *snd x* ⊆ *y* (*fst x*) ∧ *y* (*fst x*) ⊆ *B* (*fst x*)})
unfolding *H_def finite_subsets_at_top_def filtercomap_INF filtercomap_principal*
by (*simp add*: *INF_Sigma*)
also have ... ≠ *bot*
proof (*rule INF_filter_not_bot*, *subst INF_principal_finite*, *goal_cases*)

```
      case (2 X)
      define H' where
        H' = (⋂x∈X. {y. finite (y (fst x)) ∧ snd x ⊆ y (fst x) ∧ y (fst x) ⊆ B
(fst x)})
      from 2 have (λx. ⋃(y,Y)∈X. if x = y then Y else {}) ∈ H'
        by (force split: if_splits simp: H'_def)
      hence H' ≠ {} by blast
      thus principal H' ≠ bot by (simp add: principal_eq_bot_iff)
    qed
    finally show H ≠ bot .
  qed
  also have X' ⊆ X by fact
  finally show sum g Z ∈ X .
 qed (insert Y(1,2), auto simp: Y1_def)
qed

lemma has_sum_unique:
 fixes f :: _ ⇒ 'a :: {topological_comm_monoid_add, t2_space}
 assumes (f has_sum x) A (f has_sum y) A
 shows x = y
 using assms unfolding has_sum_def using tendsto_unique finite_subsets_at_top_neq_bot
by blast

lemma has_sum_SigmaI:
 fixes f :: _ ⇒ 'a :: {topological_comm_monoid_add, t3_space}
 assumes f: ⋀x. x ∈ A ⟹ ((λy. f (x, y)) has_sum g x) (B x)
 assumes g: (g has_sum S) A
 assumes summable: f summable_on Sigma A B
 shows   (f has_sum S) (Sigma A B)
 by (metis f g has_sum_SigmaD has_sum_infsum has_sum_unique local.summable)

lemma summable_on_SigmaD1:
 fixes f :: _ ⇒ _ ⇒ 'a :: {complete_uniform_space, uniform_topological_group_add,
ab_group_add, topological_comm_monoid_add}
 assumes f: (λ(x,y). f x y) summable_on Sigma A B
 assumes x: x ∈ A
 shows   f x summable_on B x
proof −
 have (λ(x,y). f x y) summable_on Sigma {x} B
   using f by (rule summable_on_subset) (use x in auto)
 also have ?this ⟷ ((λy. f x y) ∘ snd) summable_on Sigma {x} B
   by (intro summable_on_cong) auto
 also have ... ⟷ (λy. f x y) summable_on snd ‘ Sigma {x} B
   by (intro summable_on_reindex [symmetric] inj_onI) auto
 also have snd ‘ Sigma {x} B = B x
   by (force simp: Sigma_def)
 finally show ?thesis .
qed
```

lemma *has_sum_swap*:
(*f has_sum S*) (*A* × *B*) ⟷ ((λ(*x*,*y*). *f* (*y*,*x*)) *has_sum S*) (*B* × *A*)
proof −
have *bij_betw* (λ(*x*,*y*). (*y*,*x*)) (*B* × *A*) (*A* × *B*)
by (*rule bij_betwI*[*of* _ _ _ λ(*x*,*y*). (*y*,*x*)]) *auto*
from *has_sum_reindex_bij_betw*[*OF this*, **where** *f* = *f*] **show** *?thesis*
by (*simp add*: *case_prod_unfold*)
qed

lemma *summable_on_swap*:
f summable_on (*A* × *B*) ⟷ (λ(*x*,*y*). *f* (*y*,*x*)) *summable_on* (*B* × *A*)
by (*metis has_sum_swap summable_on_def*)

lemma *has_sum_cmult_right_iff*:
fixes *c* :: *'a* :: {*topological_semigroup_mult*, *field*}
assumes *c* ≠ *0*
shows ((λ*x*. *c* ∗ *f x*) *has_sum S*) *A* ⟷ (*f has_sum* (*S* / *c*)) *A*
using *has_sum_cmult_right*[*of f A S*/*c c*]
has_sum_cmult_right[*of* λ*x*. *c* ∗ *f x A S inverse c*] *assms*
by (*auto simp*: *field_simps*)

lemma *has_sum_cmult_left_iff*:
fixes *c* :: *'a* :: {*topological_semigroup_mult*, *field*}
assumes *c* ≠ *0*
shows ((λ*x*. *f x* ∗ *c*) *has_sum S*) *A* ⟷ (*f has_sum* (*S* / *c*)) *A*
by (*smt* (*verit*, *best*) *assms has_sum_cmult_right_iff has_sum_cong mult.commute*)

lemma *finite_nonzero_values_imp_summable_on*:
assumes *finite* {*x*∈*X*. *f x* ≠ *0*}
shows *f summable_on X*
by (*smt* (*verit*, *del_insts*) *Diff_iff assms mem_Collect_eq summable_on_cong_neutral summable_on_finite*)

lemma *summable_on_of_int_iff*:
(λ*x*::*'a*. *of_int* (*f x*) :: *'b* :: *real_normed_algebra_1*) *summable_on A* ⟷ *f summable_on A*
proof
assume *f summable_on A*
thus (λ*x*. *of_int* (*f x*)) *summable_on A*
by (*rule summable_on_homomorphism*) *auto*
next
assume (λ*x*. *of_int* (*f x*) :: *'b*) *summable_on A*
then obtain *S* **where** ((λ*x*. *of_int* (*f x*) :: *'b*) *has_sum S*) *A*
by (*auto simp*: *summable_on_def*)
hence (*sum* (λ*x*. *of_int* (*f x*) :: *'b*) ⟶ *S*) (*finite_subsets_at_top A*)
unfolding *has_sum_def* **.**
moreover have *1*/*2* > (*0* :: *real*)
by *auto*

```
  ultimately have eventually (λX. dist (sum (λx. of_int (f x) :: 'b) X) S < 1/2)
                   (finite_subsets_at_top A)
    unfolding tendsto_iff by blast
  then obtain X where X: finite X X ⊆ A
     ⋀Y. finite Y ⟹ X ⊆ Y ⟹ Y ⊆ A ⟹ dist (sum (λx. of_int (f x)) Y) S
< 1/2
    unfolding eventually_finite_subsets_at_top by metis

  have sum f Y = sum f X if finite Y X ⊆ Y Y ⊆ A for Y
  proof −
    have dist (sum (λx. of_int (f x)) X) S < 1/2
      by (intro X) auto
    moreover have dist (sum (λx. of_int (f x)) Y) S < 1/2
      by (intro X that)
    ultimately have dist (sum (λx. of_int (f x)) X) (sum (λx. of_int (f x) :: 'b)
Y) <
                      1/2 + 1/2
      using dist_triangle_less_add by blast
    thus ?thesis
      by (simp add: dist_norm flip: of_int_sum of_int_diff)
  qed
  then have {x∈A. f x ≠ 0} ⊆ X
   by (smt (verit) X finite_insert insert_iff mem_Collect_eq subset_eq sum.insert)
  with ‹finite X› have finite {x∈A. f x ≠ 0}
    using finite_subset by blast
  thus f summable_on A
    by (rule finite_nonzero_values_imp_summable_on)
qed

lemma summable_on_of_nat_iff:
   (λx::'a. of_nat (f x) :: 'b :: real_normed_algebra_1) summable_on A ⟷ f
summable_on A
proof
  assume f summable_on A
  thus (λx. of_nat (f x) :: 'b) summable_on A
    by (rule summable_on_homomorphism) auto
next
  assume (λx. of_nat (f x) :: 'b) summable_on A
  hence (λx. of_int (int (f x)) :: 'b) summable_on A
    by simp
  also have ?this ⟷ (λx. int (f x)) summable_on A
    by (rule summable_on_of_int_iff)
  also have ... ⟷ f summable_on A
    by (simp add: summable_on_discrete_iff)
  finally show f summable_on A .
qed

lemma infsum_of_nat:
   infsum (λx::'a. of_nat (f x) :: 'b :: {real_normed_algebra_1}) A = of_nat
```

(*infsum f A*)
by (*metis has_sum_infsum has_sum_of_nat infsumI infsum_def of_nat_0 summable_on_of_nat_iff*)

lemma *infsum_of_int*:
infsum (λx::'a. *of_int* (*f x*) :: 'b :: {*real_normed_algebra_1*}) *A* = *of_int* (*infsum f A*)
by (*metis has_sum_infsum has_sum_of_int infsumI infsum_not_exists of_int_0 summable_on_of_int_iff*)

lemma *summable_on_SigmaI*:
fixes *f* :: _ ⇒ 'a :: {*linorder_topology*, *ordered_comm_monoid_add*, *topological_comm_monoid_add*,
conditionally_complete_linorder}
assumes *f*: ⋀*x*. *x* ∈ *A* ⟹ ((λ*y*. *f* (*x*, *y*)) *has_sum g x*) (*B x*)
assumes *g*: *g summable_on A*
assumes *f_nonneg*: ⋀*x y*. *x* ∈ *A* ⟹ *y* ∈ *B x* ⟹ *f* (*x*, *y*) ≥ (*0* :: 'a)
shows *f summable_on Sigma A B*
proof −
have *g_nonneg*: *g x* ≥ *0* **if** *x* ∈ *A* **for** *x*
using *f* **by** (*rule has_sum_nonneg*) (*use f_nonneg that* **in** *auto*)
obtain *C* **where** *C*: *eventually* (λ*X*. *sum g X* ≤ *C*) (*finite_subsets_at_top A*)
using *summable_on_imp_bounded_partial_sums*[*OF g*] **by** *blast*

have *sum_g_le*: *sum g X* ≤ *C* **if** *X*: *finite X X* ⊆ *A* **for** *X*
proof −
from *C* **obtain** *X'* **where** *X'*:
finite X' X' ⊆ *A* ⋀*Y*. *finite Y* ⟹ *X'* ⊆ *Y* ⟹ *Y* ⊆ *A* ⟹ *sum g Y* ≤ *C*
unfolding *eventually_finite_subsets_at_top* **by** *metis*
have *sum g X* ≤ *sum g* (*X* ∪ *X'*)
using *X X'* **by** (*intro sum_mono2 g_nonneg*) *auto*
also have ... ≤ *C*
using *X X'*(*1*,*2*) **by** (*intro X'*(*3*)) *auto*
finally show *?thesis* **.**
qed

have *sum f Y* ≤ *C* **if** *Y*: *finite Y Y* ⊆ *Sigma A B* **for** *Y*
proof −
define *Y1* **and** *Y2* **where** *Y1* = *fst* ‘ *Y* **and** *Y2* = (λ*x*. *snd* ‘ {*z*∈*Y*. *fst z* = *x*})
have *Y12*: *Y* = *Sigma Y1 Y2*
unfolding *Y1_def Y2_def* **by** *force*
have [*intro*]: *finite Y1* ⋀*x*. *x* ∈ *Y1* ⟹ *finite* (*Y2 x*)
using *Y* **unfolding** *Y1_def Y2_def* **by** *auto*
have *Y12_subset*: *Y1* ⊆ *A* ⋀*x*. *Y2 x* ⊆ *B x*
using *Y* **by** (*auto simp*: *Y1_def Y2_def*)

have *sum f Y* = *sum f* (*Sigma Y1 Y2*)

```
    by (simp add: Y12)
  also have ... = (∑x∈Y1. ∑y∈Y2 x. f (x, y))
    by (subst sum.Sigma) auto
  also have ... ≤ (∑x∈Y1. g x)
  proof (rule sum_mono)
    fix x assume x: x ∈ Y1
    show (∑y∈Y2 x. f (x, y)) ≤ g x
    proof (rule has_sum_mono')
      show ((λy. f (x, y)) has_sum (∑y∈Y2 x. f (x, y))) (Y2 x)
        using x by (intro has_sum_finite) auto
      show ((λy. f (x, y)) has_sum g x) (B x)
        by (rule f) (use x Y12_subset in auto)
      show f (x, y) ≥ 0 if y ∈ B x - Y2 x for y
        using x that Y12_subset by (intro f_nonneg) auto
    qed (use Y12_subset in auto)
  qed
  also have ... ≤ C
    using Y12_subset by (intro sum_g_le) auto
  finally show ?thesis .
 qed

 hence ∀F X in finite_subsets_at_top (Sigma A B). sum f X ≤ C
   unfolding eventually_finite_subsets_at_top by auto
 thus ?thesis
  by (metis SigmaE f_nonneg nonneg_bounded_partial_sums_imp_summable_on)
qed

lemma summable_on_UnionI:
 fixes f :: _ ⇒ 'a :: {linorder_topology, ordered_comm_monoid_add, topological_comm_monoid_add,
                       conditionally_complete_linorder}
 assumes f: ⋀x. x ∈ A ⟹ (f has_sum g x) (B x)
 assumes g: g summable_on A
 assumes f_nonneg: ⋀x y. x ∈ A ⟹ y ∈ B x ⟹ f y ≥ (0 :: 'a)
 assumes disj: disjoint_family_on B A
 shows   f summable_on (⋃x∈A. B x)
proof -
 have f ∘ snd summable_on Sigma A B
   using assms by (intro summable_on_SigmaI[where g = g]) auto
 also have ?this ⟷ f summable_on (snd ` Sigma A B) using assms
  by (subst summable_on_reindex; force simp: disjoint_family_on_def inj_on_def)
 also have snd ` (Sigma A B) = (⋃x∈A. B x)
   by force
 finally show ?thesis .
qed

lemma summable_on_SigmaD:
 fixes f :: 'a × 'b ⇒ 'c :: {topological_comm_monoid_add,t3_space}
 assumes sum1: f summable_on (Sigma A B)
```

assumes *sum2*: $\bigwedge x.\ x \in A \Longrightarrow (\lambda y.\ f\ (x,\ y))\ summable\_on\ (B\ x)$
shows $(\lambda x.\ infsum\ (\lambda y.\ f\ (x,\ y))\ (B\ x))\ summable\_on\ A$
using *assms* **unfolding** *summable_on_def*
by (*smt* (*verit*, *del_insts*) *assms has_sum_SigmaD has_sum_cong has_sum_infsum*)

lemma *summable_on_UnionD*:
fixes $f :: 'a \Rightarrow 'c :: \{topological\_comm\_monoid\_add, t3\_space\}$
assumes *sum1*: $f\ summable\_on\ (\bigcup x{\in}A.\ B\ x)$
assumes *sum2*: $\bigwedge x.\ x \in A \Longrightarrow f\ summable\_on\ (B\ x)$
assumes *disj*: *disjoint_family_on B A*
shows $(\lambda x.\ infsum\ f\ (B\ x))\ summable\_on\ A$
proof −
have $(\bigcup x{\in}A.\ B\ x) = snd\ `\ Sigma\ A\ B$
by (*force simp*: *Sigma_def*)
with *sum1* **have** $f\ summable\_on\ (snd\ `\ Sigma\ A\ B)$
by *simp*
also have *?this* $\longleftrightarrow$ $(f \circ snd)\ summable\_on\ (Sigma\ A\ B)$
using *disj* **by** (*intro summable_on_reindex inj_onI*) (*force simp*: *disjoint_family_on_def*)
finally show $(\lambda x.\ infsum\ f\ (B\ x))\ summable\_on\ A$
using *summable_on_SigmaD*[*of f* ∘ *snd A B*] *sum2* **by** *simp*
qed

lemma *summable_on_Union_iff*:
fixes $f :: \_ \Rightarrow 'a :: \{linorder\_topology,\ ordered\_comm\_monoid\_add,\ topological\_comm\_monoid\_add,$
$conditionally\_complete\_linorder,\ t3\_space\}$
assumes *f*: $\bigwedge x.\ x \in A \Longrightarrow (f\ has\_sum\ g\ x)\ (B\ x)$
assumes *f_nonneg*: $\bigwedge x\ y.\ x \in A \Longrightarrow y \in B\ x \Longrightarrow f\ y \geq 0$
assumes *disj*: *disjoint_family_on B A*
shows $f\ summable\_on\ (\bigcup x{\in}A.\ B\ x) \longleftrightarrow g\ summable\_on\ A$
proof
assume $g\ summable\_on\ A$
thus $f\ summable\_on\ (\bigcup x{\in}A.\ B\ x)$
using *summable_on_UnionI*[*of A f B g*] *assms* **by** *auto*
next
assume $f\ summable\_on\ (\bigcup x{\in}A.\ B\ x)$
hence $(\lambda x.\ infsum\ f\ (B\ x))\ summable\_on\ A$
using *assms* **by** (*intro summable_on_UnionD*) (*auto dest*: *has_sum_imp_summable*)
also have *?this* $\longleftrightarrow$ $g\ summable\_on\ A$
using *assms* **by** (*intro summable_on_cong*) (*auto simp*: *infsumI*)
finally show $g\ summable\_on\ A$ **.**
qed

lemma *has_sum_Sigma'*:
fixes $A :: 'a\ set$ **and** $B :: 'a \Rightarrow 'b\ set$
and $f :: \langle 'a \times 'b \Rightarrow 'c :: \{comm\_monoid\_add, uniform\_space, uniform\_topological\_group\_add\}\rangle$
assumes *summableAB*: $(f\ has\_sum\ a)\ (Sigma\ A\ B)$
assumes *summableB*: $\langle \bigwedge x.\ x{\in}A \Longrightarrow ((\lambda y.\ f\ (x,\ y))\ has\_sum\ (b\ x))\ (B\ x)\rangle$
shows $(b\ has\_sum\ a)\ A$

by (*intro has_sum_Sigma*[*OF* _ *assms*] *uniformly_continuous_add*)

lemma *abs_summable_on_comparison_test'*:
assumes *g summable_on A*
assumes $\bigwedge x.\ x \in A \Longrightarrow$ *norm* (*f x*) $\leq$ *g x*
shows (*λx. norm* (*f x*)) *summable_on A*
proof (*rule Infinite_Sum.abs_summable_on_comparison_test*)
have *g summable_on A* $\longleftrightarrow$ (*λx. norm* (*g x*)) *summable_on A*
by (*metis summable_on_iff_abs_summable_on_real*)
with *assms* **show** (*λx. norm* (*g x*)) *summable_on A* **by** *blast*
qed (*use assms* **in** *fastforce*)

lemma *has_sum_geometric_from_1*:
fixes *z* :: *'a* :: {*real_normed_field*, *banach*}
assumes *norm z* < *1*
shows ((*λn. z* ^ *n*) *has_sum* (*z* / (*1* − *z*))) {*1*..}
proof −
have [*simp*]: *z* ≠ *1*
using *assms* **by** *auto*
have (*λn. z* ^ *Suc n*) *sums* (*1* / (*1* − *z*) − *1*)
using *geometric_sums*[*of z*] *assms* **by** (*subst sums_Suc_iff*) *auto*
also have *1* / (*1* − *z*) − *1* = *z* / (*1* − *z*)
by (*auto simp*: *field_simps*)
finally have (*λn. z* ^ *Suc n*) *sums* (*z* / (*1* − *z*)) **.**
moreover have *summable* (*λn. norm* (*z* ^ *Suc n*))
using *assms*
by (*subst summable_Suc_iff*) (*auto simp*: *norm_power intro*!: *summable_geometric*)
ultimately have ((*λn. z* ^ *Suc n*) *has_sum* (*z* / (*1* − *z*))) *UNIV*
by (*intro norm_summable_imp_has_sum*)
also have *?this* $\longleftrightarrow$ *?thesis*
by (*intro has_sum_reindex_bij_witness*[*of* _ *λn. n*−*1 λn. n*+*1*]) *auto*
finally show *?thesis* **.**
qed

lemma *has_sum_divide_const*:
fixes *f* :: *'a* ⇒ *'b* :: {*topological_semigroup_mult*, *field*, *semiring_0*}
shows (*f has_sum S*) *A* $\Longrightarrow$ ((*λx. f x* / *c*) *has_sum* (*S* / *c*)) *A*
using *has_sum_cmult_right*[*of f A S inverse c*] **by** (*simp add*: *field_simps*)

lemma *has_sum_uminusI*:
fixes *f* :: *'a* ⇒ *'b* :: {*topological_semigroup_mult*, *ring_1*}
shows (*f has_sum S*) *A* $\Longrightarrow$ ((*λx.* −*f x*) *has_sum* (−*S*)) *A*
using *has_sum_cmult_right*[*of f A S* −*1*] **by** *simp*

end

5.9 Ordered Euclidean Space

theory *Ordered_Euclidean_Space*

imports
Convex_Euclidean_Space Abstract_Limits
HOL−Library.Product_Order
begin

An ordering on euclidean spaces that will allow us to talk about intervals

class *ordered_euclidean_space* = *ord* + *inf* + *sup* + *abs* + *Inf* + *Sup* + *euclidean_space* +
assumes *eucl_le*: $x \leq y \longleftrightarrow (\forall i{\in}Basis.\ x \cdot i \leq y \cdot i)$
assumes *eucl_less_le_not_le*: $x < y \longleftrightarrow x \leq y \wedge \neg\ y \leq x$
assumes *eucl_inf*: $inf\ x\ y = (\sum i{\in}Basis.\ inf\ (x \cdot i)\ (y \cdot i) *_R i)$
assumes *eucl_sup*: $sup\ x\ y = (\sum i{\in}Basis.\ sup\ (x \cdot i)\ (y \cdot i) *_R i)$
assumes *eucl_Inf*: $Inf\ X = (\sum i{\in}Basis.\ (INF\ x{\in}X.\ x \cdot i) *_R i)$
assumes *eucl_Sup*: $Sup\ X = (\sum i{\in}Basis.\ (SUP\ x{\in}X.\ x \cdot i) *_R i)$
assumes *eucl_abs*: $|x| = (\sum i{\in}Basis.\ |x \cdot i| *_R i)$
begin

subclass *order*
by *standard*
(*auto simp*: *eucl_le eucl_less_le_not_le intro*!: *euclidean_eqI antisym intro*: *order.trans*)

subclass *ordered_ab_group_add_abs*
by *standard* (*auto simp*: *eucl_le inner_add_left eucl_abs abs_leI*)

subclass *ordered_real_vector*
by *standard* (*auto simp*: *eucl_le intro*!: *mult_left_mono mult_right_mono*)

subclass *lattice*
by *standard* (*auto simp*: *eucl_inf eucl_sup eucl_le*)

subclass *distrib_lattice*
by *standard* (*auto simp*: *eucl_inf eucl_sup sup_inf_distrib1 intro*!: *euclidean_eqI*)

subclass *conditionally_complete_lattice*
proof
fix *z*::*'a* **and** *X*::*'a set*
assume $X \neq \{\}$
hence $\bigwedge i.\ (\lambda x.\ x \cdot i)\ `\ X \neq \{\}$ **by** *simp*
thus $(\bigwedge x.\ x \in X \Longrightarrow z \leq x) \Longrightarrow z \leq Inf\ X\ (\bigwedge x.\ x \in X \Longrightarrow x \leq z) \Longrightarrow Sup\ X \leq z$
by (*auto simp*: *eucl_Inf eucl_Sup eucl_le*
intro!: *cInf_greatest cSup_least*)
qed (*force intro*!: *cInf_lower cSup_upper*
simp: *bdd_below_def bdd_above_def preorder_class.bdd_below_def preorder_class.bdd_above_def*
eucl_Inf eucl_Sup eucl_le)+

lemma *inner_Basis_inf_left*: $i \in Basis \Longrightarrow inf\ x\ y \cdot i = inf\ (x \cdot i)\ (y \cdot i)$
and *inner_Basis_sup_left*: $i \in Basis \Longrightarrow sup\ x\ y \cdot i = sup\ (x \cdot i)\ (y \cdot i)$

by (*simp_all add*: *eucl_inf eucl_sup inner_sum_left inner_Basis if_distrib cong*: *if_cong*)

lemma *inner_Basis_INF_left*: $i \in Basis \Longrightarrow (INF\ x{\in}X.\ f\ x) \cdot i = (INF\ x{\in}X.\ f\ x \cdot i)$
and *inner_Basis_SUP_left*: $i \in Basis \Longrightarrow (SUP\ x{\in}X.\ f\ x) \cdot i = (SUP\ x{\in}X.\ f\ x \cdot i)$
using *eucl_Sup* [*of f ‘ X*] *eucl_Inf* [*of f ‘ X*] **by** (*simp_all add*: *image_comp*)

lemma *abs_inner*: $i \in Basis \Longrightarrow |x| \cdot i = |x \cdot i|$
by (*auto simp*: *eucl_abs*)

lemma
abs_scaleR: $|a *_R b| = |a| *_R |b|$
by (*auto simp*: *eucl_abs abs_mult intro*!: *euclidean_eqI*)

lemma *interval_inner_leI*:
assumes $x \in \{a\ ..\ b\}$ $0 \le i$
shows $a{\cdot}i \le x{\cdot}i$ $x{\cdot}i \le b{\cdot}i$
using *assms*
unfolding *euclidean_inner*[*of a i*] *euclidean_inner*[*of x i*] *euclidean_inner*[*of b i*]
by (*auto intro*!: *ordered_comm_monoid_add_class.sum_mono mult_right_mono simp*: *eucl_le*)

lemma *inner_nonneg_nonneg*:
shows $0 \le a \Longrightarrow 0 \le b \Longrightarrow 0 \le a \cdot b$
using *interval_inner_leI*[*of a 0 a b*]
by *auto*

lemma *inner_Basis_mono*:
shows $a \le b \Longrightarrow c \in Basis \Longrightarrow a \cdot c \le b \cdot c$
by (*simp add*: *eucl_le*)

lemma *Basis_nonneg*[*intro*, *simp*]: $i \in Basis \Longrightarrow 0 \le i$
by (*auto simp*: *eucl_le inner_Basis*)

lemma *Sup_eq_maximum_componentwise*:
fixes *s*::*'a set*
assumes *i*: $\bigwedge b.\ b \in Basis \Longrightarrow X \cdot b = i\ b \cdot b$
assumes *sup*: $\bigwedge b\ x.\ b \in Basis \Longrightarrow x \in s \Longrightarrow x \cdot b \le X \cdot b$
assumes *i_s*: $\bigwedge b.\ b \in Basis \Longrightarrow (i\ b \cdot b) \in (\lambda x.\ x \cdot b)\ ‘\ s$
shows $Sup\ s = X$
using *assms*
unfolding *eucl_Sup euclidean_representation_sum*
by (*auto intro*!: *conditionally_complete_lattice_class.cSup_eq_maximum*)

lemma *Inf_eq_minimum_componentwise*:
assumes *i*: $\bigwedge b.\ b \in Basis \Longrightarrow X \cdot b = i\ b \cdot b$

assumes *sup*: $\bigwedge b\ x.\ b \in Basis \Longrightarrow x \in s \Longrightarrow X \cdot b \leq x \cdot b$
assumes *i_s*: $\bigwedge b.\ b \in Basis \Longrightarrow (i\ b \cdot b) \in (\lambda x.\ x \cdot b)\ `\ s$
shows $Inf\ s = X$
using *assms*
unfolding *eucl_Inf euclidean_representation_sum*
by (*auto intro*!: *conditionally_complete_lattice_class.cInf_eq_minimum*)

end

proposition *compact_attains_Inf_componentwise*:
fixes *b*::*'a*::*ordered_euclidean_space*
assumes $b \in Basis$ **assumes** $X \neq \{\}$ *compact X*
obtains *x* **where** $x \in X\ x \cdot b = Inf\ X \cdot b\ \bigwedge y.\ y \in X \Longrightarrow x \cdot b \leq y \cdot b$
proof *atomize_elim*
let *?proj* $= (\lambda x.\ x \cdot b)\ `\ X$
from *assms* **have** *compact ?proj* $?proj \neq \{\}$
by (*auto intro*!: *compact_continuous_image continuous_intros*)
from *compact_attains_inf*[*OF this*]
obtain *s x*
where *s*: $s \in (\lambda x.\ x \cdot b)\ `\ X\ \bigwedge t.\ t \in (\lambda x.\ x \cdot b)\ `\ X \Longrightarrow s \leq t$
and *x*: $x \in X\ s = x \cdot b\ \bigwedge y.\ y \in X \Longrightarrow x \cdot b \leq y \cdot b$
by *auto*
hence $Inf\ ?proj = x \cdot b$
by (*auto intro*!: *conditionally_complete_lattice_class.cInf_eq_minimum*)
hence $x \cdot b = Inf\ X \cdot b$
by (*auto simp*: *eucl_Inf inner_sum_left inner_Basis if_distrib* ‹$b \in Basis$›
cong: *if_cong*)
with *x* **show** $\exists x.\ x \in X \wedge x \cdot b = Inf\ X \cdot b \wedge (\forall y.\ y \in X \longrightarrow x \cdot b \leq y \cdot b)$
by *blast*
qed

proposition
compact_attains_Sup_componentwise:
fixes *b*::*'a*::*ordered_euclidean_space*
assumes $b \in Basis$ **assumes** $X \neq \{\}$ *compact X*
obtains *x* **where** $x \in X\ x \cdot b = Sup\ X \cdot b\ \bigwedge y.\ y \in X \Longrightarrow y \cdot b \leq x \cdot b$
proof *atomize_elim*
let *?proj* $= (\lambda x.\ x \cdot b)\ `\ X$
from *assms* **have** *compact ?proj* $?proj \neq \{\}$
by (*auto intro*!: *compact_continuous_image continuous_intros*)
from *compact_attains_sup*[*OF this*]
obtain *s x*
where *s*: $s \in (\lambda x.\ x \cdot b)\ `\ X\ \bigwedge t.\ t \in (\lambda x.\ x \cdot b)\ `\ X \Longrightarrow t \leq s$
and *x*: $x \in X\ s = x \cdot b\ \bigwedge y.\ y \in X \Longrightarrow y \cdot b \leq x \cdot b$
by *auto*
hence $Sup\ ?proj = x \cdot b$
by (*auto intro*!: *cSup_eq_maximum*)
hence $x \cdot b = Sup\ X \cdot b$
by (*auto simp*: *eucl_Sup*[**where** *'a*=*'a*] *inner_sum_left inner_Basis if_distrib*

‹$b \in Basis$›
cong: *if_cong*)
with x **show** $\exists x.\ x \in X \wedge x \cdot b = Sup\ X \cdot b \wedge (\forall y.\ y \in X \longrightarrow y \cdot b \leq x \cdot b)$
by *blast*
qed

lemma *tendsto_sup*[*tendsto_intros*]:
fixes $X :: 'a \Rightarrow 'b$::*ordered_euclidean_space*
assumes $(X \longrightarrow x)\ net\ (Y \longrightarrow y)\ net$
shows $((\lambda i.\ sup\ (X\ i)\ (Y\ i)) \longrightarrow sup\ x\ y)\ net$
unfolding *sup_max eucl_sup* **by** (*intro assms tendsto_intros*)

lemma *tendsto_inf*[*tendsto_intros*]:
fixes $X :: 'a \Rightarrow 'b$::*ordered_euclidean_space*
assumes $(X \longrightarrow x)\ net\ (Y \longrightarrow y)\ net$
shows $((\lambda i.\ inf\ (X\ i)\ (Y\ i)) \longrightarrow inf\ x\ y)\ net$
unfolding *inf_min eucl_inf* **by** (*intro assms tendsto_intros*)

lemma *tendsto_Inf*[*tendsto_intros*]:
fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c$::*ordered_euclidean_space*
assumes *finite* $K\ \bigwedge i.\ i \in K \Longrightarrow ((\lambda x.\ f\ x\ i) \longrightarrow l\ i)\ F$
shows $((\lambda x.\ Inf\ (f\ x\ `\ K)) \longrightarrow Inf\ (l\ `\ K))\ F$
using *assms*
by (*induction K rule*: *finite_induct*) (*auto simp*: *cInf_insert_If tendsto_inf*)

lemma *tendsto_Sup*[*tendsto_intros*]:
fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c$::*ordered_euclidean_space*
assumes *finite* $K\ \bigwedge i.\ i \in K \Longrightarrow ((\lambda x.\ f\ x\ i) \longrightarrow l\ i)\ F$
shows $((\lambda x.\ Sup\ (f\ x\ `\ K)) \longrightarrow Sup\ (l\ `\ K))\ F$
using *assms*
by (*induction K rule*: *finite_induct*) (*auto simp*: *cSup_insert_If tendsto_sup*)

lemma *continuous_map_Inf* [*continuous_intros*]:
fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c$::*ordered_euclidean_space*
assumes *finite* $K\ \bigwedge i.\ i \in K \Longrightarrow continuous\_map\ X\ euclidean\ (\lambda x.\ f\ x\ i)$
shows *continuous_map* X *euclidean* $(\lambda x.\ INF\ i{\in}K.\ f\ x\ i)$
using *assms* **by** (*simp add*: *continuous_map_atin tendsto_Inf*)

lemma *continuous_map_Sup* [*continuous_intros*]:
fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c$::*ordered_euclidean_space*
assumes *finite* $K\ \bigwedge i.\ i \in K \Longrightarrow continuous\_map\ X\ euclidean\ (\lambda x.\ f\ x\ i)$
shows *continuous_map* X *euclidean* $(\lambda x.\ SUP\ i{\in}K.\ f\ x\ i)$
using *assms* **by** (*simp add*: *continuous_map_atin tendsto_Sup*)

lemma *tendsto_componentwise_max*:
assumes f: $(f \longrightarrow l)\ F$ **and** g: $(g \longrightarrow m)\ F$
shows $((\lambda x.\ (\sum i{\in}Basis.\ max\ (f\ x \cdot i)\ (g\ x \cdot i) *_R i)) \longrightarrow (\sum i{\in}Basis.\ max\ (l \cdot i)\ (m \cdot i) *_R i))\ F$
by (*intro tendsto_intros assms*)

lemma *tendsto_componentwise_min*:
assumes *f*: $(f \longrightarrow l)\ F$ **and** *g*: $(g \longrightarrow m)\ F$
shows $((\lambda x.\ (\sum i{\in}Basis.\ min\ (f\ x \cdot i)\ (g\ x \cdot i) *_R i)) \longrightarrow (\sum i{\in}Basis.\ min\ (l \cdot i)\ (m \cdot i) *_R i))\ F$
by (*intro tendsto_intros assms*)

instance *real* :: *ordered_euclidean_space*
by *standard auto*

lemma *in_Basis_prod_iff*:
fixes *i*::'*a*::*euclidean_space*∗'*b*::*euclidean_space*
shows $i \in Basis \longleftrightarrow fst\ i = 0 \wedge snd\ i \in Basis \vee snd\ i = 0 \wedge fst\ i \in Basis$
by (*cases i*) (*auto simp*: *Basis_prod_def*)

instantiation *prod* :: (*abs*, *abs*) *abs*
begin

definition $|x| = (|fst\ x|, |snd\ x|)$

instance ..

end

instance *prod* :: (*ordered_euclidean_space*, *ordered_euclidean_space*) *ordered_euclidean_space*
by *standard*
(*auto intro*!: *add_mono simp add*: *euclidean_representation_sum'* *Ball_def inner_prod_def*
in_Basis_prod_iff inner_Basis_inf_left inner_Basis_sup_left inner_Basis_INF_left Inf_prod_def
inner_Basis_SUP_left Sup_prod_def less_prod_def less_eq_prod_def eucl_le[**where** '*a*='*a*]
eucl_le[**where** '*a*='*b*] *abs_prod_def abs_inner*)

Instantiation for intervals on *ordered_euclidean_space*

proposition
fixes *a* :: '*a*::*ordered_euclidean_space*
shows *cbox_interval*: *cbox a b* = {*a*..*b*}
and *interval_cbox*: {*a*..*b*} = *cbox a b*
and *eucl_le_atMost*: $\{x.\ \forall i{\in}Basis.\ x \cdot i <= a \cdot i\} = \{..a\}$
and *eucl_le_atLeast*: $\{x.\ \forall i{\in}Basis.\ a \cdot i <= x \cdot i\} = \{a..\}$
by (*auto simp*: *eucl_le*[**where** '*a*='*a*] *eucl_less_def box_def cbox_def*)

lemma *sums_vec_nth* :
assumes *f sums a*
shows $(\lambda x.\ f\ x\ \$\ i)\ sums\ a\ \$\ i$
using *assms* **unfolding** *sums_def*
by (*auto dest*: *tendsto_vec_nth* [**where** *i*=*i*])

lemma *summable_vec_nth* :
assumes *summable f*
shows *summable* (λ*x*. *f x* $ *i*)
using *assms* **unfolding** *summable_def*
by (*blast intro*: *sums_vec_nth*)

lemma *closed_eucl_atLeastAtMost*[*simp*, *intro*]:
fixes *a* :: *'a*::*ordered_euclidean_space*
shows *closed* {*a*..*b*}
by (*simp add*: *cbox_interval*[*symmetric*] *closed_cbox*)

lemma *closed_eucl_atMost*[*simp*, *intro*]:
fixes *a* :: *'a*::*ordered_euclidean_space*
shows *closed* {..*a*}
by (*simp add*: *closed_interval_left eucl_le_atMost*[*symmetric*])

lemma *closed_eucl_atLeast*[*simp*, *intro*]:
fixes *a* :: *'a*::*ordered_euclidean_space*
shows *closed* {*a*..}
by (*simp add*: *closed_interval_right eucl_le_atLeast*[*symmetric*])

lemma *bounded_closed_interval* [*simp*]:
fixes *a* :: *'a*::*ordered_euclidean_space*
shows *bounded* {*a* .. *b*}
using *bounded_cbox*[*of a b*]
by (*metis interval_cbox*)

lemma *convex_closed_interval* [*simp*]:
fixes *a* :: *'a*::*ordered_euclidean_space*
shows *convex* {*a* .. *b*}
using *convex_box*[*of a b*]
by (*metis interval_cbox*)

lemma *image_smult_interval*:(λ*x*. *m* $*_R$ (*x*::_::*ordered_euclidean_space*)) ' {*a* .. *b*} =
(*if* {*a* .. *b*} = {} *then* {} *else if* 0 ≤ *m then* {*m* $*_R$ *a* .. *m* $*_R$ *b*} *else* {*m* $*_R$ *b* .. *m* $*_R$ *a*})
using *image_smult_cbox*[*of m a b*]
by (*simp add*: *cbox_interval*)

lemma [*simp*]:
fixes *a b*::*'a*::*ordered_euclidean_space*
shows *is_interval_ic*: *is_interval* {..*a*}
and *is_interval_ci*: *is_interval* {*a*..}
and *is_interval_cc*: *is_interval* {*b*..*a*}
by (*force simp*: *is_interval_def eucl_le*[**where** *'a*=*'a*])+

lemma *connected_interval* [*simp*]:
fixes *a b*::*'a*::*ordered_euclidean_space*

shows *connected* {*a*..*b*}
using *is_interval_cc is_interval_connected* **by** *blast*

lemma *compact_interval* [*simp*]:
fixes *a b*::′*a*::*ordered_euclidean_space*
shows *compact* {*a* .. *b*}
by (*metis compact_cbox interval_cbox*)

no_notation
eucl_less (**infix** <*e* 50)

lemma *One_nonneg*: 0 ≤ (∑ *Basis*::′*a*::*ordered_euclidean_space*)
by (*auto intro*: *sum_nonneg*)

lemma
fixes *a b*::′*a*::*ordered_euclidean_space*
shows *bdd_above_cbox*[*intro*, *simp*]: *bdd_above* (*cbox a b*)
and *bdd_below_cbox*[*intro*, *simp*]: *bdd_below* (*cbox a b*)
and *bdd_above_box*[*intro*, *simp*]: *bdd_above* (*box a b*)
and *bdd_below_box*[*intro*, *simp*]: *bdd_below* (*box a b*)
unfolding *atomize_conj*
by (*metis bdd_above_Icc bdd_above_mono bdd_below_Icc bdd_below_mono bounded_box*
bounded_subset_cbox_symmetric interval_cbox)

instantiation *vec* :: (*ordered_euclidean_space*, *finite*) *ordered_euclidean_space*
begin

definition *inf x y* = (χ *i*. *inf* (*x* $ *i*) (*y* $ *i*))
definition *sup x y* = (χ *i*. *sup* (*x* $ *i*) (*y* $ *i*))
definition *Inf X* = (χ *i*. (*INF x*∈*X*. *x* $ *i*))
definition *Sup X* = (χ *i*. (*SUP x*∈*X*. *x* $ *i*))
definition |*x*| = (χ *i*. |*x* $ *i*|)

instance
apply *standard*
unfolding *euclidean_representation_sum′*
apply (*auto simp*: *less_eq_vec_def inf_vec_def sup_vec_def Inf_vec_def Sup_vec_def inner_axis*
Basis_vec_def inner_Basis_inf_left inner_Basis_sup_left inner_Basis_INF_left
inner_Basis_SUP_left eucl_le[**where** ′*a*=′*a*] *less_le_not_le abs_vec_def abs_inner*)
done

end

end

5.10 Arcwise-Connected Sets

theory *Arcwise_Connected*
imports *Path_Connected Ordered_Euclidean_Space HOL−Computational_Algebra.Primes*
begin

lemma *path_connected_interval* [*simp*]:
  fixes *a b*::*'a*::*ordered_euclidean_space*
  shows *path_connected* {*a*..*b*}
  using *is_interval_cc is_interval_path_connected* **by** *blast*

lemma *segment_to_closest_point*:
  fixes *S* :: *'a* :: *euclidean_space set*
  shows ⟦*closed S*; *S* ≠ {}⟧ ⟹ *open_segment a* (*closest_point S a*) ∩ *S* = {}
  unfolding *disjoint_iff*
  by (*metis closest_point_le dist_commute dist_in_open_segment not_le*)

lemma *segment_to_point_exists*:
  fixes *S* :: *'a* :: *euclidean_space set*
    assumes *closed S S* ≠ {}
    obtains *b* **where** *b* ∈ *S open_segment a b* ∩ *S* = {}
  by (*metis assms segment_to_closest_point closest_point_exists that*)

5.10.1 The Brouwer reduction theorem

theorem *Brouwer_reduction_theorem_gen*:
  fixes *S* :: *'a*::*euclidean_space set*
  assumes *closed S φ S*
      and *φ*: ⋀*F*. ⟦⋀*n*. *closed*(*F n*); ⋀*n*. *φ*(*F n*); ⋀*n*. *F*(*Suc n*) ⊆ *F n*⟧ ⟹ *φ*(⋂(*range F*))
  obtains *T* **where** *T* ⊆ *S closed T φ T* ⋀*U*. ⟦*U* ⊆ *S*; *closed U*; *φ U*⟧ ⟹ ¬ (*U* ⊂ *T*)
proof −
  obtain *B* :: *nat* ⇒ *'a set*
    where *inj B* ⋀*n*. *open*(*B n*) **and** *open_cov*: ⋀*S*. *open S* ⟹ ∃*K*. *S* = ⋃(*B* ‘ *K*)
      by (*metis Setcompr_eq_image that univ_second_countable_sequence*)
  define *A* **where** *A* ≡ *rec_nat S* (λ*n a*. *if* ∃ *U*. *U* ⊆ *a* ∧ *closed U* ∧ *φ U* ∧ *U* ∩ (*B n*) = {}
                                then SOME U. *U* ⊆ *a* ∧ *closed U* ∧ *φ U* ∧ *U* ∩ (*B n*) = {}
                                else a)
  have [*simp*]: *A* 0 = *S*
    by (*simp add*: *A_def*)
  have *ASuc*: *A*(*Suc n*) = (*if* ∃ *U*. *U* ⊆ *A n* ∧ *closed U* ∧ *φ U* ∧ *U* ∩ (*B n*) = {}
                    then SOME U. *U* ⊆ *A n* ∧ *closed U* ∧ *φ U* ∧ *U* ∩ (*B n*) = {}
                      else A n) **for** *n*
    by (*auto simp*: *A_def*)
  have *sub*: ⋀*n*. *A*(*Suc n*) ⊆ *A n*
    by (*auto simp*: *ASuc dest*!: *someI_ex*)

```
  have subS: A n ⊆ S for n
    by (induction n) (use sub in auto)
  have clo: closed (A n) ∧ φ (A n) for n
    by (induction n) (auto simp: assms ASuc dest!: someI_ex)
  show ?thesis
  proof
    show ⋂(range A) ⊆ S
      using ‹⋀n. A n ⊆ S› by blast
    show closed (⋂(A ‘ UNIV))
      using clo by blast
    show φ (⋂(A ‘ UNIV))
      by (simp add: clo φ sub)
    show ¬ U ⊂ ⋂(A ‘ UNIV) if U ⊆ S closed U φ U for U
    proof −
      have ∃ y. x ∉ A y if x ∉ U and Usub: U ⊆ (⋂x. A x) for x
      proof −
        obtain e where e > 0 and e: ball x e ⊆ −U
          using ‹closed U› ‹x ∉ U› openE [of −U] by blast
        moreover obtain K where K: ball x e = ⋃(B ‘ K)
          using open_cov [of ball x e] by auto
        ultimately have ⋃(B ‘ K) ⊆ −U
          by blast
        have K ≠ {}
          using ‹0 < e› ‹ball x e = ⋃(B ‘ K)› by auto
        then obtain n where n ∈ K x ∈ B n
          by (metis K UN_E ‹0 < e› centre_in_ball)
        then have U ∩ B n = {}
          using K e by auto
        show ?thesis
        proof (cases ∃ U⊆A n. closed U ∧ φ U ∧ U ∩ B n = {})
          case True
          then show ?thesis
            apply (rule_tac x=Suc n in exI)
            apply (simp add: ASuc)
            apply (erule someI2_ex)
            using ‹x ∈ B n› by blast
        next
          case False
          then show ?thesis
           by (meson Inf_lower Usub ‹U ∩ B n = {}› ‹φ U› ‹closed U› range_eqI
subset_trans)
        qed
      qed
      with that show ?thesis
        by (meson Inter_iff psubsetE rangeI subsetI)
    qed
  qed
qed
```

corollary *Brouwer_reduction_theorem*:
fixes S :: *'a::euclidean_space set*
assumes *compact* S φ S $S \neq \{\}$
and φ: $\bigwedge F$. $[\![\bigwedge n$. *compact*$(F\ n)$; $\bigwedge n$. $F\ n \neq \{\}$; $\bigwedge n$. $\varphi(F\ n)$; $\bigwedge n$. $F(Suc\ n) \subseteq F\ n]\!] \Longrightarrow \varphi(\bigcap(range\ F))$
obtains T **where** $T \subseteq S$ *compact* T $T \neq \{\}$ φ T
$\bigwedge U$. $[\![U \subseteq S$; *closed* U; $U \neq \{\}$; φ $U]\!] \Longrightarrow \neg\ (U \subset T)$
proof (*rule Brouwer_reduction_theorem_gen* [*of* S λT. $T \neq \{\} \wedge T \subseteq S \wedge \varphi$ T])
fix F
assume *cloF*: $\bigwedge n$. *closed* $(F\ n)$
and F: $\bigwedge n$. $F\ n \neq \{\} \wedge F\ n \subseteq S \wedge \varphi\ (F\ n)$ **and** *Fsub*: $\bigwedge n$. $F\ (Suc\ n) \subseteq F\ n$
show $\bigcap(F\ `\ UNIV) \neq \{\} \wedge \bigcap(F\ `\ UNIV) \subseteq S \wedge \varphi\ (\bigcap(F\ `\ UNIV))$
proof (*intro conjI*)
show $\bigcap(F\ `\ UNIV) \neq \{\}$
by (*metis F Fsub* ‹*compact S*› *cloF closed_Int_compact compact_nest inf.orderE lift_Suc_antimono_le*)
show $\bigcap(F\ `\ UNIV) \subseteq S$
using F **by** *blast*
show $\varphi\ (\bigcap(F\ `\ UNIV))$
by (*metis F Fsub* φ ‹*compact S*› *cloF closed_Int_compact inf.orderE*)
qed
next
show $S \neq \{\} \wedge S \subseteq S \wedge \varphi\ S$
by (*simp add*: *assms*)
qed (*meson assms compact_imp_closed seq_compact_closed_subset seq_compact_eq_compact*)+

5.10.2 Arcwise Connections

5.10.3 Density of points with dyadic rational coordinates

proposition *closure_dyadic_rationals*:
closure $(\bigcup k.\ \bigcup f \in Basis \to \mathbb{Z}.$
$\{\ \sum i :: 'a :: euclidean\_space \in Basis.\ (f\ i\ /\ 2\hat{}k) *_R i\ \}) = UNIV$
proof −
have $x \in$ *closure* $(\bigcup k.\ \bigcup f \in Basis \to \mathbb{Z}.\ \{\sum i \in Basis.\ (f\ i\ /\ 2\hat{}k) *_R i\})$ **for** x::*'a*
proof (*clarsimp simp*: *closure_approachable*)
fix e::*real*
assume $e > 0$
then obtain k **where** k: $(1/2)\hat{}k < e/DIM('a)$
by (*meson DIM_positive divide_less_eq_1_pos of_nat_0_less_iff one_less_numeral_iff real_arch_pow_inv semiring_norm(76) zero_less_divide_iff zero_less_numeral*)
have *dist* $(\sum i{\in}Basis.\ (real\_of\_int\ \lfloor 2\hat{}k*(x \cdot i)\rfloor\ /\ 2\hat{}k) *_R i)\ x =$
dist $(\sum i{\in}Basis.\ (real\_of\_int\ \lfloor 2\hat{}k*(x \cdot i)\rfloor\ /\ 2\hat{}k) *_R i)\ (\sum i{\in}Basis.\ (x \cdot i) *_R i)$
by (*simp add*: *euclidean_representation*)
also have ... = *norm* $((\sum i{\in}Basis.\ (real\_of\_int\ \lfloor 2\hat{}k*(x \cdot i)\rfloor\ /\ 2\hat{}k) *_R i - (x \cdot i) *_R i))$
by (*simp add*: *dist_norm sum_subtractf*)

```
    also have ... ≤ DIM('a)*((1/2)^k)
    proof (rule sum_norm_bound, simp add: algebra_simps)
      fix i::'a
      assume i ∈ Basis
      then have norm ((real_of_int ⌊x • i*2^k⌋ / 2^k) *R i − (x • i) *R i) =
                  |real_of_int ⌊x • i*2^k⌋ / 2^k − x • i|
        by (simp add: scaleR_left_diff_distrib [symmetric])
      also have ... ≤ (1/2) ^ k
        by (simp add: divide_simps) linarith
      finally show norm ((real_of_int ⌊x • i*2^k⌋ / 2^k) *R i − (x • i) *R i) ≤
(1/2) ^ k .
    qed
    also have ... < DIM('a)*(e/DIM('a))
    using DIM_positive k linordered_comm_semiring_strict_class.comm_mult_strict_left_mono
of_nat_0_less_iff by blast
    also have ... = e
      by simp
    finally have dist (∑ i∈Basis. (⌊2^k*(x • i)⌋ / 2^k) *R i) x < e .
    with Ints_of_int
    show ∃k. ∃f ∈ Basis → ℤ. dist (∑ b∈Basis. (f b / 2^k) *R b) x < e
      by fastforce
  qed
  then show ?thesis by auto
qed

corollary closure_rational_coordinates:
    closure (⋃f ∈ Basis → ℚ. { ∑ i :: 'a :: euclidean_space ∈ Basis. f i *R i }) =
UNIV
proof −
  have *: (⋃k. ⋃f ∈ Basis → ℤ. { ∑ i::'a ∈ Basis. (f i / 2^k) *R i })
          ⊆ (⋃f ∈ Basis → ℚ. { ∑ i ∈ Basis. f i *R i })
  proof clarsimp
    fix k and f :: 'a ⇒ real
    assume f: f ∈ Basis → ℤ
    show ∃x ∈ Basis → ℚ. (∑ i ∈ Basis. (f i / 2^k) *R i) = (∑ i ∈ Basis. x i *R
i)
      apply (rule_tac x=λi. f i / 2^k in bexI)
      using Ints_subset_Rats f by auto
  qed
  show ?thesis
    using closure_dyadic_rationals closure_mono [OF *] by blast
qed

lemma closure_dyadic_rationals_in_convex_set:
  ⟦convex S; interior S ≠ {}⟧
      ⟹ closure(S ∩
                 (⋃k. ⋃f ∈ Basis → ℤ.
                { ∑ i :: 'a :: euclidean_space ∈ Basis. (f i / 2^k) *R i })) =
         closure S
```

by (*simp add*: *closure_dyadic_rationals closure_convex_Int_superset*)

lemma *closure_rationals_in_convex_set*:
⟦*convex S*; *interior S* ≠ {}⟧
⟹ *closure*(*S* ∩ (⋃*f* ∈ *Basis* → ℚ. { ∑ *i* :: *'a* :: *euclidean_space* ∈ *Basis*. *f i* *$_R$ *i* })) =
closure S
by (*simp add*: *closure_rational_coordinates closure_convex_Int_superset*)

Every path between distinct points contains an arc, and hence path connection is equivalent to arcwise connection for distinct points. The proof is based on Whyburn's "Topological Analysis".

lemma *closure_dyadic_rationals_in_convex_set_pos_1*:
fixes *S* :: *real set*
assumes *convex S* **and** *intnz*: *interior S* ≠ {} **and** *pos*: ⋀*x*. *x* ∈ *S* ⟹ 0 ≤ *x*
shows *closure*(*S* ∩ (⋃*k m*. {*of_nat m* / 2^*k*})) = *closure S*
proof −
have ∃ *m*. *f* 1 / 2^*k* = *real m* / 2^*k* **if** (*f* 1) / 2^*k* ∈ *S f* 1 ∈ ℤ **for** *k* **and** *f* :: *real* ⇒ *real*
using *that* **by** (*force simp*: *Ints_def zero_le_divide_iff power_le_zero_eq dest*: *pos zero_le_imp_eq_int*)
then have *S* ∩ (⋃*k m*. {*real m* / 2^*k*}) = *S* ∩
(⋃*k*. ⋃*f*∈*Basis* → ℤ. {∑ *i*∈*Basis*. (*f i* / 2^*k*) *$_R$ *i*})
by *force*
then show *?thesis*
using *closure_dyadic_rationals_in_convex_set* [*OF* ‹*convex S*› *intnz*] **by** *simp*
qed

definition *dyadics* :: *'a*::*field_char_0 set* **where** *dyadics* ≡ ⋃*k m*. {*of_nat m* / 2^*k*}

lemma *real_in_dyadics* [*simp*]: *real m* ∈ *dyadics*
by (*simp add*: *dyadics_def*) (*metis divide_numeral_1 numeral_One power_0*)

lemma *nat_neq_4k1*: *of_nat m* ≠ (4 * *of_nat k* + 1) / (2 * 2^*n* :: *'a*::*field_char_0*)
proof
assume *of_nat m* = (4 * *of_nat k* + 1) / (2 * 2^*n* :: *'a*)
then have *of_nat* (*m* * (2 * 2^*n*)) = (*of_nat* (*Suc* (4 * *k*)) :: *'a*)
by (*simp add*: *field_split_simps*)
then have *m* * (2 * 2^*n*) = *Suc* (4 * *k*)
using *of_nat_eq_iff* **by** *blast*
then have *odd* (*m* * (2 * 2^*n*))
by *simp*
then show *False*
by *simp*
qed

lemma *nat_neq_4k3*: *of_nat m* ≠ (4 * *of_nat k* + 3) / (2 * 2^*n* :: *'a*::*field_char_0*)

```
proof
  assume of_nat m = (4 * of_nat k + 3) / (2 * 2^n :: 'a)
  then have of_nat (m * (2 * 2^n)) = (of_nat (4 * k + 3) :: 'a)
    by (simp add: field_split_simps)
  then have m * (2 * 2^n) = (4 * k) + 3
    using of_nat_eq_iff by blast
  then have odd (m * (2 * 2^n))
    by simp
  then show False
    by simp
qed

lemma iff_4k:
  assumes r = real k odd k
    shows (4 * real m + r) / (2 * 2^n) = (4 * real m' + r) / (2 * 2 ^ n') ⟷
m=m' ∧ n=n'
proof −
  { assume (4 * real m + r) / (2 * 2^n) = (4 * real m' + r) / (2 * 2 ^ n')
   then have real ((4 * m + k) * (2 * 2 ^ n')) = real ((4 * m' + k) * (2 * 2^n))
     using assms by (auto simp: field_simps)
   then have (4 * m + k) * (2 * 2 ^ n') = (4 * m' + k) * (2 * 2^n)
     using of_nat_eq_iff by blast
   then have (4 * m + k) * (2 ^ n') = (4 * m' + k) * (2^n)
     by linarith
   then obtain 4*m + k = 4*m' + k n=n'
     using prime_power_cancel2 [OF two_is_prime_nat] assms
     by (metis even_mult_iff even_numeral odd_add)
   then have m=m' n=n'
     by auto
  }
  then show ?thesis by blast
qed

lemma neq_4k1_k43: (4 * real m + 1) / (2 * 2^n) ≠ (4 * real m' + 3) / (2 *
2 ^ n')
proof
  assume (4 * real m + 1) / (2 * 2^n) = (4 * real m' + 3) / (2 * 2 ^ n')
  then have real (Suc (4 * m) * (2 * 2 ^ n')) = real ((4 * m' + 3) * (2 * 2^n))
    by (auto simp: field_simps)
  then have Suc (4 * m) * (2 * 2 ^ n') = (4 * m' + 3) * (2 * 2^n)
    using of_nat_eq_iff by blast
  then have Suc (4 * m) * (2 ^ n') = (4 * m' + 3) * (2^n)
    by linarith
  then have Suc (4 * m) = (4 * m' + 3)
    by (rule prime_power_cancel2 [OF two_is_prime_nat]) auto
  then have 1 + 2 * m' = 2 * m
    using ‹Suc (4 * m) = 4 * m' + 3› by linarith
  then show False
    using even_Suc by presburger
```

qed

lemma *dyadic_413_cases*:
obtains (*of_nat m*::'*a*::*field_char_0*) / 2^*k* ∈ *Nats*
| *m' k'* **where** *k'* < *k* (*of_nat m*:: '*a*) / 2^*k* = *of_nat* (4∗*m'* + 1) / 2^*Suc k'*
| *m' k'* **where** *k'* < *k* (*of_nat m*:: '*a*) / 2^*k* = *of_nat* (4∗*m'* + 3) / 2^*Suc k'*
proof (*cases m*>0)
case *False*
then have *m*=0 **by** *simp*
with *that* **show** *?thesis* **by** *auto*
next
case *True*
obtain *k' m'* **where** *m'*: *odd m'* **and** *k'*: *m* = *m'* ∗ 2^*k'*
using *prime_power_canonical* [*OF two_is_prime_nat True*] **by** *blast*
then obtain *q r* **where** *q*: *m'* = 4∗*q* + *r* **and** *r*: *r* < 4
by (*metis not_add_less2 split_div zero_neq_numeral*)
show *?thesis*
proof (*cases k* ≤ *k'*)
case *True*
have (*of_nat m*:: '*a*) / 2^*k* = *of_nat m'* ∗ (2 ^ *k'* / 2^*k*)
using *k'* **by** (*simp add*: *field_simps*)
also have ... = (*of_nat m'*::'*a*) ∗ 2 ^ (*k'*−*k*)
using *k' True* **by** (*simp add*: *power_diff*)
also have ... ∈ ℕ
by (*metis Nats_mult of_nat_in_Nats of_nat_numeral of_nat_power*)
finally show *?thesis* **by** (*auto simp*: *that*)
next
case *False*
then obtain *kd* **where** *kd*: *Suc kd* = *k* − *k'*
using *Suc_diff_Suc not_less* **by** *blast*
have (*of_nat m*:: '*a*) / 2^*k* = *of_nat m'* ∗ (2 ^ *k'* / 2^*k*)
using *k'* **by** (*simp add*: *field_simps*)
also have ... = (*of_nat m'*::'*a*) / 2 ^ (*k*−*k'*)
using *k' False* **by** (*simp add*: *power_diff*)
also have ... = ((*of_nat r* + 4 ∗ *of_nat q*)::'*a*) / 2 ^ (*k*−*k'*)
using *q* **by** *force*
finally have *meq*: (*of_nat m*:: '*a*) / 2^*k* = (*of_nat r* + 4 ∗ *of_nat q*) / 2 ^ (*k* − *k'*) .
have *r* ≠ 0 *r* ≠ 2
using *q m'* **by** *presburger*+
with *r* **consider** *r* = 1 | *r* = 3
by *linarith*
then show *?thesis*
proof *cases*
assume *r* = 1
with *meq kd that*(2) [*of kd q*] **show** *?thesis*
by *simp*
next
assume *r* = 3

```
    with meq kd that(3) [of kd q]  show ?thesis
      by simp
   qed
  qed
qed


lemma dyadics_iff:
  (dyadics :: 'a::field_char_0 set) =
   Nats ∪ (⋃k m. {of_nat (4*m + 1) / 2^Suc k}) ∪ (⋃k m. {of_nat (4*m +
3) / 2^Suc k})
         (is _ = ?rhs)
proof
  show dyadics ⊆ ?rhs
    unfolding dyadics_def
    apply clarify
    apply (rule dyadic_413_cases, force+)
    done
next
  have range of_nat ⊆ (⋃k m. {(of_nat m::'a) / 2 ^ k})
    by clarsimp (metis divide_numeral_1 numeral_One power_0)
  moreover have ⋀k m. ∃k' m'. ((1::'a) + 4 * of_nat m) / 2 ^ Suc k = of_nat
m' / 2 ^ k'
    by (metis (no_types) of_nat_Suc of_nat_mult of_nat_numeral)
  moreover have ⋀k m. ∃k' m'. (4 * of_nat m + (3::'a)) / 2 ^ Suc k = of_nat
m' / 2 ^ k'
    by (metis of_nat_add of_nat_mult of_nat_numeral)
  ultimately show ?rhs ⊆ dyadics
    by (auto simp: dyadics_def Nats_def)
qed


function (domintros) dyad_rec :: [nat ⇒ 'a, 'a⇒'a, 'a⇒'a, real] ⇒ 'a where
    dyad_rec b l r (real m) = b m
  | dyad_rec b l r ((4 * real m + 1) / 2 ^ (Suc n)) = l (dyad_rec b l r ((2*m +
1) / 2^n))
  | dyad_rec b l r ((4 * real m + 3) / 2 ^ (Suc n)) = r (dyad_rec b l r ((2*m +
1) / 2^n))
  | x ∉ dyadics ⟹ dyad_rec b l r x = undefined
  using iff_4k [of _ 1] iff_4k [of _ 3]
      apply (simp_all add: nat_neq_4k1 nat_neq_4k3 neq_4k1_k43 dyadics_iff
Nats_def)
  by (fastforce simp:  field_simps)+

lemma dyadics_levels: dyadics = (⋃K. ⋃k<K. ⋃ m. {of_nat m / 2^k})
  unfolding dyadics_def by auto

lemma dyad_rec_level_termination:
  assumes k < K
```

```
  shows dyad_rec_dom(b, l, r, real m / 2^k)
  using assms
proof (induction K arbitrary: k m)
  case 0
  then show ?case by auto
next
  case (Suc K)
  then consider k = K | k < K
    using less_antisym by blast
  then show ?case
  proof cases
    assume k = K
    show ?case
    proof (rule dyadic_413_cases [of m k, where 'a=real])
      show real m / 2^k ∈ ℕ ⟹ dyad_rec_dom (b, l, r, real m / 2^k)
      by (force simp: Nats_def nat_neq_4k1 nat_neq_4k3 intro: dyad_rec.domintros)
      show ?case if k' < k and eq: real m / 2^k = real (4 * m' + 1) / 2^Suc k'
for m' k'
      proof −
        have dyad_rec_dom (b, l, r, (4 * real m' + 1) / 2^Suc k')
        proof (rule dyad_rec.domintros)
          fix m n
          assume (4 * real m' + 1) / (2 * 2 ^ k') = (4 * real m + 1) / (2 * 2^n)
          then have m' = m k' = n using iff_4k [of _ 1]
            by auto
          have dyad_rec_dom (b, l, r, real (2 * m + 1) / 2 ^ k')
            using Suc.IH ‹k = K› ‹k' < k› by blast
          then show dyad_rec_dom (b, l, r, (2 * real m + 1) / 2^n)
            using ‹k' = n› by (auto simp: algebra_simps)
        next
          fix m n
          assume (4 * real m' + 1) / (2 * 2 ^ k') = (4 * real m + 3) / (2 * 2^n)
          then have False
            by (metis neq_4k1_k43)
          then show dyad_rec_dom (b, l, r, (2 * real m + 1) / 2^n) ..
        qed
        then show ?case by (simp add: eq add_ac)
      qed
      show ?case if k' < k and eq: real m / 2^k = real (4 * m' + 3) / 2^Suc k'
for m' k'
      proof −
        have dyad_rec_dom (b, l, r, (4 * real m' + 3) / 2^Suc k')
        proof (rule dyad_rec.domintros)
          fix m n
          assume (4 * real m' + 3) / (2 * 2 ^ k') = (4 * real m + 1) / (2 * 2^n)
          then have False
            by (metis neq_4k1_k43)
          then show dyad_rec_dom (b, l, r, (2 * real m + 1) / 2^n) ..
        next
```

```
        fix m n
        assume (4 * real m' + 3) / (2 * 2 ^ k') = (4 * real m + 3) / (2 * 2^n)
        then have m' = m k' = n using iff_4k [of _ 3]
          by auto
        have dyad_rec_dom (b, l, r, real (2 * m + 1) / 2 ^ k')
          using Suc.IH ‹k = K› ‹k' < k› by blast
        then show dyad_rec_dom (b, l, r, (2 * real m + 1) / 2^n)
          using ‹k' = n› by (auto simp: algebra_simps)
      qed
      then show ?case by (simp add: eq add_ac)
    qed
  qed
 next
  assume k < K
  then show ?case
   using Suc.IH by blast
 qed
qed


lemma dyad_rec_termination: x ∈ dyadics ⟹ dyad_rec_dom(b,l,r,x)
 by (auto simp: dyadics_levels intro: dyad_rec_level_termination)

lemma dyad_rec_of_nat [simp]: dyad_rec b l r (real m) = b m
 by (simp add: dyad_rec.psimps dyad_rec_termination)

lemma dyad_rec_41 [simp]: dyad_rec b l r ((4 * real m + 1) / 2 ^ (Suc n)) = l
(dyad_rec b l r ((2*m + 1) / 2^n))
proof (rule dyad_rec.psimps)
 show dyad_rec_dom (b, l, r, (4 * real m + 1) / 2 ^ Suc n)
  by (metis add.commute dyad_rec_level_termination lessI of_nat_Suc of_nat_mult
of_nat_numeral)
qed

lemma dyad_rec_43 [simp]: dyad_rec b l r ((4 * real m + 3) / 2 ^ (Suc n)) =
r (dyad_rec b l r ((2*m + 1) / 2^n))
proof (rule dyad_rec.psimps)
 show dyad_rec_dom (b, l, r, (4 * real m + 3) / 2 ^ Suc n)
  by (metis dyad_rec_level_termination lessI of_nat_add of_nat_mult of_nat_numeral)
qed

lemma dyad_rec_41_times2:
 assumes n > 0
   shows dyad_rec b l r (2 * ((4 * real m + 1) / 2^Suc n)) = l (dyad_rec b l r
(2 * (2 * real m + 1) / 2^n))
proof −
 obtain n' where n': n = Suc n'
   using assms not0_implies_Suc by blast
 have dyad_rec b l r (2 * ((4 * real m + 1) / 2^Suc n)) = dyad_rec b l r ((2 *
```

```
(4 * real m + 1)) / (2 * 2^n))
    by auto
  also have ... = dyad_rec b l r ((4 * real m + 1) / 2^n)
    by (subst mult_divide_mult_cancel_left) auto
  also have ... = l (dyad_rec b l r ((2 * real m + 1) / 2 ^ n'))
    by (simp add: add.commute [of 1] n' del: power_Suc)
  also have ... = l (dyad_rec b l r ((2 * (2 * real m + 1)) / (2 * 2 ^ n')))
    by (subst mult_divide_mult_cancel_left) auto
  also have ... = l (dyad_rec b l r (2 * (2 * real m + 1) / 2^n))
    by (simp add: add.commute n')
  finally show ?thesis .
qed

lemma dyad_rec_43_times2:
  assumes n > 0
    shows dyad_rec b l r (2 * ((4 * real m + 3) / 2^Suc n)) = r (dyad_rec b l r
(2 * (2 * real m + 1) / 2^n))
proof −
  obtain n' where n': n = Suc n'
    using assms not0_implies_Suc by blast
  have dyad_rec b l r (2 * ((4 * real m + 3) / 2^Suc n)) = dyad_rec b l r ((2 *
(4 * real m + 3)) / (2 * 2^n))
    by auto
  also have ... = dyad_rec b l r ((4 * real m + 3) / 2^n)
    by (subst mult_divide_mult_cancel_left) auto
  also have ... = r (dyad_rec b l r ((2 * real m + 1) / 2 ^ n'))
    by (simp add: n' del: power_Suc)
  also have ... = r (dyad_rec b l r ((2 * (2 * real m + 1)) / (2 * 2 ^ n')))
    by (subst mult_divide_mult_cancel_left) auto
  also have ... = r (dyad_rec b l r (2 * (2 * real m + 1) / 2^n))
    by (simp add: n')
  finally show ?thesis .
qed

definition dyad_rec2
    where dyad_rec2 u v lc rc x =
          dyad_rec (λz. (u,v)) (λ(a,b). (a, lc a b (midpoint a b))) (λ(a,b). (rc a b
(midpoint a b), b)) (2*x)

abbreviation leftrec where leftrec u v lc rc x ≡ fst (dyad_rec2 u v lc rc x)
abbreviation rightrec where rightrec u v lc rc x ≡ snd (dyad_rec2 u v lc rc x)

lemma leftrec_base: leftrec u v lc rc (real m / 2) = u
  by (simp add: dyad_rec2_def)

lemma leftrec_41: n > 0 ⟹ leftrec u v lc rc ((4 * real m + 1) / 2 ^ (Suc n))
= leftrec u v lc rc ((2 * real m + 1) / 2^n)
  unfolding dyad_rec2_def dyad_rec_41_times2
  by (simp add: case_prod_beta)
```

lemma *leftrec_43*: $n > 0 \Longrightarrow$
leftrec u v lc rc ((4 * real m + 3) / 2 ^ (Suc n)) =
rc (leftrec u v lc rc ((2 * real m + 1) / 2^n)) (rightrec u v lc rc ((2 * real m + 1) / 2^n))
(midpoint (leftrec u v lc rc ((2 * real m + 1) / 2^n)) (rightrec u v lc rc ((2 * real m + 1) / 2^n)))
unfolding *dyad_rec2_def dyad_rec_43_times2*
by (*simp add*: *case_prod_beta*)

lemma *rightrec_base*: *rightrec u v lc rc* (*real m / 2*) = *v*
by (*simp add*: *dyad_rec2_def*)

lemma *rightrec_41*: $n > 0 \Longrightarrow$
rightrec u v lc rc ((4 * real m + 1) / 2 ^ (Suc n)) =
lc (leftrec u v lc rc ((2 * real m + 1) / 2^n)) (rightrec u v lc rc ((2 * real m + 1) / 2^n))
(midpoint (leftrec u v lc rc ((2 * real m + 1) / 2^n)) (rightrec u v lc rc ((2 * real m + 1) / 2^n)))
unfolding *dyad_rec2_def dyad_rec_41_times2*
by (*simp add*: *case_prod_beta*)

lemma *rightrec_43*: $n > 0 \Longrightarrow$ rightrec u v lc rc ((4 * real m + 3) / 2 ^ (Suc n)) = rightrec u v lc rc ((2 * real m + 1) / 2^n)
unfolding *dyad_rec2_def dyad_rec_43_times2*
by (*simp add*: *case_prod_beta*)

lemma *dyadics_in_open_unit_interval*:
{0<..<1} ∩ (⋃k m. {real m / 2^k}) = (⋃k. ⋃m ∈ {0<..<2^k}. {real m / 2^k})
by (*auto simp*: *field_split_simps*)

lemma *padic_rational_approximation_straddle*:
assumes $\varepsilon > 0$ $p > 1$
obtains *n q r*
where *of_int q / p^n < x x < of_int r / p^n* $|q / p^n - r / p^n| < \varepsilon$
proof −
obtain *n* **where** *n*: $2 / \varepsilon < p \hat{} n$
using ‹*p>1*› *real_arch_pow* **by** *blast*
define *q* **where** $q \equiv \lfloor p \hat{} n * x \rfloor - 1$
show *thesis*
proof
show *q / p ^ n < x x < real_of_int (q+2) / p ^ n*
using *assms* **by** (*simp_all add*: *q_def divide_simps floor_less_cancel mult.commute*)
show $|q / p \hat{} n - real\_of\_int\ (q+2) / p \hat{} n| < \varepsilon$
using *assms n* **by** (*simp add*: *q_def divide_simps mult.commute*)
qed

qed

lemma *padic_rational_approximation_straddle_pos*:
assumes $\varepsilon > 0$ $p > 1$ $x > 0$
obtains *n q r*
where *of_nat q / p^n < x x < of_nat r / p^n |q / p^n − r / p^n | < ε*
proof −
obtain *n q r*
where ∗: *of_int q / p^n < x x < of_int r / p^n |q / p^n − r / p^n | < ε*
using *padic_rational_approximation_straddle assms* **by** *metis*
then have $r \geq 0$
using *assms* **by** (*smt* (*verit*, *best*) *divide_nonpos_pos of_int_0_le_iff zero_less_power*)
show *thesis*
proof
show *real* (*max 0* (*nat q*)) / *p* ^ *n* < *x*
using ∗ **by** (*metis assms*(*3*) *div_0 max_nat.left_neutral nat_eq_iff2 of_nat_0 of_nat_nat*)
show *x* < *real* (*nat r*) / *p* ^ *n*
using ‹*r* ≥ *0*› ∗ **by** *force*
show |*real* (*max 0* (*nat q*)) / *p* ^ *n* − *real* (*nat r*) / *p* ^ *n*| < *ε*
using ∗ *assms* **by** (*simp add*: *divide_simps*)
qed
qed

lemma *padic_rational_approximation_straddle_pos_le*:
assumes $\varepsilon > 0$ $p > 1$ $x \geq 0$
obtains *n q r*
where *of_nat q / p^n ≤ x x < of_nat r / p^n |q / p^n − r / p^n | < ε*
proof −
obtain *n q r*
where ∗: *of_int q / p^n < x x < of_int r / p^n |q / p^n − r / p^n | < ε*
using *padic_rational_approximation_straddle assms* **by** *metis*
then have $r \geq 0$
using *assms* **by** (*smt* (*verit*, *best*) *divide_nonpos_pos of_int_0_le_iff zero_less_power*)
show *thesis*
proof
show *real* (*max 0* (*nat q*)) / *p* ^ *n* ≤ *x*
using ∗ *assms*(*3*) *nle_le* **by** *fastforce*
show *x* < *real* (*nat r*) / *p* ^ *n*
using ‹*r* ≥ *0*› ∗ **by** *force*
show |*real* (*max 0* (*nat q*)) / *p* ^ *n* − *real* (*nat r*) / *p* ^ *n*| < *ε*
using ∗ *assms* **by** (*simp add*: *divide_simps*)
qed
qed

Definition by recursion on dyadic rationals in [0,1]

lemma *recursion_on_dyadic_fractions*:
assumes *base*: *R a b*

```
    and step: ⋀x y. R x y ⟹ ∃z. R x z ∧ R z y and trans: ⋀x y z. ⟦R x y; R y z⟧ ⟹ R x z
  shows ∃f :: real ⇒ 'a. f 0 = a ∧ f 1 = b ∧
            (∀x ∈ dyadics ∩ {0..1}. ∀y ∈ dyadics ∩ {0..1}. x < y ⟶ R (f x) (f y))
proof −
  obtain mid where mid: R x y ⟹ R x (mid x y) R x y ⟹ R (mid x y) y for x y
    using step by metis
  define g where g ≡ rec_nat (λk. if k = 0 then a else b) (λn r k. if even k then r (k div 2) else mid (r ((k − 1) div 2)) (r ((Suc k) div 2)))
  have g0 [simp]: g 0 = (λk. if k = 0 then a else b)
    by (simp add: g_def)
  have gSuc [simp]: ⋀n. g(Suc n) = (λk. if even k then g n (k div 2) else mid (g n ((k − 1) div 2)) (g n ((Suc k) div 2)))
    by (auto simp: g_def)
  have g_eq_g: 2 ^ d * k = k' ⟹ g n k = g (n + d) k' for n d k k'
    by (induction d arbitrary: k k') auto
  have g n k = g n' k' if real k / 2^n = real k' / 2^n' n' ≤ n for k n k' n'
  proof −
    have real k = real k' * 2 ^ (n−n')
      using that by (simp add: power_diff divide_simps)
    then have k = k' * 2 ^ (n−n')
      using of_nat_eq_iff by fastforce
    with g_eq_g show ?thesis
      by (metis le_add_diff_inverse mult.commute that(2))
  qed
  then have g_eq_g: g n k = g n' k' if real k / 2 ^ n = real k' / 2 ^ n' for k n k' n'
    by (metis nat_le_linear that)
  then obtain f where (λ(k,n). g n k) = f ∘ (λ(k,n). k / 2 ^ n)
    using function_factors_left by (smt (verit, del_insts) case_prod_beta')
  then have f_eq_g: ⋀k n. f(real k / 2 ^ n) = g n k
    by (simp add: fun_eq_iff)
  show ?thesis
  proof (intro exI conjI strip)
    show f 0 = a
      by (metis f_eq_g g0 div_0 of_nat_0)
    show f 1 = b
      by (metis f_eq_g g0 div_by_1 of_nat_1_eq_iff power_0 zero_neq_one)
    show R (f x) (f y)
      if x: x ∈ dyadics ∩ {0..1} and y: y ∈ dyadics ∩ {0..1} and x < y for x y
    proof −
      obtain n1 k1 where xeq: x = real k1 / 2^n1 k1 ≤ 2^n1
        using x by (auto simp: dyadics_def)
      obtain n2 k2 where yeq: y = real k2 / 2^n2 k2 ≤ 2^n2
        using y by (auto simp: dyadics_def)
      have xcommon: x = real(2^n2 * k1) / 2 ^ (n1+n2)
        using xeq by (simp add: power_add)
```

```
    have ycommon: y = real(2^n1 * k2) / 2 ^ (n1+n2)
      using yeq by (simp add: power_add)
    have *: R (g n j) (g n k) if j < k k ≤ 2^n for n j k
      using that
    proof (induction n arbitrary: j k)
      case 0
      then show ?case
        by (simp add: base)
    next
      case (Suc n)
      show ?case
      proof (cases even j)
        case True
        then obtain a where [simp]: j = 2*a
          by blast
        show ?thesis
        proof (cases even k)
          case True
          with Suc show ?thesis
            by (auto elim!: evenE)
        next
          case False
          then obtain b where [simp]: k = Suc (2*b)
            using oddE by fastforce
          show ?thesis
            using Suc
            apply simp
              by (smt (verit, ccfv_SIG) less_Suc_eq linorder_not_le local.trans
mid(1) nat_mult_less_cancel1 pos2)
        qed
      next
        case False
        then obtain a where [simp]: j = Suc (2*a)
          using oddE by fastforce
        show ?thesis
        proof (cases even k)
          case True
          then obtain b where [simp]: k = 2*b
            by blast
          show ?thesis
            using Suc
            apply simp
        by (smt (verit, ccfv_SIG) Suc_leI Suc_lessD le_trans lessI linorder_neqE_nat
linorder_not_le local.trans mid(2) nat_mult_less_cancel1 pos2)
        next
          case False
          then obtain b where [simp]: k = Suc (2*b)
            using oddE by fastforce
          show ?thesis
```

```
            using Suc
            apply simp
          by (smt (verit) Suc_leI le_trans lessI less_or_eq_imp_le linorder_neqE_nat
linorder_not_le local.trans mid(1) mid(2) nat_mult_less_cancel1 pos2)
        qed
      qed
    qed
    show ?thesis
      unfolding xcommon ycommon f_eq_g
    proof (rule *)
      show 2 ^ n2 * k1 < 2 ^ n1 * k2
        using of_nat_less_iff ‹x < y› by (fastforce simp: xeq yeq field_simps)
      show 2 ^ n1 * k2 ≤ 2 ^ (n1 + n2)
        by (simp add: power_add yeq)
    qed
  qed
 qed
qed

lemma dyadics_add:
 assumes x ∈ dyadics y ∈ dyadics
 shows x+y ∈ dyadics
proof −
 obtain i j m n where x: x = of_nat i / 2 ^ m and y: y = of_nat j / 2 ^ n
   using assms by (auto simp: dyadics_def)
 have xcommon: x = of_nat(2^n * i) / 2 ^ (m+n)
   using x by (simp add: power_add)
 moreover
 have ycommon: y = of_nat(2^m * j) / 2 ^ (m+n)
   using y by (simp add: power_add)
 ultimately have x+y = (of_nat(2^n * i + 2^m * j)) / 2 ^ (m+n)
   by (simp add: field_simps)
 then show ?thesis
   unfolding dyadics_def by blast
qed

lemma dyadics_diff:
 fixes x :: 'a::linordered_field
 assumes x ∈ dyadics y ∈ dyadics y ≤ x
 shows x−y ∈ dyadics
proof −
 obtain i j m n where x: x = of_nat i / 2 ^ m and y: y = of_nat j / 2 ^ n
   using assms by (auto simp: dyadics_def)
 have j_le_i: j * 2 ^ m ≤ i * 2 ^ n
    using of_nat_le_iff ‹y ≤ x› unfolding x y by (fastforce simp add: di-
vide_simps)
 have xcommon: x = of_nat(2^n * i) / 2 ^ (m+n)
   using x by (simp add: power_add)
 moreover
```

```
  have ycommon: y = of_nat(2^m * j) / 2 ^(m+n)
    using y by (simp add: power_add)
  ultimately have x-y = (of_nat(2^n * i - 2^m * j)) / 2 ^(m+n)
    by (simp add: xcommon ycommon field_simps j_le_i of_nat_diff)
  then show ?thesis
    unfolding dyadics_def by blast
qed



theorem homeomorphic_monotone_image_interval:
  fixes f :: real ⇒ 'a::{real_normed_vector,complete_space}
  assumes cont_f: continuous_on {0..1} f
      and conn: ⋀y. connected ({0..1} ∩ f -` {y})
      and f_1not0: f 1 ≠ f 0
    shows (f ` {0..1}) homeomorphic {0..1::real}
proof -
  have ∃c d. a ≤ c ∧ c ≤ m ∧ m ≤ d ∧ d ≤ b ∧
             (∀x ∈ {c..d}. f x = f m) ∧
             (∀x ∈ {a..<c}. (f x ≠ f m)) ∧
             (∀x ∈ {d<..b}. (f x ≠ f m)) ∧
             (∀x ∈ {a..<c}. ∀y ∈ {d<..b}. f x ≠ f y)
    if m: m ∈ {a..b} and ab01: {a..b} ⊆ {0..1} for a b m
  proof -
    have comp: compact (f -` {f m} ∩ {0..1})
      by (simp add: compact_eq_bounded_closed bounded_Int closed_vimage_Int
cont_f)
    obtain c0 d0 where cd0: {0..1} ∩ f -` {f m} = {c0..d0}
      using connected_compact_interval_1 [of {0..1} ∩ f -` {f m}] conn comp
      by (metis Int_commute)
    with that have m ∈ cbox c0 d0
      by auto
    obtain c d where cd: {a..b} ∩ f -` {f m} = {c..d}
      using ab01 cd0
      by (rule_tac c=max a c0 and d=min b d0 in that) auto
    then have cdab: {c..d} ⊆ {a..b}
      by blast
    show ?thesis
    proof (intro exI conjI ballI)
      show a ≤ c d ≤ b
        using cdab cd m by auto
      show c ≤ m m ≤ d
        using cd m by auto
      show ⋀x. x ∈ {c..d} ⟹ f x = f m
        using cd by blast
      show f x ≠ f m if x ∈ {a..<c} for x
        using that m cd [THEN equalityD1, THEN subsetD] ‹c ≤ m› by force
      show f x ≠ f m if x ∈ {d<..b} for x
       using that m cd [THEN equalityD1, THEN subsetD, of x] ‹m ≤ d› by force
```

```
      show f x ≠ f y if x ∈ {a..<c} y ∈ {d<..b} for x y
      proof (cases f x = f m ∨ f y = f m)
        case True
        then show ?thesis
          using ‹⋀x. x ∈ {a..<c} ⟹ f x ≠ f m› that by auto
      next
        case False
        have False if f x = f y
        proof −
          have x ≤ m m ≤ y
            using ‹c ≤ m› ‹x ∈ {a..<c}›  ‹m ≤ d› ‹y ∈ {d<..b}› by auto
          then have x ∈ ({0..1} ∩ f −‘ {f y}) y ∈ ({0..1} ∩ f −‘ {f y})
            using ‹x ∈ {a..<c}› ‹y ∈ {d<..b}› ab01 by (auto simp: that)
          then have m ∈ ({0..1} ∩ f −‘ {f y})
              by (meson ‹m ≤ y› ‹x ≤ m› is_interval_connected_1 conn [of f y]
is_interval_1)
          with False show False by auto
        qed
        then show ?thesis by auto
      qed
    qed
  qed
  then obtain leftcut rightcut where LR:
    ⋀a b m. ⟦m ∈ {a..b}; {a..b} ⊆ {0..1}⟧ ⟹
          (a ≤ leftcut a b m ∧ leftcut a b m ≤ m ∧ m ≤ rightcut a b m ∧ rightcut
a b m ≤ b ∧
           (∀x ∈ {leftcut a b m..rightcut a b m}. f x = f m) ∧
           (∀x ∈ {a..<leftcut a b m}. f x ≠ f m) ∧
           (∀x ∈ {rightcut a b m<..b}. f x ≠ f m) ∧
           (∀x ∈ {a..<leftcut a b m}. ∀y ∈ {rightcut a b m<..b}. f x ≠ f y))
    apply atomize
    apply (clarsimp simp only: imp_conjL [symmetric] choice_iff choice_iff')
    apply (rule that, blast)
    done
  then have left_right: ⋀a b m. ⟦m ∈ {a..b}; {a..b} ⊆ {0..1}⟧ ⟹ a ≤ leftcut a
b m ∧ rightcut a b m ≤ b
       and  left_right_m: ⋀a b m. ⟦m ∈ {a..b}; {a..b} ⊆ {0..1}⟧ ⟹ leftcut a b m
≤ m ∧ m ≤ rightcut a b m
    by auto
  have left_neq: ⟦a ≤ x; x < leftcut a b m; a ≤ m; m ≤ b; {a..b} ⊆ {0..1}⟧ ⟹
f x ≠ f m
     and right_neq: ⟦rightcut a b m < x; x ≤ b; a ≤ m; m ≤ b; {a..b} ⊆ {0..1}⟧
⟹ f x ≠ f m
    and left_right_neq: ⟦a ≤ x; x < leftcut a b m; rightcut a b m < y; y ≤ b; a ≤
m; m ≤ b; {a..b} ⊆ {0..1}⟧ ⟹ f x ≠ f m
     and feqm: ⟦leftcut a b m ≤ x; x ≤ rightcut a b m; a ≤ m; m ≤ b; {a..b} ⊆
{0..1}⟧
                        ⟹ f x = f m for a b m x y
   by (meson atLeastAtMost_iff greaterThanAtMost_iff atLeastLessThan_iff LR)+
```

have *f_eqI*: $\bigwedge a\ b\ m\ x\ y.\ [\![leftcut\ a\ b\ m \le x;\ x \le rightcut\ a\ b\ m;\ leftcut\ a\ b\ m \le y;\ y \le rightcut\ a\ b\ m;$
$a \le m;\ m \le b;\ \{a..b\} \subseteq \{0..1\}]\!] \Longrightarrow f\ x = f\ y$
by (*metis feqm*)
define *u* **where** $u \equiv rightcut\ 0\ 1\ 0$
have *lc*[*simp*]: $leftcut\ 0\ 1\ 0 = 0$ **and** *u01*: $0 \le u\ u \le 1$
using *LR* [*of 0 0 1*] **by** (*auto simp*: *u_def*)
have *f0u*: $\bigwedge x.\ x \in \{0..u\} \Longrightarrow f\ x = f\ 0$
using *LR* [*of 0 0 1*] **unfolding** *u_def* [*symmetric*]
by (*metis* ‹$leftcut\ 0\ 1\ 0 = 0$› *atLeastAtMost_iff order_refl zero_le_one*)
have *fu1*: $\bigwedge x.\ x \in \{u<..1\} \Longrightarrow f\ x \neq f\ 0$
using *LR* [*of 0 0 1*] **unfolding** *u_def* [*symmetric*] **by** *fastforce*
define *v* **where** $v \equiv leftcut\ u\ 1\ 1$
have *rc*[*simp*]: $rightcut\ u\ 1\ 1 = 1$ **and** *v01*: $u \le v\ v \le 1$
using *LR* [*of 1 u 1*] *u01* **by** (*auto simp*: *v_def*)
have *fuv*: $\bigwedge x.\ x \in \{u..<v\} \Longrightarrow f\ x \neq f\ 1$
using *LR* [*of 1 u 1*] *u01 v_def* **by** *fastforce*
have *f0v*: $\bigwedge x.\ x \in \{0..<v\} \Longrightarrow f\ x \neq f\ 1$
by (*metis f_1not0 atLeastAtMost_iff atLeastLessThan_iff f0u fuv linear*)
have *fv1*: $\bigwedge x.\ x \in \{v..1\} \Longrightarrow f\ x = f\ 1$
using *LR* [*of 1 u 1*] *u01 v_def* **by** (*metis atLeastAtMost_iff atLeastatMost_subset_iff order_refl rc*)
define *a* **where** $a \equiv leftrec\ u\ v\ leftcut\ rightcut$
define *b* **where** $b \equiv rightrec\ u\ v\ leftcut\ rightcut$
define *c* **where** $c \equiv \lambda x.\ midpoint\ (a\ x)\ (b\ x)$
have *a_real* [*simp*]: $a\ (real\ j) = u$ **for** *j*
using *a_def leftrec_base*
by (*metis nonzero_mult_div_cancel_right of_nat_mult of_nat_numeral zero_neq_numeral*)
have *b_real* [*simp*]: $b\ (real\ j) = v$ **for** *j*
using *b_def rightrec_base*
by (*metis nonzero_mult_div_cancel_right of_nat_mult of_nat_numeral zero_neq_numeral*)
have *a41*: $a\ ((4 * real\ m + 1) / 2\hat{}Suc\ n) = a\ ((2 * real\ m + 1) / 2\hat{}n)$ **if** $n > 0$ **for** *m n*
using *that a_def leftrec_41* **by** *blast*
have *b41*: $b\ ((4 * real\ m + 1) / 2\hat{}Suc\ n) =$
$leftcut\ (a\ ((2 * real\ m + 1) / 2\hat{}n))$
$(b\ ((2 * real\ m + 1) / 2\hat{}n))$
$(c\ ((2 * real\ m + 1) / 2\hat{}n))$ **if** $n > 0$ **for** *m n*
using *that a_def b_def c_def rightrec_41* **by** *blast*
have *a43*: $a\ ((4 * real\ m + 3) / 2\hat{}Suc\ n) =$
$rightcut\ (a\ ((2 * real\ m + 1) / 2\hat{}n))$
$(b\ ((2 * real\ m + 1) / 2\hat{}n))$
$(c\ ((2 * real\ m + 1) / 2\hat{}n))$ **if** $n > 0$ **for** *m n*
using *that a_def b_def c_def leftrec_43* **by** *blast*
have *b43*: $b\ ((4 * real\ m + 3) / 2\hat{}Suc\ n) = b\ ((2 * real\ m + 1) / 2\hat{}n)$ **if** $n > 0$ **for** *m n*
using *that b_def rightrec_43* **by** *blast*
have *uabv*: $u \le a\ (real\ m / 2\ \hat{}\ n) \wedge a\ (real\ m / 2\ \hat{}\ n) \le b\ (real\ m / 2\ \hat{}\ n) \wedge b\ (real\ m / 2\ \hat{}\ n) \le v$ **for** *m n*

```
  proof (induction n arbitrary: m)
    case 0
    then show ?case by (simp add: v01)
  next
    case (Suc n p)
    show ?case
    proof (cases even p)
      case True
      then obtain m where p = 2*m by (metis evenE)
      then show ?thesis
        by (simp add: Suc.IH)
    next
      case False
      then obtain m where m: p = 2*m + 1 by (metis oddE)
      show ?thesis
      proof (cases n)
        case 0
        then show ?thesis
          by (simp add: a_def b_def leftrec_base rightrec_base v01)
      next
        case (Suc n')
        then have n > 0 by simp
        have a_le_c: a (real m / 2^n) ≤ c (real m / 2^n) for m
          unfolding c_def by (metis Suc.IH ge_midpoint_1)
        have c_le_b: c (real m / 2^n) ≤ b (real m / 2^n) for m
          unfolding c_def by (metis Suc.IH le_midpoint_1)
        have c_ge_u: c (real m / 2^n) ≥ u for m
          using Suc.IH a_le_c order_trans by blast
        have c_le_v: c (real m / 2^n) ≤ v for m
          using Suc.IH c_le_b order_trans by blast
        have a_ge_0: 0 ≤ a (real m / 2^n) for m
          using Suc.IH order_trans u01(1) by blast
        have b_le_1: b (real m / 2^n) ≤ 1 for m
          using Suc.IH order_trans v01(2) by blast
        have left_le: leftcut (a ((real m) / 2^n)) (b ((real m) / 2^n)) (c ((real m)
/ 2^n)) ≤ c ((real m) / 2^n) for m
          by (simp add: LR a_ge_0 a_le_c b_le_1 c_le_b)
        have right_ge: rightcut (a ((real m) / 2^n)) (b ((real m) / 2^n)) (c ((real
m) / 2^n)) ≥ c ((real m) / 2^n) for m
          by (simp add: LR a_ge_0 a_le_c b_le_1 c_le_b)
        show ?thesis
        proof (cases even m)
          case True
          then obtain r where r: m = 2*r by (metis evenE)
          show ?thesis
            using order_trans [OF left_le c_le_v, of 1+2*r] Suc.IH [of m+1]
            using a_le_c [of m+1] c_le_b [of m+1] a_ge_0 [of m+1] b_le_1 [of
m+1] left_right ‹n > 0›
            by (simp_all add: r m add.commute [of 1]  a41 b41 del: power_Suc)
```

```
      next
        case False
        then obtain r where r: m = 2*r + 1 by (metis oddE)
        show ?thesis
          using order_trans [OF c_ge_u right_ge, of 1+2*r] Suc.IH [of m]
            using a_le_c [of m] c_le_b [of m] a_ge_0 [of m] b_le_1 [of m]
left_right ‹n > 0›
          apply (simp_all add: r m add.commute [of 3] a43 b43 del: power_Suc)
          by (simp add: add.commute)
      qed
    qed
  qed
  have a_ge_0 [simp]: 0 ≤ a(m / 2^n) and b_le_1 [simp]: b(m / 2^n) ≤ 1 for
m::nat and n
    using uabv order_trans u01 v01 by blast+
  then have b_ge_0 [simp]: 0 ≤ b(m / 2^n) and a_le_1 [simp]: a(m / 2^n) ≤
1 for m::nat and n
    using uabv order_trans by blast+
  have alec [simp]: a(m / 2^n) ≤ c(m / 2^n) and cleb [simp]: c(m / 2^n) ≤ b(m
/ 2^n) for m::nat and n
    by (auto simp: c_def ge_midpoint_1 le_midpoint_1 uabv)
  have c_ge_0 [simp]: 0 ≤ c(m / 2^n) and c_le_1 [simp]: c(m / 2^n) ≤ 1 for
m::nat and n
    using a_ge_0 alec b_le_1 cleb order_trans by blast+
  have ⟦d = m−n; odd j; |real i / 2^m − real j / 2^n| < 1/2 ^ n⟧
       ⟹ (a(j / 2^n)) ≤ (c(i / 2^m)) ∧ (c(i / 2^m)) ≤ (b(j / 2^n)) for d i j m
n
  proof (induction d arbitrary: j n rule: less_induct)
    case (less d j n)
    show ?case
    proof (cases m ≤ n)
      case True
      have |2^n| * |real i / 2^m − real j / 2^n| = 0
      proof (rule Ints_nonzero_abs_less1)
        have (real i * 2^n − real j * 2^m) / 2^m = (real i * 2^n) / 2^m − (real j
* 2^m) / 2^m
          using diff_divide_distrib by blast
        also have ... = (real i * 2 ^ (n−m)) − (real j)
          using True by (auto simp: power_diff field_simps)
        also have ... ∈ ℤ
          by simp
        finally have (real i * 2^n − real j * 2^m) / 2^m ∈ ℤ .
        with True Ints_abs show |2^n| * |real i / 2^m − real j / 2^n| ∈ ℤ
          by (fastforce simp: field_split_simps)
        show ||2^n| * |real i / 2^m − real j / 2^n|| < 1
          using less.prems by (auto simp: field_split_simps)
      qed
      then have real i / 2^m = real j / 2^n
```

```
      by auto
    then show ?thesis
      by auto
  next
    case False
    then have n < m by auto
    obtain k where k: j = Suc (2*k)
      using ‹odd j› oddE by fastforce
    show ?thesis
    proof (cases n > 0)
      case False
      then have a (real j / 2^n) = u
        by simp
      also have ... ≤ c (real i / 2^m)
        using alec uabv by (blast intro: order_trans)
      finally have ac: a (real j / 2^n) ≤ c (real i / 2^m) .
      have c (real i / 2^m) ≤ v
        using cleb uabv by (blast intro: order_trans)
      also have ... = b (real j / 2^n)
        using False by simp
      finally show ?thesis
        by (auto simp: ac)
    next
      case True show ?thesis
      proof (cases i / 2^m j / 2^n rule: linorder_cases)
        case less
        moreover have real (4 * k + 1) / 2 ^ Suc n + 1 / (2 ^ Suc n) = real j
/ 2 ^ n
          using k by (force simp: field_split_simps)
        moreover have |real i / 2 ^ m − j / 2 ^ n| < 2 / (2 ^ Suc n)
          using less.prems by simp
        ultimately have closer: |real i / 2 ^ m − real (4 * k + 1) / 2 ^ Suc n|
< 1 / (2 ^ Suc n)
          using less.prems by linarith
        have a (real (4 * k + 1) / 2 ^ Suc n) ≤ c (i / 2 ^ m) ∧
              c (real i / 2 ^ m) ≤ b (real (4 * k + 1) / 2 ^ Suc n)
        proof (rule less.IH [OF _ refl])
          show m − Suc n < d
            using ‹n < m› diff_less_mono2 less.prems(1) lessI by presburger
          show |real i / 2 ^ m − real (4 * k + 1) / 2 ^ Suc n| < 1 / 2 ^ Suc n
            using closer ‹n < m› ‹d = m − n› by (auto simp: field_split_simps
‹n < m› diff_less_mono2)
        qed auto
        then show ?thesis
          using LR [of c((2*k + 1) / 2^n) a((2*k + 1) / 2^n) b((2*k + 1) /
2^n)]
          using alec [of 2*k+1] cleb [of 2*k+1] a_ge_0 [of 2*k+1] b_le_1 [of
2*k+1]
          using k a41 b41 ‹0 < n›
```

```
        by (simp add: add.commute)
      next
        case equal then show ?thesis by simp
      next
        case greater
        moreover have real (4 * k + 3) / 2 ^ Suc n - 1 / (2 ^ Suc n) = real j
/ 2 ^ n
          using k by (force simp: field_split_simps)
        moreover have |real i / 2 ^ m - real j / 2 ^ n| < 2 * 1 / (2 ^ Suc n)
          using less.prems by simp
        ultimately have closer: |real i / 2 ^ m - real (4 * k + 3) / 2 ^ Suc n|
< 1 / (2 ^ Suc n)
          using less.prems by linarith
        have a (real (4 * k + 3) / 2 ^ Suc n) ≤ c (real i / 2 ^ m) ∧
              c (real i / 2 ^ m) ≤ b (real (4 * k + 3) / 2 ^ Suc n)
        proof (rule less.IH [OF _ refl])
          show m - Suc n < d
            using ‹n < m› diff_less_mono2 less.prems(1) by blast
          show |real i / 2 ^ m - real (4 * k + 3) / 2 ^ Suc n| < 1 / 2 ^ Suc n
            using closer ‹n < m› ‹d = m - n› by (auto simp: field_split_simps
‹n < m› diff_less_mono2)
        qed auto
        then show ?thesis
          using LR [of c((2*k + 1) / 2^n) a((2*k + 1) / 2^n) b((2*k + 1) /
2^n)]
          using alec [of 2*k+1] cleb [of 2*k+1] a_ge_0 [of 2*k+1] b_le_1 [of
2*k+1]
          using k a43 b43 ‹0 < n›
          by (simp add: add.commute)
      qed
    qed
  qed
  then have aj_le_ci: a (real j / 2 ^ n) ≤ c (real i / 2 ^ m)
    and ci_le_bj: c (real i / 2 ^ m) ≤ b (real j / 2 ^ n) if odd j |real i / 2^m -
real j / 2^n| < 1/2 ^ n for i j m n
    using that by blast+
  have close_ab: odd m ⟹ |a (real m / 2 ^ n) - b (real m / 2 ^ n)| ≤ 2 / 2^n
for m n
  proof (induction n arbitrary: m)
    case 0
    with u01 v01 show ?case by auto
  next
    case (Suc n m)
    with oddE obtain k where k: m = Suc (2*k) by fastforce
    show ?case
    proof (cases n > 0)
      case False
      with u01 v01 show ?thesis
```

```
      by (simp add: a_def b_def leftrec_base rightrec_base)
    next
      case True
      show ?thesis
      proof (cases even k)
        case True
        then obtain j where j: k = 2*j by (metis evenE)
        have |a ((2 * real j + 1) / 2 ^ n) - (b ((2 * real j + 1) / 2 ^ n))| ≤ 2/2
^ n
        proof -
          have odd (Suc k)
            using True by auto
          then show ?thesis
         by (metis (no_types) Groups.add_ac(2) Suc.IH j of_nat_Suc of_nat_mult
of_nat_numeral)
        qed
        moreover have a ((2 * real j + 1) / 2 ^ n) ≤
                      leftcut (a ((2 * real j + 1) / 2 ^ n)) (b ((2 * real j + 1) / 2 ^
n)) (c ((2 * real j + 1) / 2 ^ n))
          using alec [of 2*j+1] cleb [of 2*j+1] a_ge_0 [of 2*j+1]  b_le_1 [of
2*j+1]
          by (auto simp: add.commute left_right)
        moreover have leftcut (a ((2 * real j + 1) / 2 ^ n)) (b ((2 * real j + 1)
/ 2 ^ n)) (c ((2 * real j + 1) / 2 ^ n)) ≤
                        c ((2 * real j + 1) / 2 ^ n)
          using alec [of 2*j+1] cleb [of 2*j+1] a_ge_0 [of 2*j+1]  b_le_1 [of
2*j+1]
          by (auto simp: add.commute left_right_m)
        ultimately have |a ((2 * real j + 1) / 2 ^ n) -
                         leftcut (a ((2 * real j + 1) / 2 ^ n)) (b ((2 * real j + 1) / 2
^ n)) (c ((2 * real j + 1) / 2 ^ n))|
                    ≤ 2/2 ^ Suc n
          by (simp add: c_def midpoint_def)
        with j k ‹n > 0› show ?thesis
          by (simp add: add.commute [of 1] a41 b41 del: power_Suc)
      next
        case False
        then obtain j where j: k = 2*j + 1 by (metis oddE)
        have |a ((2 * real j + 1) / 2 ^ n) - (b ((2 * real j + 1) / 2 ^ n))| ≤ 2/2
^ n
          using Suc.IH [OF False] j by (auto simp: algebra_simps)
        moreover have c ((2 * real j + 1) / 2 ^ n) ≤
                      rightcut (a ((2 * real j + 1) / 2 ^ n)) (b ((2 * real j + 1) / 2
^ n)) (c ((2 * real j + 1) / 2 ^ n))
          using alec [of 2*j+1] cleb [of 2*j+1] a_ge_0 [of 2*j+1]  b_le_1 [of
2*j+1]
          by (auto simp: add.commute left_right_m)
        moreover have rightcut (a ((2 * real j + 1) / 2 ^ n)) (b ((2 * real j + 1)
/ 2 ^ n)) (c ((2 * real j + 1) / 2 ^ n)) ≤
```

```
                    b ((2 * real j + 1) / 2 ^ n)
          using alec [of 2*j+1] cleb [of 2*j+1] a_ge_0 [of 2*j+1]  b_le_1 [of
2*j+1]
        by (auto simp: add.commute left_right)
       ultimately have |rightcut (a ((2 * real j + 1) / 2 ^ n)) (b ((2 * real j +
1) / 2 ^ n)) (c ((2 * real j + 1) / 2 ^ n)) -
                    b ((2 * real j + 1) / 2 ^ n)|  ≤  2/2 ^ Suc n
        by (simp add: c_def midpoint_def)
       with j k ‹n > 0› show ?thesis
        by (simp add: add.commute [of 3] a43 b43 del: power_Suc)
     qed
   qed
  qed
  have m1_to_3: 4 * real k - 1 = real (4 * (k-1)) + 3 if 0 < k for k
   using that by auto
  have fb_eq_fa: ⟦0 < j; 2*j < 2 ^ n⟧ ⟹ f(b((2 * real j - 1) / 2^n)) = f(a((2
* real j + 1) / 2^n)) for n j
  proof (induction n arbitrary: j)
   case 0
   then show ?case by auto
  next
   case (Suc n j) show ?case
   proof (cases n > 0)
    case False
    with Suc.prems show ?thesis by auto
   next
    case True
    show ?thesis proof (cases even j)
     case True
     then obtain k where k: j = 2*k by (metis evenE)
     with ‹0 < j› have k > 0 2 * k < 2 ^ n
      using Suc.prems(2) k by auto
     with k ‹0 < n› Suc.IH [of k] show ?thesis
      by (simp add: m1_to_3 a41 b43 del: power_Suc) (auto simp: of_nat_diff)
    next
     case False
     then obtain k where k: j = 2*k + 1 by (metis oddE)
     have f (leftcut (a ((2 * k + 1) / 2^n)) (b ((2 * k + 1) / 2^n)) (c ((2 * k
+ 1) / 2^n)))
           = f (c ((2 * k + 1) / 2^n))
       f (c ((2 * k + 1) / 2^n))
           = f (rightcut (a ((2 * k + 1) / 2^n)) (b ((2 * k + 1) / 2^n)) (c ((2
* k + 1) / 2^n)))
      using alec [of 2*k+1 n] cleb [of 2*k+1 n] a_ge_0 [of 2*k+1 n]  b_le_1
[of 2*k+1 n] k
      using left_right_m [of c((2*k + 1) / 2^n) a((2*k + 1) / 2^n) b((2*k +
1) / 2^n)]
     by (auto simp: add.commute  feqm [OF order_refl] feqm [OF _ order_refl,
symmetric])
```

```
      then
      show ?thesis
        by (simp add: k add.commute [of 1] add.commute [of 3] a43 b41‹0 < n›
del: power_Suc)
    qed
  qed
  qed
  have f_eq_fc: ⟦0 < j; j < 2 ^ n⟧
              ⟹ f(b((2*j - 1) / 2 ^ (Suc n))) = f(c(j / 2^n)) ∧
                 f(a((2*j + 1) / 2 ^ (Suc n))) = f(c(j / 2^n)) for n and j::nat
  proof (induction n arbitrary: j)
    case 0
    then show ?case by auto
  next
    case (Suc n)
    show ?case
    proof (cases even j)
      case True
      then obtain k where k: j = 2*k by (metis evenE)
      then have less2n: k < 2 ^ n
        using Suc.prems(2) by auto
      have 0 < k using ‹0 < j› k by linarith
      then have m1_to_3: real (4 * k - Suc 0) = real (4 * (k-1)) + 3
        by auto
      then show ?thesis
        using Suc.IH [of k] k ‹0 < k›
       by (simp add: less2n add.commute [of 1] m1_to_3 a41 b43 del: power_Suc)
(auto simp: of_nat_diff)
    next
      case False
      then obtain k where k: j = 2*k + 1 by (metis oddE)
      with Suc.prems have k < 2^n by auto
      show ?thesis
        using alec [of 2*k+1 Suc n] cleb [of 2*k+1 Suc n] a_ge_0 [of 2*k+1 Suc
n]  b_le_1 [of 2*k+1 Suc n] k
       using left_right_m [of c((2*k + 1) / 2 ^ Suc n) a((2*k + 1) / 2 ^ Suc n)
b((2*k + 1) / 2 ^ Suc n)]
       apply (simp add: add.commute [of 1] add.commute [of 3] m1_to_3 b41 a43
del: power_Suc)
        apply (force intro: feqm)
        done
    qed
  qed
  define D01 where D01 ≡ {0<..<1} ∩ (⋃k m. {real m / 2^k})
  have cloD01 [simp]: closure D01 = {0..1}
    unfolding D01_def
    by (subst closure_dyadic_rationals_in_convex_set_pos_1) auto
  have uniformly_continuous_on D01 (f ∘ c)
  proof (clarsimp simp: uniformly_continuous_on_def)
```

```
    fix e::real
    assume 0 < e
    have ucontf: uniformly_continuous_on {0..1} f
      by (simp add: compact_uniformly_continuous [OF cont_f])
    then obtain d where 0 < d and d: ⋀x x'. ⟦x ∈ {0..1}; x' ∈ {0..1}; norm
(x' − x) < d⟧ ⟹ norm (f x' − f x) < e/2
      unfolding uniformly_continuous_on_def dist_norm
      by (metis ‹0 < e› less_divide_eq_numeral1(1) mult_zero_left)
    obtain n where n: 1/2^n < min d 1
    by (metis ‹0 < d› divide_less_eq_1 less_numeral_extra(1) min_def one_less_numeral_iff
power_one_over real_arch_pow_inv semiring_norm(76) zero_less_numeral)
    with gr0I have n > 0
      by (force simp: field_split_simps)
    show ∃ d>0. ∀ x∈D01. ∀ x'∈D01. dist x' x < d ⟶ dist (f (c x')) (f (c x)) <
e
    proof (intro exI ballI impI conjI)
      show (0::real) < 1/2^n by auto
    next
      have dist_fc_close: dist (f(c(real i / 2^m))) (f(c(real j / 2^n))) < e/2
        if i: 0 < i i < 2 ^ m and j: 0 < j j < 2 ^ n and clo: abs(i / 2^m − j /
2^n) < 1/2 ^ n for i j m
      proof −
        have abs3: |x − a| < e ⟹ x = a ∨ |x − (a − e/2)| < e/2 ∨ |x − (a +
e/2)| < e/2 for x a e::real
          by linarith
        consider i / 2 ^ m = j / 2 ^ n
          | |i / 2 ^ m − (2 * j − 1) / 2 ^ Suc n| < 1/2 ^ Suc n
          | |i / 2 ^ m − (2 * j + 1) / 2 ^ Suc n| < 1/2 ^ Suc n
          using abs3 [OF clo] j by (auto simp: field_simps of_nat_diff)
        then show ?thesis
        proof cases
          case 1 with ‹0 < e› show ?thesis by auto
        next
          case 2
          have ∗: abs(a − b) ≤ 1/2 ^ n ∧ 1/2 ^ n < d ∧ a ≤ c ∧ c ≤ b ⟹ b −
c < d for a b c
            by auto
          have norm (c (real i / 2 ^ m) − b (real (2 * j − 1) / 2 ^ Suc n)) < d
            using 2 j n close_ab [of 2∗j−1 Suc n]
            using b_ge_0 [of 2∗j−1 Suc n] b_le_1 [of 2∗j−1 Suc n]
            using aj_le_ci [of 2∗j−1 i m Suc n]
            using ci_le_bj [of 2∗j−1 i m Suc n]
            apply (simp add: divide_simps of_nat_diff del: power_Suc)
            apply (auto simp: divide_simps intro!: ∗)
            done
          moreover have f(c(j / 2^n)) = f(b ((2∗j − 1) / 2 ^ (Suc n)))
            using f_eq_fc [OF j] by metis
          ultimately show ?thesis
           by (metis dist_norm atLeastAtMost_iff b_ge_0 b_le_1 c_ge_0 c_le_1
```

```
d)
      next
        case 3
        have ∗: abs(a − b) ≤ 1/2 ^ n ∧ 1/2 ^ n < d ∧ a ≤ c ∧ c ≤ b ⟹ c −
a < d for a b c
          by auto
        have norm (c (real i / 2 ^ m) − a (real (2 ∗ j + 1) / 2 ^ Suc n)) < d
          using 3 j n close_ab [of 2∗j+1 Suc n]
          using b_ge_0 [of 2∗j+1 Suc n] b_le_1 [of 2∗j+1 Suc n]
          using aj_le_ci [of 2∗j+1 i m Suc n]
          using ci_le_bj [of 2∗j+1 i m Suc n]
          apply (simp add: divide_simps of_nat_diff del: power_Suc)
          apply (auto simp: divide_simps intro!: ∗)
          done
        moreover have f(c(j / 2^n)) = f(a ((2∗j + 1) / 2 ^ (Suc n)))
          using f_eq_fc [OF j] by metis
        ultimately show ?thesis
         by (metis dist_norm a_ge_0 atLeastAtMost_iff a_ge_0 a_le_1 c_ge_0
c_le_1 d)
      qed
    qed
    show dist (f (c x′)) (f (c x)) < e
      if x ∈ D01 x′ ∈ D01 dist x′ x < 1/2^n for x x′
      using that unfolding D01_def dyadics_in_open_unit_interval
    proof clarsimp
      fix i k::nat and m p
      assume i: 0 < i i < 2 ^ m and k: 0<k k < 2 ^ p
      assume clo: dist (real k / 2 ^ p) (real i / 2 ^ m) < 1/2 ^ n
      obtain j::nat where 0 < j j < 2 ^ n
        and clo_ij: abs(i / 2^m − j / 2^n) < 1/2 ^ n
        and clo_kj: abs(k / 2^p − j / 2^n) < 1/2 ^ n
      proof −
        have max (2^n ∗ i / 2^m) (2^n ∗ k / 2^p) ≥ 0
          by (auto simp: le_max_iff_disj)
        then obtain j where floor (max (2^n∗i / 2^m) (2^n∗k / 2^p)) = int j
          using zero_le_floor zero_le_imp_eq_int by blast
        then have j_le: real j ≤ max (2^n ∗ i / 2^m) (2^n ∗ k / 2^p)
          and less_j1: max (2^n ∗ i / 2^m) (2^n ∗ k / 2^p) < real j + 1
            using floor_correct [of max (2^n ∗ i / 2^m) (2^n ∗ k / 2^p)] by
linarith+
        show thesis
        proof (cases j = 0)
          case True
          show thesis
          proof
            show (1::nat) < 2 ^ n
              by (metis Suc_1 ‹0 < n› lessI one_less_power)
            show |real i / 2 ^ m − real 1/2 ^ n| < 1/2 ^ n
              using i less_j1 by (simp add: dist_norm field_simps True)
```

```
      show |real k / 2 ^ p − real 1/2 ^ n| < 1/2 ^ n
        using k less_j1 by (simp add: dist_norm field_simps True)
    qed simp
  next
    case False
    have 1: real j * 2 ^ m < real i * 2 ^ n
      if j: real j * 2 ^ p ≤ real k * 2 ^ n and k: real k * 2 ^ m < real i * 2
^ p
      for i k m p
    proof −
      have real j * 2 ^ p * 2 ^ m ≤ real k * 2 ^ n * 2 ^ m
        using j by simp
      moreover have real k * 2 ^ m * 2 ^ n < real i * 2 ^ p * 2 ^ n
        using k by simp
      ultimately have real j * 2 ^ p * 2 ^ m < real i * 2 ^ p * 2 ^ n
        by (simp only: mult_ac)
      then show ?thesis
        by simp
    qed
    have 2: real j * 2 ^ m < 2 ^ m + real i * 2 ^ n
      if j: real j * 2 ^ p ≤ real k * 2 ^ n and k: real k * (2 ^ m * 2 ^ n) <
2 ^ m * 2 ^ p + real i * (2 ^ n * 2 ^ p)
      for i k m p
    proof −
      have real j * 2 ^ p * 2 ^ m ≤ real k * (2 ^ m * 2 ^ n)
        using j by simp
      also have ... < 2 ^ m * 2 ^ p + real i * (2 ^ n * 2 ^ p)
        by (rule k)
     finally have (real j * 2 ^ m) * 2 ^ p < (2 ^ m + real i * 2 ^ n) * 2 ^ p
        by (simp add: algebra_simps)
      then show ?thesis
        by simp
    qed
    have 3: real j * 2 ^ p < 2 ^ p + real k * 2 ^ n
     if j: real j * 2 ^ m ≤ real i * 2 ^ n and i: real i * 2 ^ p ≤ real k * 2 ^
m
    proof −
      have real j * 2 ^ m * 2 ^ p ≤ real i * 2 ^ n * 2 ^ p
        using j by simp
      moreover have real i * 2 ^ p * 2 ^ n ≤ real k * 2 ^ m * 2 ^ n
        using i by simp
      ultimately have real j * 2 ^ m * 2 ^ p ≤ real k * 2 ^ m * 2 ^ n
        by (simp only: mult_ac)
      then have real j * 2 ^ p ≤ real k * 2 ^ n
        by simp
      also have ... < 2 ^ p + real k * 2 ^ n
        by auto
      finally show ?thesis by simp
    qed
```

```
          show ?thesis
          proof
            have 2 ^ n * real i / 2 ^ m < 2 ^ n 2 ^ n * real k / 2 ^ p < 2 ^ n
              using i k by (auto simp: field_simps)
            then have max (2^n * i / 2^m) (2^n * k / 2^p) < 2^n
              by simp
            with j_le have real j < 2 ^ n by linarith
            then show j < 2 ^ n
              by auto
            have |real i * 2 ^ n - real j * 2 ^ m| < 2 ^ m
              using clo less_j1 j_le
               by (auto simp: le_max_iff_disj field_split_simps dist_norm abs_if
split: if_split_asm dest: 1 2)
            then show |real i / 2 ^ m - real j / 2 ^ n| < 1/2 ^ n
              by (auto simp: field_split_simps)
            have |real k * 2 ^ n - real j * 2 ^ p| < 2 ^ p
              using clo less_j1 j_le
               by (auto simp: le_max_iff_disj field_split_simps dist_norm abs_if
split: if_split_asm dest: 3 2)
            then show |real k / 2 ^ p - real j / 2 ^ n| < 1/2 ^ n
              by (auto simp: le_max_iff_disj field_split_simps dist_norm)
          qed (use False in simp)
        qed
      qed
      show dist (f (c (real k / 2 ^ p))) (f (c (real i / 2 ^ m))) < e
      proof (rule dist_triangle_half_l)
        show dist (f (c (real k / 2 ^ p))) (f(c(j / 2^n))) < e/2
          using ‹0 < j› ‹j < 2 ^ n› k clo_kj
          by (intro dist_fc_close) auto
        show dist (f (c (real i / 2 ^ m))) (f (c (real j / 2 ^ n))) < e/2
          using ‹0 < j› ‹j < 2 ^ n› i clo_ij
          by (intro dist_fc_close) auto
      qed
    qed
  qed
  then obtain h where ucont_h: uniformly_continuous_on {0..1} h
    and fc_eq: ⋀x. x ∈ D01 ⟹ (f ∘ c) x = h x
  proof (rule uniformly_continuous_on_extension_on_closure [of D01 f ∘ c])
  qed (use closure_subset [of D01] in ‹auto intro!: that›)
  then have cont_h: continuous_on {0..1} h
    using uniformly_continuous_imp_continuous by blast
  have h_eq: h (real k / 2 ^ m) = f (c (real k / 2 ^ m)) if 0 < k k < 2^m for k
m
    using fc_eq that by (force simp: D01_def)
  have h ‘ {0..1} = f ‘ {0..1}
  proof
    have h ‘ (closure D01) ⊆ f ‘ {0..1}
    proof (rule image_closure_subset)
```

```
    show continuous_on (closure D01) h
      using cont_h by simp
    show closed (f ‘ {0..1})
      using compact_continuous_image [OF cont_f] compact_imp_closed by
blast
    show h ‘ D01 ⊆ f ‘ {0..1}
      by (force simp: dyadics_in_open_unit_interval D01_def h_eq)
  qed
  with cloD01 show h ‘ {0..1} ⊆ f ‘ {0..1} by simp
  have a12 [simp]: a (1/2) = u
    by (metis a_def leftrec_base numeral_One of_nat_numeral)
  have b12 [simp]: b (1/2) = v
    by (metis b_def rightrec_base numeral_One of_nat_numeral)
  have f ‘ {0..1} ⊆ closure(h ‘ D01)
   proof (clarsimp simp: closure_approachable dyadics_in_open_unit_interval
D01_def)
    fix x e::real
    assume 0 ≤ x x ≤ 1 0 < e
    have ucont_f: uniformly_continuous_on {0..1} f
      using compact_uniformly_continuous cont_f by blast
    then obtain δ where δ > 0
      and δ: ⋀x x'. ⟦x ∈ {0..1}; x' ∈ {0..1}; dist x' x < δ⟧ ⟹ norm (f x' − f
x) < e
      using ‹0 < e› by (auto simp: uniformly_continuous_on_def dist_norm)
     have ∗: ∃ m::nat. ∃ y. odd m ∧ 0 < m ∧ m < 2 ^ n ∧ y ∈ {a(m / 2^n) ..
b(m / 2^n)} ∧ f y = f x
      if n ≠ 0 for n
      using that
    proof (induction n)
      case 0 then show ?case by auto
    next
      case (Suc n)
      show ?case
      proof (cases n=0)
        case True
        consider x ∈ {0..u} | x ∈ {u..v} | x ∈ {v..1}
          using ‹0 ≤ x› ‹x ≤ 1› by force
        then have ∃ y≥a (real 1/2). y ≤ b (real 1/2) ∧ f y = f x
        proof cases
          case 1
          then show ?thesis
            using uabv [of 1 1] f0u [of u] f0u [of x] by force
        next
          case 2
          then show ?thesis
            by (rule_tac x=x in exI) auto
        next
          case 3
          then show ?thesis
```

```
          using uabv [of 1 1] fv1 [of v] fv1 [of x] by force
        qed
        with ‹n=0› show ?thesis
          by (rule_tac x=1 in exI) auto
      next
        case False
        with Suc obtain m y
          where odd m 0 < m and mless: m < 2 ^ n
            and y: y ∈ {a (real m / 2 ^ n)..b (real m / 2 ^ n)} and feq: f y = f x
          by metis
        then obtain j where j: m = 2*j + 1 by (metis oddE)
        have j4: 4 * j + 1 < 2 ^ Suc n
          using mless j by (simp add: algebra_simps)

        consider y ∈ {a((2*j + 1) / 2^n) .. b((4*j + 1) / 2 ^ (Suc n))}
          | y ∈ {b((4*j + 1) / 2 ^ (Suc n)) .. a((4*j + 3) / 2 ^ (Suc n))}
          | y ∈ {a((4*j + 3) / 2 ^ (Suc n)) .. b((2*j + 1) / 2^n)}
          using y j by force
        then show ?thesis
        proof cases
          case 1
          show ?thesis
          proof (intro exI conjI)
            show y ∈ {a (real (4 * j + 1) / 2 ^ Suc n)..b (real (4 * j + 1) / 2 ^
Suc n)}
              using mless j ‹n ≠ 0› 1 by (simp add: a41 b41 add.commute [of 1]
del: power_Suc)
          qed (use feq j4 in auto)
        next
          case 2
          show ?thesis
          proof (intro exI conjI)
            show b (real (4 * j + 1) / 2 ^ Suc n) ∈ {a (real (4 * j + 1) / 2 ^
Suc n)..b (real (4 * j + 1) / 2 ^ Suc n)}
              using ‹n ≠ 0› alec [of 2*j+1 n] cleb [of 2*j+1 n] a_ge_0 [of 2*j+1
n] b_le_1 [of 2*j+1 n]
              using left_right [of c((2*j + 1) / 2^n) a((2*j + 1) / 2^n) b((2*j
+ 1) / 2^n)]
              by (simp add: a41 b41 add.commute [of 1] del: power_Suc)
            show f (b (real (4 * j + 1) / 2 ^ Suc n)) = f x
              using ‹n ≠ 0› 2
                using alec [of 2*j+1 n] cleb [of 2*j+1 n] a_ge_0 [of 2*j+1 n]
b_le_1 [of 2*j+1 n]
              by (force simp add: b41 a43 add.commute [of 1] feq [symmetric] simp
del: power_Suc intro: f_eqI)
          qed (use j4 in auto)
        next
          case 3
          show ?thesis
```

```
          proof (intro exI conjI)
            show 4 * j + 3 < 2 ^ Suc n
              using mless j by simp
            show f y = f x
              by fact
            show y ∈ {a (real (4 * j + 3) / 2 ^ Suc n) .. b (real (4 * j + 3) / 2
^ Suc n)}
              using 3 False b43 [of n j] by (simp add: add.commute)
          qed (use 3 in auto)
        qed
      qed
    qed
    obtain n where n: 1/2^n < min (δ / 2) 1
    by (metis ‹0 < δ› divide_less_eq_1 less_numeral_extra(1) min_less_iff_conj
one_less_numeral_iff power_one_over real_arch_pow_inv semiring_norm(76)
zero_less_divide_iff zero_less_numeral)
    with gr0I have n ≠ 0
      by fastforce
    with * obtain m::nat and y
      where odd m 0 < m and mless: m < 2 ^ n
        and y: a(m / 2^n) ≤ y ∧ y ≤ b(m / 2^n) and feq: f x = f y
      by (metis atLeastAtMost_iff)
    then have 0 ≤ y y ≤ 1
      by (meson a_ge_0 b_le_1 order.trans)+
    moreover have y < δ + c (real m / 2 ^ n) c (real m / 2 ^ n) < δ + y
      using y alec [of m n] cleb [of m n] n field_sum_of_halves close_ab [OF
‹odd m›, of n]
      by linarith+
    moreover note ‹0 < m› mless ‹0 ≤ x› ‹x ≤ 1›
    ultimately have dist (h (real m / 2 ^ n)) (f x) < e
      by (auto simp: dist_norm h_eq feq δ)
    then show ∃k. ∃m∈{0<..<2 ^ k}. dist (h (real m / 2 ^ k)) (f x) < e
      using ‹0 < m› greaterThanLessThan_iff mless by blast
  qed
  also have ... ⊆ h ‘ {0..1}
  proof (rule closure_minimal)
    show h ‘ D01 ⊆ h ‘ {0..1}
      using cloD01 closure_subset by blast
    show closed (h ‘ {0..1})
      using compact_continuous_image [OF cont_h] compact_imp_closed by
auto
  qed
  finally show f ‘ {0..1} ⊆ h ‘ {0..1} .
qed
moreover have inj_on h {0..1}
proof −
  have u < v
  by (metis atLeastAtMost_iff f0u f_1not0 fv1 order.not_eq_order_implies_strict
u01(1) u01(2) v01(1))
```

```
    have f_not_fu: ⋀x. ⟦u < x; x ≤ v⟧ ⟹ f x ≠ f u
      by (metis atLeastAtMost_iff f0u fu1 greaterThanAtMost_iff order_refl or-
der_trans u01(1) v01(2))
    have f_not_fv: ⋀x. ⟦u ≤ x; x < v⟧ ⟹ f x ≠ f v
    by (metis atLeastAtMost_iff order_refl order_trans v01(2) atLeastLessThan_iff
fuv fv1)
    have a_less_b:
        a(j / 2^n) < b(j / 2^n) ∧
         (∀x. a(j / 2^n) < x ⟶ x ≤ b(j / 2^n) ⟶ f x ≠ f(a(j / 2^n))) ∧
         (∀x. a(j / 2^n) ≤ x ⟶ x < b(j / 2^n) ⟶ f x ≠ f(b(j / 2^n))) for n
and j::nat
    proof (induction n arbitrary: j)
      case 0 then show ?case
        by (simp add: ‹u < v› f_not_fu f_not_fv)
    next
      case (Suc n j) show ?case
      proof (cases n > 0)
        case False then show ?thesis
          by (auto simp: a_def b_def leftrec_base rightrec_base ‹u < v› f_not_fu
f_not_fv)
      next
        case True show ?thesis
        proof (cases even j)
          case True
          with ‹0 < n› Suc.IH show ?thesis
            by (auto elim!: evenE)
        next
          case False
          then obtain k where k: j = 2*k + 1 by (metis oddE)
          then show ?thesis
          proof (cases even k)
            case True
            then obtain m where m: k = 2*m by (metis evenE)
             have fleft: f (leftcut (a ((2*m + 1) / 2^n)) (b ((2*m + 1) / 2^n)) (c
((2*m + 1) / 2^n))) =
                          f (c((2*m + 1) / 2^n))
               using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
                 using left_right_m [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
b((2*m + 1) / 2^n)]
              by (auto intro: f_eqI)
            show ?thesis
            proof (intro conjI impI notI allI)
              have False if b (real j / 2 ^ Suc n) ≤ a (real j / 2 ^ Suc n)
              proof −
                have f (c ((1 + real m * 2) / 2 ^ n)) = f (a ((1 + real m * 2) / 2
^ n))
                  using k m ‹0 < n› fleft that a41 [of n m] b41 [of n m]
                  using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
```

```
b_le_1 [of 2*m+1 n]
                  using left_right [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
b((2*m + 1) / 2^n)]
                by (auto simp: algebra_simps)
               moreover have a (real (1 + m * 2) / 2 ^ n) < c (real (1 + m *
2) / 2 ^ n)
                using Suc.IH [of 1 + m * 2] by (simp add: c_def midpoint_def)
              moreover have c (real (1 + m * 2) / 2 ^ n) ≤ b (real (1 + m * 2)
/ 2 ^ n)
                using cleb by blast
              ultimately show ?thesis
                using Suc.IH [of 1 + m * 2] by force
            qed
            then show a (real j / 2 ^ Suc n) < b (real j / 2 ^ Suc n) by force
          next
            fix x
            assume a (real j / 2 ^ Suc n) < x x ≤ b (real j / 2 ^ Suc n) f x = f
(a (real j / 2 ^ Suc n))
            then show False
              using Suc.IH [of 1 + m * 2, THEN conjunct2, THEN conjunct1]
              using k m ‹0 < n› a41 [of n m] b41 [of n m]
               using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
                using left_right_m [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
b((2*m + 1) / 2^n)]
              by (auto simp: algebra_simps)
          next
            fix x
            assume a (real j / 2 ^ Suc n) ≤ x x < b (real j / 2 ^ Suc n) f x = f
(b (real j / 2 ^ Suc n))
            then show False
              using k m ‹0 < n› a41 [of n m] b41 [of n m] fleft left_neq
               using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
              by (auto simp: algebra_simps)
          qed
        next
          case False
          with oddE obtain m where m: k = Suc (2*m) by fastforce
          have fright: f (rightcut (a ((2*m + 1) / 2^n)) (b ((2*m + 1) / 2^n))
(c ((2*m + 1) / 2^n))) = f (c((2*m + 1) / 2^n))
            using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
             using left_right_m [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
b((2*m + 1) / 2^n)]
            by (auto intro: f_eqI [OF _ order_refl])
          show ?thesis
          proof (intro conjI impI notI allI)
            have False if b (real j / 2 ^ Suc n) ≤ a (real j / 2 ^ Suc n)
```

```
          proof −
            have f (c ((1 + real m * 2) / 2 ^ n)) = f (b ((1 + real m * 2) / 2
^ n))
              using k m ‹0 < n› fright that a43 [of n m] b43 [of n m]
              using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
                  using left_right [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
b((2*m + 1) / 2^n)]
              by (auto simp: algebra_simps)
             moreover have a (real (1 + m * 2) / 2 ^ n) ≤ c (real (1 + m *
2) / 2 ^ n)
              using alec by blast
            moreover have c (real (1 + m * 2) / 2 ^ n) < b (real (1 + m * 2)
/ 2 ^ n)
              using Suc.IH [of 1 + m * 2] by (simp add: c_def midpoint_def)
            ultimately show ?thesis
              using Suc.IH [of 1 + m * 2] by force
          qed
          then show a (real j / 2 ^ Suc n) < b (real j / 2 ^ Suc n) by force
        next
          fix x
          assume a (real j / 2 ^ Suc n) < x x ≤ b (real j / 2 ^ Suc n) f x = f
(a (real j / 2 ^ Suc n))
          then show False
            using k m ‹0 < n› a43 [of n m] b43 [of n m] fright right_neq
             using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
            by (auto simp: algebra_simps)
        next
          fix x
          assume a (real j / 2 ^ Suc n) ≤ x x < b (real j / 2 ^ Suc n) f x = f
(b (real j / 2 ^ Suc n))
          then show False
            using Suc.IH [of 1 + m * 2, THEN conjunct2, THEN conjunct2]
            using k m ‹0 < n› a43 [of n m] b43 [of n m]
             using alec [of 2*m+1 n] cleb [of 2*m+1 n] a_ge_0 [of 2*m+1 n]
b_le_1 [of 2*m+1 n]
              using left_right_m [of c((2*m + 1) / 2^n) a((2*m + 1) / 2^n)
b((2*m + 1) / 2^n)]
            by (auto simp: algebra_simps fright simp del: power_Suc)
        qed
      qed
    qed
  qed
  have c_gt_0 [simp]: 0 < c(m / 2^n) and c_less_1 [simp]: c(m / 2^n) < 1
for m::nat and n
    using a_less_b [of m n] apply (simp_all add: c_def midpoint_def)
    using a_ge_0 [of m n] b_le_1 [of m n] by linarith+
```

```
    have approx: ∃j n. odd j ∧ n ≠ 0 ∧
                       real i / 2^m ≤ real j / 2^n ∧
                       real j / 2^n ≤ real k / 2^p ∧
                       |real i / 2 ^ m − real j / 2 ^ n| < 1/2^n ∧
                       |real k / 2 ^ p − real j / 2 ^ n| < 1/2^n
      if 0 < i i < 2 ^ m 0 < k k < 2 ^ p i / 2^m < k / 2^p m + p = N for N m
p i k
      using that
    proof (induction N arbitrary: m p i k rule: less_induct)
      case (less N)
      then consider i / 2^m ≤ 1/2 1/2 ≤ k / 2^p | k / 2^p < 1/2 | k / 2^p ≥
1/2 1/2 < i / 2^m
        by linarith
      then show ?case
      proof cases
        case 1
        with less.prems show ?thesis
          by (rule_tac x=1 in exI)+ (fastforce simp: field_split_simps)
      next
        case 2 show ?thesis
        proof (cases m)
          case 0 with less.prems show ?thesis
            by auto
        next
          case (Suc m') show ?thesis
          proof (cases p)
            case 0 with less.prems show ?thesis by auto
          next
            case (Suc p')
            have §: False if real i * 2 ^ p' < real k * 2 ^ m' k < 2 ^ p' 2 ^ m' ≤ i
            proof −
              have real k * 2 ^ m' < 2 ^ p' * 2 ^ m'
                using that by simp
              then have real i * 2 ^ p' < 2 ^ p' * 2 ^ m'
                using that by linarith
              with that show ?thesis by simp
            qed
            moreover have ∗: real i / 2 ^ m' < real k / 2^p'  k < 2 ^ p'
           using less.prems ‹m = Suc m'› 2 Suc by (force simp: field_split_simps)+
            moreover have i < 2 ^ m'
              using § ∗ by (clarsimp simp: divide_simps linorder_not_le) (meson
linorder_not_le)
            ultimately show ?thesis
              using less.IH [of m'+p' i m' k p'] less.prems ‹m = Suc m'› 2 Suc
              by (force simp: field_split_simps)
          qed
        qed
      next
        case 3 show ?thesis
```

```
      proof (cases m)
        case 0 with less.prems show ?thesis
          by auto
      next
        case (Suc m')
        proof (cases p)
          case 0 with less.prems show ?thesis by auto
        next
          case (Suc p')
          have real (i - 2 ^ m') / 2 ^ m' < real (k - 2 ^ p') / 2 ^ p'
              using less.prems ‹m = Suc m'› Suc 3  by (auto simp: field_simps
of_nat_diff)
          moreover have k - 2 ^ p' < 2 ^ p' i - 2 ^ m' < 2 ^ m'
            using less.prems Suc ‹m = Suc m'› by auto
          moreover
          have 2 ^ p' ≤ k 2 ^ p' ≠ k
            using less.prems ‹m = Suc m'› Suc 3 by auto
          then have 2 ^ p' < k
            by linarith
          ultimately show ?thesis
            using less.IH [of m'+p' i - 2^m' m' k - 2 ^ p' p'] less.prems ‹m =
Suc m'› Suc 3
            apply (clarsimp simp: field_simps of_nat_diff)
            apply (rule_tac x=2 ^ n + j in exI, simp)
            apply (rule_tac x=Suc n in exI)
            apply (auto simp: field_simps)
            done
        qed
      qed
    qed
    have clec: c(real i / 2^m) ≤ c(real j / 2^n)
      if i: 0 < i i < 2 ^ m and j: 0 < j j < 2 ^ n and ij: i / 2^m < j / 2^n for
m i n j
    proof -
      obtain j' n' where odd j' n' ≠ 0
        and i_le_j: real i / 2 ^ m ≤ real j' / 2 ^ n'
        and j_le_j: real j' / 2 ^ n' ≤ real j / 2 ^ n
        and clo_ij: |real i / 2 ^ m - real j' / 2 ^ n'| < 1/2 ^ n'
        and clo_jj: |real j / 2 ^ n - real j' / 2 ^ n'| < 1/2 ^ n'
        using approx [of i m j n m+n] that i j ij by auto
      with oddE obtain q where q: j' = Suc (2*q) by fastforce
      have c (real i / 2 ^ m) ≤ c((2*q + 1) / 2^n')
      proof (cases i / 2^m = (2*q + 1) / 2^n')
        case True then show ?thesis by simp
      next
        case False
        with i_le_j clo_ij q have |real i / 2 ^ m - real (4 * q + 1) / 2 ^ Suc n'|
< 1 / 2 ^ Suc n'
```

```
        by (auto simp: field_split_simps)
      then have c(i / 2^m) ≤ b(real(4 * q + 1) / 2 ^ (Suc n'))
        by (meson ci_le_bj even_mult_iff even_numeral even_plus_one_iff)
      then show ?thesis
       using alec [of 2*q+1 n'] cleb [of 2*q+1 n'] a_ge_0 [of 2*q+1 n'] b_le_1
[of 2*q+1 n'] b41 [of n' q] ‹n' ≠ 0›
        using left_right_m [of c((2*q + 1) / 2^n') a((2*q + 1) / 2^n') b((2*q
+ 1) / 2^n')]
        by (auto simp: algebra_simps)
    qed
    also have ... ≤ c(real j / 2^n)
    proof (cases j / 2^n = (2*q + 1) / 2^n')
      case True
      then show ?thesis by simp
    next
      case False
      with j_le_j q have less: (2*q + 1) / 2^n' < j / 2^n
        by auto
      have *: ⟦q < i; abs(i − q) < s*2; r = q + s⟧ ⟹ abs(i − r) < s for i q s
r::real
        by auto
      have |real j / 2 ^ n − real (4 * q + 3) / 2 ^ Suc n'| < 1 / 2 ^ Suc n'
      by (rule * [OF less]) (use j_le_j clo_jj q in ‹auto simp: field_split_simps›)
      then have a(real(4*q + 3) / 2 ^ (Suc n')) ≤ c(j / 2^n)
       by (metis Suc3_eq_add_3 add.commute aj_le_ci even_Suc even_mult_iff
even_numeral)
      then show ?thesis
       using alec [of 2*q+1 n'] cleb [of 2*q+1 n'] a_ge_0 [of 2*q+1 n'] b_le_1
[of 2*q+1 n'] a43 [of n' q] ‹n' ≠ 0›
        using left_right_m [of c((2*q + 1) / 2^n') a((2*q + 1) / 2^n') b((2*q
+ 1) / 2^n')]
        by (auto simp: algebra_simps)
    qed
    finally show ?thesis .
  qed
  have x = y if 0 ≤ x x ≤ 1 0 ≤ y y ≤ 1 h x = h y for x y
    using that
  proof (induction x y rule: linorder_class.linorder_less_wlog)
    case (less x1 x2)
    obtain m n where m: 0 < m m < 2 ^ n
      and x12: x1 < m / 2^n m / 2^n < x2
      and neq: h x1 ≠ h (real m / 2^n)
    proof −
      have (x1 + x2) / 2 ∈ closure D01
        using cloD01 less.hyps less.prems by auto
      with less obtain y where y ∈ D01 and dist_y: dist y ((x1 + x2) / 2) <
(x2 − x1) / 64
        unfolding closure_approachable
        by (metis diff_gt_0_iff_gt less_divide_eq_numeral1(1) mult_zero_left)
```

```
      obtain m n where m: 0 < m m < 2 ^ n
                  and clo: |real m / 2 ^ n - (x1 + x2) / 2| < (x2 - x1) / 64
                  and n: 1/2^n < (x2 - x1) / 128
      proof -
        have min 1 ((x2 - x1) / 128) > 0 1/2 < (1::real)
          using less by auto
        then obtain N where N: 1/2^N < min 1 ((x2 - x1) / 128)
          by (metis power_one_over real_arch_pow_inv)
        then have N > 0
          using less_divide_eq_1 by force
        obtain p q where p: p < 2 ^ q p ≠ 0 and yeq: y = real p / 2 ^ q
          using ‹y ∈ D01› by (auto simp: zero_less_divide_iff D01_def)
        show ?thesis
        proof
          show 0 < 2^N * p
            using p by auto
          show 2 ^ N * p < 2 ^ (N+q)
            by (simp add: p power_add)
          have |real (2 ^ N * p) / 2 ^ (N + q) - (x1 + x2) / 2| = |real p / 2 ^
q - (x1 + x2) / 2|
            by (simp add: power_add)
          also have ... = |y - (x1 + x2) / 2|
            by (simp add: yeq)
          also have ... < (x2 - x1) / 64
            using dist_y by (simp add: dist_norm)
          finally show |real (2 ^ N * p) / 2 ^ (N + q) - (x1 + x2) / 2| < (x2
- x1) / 64 .
          have (1::real) / 2 ^ (N + q) ≤ 1/2^N
            by (simp add: field_simps)
          also have ... < (x2 - x1) / 128
            using N by force
          finally show 1/2 ^ (N + q) < (x2 - x1) / 128 .
        qed
      qed
      obtain m' n' m'' n'' where 0 < m' m' < 2 ^ n' x1 < m' / 2^n' m' / 2^n'
< x2
        and 0 < m'' m'' < 2 ^ n'' x1 < m'' / 2^n'' m'' / 2^n'' < x2
        and neq: h (real m'' / 2^n'') ≠ h (real m' / 2^n')
      proof
        show 0 < Suc (2*m)
          by simp
        show m21: Suc (2*m) < 2 ^ Suc n
          using m by auto
        show x1 < real (Suc (2 * m)) / 2 ^ Suc n
          using clo by (simp add: field_simps abs_if split: if_split_asm)
        show real (Suc (2 * m)) / 2 ^ Suc n < x2
          using n clo by (simp add: field_simps abs_if split: if_split_asm)
        show 0 < 4*m + 3
          by simp
```

```
      have m+1 ≤ 2 ^ n
        using m by simp
      then have 4 * (m+1) ≤ 4 * (2 ^ n)
        by simp
      then show m43: 4*m + 3 < 2 ^ (n+2)
        by (simp add: algebra_simps)
      show x1 < real (4 * m + 3) / 2 ^ (n + 2)
        using clo by (simp add: field_simps abs_if split: if_split_asm)
      show real (4 * m + 3) / 2 ^ (n + 2) < x2
        using n clo by (simp add: field_simps abs_if split: if_split_asm)
     have c_fold: midpoint (a ((2 * real m + 1) / 2 ^ Suc n)) (b ((2 * real m
+ 1) / 2 ^ Suc n)) = c ((2 * real m + 1) / 2 ^ Suc n)
        by (simp add: c_def)
      define R where R ≡ rightcut (a ((2 * real m + 1) / 2 ^ Suc n)) (b ((2
* real m + 1) / 2 ^ Suc n)) (c ((2 * real m + 1) / 2 ^ Suc n))
      have R < b ((2 * real m + 1) / 2 ^ Suc n)
        unfolding R_def using a_less_b [of 4*m + 3 n+2] a43 [of Suc n m]
b43 [of Suc n m]
        by simp
      then have Rless: R < midpoint R (b ((2 * real m + 1) / 2 ^ Suc n))
        by (simp add: midpoint_def)
      have midR_le: midpoint R (b ((2 * real m + 1) / 2 ^ Suc n)) ≤ b ((2 *
real m + 1) / (2 * 2 ^ n))
        using ‹R < b ((2 * real m + 1) / 2 ^ Suc n)›
        by (simp add: midpoint_def)
      have (real (Suc (2 * m)) / 2 ^ Suc n) ∈ D01  real (4 * m + 3) / 2 ^ (n
+ 2) ∈ D01
          by (simp_all add: D01_def m21 m43 del: power_Suc of_nat_Suc
of_nat_add add_2_eq_Suc') blast+
      then show h (real (4 * m + 3) / 2 ^ (n + 2)) ≠ h (real (Suc (2 * m))
/ 2 ^ Suc n)
        using a_less_b [of 4*m + 3 n+2, THEN conjunct1]
        using a43 [of Suc n m] b43 [of Suc n m]
       using alec [of 2*m+1 Suc n] cleb [of 2*m+1 Suc n] a_ge_0 [of 2*m+1
Suc n]  b_le_1 [of 2*m+1 Suc n]
        apply (simp add: fc_eq [symmetric] c_def del: power_Suc)
        apply (simp only: add.commute [of 1] c_fold R_def [symmetric])
        apply (rule right_neq)
        using Rless apply (simp add: R_def)
          apply (rule midR_le, auto)
        done
    qed
    then show ?thesis by (metis that)
  qed
  have m_div: 0 < m / 2^n m / 2^n < 1
    using m  by (auto simp: field_split_simps)
  have closure0m: {0..m / 2^n} = closure ({0<..< m / 2^n} ∩ (⋃k m. {real
m / 2 ^ k}))
    by (subst closure_dyadic_rationals_in_convex_set_pos_1, simp_all add:
```

```
not_le m)
    have 2^n > m
      by (simp add: m(2) not_le)
    then have closurem1: {m / 2^n .. 1} = closure ({m / 2^n <..< 1} ∩ (⋃k
m. {real m / 2 ^ k}))
      using closure_dyadic_rationals_in_convex_set_pos_1 m_div(1) by fast-
force
     have cont_h': continuous_on (closure ({u<..<v} ∩ (⋃k m. {real m / 2 ^
k}))) h
      if 0 ≤ u v ≤ 1 for u v
      using that by (intro continuous_on_subset [OF cont_h] closure_minimal
[OF subsetI]) auto
    have closed_f': closed (f ` {u..v}) if 0 ≤ u v ≤ 1 for u v
      by (metis compact_continuous_image cont_f compact_interval atLeastat-
Most_subset_iff
          compact_imp_closed continuous_on_subset that)
    have less_2I: ⋀k i. real i / 2 ^ k < 1 ⟹ i < 2 ^ k
      by simp
    have h ` ({0<..<m / 2 ^ n} ∩ (⋃ q p. {real p / 2 ^ q})) ⊆ f ` {0..c (m / 2
^ n)}
    proof clarsimp
      fix p q
      assume p: 0 < real p / 2 ^ q real p / 2 ^ q < real m / 2 ^ n
      then have [simp]: 0 < p
        by (simp add: field_split_simps)
      have [simp]: p < 2 ^ q
        by (blast intro: p less_2I m_div less_trans)
      have f (c (real p / 2 ^ q)) ∈ f ` {0..c (real m / 2 ^ n)}
        by (auto simp: clec p m)
      then show h (real p / 2 ^ q) ∈ f ` {0..c (real m / 2 ^ n)}
        by (simp add: h_eq)
    qed
    with m_div have h ` {0 .. m / 2^n} ⊆ f ` {0 .. c(m / 2^n)}
      apply (subst closure0m)
      by (rule image_closure_subset [OF cont_h' closed_f']) auto
    then have hx1: h x1 ∈ f ` {0 .. c(m / 2^n)}
      using x12 less.prems(1) by auto
    then obtain t1 where t1: h x1 = f t1 0 ≤ t1 t1 ≤ c (m / 2 ^ n)
      by auto
    have h ` ({m / 2 ^ n<..<1} ∩ (⋃ q p. {real p / 2 ^ q})) ⊆ f ` {c (m / 2 ^
n)..1}
    proof clarsimp
      fix p q
      assume p: real m / 2 ^ n < real p / 2 ^ q and [simp]: p < 2 ^ q
      then have [simp]: 0 < p
        using gr_zeroI m_div by fastforce
      have f (c (real p / 2 ^ q)) ∈ f ` {c (m / 2 ^ n)..1}
        by (auto simp: clec p m)
      then show h (real p / 2 ^ q) ∈ f ` {c (real m / 2 ^ n)..1}
```

```
        by (simp add: h_eq)
    qed
    with m have h ‘ {m / 2^n .. 1} ⊆ f ‘ {c(m / 2^n) .. 1}
      apply (subst closurem1)
      by (rule image_closure_subset [OF cont_h' closed_f']) auto
    then have hx2: h x2 ∈ f ‘ {c(m / 2^n)..1}
      using x12 less.prems by auto
    then obtain t2 where t2: h x2 = f t2 c (m / 2 ^ n) ≤ t2 t2 ≤ 1
      by auto
    with t1 less neq have False
    using conn [of h x2, unfolded is_interval_connected_1 [symmetric] is_interval_1,
rule_format, of t1 t2 c(m / 2^n)]
      by (simp add: h_eq m)
    then show ?case by blast
  qed auto
  then show ?thesis
    by (auto simp: inj_on_def)
 qed
 ultimately have {0..1::real} homeomorphic f ‘ {0..1}
  using homeomorphic_compact [OF _ cont_h] by blast
 then show ?thesis
  using homeomorphic_sym by blast
qed


theorem path_contains_arc:
 fixes p :: real ⇒ 'a::{complete_space,real_normed_vector}
 assumes path p and a: pathstart p = a and b: pathfinish p = b and a ≠ b
 obtains q where arc q path_image q ⊆ path_image p pathstart q = a pathfinish
q = b
proof −
 have ucont_p: uniformly_continuous_on {0..1} p
  using ‹path p› unfolding path_def
  by (metis compact_Icc compact_uniformly_continuous)
 define φ where φ ≡ λS. S ⊆ {0..1} ∧ 0 ∈ S ∧ 1 ∈ S ∧
                    (∀x ∈ S. ∀y ∈ S. open_segment x y ∩ S = {} ⟶ p x = p y)
 obtain T where closed T φ T and T: ⋀U. ⟦closed U; φ U⟧ ⟹ ¬ (U ⊂ T)
 proof (rule Brouwer_reduction_theorem_gen [of {0..1} φ])
  have ∗: {x<..<y} ∩ {0..1} = {x<..<y} if 0 ≤ x y ≤ 1 x ≤ y for x y::real
   using that by auto
  show φ {0..1}
   by (auto simp: φ_def open_segment_eq_real_ivl ∗)
  show φ (⋂(F ‘ UNIV))
   if ⋀n. closed (F n) and φ: ⋀n. φ (F n) and Fsub: ⋀n. F (Suc n) ⊆ F n for
F
  proof −
   have F01: ⋀n. F n ⊆ {0..1} ∧ 0 ∈ F n ∧ 1 ∈ F n
    and peq: ⋀n x y. ⟦x ∈ F n; y ∈ F n; open_segment x y ∩ F n = {}⟧ ⟹ p
x = p y
```

```
      by (metis φ φ_def)+
    have pqF: False if ∀u. x ∈ F u ∀x. y ∈ F x open_segment x y ∩ (⋂x. F x)
= {} and neg: p x ≠ p y
      for x y
      using that
    proof (induction x y rule: linorder_class.linorder_less_wlog)
      case (less x y)
      have xy: x ∈ {0..1} y ∈ {0..1}
        by (metis less.prems subsetCE F01)+
      have norm(p x − p y) / 2 > 0
        using less by auto
      then obtain e where e > 0
        and e: ⋀u v. ⟦u ∈ {0..1}; v ∈ {0..1}; dist v u < e⟧ ⟹ dist (p v) (p u)
< norm(p x − p y) / 2
        by (metis uniformly_continuous_onE [OF ucont_p])
      have minxy: min e (y − x) < (y − x) ∗ (3 / 2)
        by (subst min_less_iff_disj) (simp add: less)
      define w where w ≡ x + (min e (y − x) / 3)
      define z where z ≡y − (min e (y − x) / 3)
      have w < z and w: w ∈ {x<..<y} and z: z ∈ {x<..<y}
        and wxe: norm(w − x) < e and zye: norm(z − y) < e
        using minxy ‹0 < e› less unfolding w_def z_def by auto
      have Fclo: ⋀T. T ∈ range F ⟹ closed T
        by (metis ‹⋀n. closed (F n)› image_iff)
      have eq: {w..z} ∩ ⋂(F ‘ UNIV) = {}
        using less w z by (simp add: open_segment_eq_real_ivl disjoint_iff)
      then obtain K where finite K and K: {w..z} ∩ (⋂ (F ‘ K)) = {}
        by (metis finite_subset_image compact_imp_fip [OF compact_interval
Fclo])
      then have K ≠ {}
        using ‹w < z› ‹{w..z} ∩ ⋂(F ‘ K) = {}› by auto
      define n where n ≡ Max K
      have n ∈ K unfolding n_def by (metis ‹K ≠ {}› ‹finite K› Max_in)
      have F n ⊆ ⋂ (F ‘ K)
     unfolding n_def by (metis Fsub Max_ge ‹K ≠ {}› ‹finite K› cINF_greatest
lift_Suc_antimono_le)
      with K have wzF_null: {w..z} ∩ F n = {}
        by (metis disjoint_iff_not_equal subset_eq)
      obtain u where u: u ∈ F n u ∈ {x..w} ({u..w} − {u}) ∩ F n = {}
      proof (cases w ∈ F n)
        case True
        then show ?thesis
          by (metis wzF_null ‹w < z› atLeastAtMost_iff disjoint_iff_not_equal
less_eq_real_def)
      next
        case False
        obtain u where u ∈ F n u ∈ {x..w} {u<..<w} ∩ F n = {}
        proof (rule segment_to_point_exists [of F n ∩ {x..w} w])
          show closed (F n ∩ {x..w})
```

```
          by (metis ‹⋀n. closed (F n)› closed_Int closed_real_atLeastAtMost)
        show F n ∩ {x..w} ≠ {}
       by (metis atLeastAtMost_iff disjoint_iff_not_equal greaterThanLessThan_iff
less.prems(1) less_eq_real_def w)
       qed (auto simp: open_segment_eq_real_ivl intro!: that)
       with False show thesis
        by (auto simp add: disjoint_iff less_eq_real_def intro!: that)
      qed
      obtain v where v: v ∈ F n v ∈ {z..y} ({z..v} − {v}) ∩ F n = {}
      proof (cases z ∈ F n)
       case True
       have z ∈ {w..z}
        using ‹w < z› by auto
       then show ?thesis
        by (metis wzF_null Int_iff True empty_iff)
      next
       case False
       show ?thesis
       proof (rule segment_to_point_exists [of F n ∩ {z..y} z])
        show closed (F n ∩ {z..y})
         by (metis ‹⋀n. closed (F n)› closed_Int closed_atLeastAtMost)
        show F n ∩ {z..y} ≠ {}
       by (metis atLeastAtMost_iff disjoint_iff_not_equal greaterThanLessThan_iff
less.prems(2) less_eq_real_def z)
        show ⋀b. ⟦b ∈ F n ∩ {z..y}; open_segment z b ∩ (F n ∩ {z..y}) = {}⟧
⟹ thesis
         proof
          show ⋀b. ⟦b ∈ F n ∩ {z..y}; open_segment z b ∩ (F n ∩ {z..y}) = {}⟧
⟹ ({z..b} − {b}) ∩ F n = {}
         using False by (auto simp: open_segment_eq_real_ivl less_eq_real_def)
         qed auto
       qed
      qed
      obtain u v where u ∈ {0..1} v ∈ {0..1} norm(u − x) < e norm(v − y)
< e p u = p v
      proof
       show u ∈ {0..1} v ∈ {0..1}
        by (metis F01 ‹u ∈ F n› ‹v ∈ F n› subsetD)+
       show norm(u − x) < e norm (v − y) < e
        using ‹u ∈ {x..w}› ‹v ∈ {z..y}› atLeastAtMost_iff real_norm_def wxe
zye by auto
       show p u = p v
       proof (rule peq)
        show u ∈ F n v ∈ F n
         by (auto simp: u v)
        have False if ξ ∈ F n u < ξ ξ < v for ξ
        proof −
         have ξ ∉ {z..v}
         by (metis DiffI disjoint_iff_not_equal less_irrefl singletonD that(1,3)
```

```
v(3))
            moreover have ξ ∉ {w..z} ∩ F n
              by (metis equals0D wzF_null)
            ultimately have ξ ∈ {u..w}
              using that by auto
            then show ?thesis
                 by (metis DiffI disjoint_iff_not_equal less_eq_real_def not_le
singletonD that(1,2) u(3))
          qed
          moreover
          have ⟦ξ ∈ F n; v < ξ; ξ < u⟧ ⟹ False for ξ
            using ‹u ∈ {x..w}› ‹v ∈ {z..y}› ‹w < z› by simp
          ultimately
          show open_segment u v ∩ F n = {}
            by (force simp: open_segment_eq_real_ivl)
        qed
      qed
      then show ?case
        using e [of x u] e [of y v] xy
        by (metis dist_norm dist_triangle_half_r order_less_irrefl)
    qed (auto simp: open_segment_commute)
    show ?thesis
      unfolding φ_def by (metis (no_types, opaque_lifting) INT_I Inf_lower2
rangeI that(3) F01 subsetCE pqF)
  qed
  show closed {0..1::real} by auto
 qed (meson φ_def)
 then have T ⊆ {0..1} 0 ∈ T 1 ∈ T
  and peq: ⋀x y. ⟦x ∈ T; y ∈ T; open_segment x y ∩ T = {}⟧ ⟹ p x = p y
  unfolding φ_def by metis+
 then have T ≠ {} by auto
 define h where h ≡ λx. p(SOME y. y ∈ T ∧ open_segment x y ∩ T = {})
 have p y = p z if y ∈ T z ∈ T and xyT: open_segment x y ∩ T = {} and xzT:
open_segment x z ∩ T = {}
  for x y z
 proof (cases x ∈ T)
  case True
  with that show ?thesis by (metis ‹φ T› φ_def)
 next
  case False
  have insert x (open_segment x y ∪ open_segment x z) ∩ T = {}
    by (metis False Int_Un_distrib2 Int_insert_left Un_empty_right xyT xzT)
   moreover have open_segment y z ∩ T ⊆ insert x (open_segment x y ∪
open_segment x z) ∩ T
    by (auto simp: open_segment_eq_real_ivl)
  ultimately have open_segment y z ∩ T = {}
    by blast
  with that peq show ?thesis by metis
 qed
```

then have *h_eq_p_gen*: *h x* = *p y* **if** *y* ∈ *T open_segment x y* ∩ *T* = {} **for** *x y*
using *that* **unfolding** *h_def*
by (*metis* (*mono_tags*, *lifting*) *some_eq_ex*)
then have *h_eq_p*: ⋀*x*. *x* ∈ *T* ⟹ *h x* = *p x*
by *simp*
have *disjoint*: ⋀*x*. ∃ *y*. *y* ∈ *T* ∧ *open_segment x y* ∩ *T* = {}
by (*meson* ‹*T* ≠ {}› ‹*closed T*› *segment_to_point_exists*)
have *heq*: *h x* = *h x'* **if** *open_segment x x'* ∩ *T* = {} **for** *x x'*
proof (*cases x* ∈ *T* ∨ *x'* ∈ *T*)
case *True*
then show *?thesis*
by (*metis h_eq_p h_eq_p_gen open_segment_commute that*)
next
case *False*
obtain *y y'* **where** *y* ∈ *T open_segment x y* ∩ *T* = {} *h x* = *p y*
y' ∈ *T open_segment x' y'* ∩ *T* = {} *h x'* = *p y'*
by (*meson disjoint h_eq_p_gen*)
moreover have *open_segment y y'* ⊆ (*insert x* (*insert x'* (*open_segment x y* ∪ *open_segment x' y'* ∪ *open_segment x x'*)))
by (*auto simp*: *open_segment_eq_real_ivl*)
ultimately show *?thesis*
using *False that* **by** (*fastforce simp add*: *h_eq_p intro*!: *peq*)
qed
have *h* ‘ {*0..1*} *homeomorphic* {*0..1*::*real*}
proof (*rule homeomorphic_monotone_image_interval*)
show *continuous_on* {*0..1*} *h*
proof (*clarsimp simp add*: *continuous_on_iff*)
fix *u* ε::*real*
assume *0* < ε *0* ≤ *u u* ≤ *1*
then obtain δ **where** δ > *0* **and** δ: ⋀*v*. *v* ∈ {*0..1*} ⟹ *dist v u* < δ ⟶ *dist* (*p v*) (*p u*) < ε / *2*
using *ucont_p* [*unfolded uniformly_continuous_on_def*]
by (*metis atLeastAtMost_iff half_gt_zero_iff*)
then have *dist* (*h v*) (*h u*) < ε **if** *v* ∈ {*0..1*} *dist v u* < δ **for** *v*
proof (*cases open_segment u v* ∩ *T* = {})
case *True*
then show *?thesis*
using ‹*0* < ε› *heq* **by** *auto*
next
case *False*
have *uvT*: *closed* (*closed_segment u v* ∩ *T*) *closed_segment u v* ∩ *T* ≠ {}
using *False open_closed_segment* **by** (*auto simp*: ‹*closed T*› *closed_Int*)
obtain *w* **where** *w* ∈ *T* **and** *w*: *w* ∈ *closed_segment u v open_segment u w* ∩ *T* = {}
proof (*rule segment_to_point_exists* [*OF uvT*])
fix *b*
assume *b* ∈ *closed_segment u v* ∩ *T open_segment u b* ∩ (*closed_segment u v* ∩ *T*) = {}

```
        then show thesis
            by (metis IntD1 IntD2 ends_in_segment(1) inf.orderE inf_assoc
subset_oc_segment that)
      qed
      then have puw: dist (p u) (p w) < ε / 2
          by (metis (no_types) ‹T ⊆ {0..1}› ‹dist v u < δ› δ dist_commute
dist_in_closed_segment le_less_trans subsetCE)
       obtain z where z ∈ T and z: z ∈ closed_segment u v open_segment v z
∩ T = {}
      proof (rule segment_to_point_exists [OF uvT])
        fix b
       assume b ∈ closed_segment u v ∩ T open_segment v b ∩ (closed_segment
u v ∩ T) = {}
        then show thesis
            by (metis IntD1 IntD2 ends_in_segment(2) inf.orderE inf_assoc
subset_oc_segment that)
      qed
      then have dist (p u) (p z) < ε / 2
     by (metis ‹T ⊆ {0..1}› ‹dist v u < δ› δ dist_commute dist_in_closed_segment
le_less_trans subsetCE)
      then show ?thesis
          using puw by (metis (no_types) ‹w ∈ T› ‹z ∈ T› dist_commute
dist_triangle_half_l h_eq_p_gen w(2) z(2))
    qed
    with ‹0 < δ› show ∃δ>0. ∀v∈{0..1}. dist v u < δ ⟶ dist (h v) (h u) < ε
by blast
  qed
  show connected ({0..1} ∩ h −‘ {z}) for z
  proof (clarsimp simp add: connected_iff_connected_component)
    fix u v
    assume huv_eq: h v = h u and uv: 0 ≤ u u ≤ 1 0 ≤ v v ≤ 1
    have ∃ T. connected T ∧ T ⊆ {0..1} ∧ T ⊆ h −‘ {h u} ∧ u ∈ T ∧ v ∈ T
    proof (intro exI conjI)
      show connected (closed_segment u v)
        by simp
      show closed_segment u v ⊆ {0..1}
        by (simp add: uv closed_segment_eq_real_ivl)
      have pxy: p x = p y
        if T ⊆ {0..1} 0 ∈ T 1 ∈ T x ∈ T y ∈ T
        and disjT: open_segment x y ∩ (T − open_segment u v) = {}
        and xynot: x ∉ open_segment u v y ∉ open_segment u v
        for x y
      proof (cases open_segment x y ∩ open_segment u v = {})
        case True
        then show ?thesis
          by (metis Diff_Int_distrib Diff_empty peq disjT ‹x ∈ T› ‹y ∈ T›)
      next
        case False
         then have open_segment x u ∪ open_segment y v ⊆ open_segment x y
```

```
− open_segment u v ∨
                    open_segment y u ∪ open_segment x v ⊆ open_segment x y −
open_segment u v (is ?xuyv ∨ ?yuxv)
          using xynot by (fastforce simp add: open_segment_eq_real_ivl not_le
not_less split: if_split_asm)
        then show p x = p y
        proof
          assume ?xuyv
          then have open_segment x u ∩ T = {} open_segment y v ∩ T = {}
            using disjT by auto
          then have h x = h y
            using heq huv_eq by auto
          then show ?thesis
            using h_eq_p ‹x ∈ T› ‹y ∈ T› by auto
        next
          assume ?yuxv
          then have open_segment y u ∩ T = {} open_segment x v ∩ T = {}
            using disjT by auto
          then have h x = h y
            using heq [of y u]  heq [of x v] huv_eq by auto
          then show ?thesis
            using h_eq_p ‹x ∈ T› ‹y ∈ T› by auto
        qed
      qed
      have ¬ T − open_segment u v ⊂ T
      proof (rule T)
        show closed (T − open_segment u v)
          by (simp add: closed_Diff [OF ‹closed T›] open_segment_eq_real_ivl)
        have 0 ∉ open_segment u v 1 ∉ open_segment u v
          using open_segment_eq_real_ivl uv by auto
        then show φ (T − open_segment u v)
          using ‹T ⊆ {0..1}› ‹0 ∈ T› ‹1 ∈ T›
          by (auto simp: φ_def) (meson peq pxy)
      qed
      then have open_segment u v ∩ T = {}
        by blast
      then show closed_segment u v ⊆ h −‘ {h u}
    by (force intro: heq simp: open_segment_eq_real_ivl closed_segment_eq_real_ivl
split: if_split_asm)+
    qed auto
    then show connected_component ({0..1} ∩ h −‘ {h u}) u v
      by (simp add: connected_component_def)
  qed
  show h 1 ≠ h 0
    by (metis ‹φ T› φ_def a ‹a ≠ b› b h_eq_p pathfinish_def pathstart_def)
 qed
 then obtain f and g :: real ⇒ 'a
   where gfeq: (∀x∈h ‘ {0..1}. (g(f x) = x)) and fhim: f ‘ h ‘ {0..1} = {0..1}
and contf: continuous_on (h ‘ {0..1}) f
```

and *fgeq*: $(\forall y{\in}\{0..1\}.\ (f(g\ y) = y))$ **and** *pag*: *path_image* $g = h\ `\ \{0..1\}$ **and** *contg*: *continuous_on* $\{0..1\}$ *g*
by (*auto simp*: *homeomorphic_def homeomorphism_def path_image_def*)
then have *arc g*
by (*metis arc_def path_def inj_on_def*)
obtain *u v* **where** $u \in \{0..1\}$ $a = g\ u$ $v \in \{0..1\}$ $b = g\ v$
by (*metis* (*mono_tags*, *opaque_lifting*) ‹φ *T*› φ*_def a b fhim gfeq h_eq_p imageI path_image_def pathfinish_def pathfinish_in_path_image pathstart_def pathstart_in_path_image*)
then have $a \in$ *path_image g* $b \in$ *path_image g*
using *path_image_def* **by** *blast+*
have *ph*: *path_image h* $\subseteq$ *path_image p*
by (*metis image_mono image_subset_iff path_image_def disjoint h_eq_p_gen* ‹*T* $\subseteq \{0..1\}$›)
show *?thesis*
proof
show *pathstart* (*subpath u v g*) = *a pathfinish* (*subpath u v g*) = *b*
by (*simp_all add*: ‹$a = g\ u$› ‹$b = g\ v$›)
show *path_image* (*subpath u v g*) $\subseteq$ *path_image p*
by (*metis* ‹$u \in \{0..1\}$› ‹$v \in \{0..1\}$› *order_trans pag path_image_def path_image_subpath_subset ph*)
show *arc* (*subpath u v g*)
using ‹*arc g*› ‹$a = g\ u$› ‹$b = g\ v$› ‹$u \in \{0..1\}$› ‹$v \in \{0..1\}$› *arc_subpath_arc* ‹$a \neq b$› **by** *blast*
qed
qed

corollary *path_connected_arcwise*:
fixes *S* :: *'a*::{*complete_space*,*real_normed_vector*} *set*
shows *path_connected S* $\longleftrightarrow$
$(\forall x \in S.\ \forall y \in S.\ x \neq y \longrightarrow (\exists g.\ arc\ g \wedge path\_image\ g \subseteq S \wedge pathstart\ g = x \wedge pathfinish\ g = y))$
(**is** *?lhs* = *?rhs*)
proof (*intro iffI impI ballI*)
fix *x y*
assume *path_connected S x* $\in$ *S y* $\in$ *S x* $\neq$ *y*
then obtain *p* **where** *p*: *path p path_image p* $\subseteq$ *S pathstart p* = *x pathfinish p* = *y*
by (*force simp*: *path_connected_def*)
then show $\exists g.$ *arc g* $\wedge$ *path_image g* $\subseteq$ *S* $\wedge$ *pathstart g* = *x* $\wedge$ *pathfinish g* = *y*
by (*metis* ‹$x \neq y$› *order_trans path_contains_arc*)
next
assume *R* [*rule_format*]: *?rhs*
show *?lhs*
unfolding *path_connected_def*
proof (*intro ballI*)
fix *x y*
assume $x \in S\ y \in S$

```
    show ∃g. path g ∧ path_image g ⊆ S ∧ pathstart g = x ∧ pathfinish g = y
    proof (cases x = y)
      case True with ‹x ∈ S› path_component_def path_component_refl show
?thesis
       by blast
    next
      case False with R [OF ‹x ∈ S› ‹y ∈ S›] show ?thesis
       by (auto intro: arc_imp_path)
    qed
  qed
qed


corollary arc_connected_trans:
  fixes g :: real ⇒ 'a::{complete_space,real_normed_vector}
  assumes arc g arc h pathfinish g = pathstart h pathstart g ≠ pathfinish h
  obtains i where arc i path_image i ⊆ path_image g ∪ path_image h
                  pathstart i = pathstart g pathfinish i = pathfinish h
  by (metis (no_types, opaque_lifting) arc_imp_path assms path_contains_arc
path_image_join path_join pathfinish_join pathstart_join)
```

5.10.4 Accessibility of frontier points

```
lemma dense_accessible_frontier_points:
  fixes S :: 'a::{complete_space,real_normed_vector} set
  assumes open S and opeSV: openin (top_of_set (frontier S)) V and V ≠ {}
  obtains g where arc g g ‘ {0..<1} ⊆ S pathstart g ∈ S pathfinish g ∈ V
proof −
  obtain z where z ∈ V
    using ‹V ≠ {}› by auto
  then obtain r where r > 0 and r: ball z r ∩ frontier S ⊆ V
    by (metis openin_contains_ball opeSV)
  then have z ∈ frontier S
    using ‹z ∈ V› opeSV openin_contains_ball by blast
  then have z ∈ closure S z ∉ S
    by (simp_all add: frontier_def assms interior_open)
  with ‹r > 0› have infinite (S ∩ ball z r)
    by (auto simp: closure_def islimpt_eq_infinite_ball)
  then obtain y where y ∈ S and y: y ∈ ball z r
    using infinite_imp_nonempty by force
  then have y ∉ frontier S
    by (meson ‹open S› disjoint_iff_not_equal frontier_disjoint_eq)
  have y ≠ z
    using ‹y ∈ S› ‹z ∉ S› by blast
  have path_connected(ball z r)
    by (simp add: convex_imp_path_connected)
  with y ‹r > 0›  obtain g where arc g and pig: path_image g ⊆ ball z r
                              and g: pathstart g = y pathfinish g = z
    using ‹y ≠ z› by (force simp: path_connected_arcwise)
```

```
  have continuous_on {0..1} g
    using ‹arc g› arc_imp_path path_def by blast
  then have compact (g -‘ frontier S ∩ {0..1})
   by (simp add: bounded_Int closed_Diff closed_vimage_Int compact_eq_bounded_closed)
  moreover have g -‘ frontier S ∩ {0..1} ≠ {}
  proof -
    have ∃r. r ∈ g -‘ frontier S ∧ r ∈ {0..1}
    by (metis ‹z ∈ frontier S› g(2) imageE path_image_def pathfinish_in_path_image
vimageI2)
    then show ?thesis
      by blast
  qed
  ultimately obtain t where gt: g t ∈ frontier S and 0 ≤ t t ≤ 1
              and t: ⋀u. ⟦g u ∈ frontier S; 0 ≤ u; u ≤ 1⟧ ⟹ t ≤ u
    by (force simp: dest!: compact_attains_inf)
  moreover have t ≠ 0
    by (metis ‹y ∉ frontier S› g(1) gt pathstart_def)
  ultimately have  t01: 0 < t t ≤ 1
    by auto
  have V ⊆ frontier S
    using opeSV openin_contains_ball by blast
  show ?thesis
  proof
    show arc (subpath 0 t g)
      by (simp add: ‹0 ≤ t› ‹t ≤ 1› ‹arc g› ‹t ≠ 0› arc_subpath_arc)
    have g 0 ∈ S
      by (metis ‹y ∈ S› g(1) pathstart_def)
    then show pathstart (subpath 0 t g) ∈ S
      by auto
    have g t ∈ V
     by (metis IntI atLeastAtMost_iff gt image_eqI path_image_def pig r subsetCE
‹0 ≤ t› ‹t ≤ 1›)
    then show pathfinish (subpath 0 t g) ∈ V
      by auto
    then have inj_on (subpath 0 t g) {0..1}
      using t01 ‹arc (subpath 0 t g)› arc_imp_inj_on by blast
    then have subpath 0 t g ‘ {0..<1} ⊆ subpath 0 t g ‘ {0..1} - {subpath 0 t g
1}
      by (force simp: dest: inj_onD)
    moreover have False if subpath 0 t g ‘ ({0..<1}) - S ≠ {}
    proof -
      have contg: continuous_on {0..1} g
        using ‹arc g› by (auto simp: arc_def path_def)
      have subpath 0 t g ‘ {0..<1} ∩ frontier S ≠ {}
      proof (rule connected_Int_frontier [OF _ _ that])
        show connected (subpath 0 t g ‘ {0..<1})
        proof (rule connected_continuous_image)
          show continuous_on {0..<1} (subpath 0 t g)
        by (meson ‹arc (subpath 0 t g)› arc_def atLeastLessThan_subseteq_atLeastAtMost_iff
```

```
continuous_on_subset order_refl path_def)
      qed auto
      show subpath 0 t g ‘ {0..<1} ∩ S ≠ {}
       using ‹y ∈ S› g(1) by (force simp: subpath_def image_def pathstart_def)
    qed
    then obtain x where x ∈ subpath 0 t g ‘ {0..<1} x ∈ frontier S
     by blast
    with t01 ‹0 ≤ t› mult_le_one t show False
     by (fastforce simp: subpath_def)
  qed
  then have subpath 0 t g ‘ {0..1} − {subpath 0 t g 1} ⊆ S
   using subsetD by fastforce
  ultimately show subpath 0 t g ‘ {0..<1} ⊆ S
   by auto
 qed
qed


lemma dense_accessible_frontier_points_connected:
 fixes S :: 'a::{complete_space,real_normed_vector} set
 assumes open S connected S x ∈ S V ≠ {}
    and ope: openin (top_of_set (frontier S)) V
 obtains g where arc g g ‘ {0..<1} ⊆ S pathstart g = x pathfinish g ∈ V
proof −
 have V ⊆ frontier S
   using ope openin_imp_subset by blast
 with ‹open S› ‹x ∈ S› have x ∉ V
   using interior_open by (auto simp: frontier_def)
 obtain g where arc g and g: g ‘ {0..<1} ⊆ S pathstart g ∈ S pathfinish g ∈ V
   by (metis dense_accessible_frontier_points [OF ‹open S› ope ‹V ≠ {}›])
 then have path_connected S
   by (simp add: assms connected_open_path_connected)
 with ‹pathstart g ∈ S› ‹x ∈ S› have path_component S x (pathstart g)
   by (simp add: path_connected_component)
 then obtain f where path f and f: path_image f ⊆ S pathstart f = x pathfinish
f = pathstart g
   by (auto simp: path_component_def)
 then have path (f +++ g)
   by (simp add: ‹arc g› arc_imp_path)
 then obtain h where arc h
                 and h: path_image h ⊆ path_image (f +++ g) pathstart h = x
pathfinish h = pathfinish g
   using path_contains_arc [of f +++ g x pathfinish g] ‹x ∉ V› ‹pathfinish g ∈
V› f
   by (metis pathfinish_join pathstart_join)
 have path_image h ⊆ path_image f ∪ path_image g
   using h(1) path_image_join_subset by auto
 then have h ‘ {0..1} − {h 1} ⊆ S
   using f g h
```

```
  apply (simp add: path_image_def pathfinish_def subset_iff image_def Bex_def)
   by (metis le_less)
 then have h ‘ {0..<1} ⊆ S
   using ‹arc h› by (force simp: arc_def dest: inj_onD)
 then show thesis
   using ‹arc h› g(3) h that by presburger
qed

lemma dense_access_fp_aux:
 fixes S :: 'a::{complete_space,real_normed_vector} set
 assumes S: open S connected S
     and openSU: openin (top_of_set (frontier S)) U
     and openSV: openin (top_of_set (frontier S)) V
     and V ≠ {} ¬ U ⊆ V
 obtains g where arc g pathstart g ∈ U pathfinish g ∈ V g ‘ {0<..<1} ⊆ S
proof −
 have S ≠ {}
   using openSV ‹V ≠ {}› by (metis frontier_empty openin_subtopology_empty)
 then obtain x where x ∈ S by auto
 obtain g where arc g and g: g ‘ {0..<1} ⊆ S pathstart g = x pathfinish g ∈ V
   using dense_accessible_frontier_points_connected [OF S ‹x ∈ S› ‹V ≠ {}›
openSV] by blast
 obtain h where arc h and h: h ‘ {0..<1} ⊆ S pathstart h = x pathfinish h ∈ U
− {pathfinish g}
 proof (rule dense_accessible_frontier_points_connected [OF S ‹x ∈ S›])
   show U − {pathfinish g} ≠ {}
     using ‹pathfinish g ∈ V› ‹¬ U ⊆ V› by blast
   show openin (top_of_set (frontier S)) (U − {pathfinish g})
     by (simp add: openSU openin_delete)
 qed auto
 obtain γ where arc γ
          and γ: path_image γ ⊆ path_image (reversepath h +++ g)
                 pathstart γ = pathfinish h pathfinish γ = pathfinish g
 proof (rule path_contains_arc [of (reversepath h +++ g) pathfinish h pathfinish
g])
   show path (reversepath h +++ g)
    by (simp add: ‹arc g› ‹arc h› ‹pathstart g = x› ‹pathstart h = x› arc_imp_path)
   show pathstart (reversepath h +++ g) = pathfinish h
        pathfinish (reversepath h +++ g) = pathfinish g
     by auto
   show pathfinish h ≠ pathfinish g
     using ‹pathfinish h ∈ U − {pathfinish g}› by auto
 qed auto
 show ?thesis
 proof
   show arc γ pathstart γ ∈ U pathfinish γ ∈ V
     using γ ‹arc γ› ‹pathfinish h ∈ U − {pathfinish g}› ‹pathfinish g ∈ V› by
auto
   have path_image γ ⊆ path_image h ∪ path_image g
```

```
    by (metis γ(1) g(2) h(2) path_image_join path_image_reversepath pathfinish_reversepath)
   then have γ ` {0..1} - {γ 0, γ 1} ⊆ S
    using γ g h
     apply (simp add: path_image_def pathstart_def pathfinish_def subset_iff image_def Bex_def)
    by (metis linorder_neqE_linordered_idom not_less)
   then show γ ` {0<..<1} ⊆ S
    using ‹arc h› ‹arc γ›
   by (metis arc_imp_simple_path path_image_def pathfinish_def pathstart_def simple_path_endless)
  qed
qed

lemma dense_accessible_frontier_point_pairs:
  fixes S :: 'a::{complete_space,real_normed_vector} set
  assumes S: open S connected S
     and opeSU: openin (top_of_set (frontier S)) U
     and opeSV: openin (top_of_set (frontier S)) V
     and U ≠ {} V ≠ {} U ≠ V
   obtains g where arc g pathstart g ∈ U pathfinish g ∈ V g ` {0<..<1} ⊆ S
proof -
  consider ¬ U ⊆ V | ¬ V ⊆ U
   using ‹U ≠ V› by blast
  then show ?thesis
  proof cases
   case 1 then show ?thesis
    using assms dense_access_fp_aux [OF S opeSU opeSV] that by blast
  next
   case 2
   obtain g where arc g and g: pathstart g ∈ V pathfinish g ∈ U g ` {0<..<1} ⊆ S
    using assms dense_access_fp_aux [OF S opeSV opeSU] 2 by blast
   show ?thesis
   proof
    show arc (reversepath g)
     by (simp add: ‹arc g› arc_reversepath)
    show pathstart (reversepath g) ∈ U pathfinish (reversepath g) ∈ V
     using g by auto
    show reversepath g ` {0<..<1} ⊆ S
     using g by (auto simp: reversepath_def)
   qed
  qed
qed

end
```

5.11 The Urysohn lemma, its consequences and other advanced material about metric spaces

theory *Urysohn*
imports *Abstract_Topological_Spaces Abstract_Metric_Spaces Infinite_Sum Arcwise_Connected*
begin

5.11.1 Urysohn lemma and Tietze's theorem

proposition *Urysohn_lemma*:
fixes *a b* :: *real*
assumes *normal_space X closedin X S closedin X T disjnt S T a* ≤ *b*
obtains *f* **where** *continuous_map X* (*top_of_set* {*a*..*b*}) *f f* ‘ *S* ⊆ {*a*} *f* ‘ *T* ⊆ {*b*}
proof −
obtain *U* **where** *openin X U S* ⊆ *U X closure_of U* ⊆ *topspace X* − *T*
using *assms* **unfolding** *normal_space_alt disjnt_def*
by (*metis Diff_mono Un_Diff_Int closedin_def subset_eq sup_bot_right*)
have ∃ *G* :: *real* ⇒ *'a set*. *G 0* = *U* ∧ *G 1* = *topspace X* − *T* ∧
(∀ *x* ∈ *dyadics* ∩ {*0..1*}. ∀ *y* ∈ *dyadics* ∩ {*0..1*}. *x* < *y* ⟶ *openin X* (*G x*) ∧ *openin X* (*G y*) ∧ *X closure_of* (*G x*) ⊆ *G y*)
proof (*rule recursion_on_dyadic_fractions*)
show *openin X U* ∧ *openin X* (*topspace X* − *T*) ∧ *X closure_of U* ⊆ *topspace X* − *T*
using ‹*X closure_of U* ⊆ *topspace X* − *T*› ‹*openin X U*› ‹*closedin X T*› **by** *blast*
show ∃ *z*. (*openin X x* ∧ *openin X z* ∧ *X closure_of x* ⊆ *z*) ∧ *openin X z* ∧ *openin X y* ∧ *X closure_of z* ⊆ *y*
if *openin X x* ∧ *openin X y* ∧ *X closure_of x* ⊆ *y* **for** *x y*
by (*meson that closedin_closure_of normal_space_alt* ‹*normal_space X*›)
show *openin X x* ∧ *openin X z* ∧ *X closure_of x* ⊆ *z*
if *openin X x* ∧ *openin X y* ∧ *X closure_of x* ⊆ *y* **and** *openin X y* ∧ *openin X z* ∧ *X closure_of y* ⊆ *z* **for** *x y z*
by (*meson that closure_of_subset openin_subset subset_trans*)
qed
then obtain *G* :: *real* ⇒ *'a set*
where *G0*: *G 0* = *U* **and** *G1*: *G 1* = *topspace X* − *T*
and *G*: ⋀*x y*. ⟦*x* ∈ *dyadics*; *y* ∈ *dyadics*; *0* ≤ *x*; *x* < *y*; *y* ≤ *1*⟧
⟹ *openin X* (*G x*) ∧ *openin X* (*G y*) ∧ *X closure_of* (*G x*) ⊆ *G y*
by (*smt* (*verit*, *del_insts*) *Int_iff atLeastAtMost_iff*)
define *f* **where** *f* ≡ λ*x*. *Inf*(*insert 1* {*r*. *r* ∈ *dyadics* ∩ {*0..1*} ∧ *x* ∈ *G r*})
have *f_ge*: *f x* ≥ *0* **if** *x* ∈ *topspace X* **for** *x*
unfolding *f_def* **by** (*force intro*: *cInf_greatest*)
moreover have *f_le1*: *f x* ≤ *1* **if** *x* ∈ *topspace X* **for** *x*
proof −
have *bdd_below* {*r* ∈ *dyadics* ∩ {*0..1*}. *x* ∈ *G r*}
by (*auto simp*: *bdd_below_def*)

```
    then show ?thesis
      by (auto simp: f_def cInf_lower)
  qed
  ultimately have fim: f ‘ topspace X ⊆ {0..1}
    by (auto simp: f_def)
  have 0: 0 ∈ dyadics ∩ {0..1::real} and 1: 1 ∈ dyadics ∩ {0..1::real}
    by (force simp: dyadics_def)+
  then have opeG: openin X (G r) if r ∈ dyadics ∩ {0..1} for r
    using G G0 ‹openin X U› less_eq_real_def that by auto
  have x ∈ G 0 if x ∈ S for x
    using G0 ‹S ⊆ U› that by blast
  with 0 have fimS: f ‘ S ⊆ {0}
    unfolding f_def by (force intro!: cInf_eq_minimum)
  have False if r ∈ dyadics 0 ≤ r r < 1 x ∈ G r x ∈ T for r x
    using G [of r 1] 1
   by (smt (verit, best) DiffD2 G1 Int_iff closure_of_subset inf.orderE openin_subset
that)
  then have r≥1 if r ∈ dyadics 0 ≤ r r ≤ 1 x ∈ G r x ∈ T for r x
    using linorder_not_le that by blast
  then have fimT: f ‘ T ⊆ {1}
    unfolding f_def by (force intro!: cInf_eq_minimum)
  have fle1: f z ≤ 1 for z
    by (force simp: f_def intro: cInf_lower)
  have fle: f z ≤ x if x ∈ dyadics ∩ {0..1} z ∈ G x for z x
    using that by (force simp: f_def intro: cInf_lower)
  have ∗: b ≤ f z if b ≤ 1 ⋀x. ⟦x ∈ dyadics ∩ {0..1}; z ∈ G x⟧ ⟹ b ≤ x for z b
    using that by (force simp: f_def intro: cInf_greatest)
  have ∗∗: r ≤ f x if r: r ∈ dyadics ∩ {0..1} x ∉ G r for r x
  proof (rule ∗)
    show r ≤ s if s ∈ dyadics ∩ {0..1} and x ∈ G s for s :: real
     using that r G [of s r] by (force simp: dest: closure_of_subset openin_subset)
  qed (use that in force)

  have ∃ U. openin X U ∧ x ∈ U ∧ (∀ y ∈ U. |f y − f x| < ε)
    if x ∈ topspace X and 0 < ε for x ε
  proof −
    have A: ∃ r. r ∈ dyadics ∩ {0..1} ∧ r < y ∧ |r − y| < d if 0 < y y ≤ 1 0 <
d for y d::real
    proof −
      obtain n q r
        where of_nat q / 2^n < y y < of_nat r / 2^n |q / 2^n − r / 2^n | < d
        by (smt (verit, del_insts) padic_rational_approximation_straddle_pos ‹0
< d› ‹0 < y›)
      then show ?thesis
        unfolding dyadics_def
        using divide_eq_0_iff that(2) by fastforce
    qed
    have B: ∃ r. r ∈ dyadics ∩ {0..1} ∧ y < r ∧ |r − y| < d if 0 ≤ y y < 1 0 <
d for y d::real
```

```
    proof −
      obtain n q r
        where of_nat q / 2^n ≤ y y < of_nat r / 2^n |q / 2^n − r / 2^n | < d
        using padic_rational_approximation_straddle_pos_le
        by (smt (verit, del_insts) ‹0 < d› ‹0 ≤ y›)
      then show ?thesis
        apply (clarsimp simp: dyadics_def)
        using divide_eq_0_iff ‹y < 1›
      by (smt (verit) divide_nonneg_nonneg divide_self of_nat_0_le_iff of_nat_1
power_0 zero_le_power)
    qed
    show ?thesis
    proof (cases f x = 0)
      case True
      with B[of 0] obtain r where r: r ∈ dyadics ∩ {0..1} 0 < r |r| < ε/2
        by (smt (verit) ‹0 < ε› half_gt_zero)
      show ?thesis
      proof (intro exI conjI)
        show openin X (G r)
          using opeG r(1) by blast
        show x ∈ G r
          using True ∗∗ r by force
        show ∀ y ∈ G r. |f y − f x| < ε
         using f_ge ‹openin X (G r)› fle openin_subset r by (fastforce simp: True)
      qed
    next
      case False
      show ?thesis
      proof (cases f x = 1)
        case True
        with A[of 1] obtain r where r: r ∈ dyadics ∩ {0..1} r < 1 |r−1| < ε/2
          by (smt (verit) ‹0 < ε› half_gt_zero)
        define G' where G' ≡ topspace X − X closure_of G r
        show ?thesis
        proof (intro exI conjI)
          show openin X G'
            unfolding G'_def by fastforce
          obtain r' where r' ∈ dyadics ∧ 0 ≤ r' ∧ r' ≤ 1 ∧ r < r' ∧ |r' − r| <
1 − r
            using B r by force
          moreover
          have 1 − r ∈ dyadics 0 ≤ r
            using 1 r dyadics_diff by force+
          ultimately have x ∉ X closure_of G r
            using G True r fle by force
          then show x ∈ G'
            by (simp add: G'_def that)
          show ∀ y ∈ G'. |f y − f x| < ε
            using ∗∗ f_le1 in_closure_of r by (fastforce simp: True G'_def)
```

```
      qed
    next
      case False
      have 0 < f x f x < 1
        using fle1 f_ge that(1) ‹f x ≠ 0› ‹f x ≠ 1› by (metis order_le_less) +
      obtain r where r: r ∈ dyadics ∩ {0..1} r < f x |r − f x| < ε / 2
        using A ‹0 < ε› ‹0 < f x› ‹f x < 1› by (smt (verit, best) half_gt_zero)
      obtain r' where r': r' ∈ dyadics ∩ {0..1} f x < r' |r' − f x| < ε / 2
        using B ‹0 < ε› ‹0 < f x› ‹f x < 1› by (smt (verit, best) half_gt_zero)
      have r < 1
        using ‹f x < 1› r(2) by force
      show ?thesis
      proof (intro conjI exI)
        show openin X (G r' − X closure_of G r)
          using closedin_closure_of opeG r' by blast
        have x ∈ X closure_of G r ⟹ False
          using B [of r f x − r] r ‹r < 1› G [of r] fle by force
        then show x ∈ G r' − X closure_of G r
          using ** r' by fastforce
        show ∀ y∈G r' − X closure_of G r. |f y − f x| < ε
        using r r' ** G closure_of_subset field_sum_of_halves fle openin_subset
subset_eq
          by (smt (verit) DiffE opeG)
      qed
    qed
  qed
  then have contf: continuous_map X (top_of_set {0..1}) f
    by (force simp: Met_TC.continuous_map_to_metric dist_real_def continu-
ous_map_in_subtopology fim simp flip: mtopology_is_euclidean)
  define g where g ≡ λx. a + (b − a) * f x
  show thesis
  proof
    have continuous_map X euclideanreal g
      using contf ‹a ≤ b› unfolding g_def by (auto simp: continuous_intros
continuous_map_in_subtopology)
    moreover have g ‘ (topspace X) ⊆ {a..b}
      using mult_left_le [of f _ b−a] contf ‹a ≤ b›
      by (simp add: g_def add.commute continuous_map_in_subtopology im-
age_subset_iff le_diff_eq)
    ultimately show continuous_map X (top_of_set {a..b}) g
      by (meson continuous_map_in_subtopology)
    show g ‘ S ⊆ {a} g ‘ T ⊆ {b}
      using fimS fimT by (auto simp: g_def)
  qed
qed

lemma Urysohn_lemma_alt:
  fixes a b :: real
```

assumes *normal_space X closedin X S closedin X T disjnt S T*
obtains *f* **where** *continuous_map X euclideanreal f f ‘ S ⊆ {a} f ‘ T ⊆ {b}*
by (*metis Urysohn_lemma assms continuous_map_in_subtopology disjnt_sym linear*)

lemma *normal_space_iff_Urysohn_gen_alt*:
assumes *a ≠ b*
shows *normal_space X ⟷*
(∀ S T. closedin X S ∧ closedin X T ∧ disjnt S T
⟶ (∃f. continuous_map X euclideanreal f ∧ f ‘ S ⊆ {a} ∧ f ‘ T ⊆ {b}))
(**is** *?lhs=?rhs*)
proof
show *?lhs ⟹ ?rhs*
by (*metis Urysohn_lemma_alt*)
next
assume *R*: *?rhs*
show *?lhs*
unfolding *normal_space_def*
proof *clarify*
fix *S T*
assume *closedin X S* **and** *closedin X T* **and** *disjnt S T*
with *R* **obtain** *f* **where** *contf*: *continuous_map X euclideanreal f* **and** *f ‘ S ⊆ {a} f ‘ T ⊆ {b}*
by *meson*
show *∃ U V. openin X U ∧ openin X V ∧ S ⊆ U ∧ T ⊆ V ∧ disjnt U V*
proof (*intro conjI exI*)
show *openin X {x ∈ topspace X. f x ∈ ball a (|a − b| / 2)}*
by (*force intro*!: *openin_continuous_map_preimage* [*OF contf*])
show *openin X {x ∈ topspace X. f x ∈ ball b (|a − b| / 2)}*
by (*force intro*!: *openin_continuous_map_preimage* [*OF contf*])
show *S ⊆ {x ∈ topspace X. f x ∈ ball a (|a − b| / 2)}*
using ‹*closedin X S*› *closedin_subset* ‹*f ‘ S ⊆ {a}*› *assms* **by** *force*
show *T ⊆ {x ∈ topspace X. f x ∈ ball b (|a − b| / 2)}*
using ‹*closedin X T*› *closedin_subset* ‹*f ‘ T ⊆ {b}*› *assms* **by** *force*
have *⋀x. ⟦x ∈ topspace X; dist a (f x) < |a−b|/2; dist b (f x) < |a−b|/2⟧ ⟹ False*
by (*smt* (*verit*, *best*) *dist_real_def dist_triangle_half_l*)
then show *disjnt {x ∈ topspace X. f x ∈ ball a (|a−b| / 2)} {x ∈ topspace X. f x ∈ ball b (|a−b| / 2)}*
using *disjnt_iff* **by** *fastforce*
qed
qed
qed

lemma *normal_space_iff_Urysohn_gen*:
fixes *a b*::*real*
shows
a < b ⟹

normal_space X ⟷
(∀ *S T*. *closedin X S* ∧ *closedin X T* ∧ *disjnt S T*
⟶ (∃ *f*. *continuous_map X* (*top_of_set* {*a*..*b*}) *f* ∧
f ‘ *S* ⊆ {*a*} ∧ *f* ‘ *T* ⊆ {*b*}))
by (*metis linear not_le Urysohn_lemma normal_space_iff_Urysohn_gen_alt continuous_map_in_subtopology*)

lemma *normal_space_iff_Urysohn_alt*:
normal_space X ⟷
(∀ *S T*. *closedin X S* ∧ *closedin X T* ∧ *disjnt S T*
⟶ (∃ *f*. *continuous_map X euclideanreal f* ∧
f ‘ *S* ⊆ {*0*} ∧ *f* ‘ *T* ⊆ {*1*}))
by (*rule normal_space_iff_Urysohn_gen_alt*) *auto*

lemma *normal_space_iff_Urysohn*:
normal_space X ⟷
(∀ *S T*. *closedin X S* ∧ *closedin X T* ∧ *disjnt S T*
⟶ (∃ *f*::*'a*⇒*real*. *continuous_map X* (*top_of_set* {*0*..*1*}) *f* ∧
f ‘ *S* ⊆ {*0*} ∧ *f* ‘ *T* ⊆ {*1*}))
by (*rule normal_space_iff_Urysohn_gen*) *auto*

lemma *normal_space_perfect_map_image*:
⟦*normal_space X*; *perfect_map X Y f*⟧ ⟹ *normal_space Y*
unfolding *perfect_map_def proper_map_def*
using *normal_space_continuous_closed_map_image* **by** *fastforce*

lemma *Hausdorff_normal_space_closed_continuous_map_image*:
⟦*normal_space X*; *closed_map X Y f*; *continuous_map X Y f*;
f ‘ *topspace X* = *topspace Y*; *t1_space Y*⟧
⟹ *Hausdorff_space Y*
by (*metis normal_space_continuous_closed_map_image normal_t1_imp_Hausdorff_space*)

lemma *normal_Hausdorff_space_closed_continuous_map_image*:
⟦*normal_space X*; *Hausdorff_space X*; *closed_map X Y f*;
continuous_map X Y f; *f* ‘ *topspace X* = *topspace Y*⟧
⟹ *normal_space Y* ∧ *Hausdorff_space Y*
by (*meson normal_space_continuous_closed_map_image normal_t1_eq_Hausdorff_space t1_space_closed_map_image*)

lemma *Lindelof_cover*:
assumes *regular_space X* **and** *Lindelof_space X* **and** *S* ≠ {}
and *clo*: *closedin X S closedin X T disjnt S T*
obtains *h* :: *nat* ⇒ *'a set* **where**
⋀*n*. *openin X* (*h n*) ⋀*n*. *disjnt T* (*X closure_of* (*h n*)) **and** *S* ⊆ ⋃(*range h*)
proof −
have ∃ *U*. *openin X U* ∧ *x* ∈ *U* ∧ *disjnt T* (*X closure_of U*)
if *x* ∈ *S* **for** *x*
using ‹*regular_space X*› **unfolding** *regular_space*
by (*metis* (*full_types*) *Diff_iff* ‹*disjnt S T*› *clo closure_of_eq disjnt_iff in_closure_of*

```
that)
  then obtain h where oh: ⋀x. x ∈ S ⟹ openin X (h x)
    and xh: ⋀x. x ∈ S ⟹ x ∈ h x
    and dh: ⋀x. x ∈ S ⟹ disjnt T (X closure_of h x)
    by metis
  have Lindelof_space(subtopology X S)
   by (simp add: Lindelof_space_closedin_subtopology ‹Lindelof_space X› ‹closedin
X S›)
  then obtain 𝒰 where 𝒰: countable 𝒰 ∧ 𝒰 ⊆ h ` S ∧ S ⊆ ⋃𝒰
   unfolding Lindelof_space_subtopology_subset [OF closedin_subset [OF ‹closedin
X S›]]
    by (smt (verit, del_insts) oh xh UN_I image_iff subsetI)
  with ‹S ≠ {}› have 𝒰 ≠ {}
    by blast
  show ?thesis
  proof
    show openin X (from_nat_into 𝒰 n) for n
      by (metis 𝒰 from_nat_into image_iff ‹𝒰 ≠ {}› oh subsetD)
    show disjnt T (X closure_of (from_nat_into 𝒰) n) for n
      using dh from_nat_into [OF ‹𝒰 ≠ {}›]
      by (metis 𝒰 f_inv_into_f inv_into_into subset_eq)
    show S ⊆ ⋃(range (from_nat_into 𝒰))
      by (simp add: 𝒰 ‹𝒰 ≠ {}›)
  qed
qed

lemma regular_Lindelof_imp_normal_space:
  assumes regular_space X and Lindelof_space X
  shows normal_space X
  unfolding normal_space_def
proof clarify
  fix S T
  assume clo: closedin X S closedin X T and disjnt S T
  show ∃ U V. openin X U ∧ openin X V ∧ S ⊆ U ∧ T ⊆ V ∧ disjnt U V
  proof (cases S={} ∨ T={})
    case True
    with clo show ?thesis
     by (meson closedin_def disjnt_empty1 disjnt_empty2 openin_empty openin_topspace
subset_empty)
  next
    case False
    obtain h :: nat ⇒ 'a set where
      openh: ⋀n. openin X (h n) and dish: ⋀n. disjnt T (X closure_of (h n))
      and Sh: S ⊆ ⋃(range h)
      by (metis Lindelof_cover False ‹disjnt S T› assms clo)
    obtain k :: nat ⇒ 'a set where
      openk: ⋀n. openin X (k n) and disk: ⋀n. disjnt S (X closure_of (k n))
      and Tk: T ⊆ ⋃(range k)
      by (metis Lindelof_cover False ‹disjnt S T› assms clo disjnt_sym)
```

```
    define U where U ≡ ⋃i. h i − (⋃j<i. X closure_of k j)
    define V where V ≡ ⋃i. k i − (⋃j≤i. X closure_of h j)
    show ?thesis
    proof (intro exI conjI)
      show openin X U openin X V
        unfolding U_def V_def
        by (force intro!: opek opeh closedin_Union closedin_closure_of)+
      show S ⊆ U T ⊆ V
        using Sh Tk dish disk by (fastforce simp: U_def V_def disjnt_iff)+
      have ⋀x i j. ⟦x ∈ k i; x ∈ h j; ∀j≤i. x ∉ X closure_of h j⟧
              ⟹ ∃i<j. x ∈ X closure_of k i
        by (metis in_closure_of linorder_not_less opek openin_subset subsetD)
      then show disjnt U V
        by (force simp: U_def V_def disjnt_iff)
    qed
  qed
qed

theorem Tietze_extension_closed_real_interval:
  assumes normal_space X and closedin X S
    and contf: continuous_map (subtopology X S) euclideanreal f
    and fim: f ‘ S ⊆ {a..b} and a ≤ b
  obtains g
  where continuous_map X euclideanreal g
        ⋀x. x ∈ S ⟹ g x = f x g ‘ topspace X ⊆ {a..b}
proof −
  define c where c ≡ max |a| |b| + 1
  have 0 < c and c: ⋀x. x ∈ S ⟹ |f x| ≤ c
    using fim by (auto simp: c_def image_subset_iff)
  define good where
    good ≡ λg n. continuous_map X euclideanreal g ∧ (∀x ∈ S. |f x − g x| ≤ c *
(2/3)^n)
  have step: ∃g. good g (Suc n) ∧
             (∀x ∈ topspace X. |g x − h x| ≤ c * (2/3)^n / 3)
    if h: good h n for n h
  proof −
    have pos: 0 < c * (2/3) ^ n
      by (simp add: ‹0 < c›)
    have S_eq: S = topspace(subtopology X S) and S ⊆ topspace X
      using ‹closedin X S› closedin_subset by auto
    define d where d ≡ c/3 * (2/3) ^ n
    define SA where SA ≡ {x ∈ S. f x − h x ∈ {..−d}}
    define SB where SB ≡ {x ∈ S. f x − h x ∈ {d..}}
    have contfh: continuous_map (subtopology X S) euclideanreal (λx. f x − h x)
      using that
    by (simp add: contf good_def continuous_map_diff continuous_map_from_subtopology)
    then have closedin (subtopology X S) SA
      unfolding SA_def
      by (smt (verit, del_insts) closed_closedin continuous_map_closedin Col-
```

```
lect_cong S_eq closed_real_atMost)
    then have closedin X SA
      using ‹closedin X S› closedin_trans_full by blast
    moreover have  closedin (subtopology X S) SB
      unfolding SB_def
      using closedin_continuous_map_preimage_gen [OF contfh]
     by (metis (full_types) S_eq closed_atLeast closed_closedin closedin_topspace)
    then have closedin X SB
      using ‹closedin X S› closedin_trans_full by blast
    moreover have disjnt SA SB
      using pos by (auto simp: d_def disjnt_def SA_def SB_def)
    moreover have −d ≤ d
      using pos by (auto simp: d_def)
    ultimately
    obtain g where contg: continuous_map X (top_of_set {− d..d}) g
      and ga: g ‘ SA ⊆ {− d} and gb: g ‘ SB ⊆ {d}
      using Urysohn_lemma ‹normal_space X› by metis
    then have g_le_d: ⋀x. x ∈ topspace X ⟹ |g x| ≤ d
      by (fastforce simp: abs_le_iff continuous_map_def minus_le_iff)
    have g_eq_d: ⋀x. ⟦x ∈ S; f x − h x ≤ −d⟧ ⟹ g x = −d
      using ga by (auto simp: SA_def)
    have g_eq_negd: ⋀x. ⟦x ∈ S; f x − h x ≥ d⟧ ⟹ g x = d
      using gb by (auto simp: SB_def)
    show ?thesis
      unfolding good_def
    proof (intro conjI strip exI)
      show continuous_map X euclideanreal (λx. h x + g x)
        using contg continuous_map_add continuous_map_in_subtopology that
        unfolding good_def by blast
      show |f x − (h x + g x)| ≤ c ∗ (2 / 3) ^ Suc n if x ∈ S for x
      proof −
        have x: x ∈ topspace X
          using ‹S ⊆ topspace X› that by auto
        have |f x − h x| ≤ c ∗ (2/3) ^ n
          using good_def h that by blast
        with g_eq_d [OF that] g_eq_negd [OF that] g_le_d [OF x]
        have |f x − (h x + g x)| ≤ d + d
          unfolding d_def by linarith
        then show ?thesis
          by (simp add: d_def)
      qed
      show |h x + g x − h x| ≤ c ∗ (2 / 3) ^ n / 3 if x ∈ topspace X for x
        using that d_def g_le_d by auto
    qed
  qed
  then obtain nxtg where nxtg: ⋀h n. good h n ⟹
        good (nxtg h n) (Suc n) ∧ (∀x ∈ topspace X. |nxtg h n x − h x| ≤ c ∗
(2/3)^n / 3)
    by metis
```

```
  define g where g ≡ rec_nat (λx. 0) (λn r. nxtg r n)
  have [simp]: g 0 x = 0 for x
    by (auto simp: g_def)
  have g_Suc: g(Suc n) = nxtg (g n) n for n
    by (auto simp: g_def)
  have good: good (g n) n for n
  proof (induction n)
    case 0
    with c show ?case
      by (auto simp: good_def)
  qed (simp add: g_Suc nxtg)
  have ∗: ⋀n x. x ∈ topspace X ⟹ |g(Suc n) x − g n x| ≤ c ∗ (2/3) ^ n / 3
    using nxtg g_Suc good by force
  obtain h where conth:  continuous_map X euclideanreal h
    and h: ⋀ε. 0 < ε ⟹ ∀F n in sequentially. ∀x∈topspace X. dist (g n x) (h x)
< ε
  proof (rule Met_TC.continuous_map_uniformly_Cauchy_limit)
    show ∀F n in sequentially. continuous_map X (Met_TC.mtopology) (g n)
      using good good_def by fastforce
    show ∃N. ∀m n x. N ≤ m ⟶ N ≤ n ⟶ x ∈ topspace X ⟶ dist (g m x)
(g n x) < ε
      if ε > 0 for ε
    proof −
      have ∀F n in sequentially. |(2/3) ^ n| < ε/c
      proof (rule Archimedean_eventually_pow_inverse)
        show 0 < ε / c
          by (simp add: ‹0 < c› that)
      qed auto
      then obtain N where N: ⋀n. n ≥ N ⟹ |(2/3) ^ n| < ε/c
        by (meson eventually_sequentially order_le_less_trans)
      have |g m x − g n x| < ε
        if N ≤ m N ≤ n and x: x ∈ topspace X m ≤ n for m n x
      proof (cases m < n)
        case True
        have 23: (∑k = m..<n. (2/3)^k) = 3 ∗ ((2/3) ^ m − (2/3::real) ^ n)
          using ‹m ≤ n›
          by (induction n) (auto simp: le_Suc_eq)
        have |g m x − g n x| ≤ |∑k = m..<n. g (Suc k) x − g k x|
          by (subst sum_Suc_diff' [OF ‹m ≤ n›]) linarith
        also have ... ≤ (∑k = m..<n. |g (Suc k) x − g k x|)
          by (rule sum_abs)
        also have ... ≤ (∑k = m..<n. c ∗ (2/3)^k / 3)
          by (meson ∗ sum_mono x(1))
        also have ... = (c/3) ∗ (∑k = m..<n. (2/3)^k)
          by (simp add: sum_distrib_left)
        also have ... = (c/3) ∗ 3 ∗ ((2/3) ^ m − (2/3) ^ n)
          by (simp add: sum_distrib_left 23)
        also have ... < (c/3) ∗ 3 ∗ ((2/3) ^ m)
          using ‹0 < c› by auto
```

```
      also have ... < ε
        using N [OF ‹N ≤ m›] ‹0 < c› by (simp add: field_simps)
      finally show ?thesis .
    qed (use ‹0 < ε› ‹m ≤ n› in auto)
    then show ?thesis
      by (metis dist_commute_lessI dist_real_def nle_le)
  qed
 qed auto
 define φ where φ ≡ λx. max a (min (h x) b)
 show thesis
 proof
  show continuous_map X euclidean φ
    unfolding φ_def using conth by (intro continuous_intros) auto
  show φ x = f x if x ∈ S for x
  proof −
    have x: x ∈ topspace X
      using ‹closedin X S› closedin_subset that by blast
    have h x = f x
    proof (rule Met_TC.limitin_metric_unique)
      show limitin Met_TC.mtopology (λn. g n x) (h x) sequentially
        using h x by (force simp: tendsto_iff eventually_sequentially)
      show limitin Met_TC.mtopology (λn. g n x) (f x) sequentially
      proof (clarsimp simp: tendsto_iff)
        fix ε::real
        assume ε > 0
        then have ∀F n in sequentially. |(2/3) ^ n| < ε/c
          by (intro Archimedean_eventually_pow_inverse) (auto simp: ‹c > 0›)
        then show ∀F n in sequentially. dist (g n x) (f x) < ε
          apply eventually_elim
           by (smt (verit) good x good_def ‹c > 0› dist_real_def mult.commute
pos_less_divide_eq that)
      qed
    qed auto
    then show ?thesis
      using that fim by (auto simp: φ_def)
  qed
  then show φ ‘ topspace X ⊆ {a..b}
    using fim ‹a ≤ b› by (auto simp: φ_def)
 qed
qed


theorem Tietze_extension_realinterval:
 assumes XS: normal_space X closedin X S and T: is_interval T T ≠ {}
   and contf: continuous_map (subtopology X S) euclideanreal f
   and f ‘ S ⊆ T
 obtains g where continuous_map X euclideanreal g  g ‘ topspace X ⊆ T  ⋀x.
x ∈ S ⟹ g x = f x
proof −
```

```
  define Φ where
        Φ ≡ λT::real set. ∀f. continuous_map (subtopology X S) euclidean f ⟶
f‘S ⊆ T
              ⟶ (∃g. continuous_map X euclidean g ∧ g ‘ topspace X ⊆ T ∧ (∀x
∈ S. g x = f x))
  have Φ T
    if ∗: ⋀T. ⟦bounded T; is_interval T; T ≠ {}⟧ ⟹ Φ T
      and is_interval T T ≠ {} for T
    unfolding Φ_def
  proof (intro strip)
    fix f
    assume contf: continuous_map (subtopology X S) euclideanreal f
      and f ‘ S ⊆ T
    have ΦT: Φ ((λx. x / (1 + |x|)) ‘ T)
    proof (rule ∗)
      show bounded ((λx. x / (1 + |x|)) ‘ T)
        using shrink_range [of T] by (force intro: boundedI [where B=1])
      show is_interval ((λx. x / (1 + |x|)) ‘ T)
        using connected_shrink that(2) is_interval_connected_1 by blast
      show (λx. x / (1 + |x|)) ‘ T ≠ {}
        using ‹T ≠ {}› by auto
    qed
    moreover have continuous_map (subtopology X S) euclidean ((λx. x / (1 +
|x|)) ∘ f)
     by (metis contf continuous_map_compose continuous_map_into_fulltopology
continuous_map_real_shrink)
    moreover have ((λx. x / (1 + |x|)) ∘ f) ‘ S ⊆ (λx. x / (1 + |x|)) ‘ T
      using ‹f ‘ S ⊆ T› by auto
    ultimately obtain g
      where contg: continuous_map X euclidean g
        and gim: g ‘ topspace X ⊆ (λx. x / (1 + |x|)) ‘ T
        and geq: ⋀x. x ∈ S ⟹ g x = ((λx. x / (1 + |x|)) ∘ f) x
      using ΦT unfolding Φ_def by force
    show ∃g. continuous_map X euclideanreal g ∧ g ‘ topspace X ⊆ T ∧ (∀x∈S.
g x = f x)
    proof (intro conjI exI)
      have continuous_map X (top_of_set {−1<..<1}) g
        using contg continuous_map_in_subtopology gim shrink_range by blast
      then show continuous_map X euclideanreal ((λx. x / (1 − |x|)) ∘ g)
       by (rule continuous_map_compose) (auto simp: continuous_on_real_grow)
      show ((λx. x / (1 − |x|)) ∘ g) ‘ topspace X ⊆ T
        using gim real_grow_shrink by fastforce
      show ∀x∈S. ((λx. x / (1 − |x|)) ∘ g) x = f x
        using geq real_grow_shrink by force
    qed
  qed
  moreover have Φ T
    if bounded T is_interval T T ≠ {} for T
    unfolding Φ_def
```

```
  proof (intro strip)
    fix f
    assume contf: continuous_map (subtopology X S) euclideanreal f
      and f ‘ S ⊆ T
    obtain a b where ab: closure T = {a..b}
    by (meson ‹bounded T› ‹is_interval T› compact_closure connected_compact_interval_1

          connected_imp_connected_closure is_interval_connected)
    with ‹T ≠ {}› have a ≤ b by auto
    have f ‘ S ⊆ {a..b}
      using ‹f ‘ S ⊆ T› ab closure_subset by auto
    then obtain g where contg: continuous_map X euclideanreal g
      and gf: ⋀x. x ∈ S ⟹ g x = f x and gim: g ‘ topspace X ⊆ {a..b}
       using Tietze_extension_closed_real_interval [OF XS contf _ ‹a ≤ b›] by
metis
    define W where W ≡ {x ∈ topspace X. g x ∈ closure T − T}
    have {a..b} − {a, b} ⊆ T
      using that
    by (metis ab atLeastAtMost_diff_ends convex_interior_closure interior_atLeastAtMost_real

        interior_subset is_interval_convex)
    with finite_imp_compact have compact (closure T − T)
     by (metis Diff_eq_empty_iff Diff_insert2 ab finite.emptyI finite_Diff_insert)
    then have closedin X W
       unfolding W_def using closedin_continuous_map_preimage [OF contg]
compact_imp_closed by force
    moreover have disjnt W S
      unfolding W_def disjnt_iff using ‹f ‘ S ⊆ T› gf by blast
    ultimately obtain h :: 'a ⇒ real
      where conth: continuous_map X (top_of_set {0..1}) h
          and him: h ‘ W ⊆ {0} h ‘ S ⊆ {1}
      by (metis XS normal_space_iff_Urysohn)
    then have him01: h ‘ topspace X ⊆ {0..1}
      by (meson continuous_map_in_subtopology)
    obtain z where z ∈ T
      using ‹T ≠ {}› by blast
    define g' where g' ≡ λx. z + h x ∗ (g x − z)
    show ∃g. continuous_map X euclidean g ∧ g ‘ topspace X ⊆ T ∧ (∀x∈S. g x
= f x)
    proof (intro exI conjI)
      show continuous_map X euclideanreal g'
        unfolding g'_def using contg conth continuous_map_in_subtopology
        by (intro continuous_intros) auto
      show g' ‘ topspace X ⊆ T
        unfolding g'_def
      proof clarify
        fix x
        assume x ∈ topspace X
        show z + h x ∗ (g x − z) ∈ T
```

```
      proof (cases g x ∈ T)
        case True
        define w where w ≡ z + h x * (g x − z)
        have |h x| * |g x − z| ≤ |g x − z| |1 − h x| * |g x − z| ≤ |g x − z|
       using him01 ‹x ∈ topspace X› by (force simp: intro: mult_left_le_one_le)+
        then consider z ≤ w ∧ w ≤ g x | g x ≤ w ∧ w ≤ z
          unfolding w_def by (smt (verit) left_diff_distrib mult_cancel_right2
mult_minus_right zero_less_mult_iff)
        then show ?thesis
          using ‹is_interval T› unfolding w_def is_interval_1 by (metis True
‹z ∈ T›)
      next
        case False
        then have g x ∈ closure T
          using ‹x ∈ topspace X› ab gim by blast
        then have h x = 0
       using him False ‹x ∈ topspace X› by (auto simp: W_def image_subset_iff)
        then show ?thesis
          by (simp add: ‹z ∈ T›)
      qed
    qed
    show ∀x∈S. g' x = f x
      using gf him by (auto simp: W_def g'_def)
  qed
 qed
 ultimately show thesis
   using assms that unfolding Φ_def by best
qed

lemma normal_space_iff_Tietze:
  normal_space X ⟷
   (∀f S. closedin X S ∧
         continuous_map (subtopology X S) euclidean f
         ⟶ (∃g::'a ⇒ real. continuous_map X euclidean g ∧ (∀x ∈ S. g x = f
x)))
  (is ?lhs ⟷ ?rhs)
proof
 assume ?lhs
 then show ?rhs
   by (metis Tietze_extension_realinterval empty_not_UNIV is_interval_univ
subset_UNIV)
next
 assume R: ?rhs
 show ?lhs
   unfolding normal_space_iff_Urysohn_alt
 proof clarify
   fix S T
   assume closedin X S
     and closedin X T
```

and *disjnt S T*
then have *cloST*: *closedin X* (*S* ∪ *T*)
by (*simp add*: *closedin_Un*)
moreover
have *continuous_map* (*subtopology X* (*S* ∪ *T*)) *euclideanreal* (λ*x*. *if x* ∈ *S then 0 else 1*)
unfolding *continuous_map_closedin*
proof (*intro conjI strip*)
fix *C* :: *real set*
define *D* **where** *D* ≡ {*x* ∈ *topspace X*. *if x* ∈ *S then 0* ∈ *C else x* ∈ *T* ∧ *1* ∈ *C*}
have *D* ∈ {{}, *S*, *T*, *S* ∪ *T*}
unfolding *D_def*
using *closedin_subset* [*OF* ‹*closedin X S*›] *closedin_subset* [*OF* ‹*closedin X T*›] ‹*disjnt S T*›
by (*auto simp*: *disjnt_iff*)
then have *closedin X D*
using ‹*closedin X S*› ‹*closedin X T*› *closedin_empty* **by** *blast*
with *closedin_subset_topspace*
show *closedin* (*subtopology X* (*S* ∪ *T*)) {*x* ∈ *topspace* (*subtopology X* (*S* ∪ *T*)). (*if x* ∈ *S then 0*::*real else 1*) ∈ *C*}
apply (*simp add*: *D_def*)
by (*smt* (*verit*, *best*) *Collect_cong Collect_mono_iff Un_def closedin_subset_topspace*)
qed *auto*
ultimately obtain *g* :: *'a* ⇒ *real* **where**
contg: *continuous_map X euclidean g* **and** *gf*: ∀ *x* ∈ *S* ∪ *T*. *g x* = (*if x* ∈ *S then 0 else 1*)
using *R* **by** *blast*
then show ∃ *f*. *continuous_map X euclideanreal f* ∧ *f* ‘ *S* ⊆ {*0*} ∧ *f* ‘ *T* ⊆ {*1*}
by (*smt* (*verit*) *Un_iff* ‹*disjnt S T*› *disjnt_iff image_subset_iff insert_iff*)
qed
qed

5.11.2 Random metric space stuff

lemma *metrizable_imp_k_space*:
assumes *metrizable_space X*
shows *k_space X*
proof −
obtain *M d* **where** *Metric_space M d* **and** *Xeq*: *X* = *Metric_space.mtopology M d*
using *assms* **unfolding** *metrizable_space_def* **by** *metis*
then interpret *Metric_space M d*
by *blast*
show *?thesis*
unfolding *k_space Xeq*
proof *clarsimp*
fix *S*

assume $S \subseteq M$ **and** S: $\forall K.$ *compactin mtopology* $K \longrightarrow$ *closedin* (*subtopology mtopology* K) $(K \cap S)$
have $l \in S$
if σ: *range* $\sigma \subseteq S$ **and** l: *limitin mtopology* σ l *sequentially* **for** σ l
proof −
define K **where** $K \equiv$ *insert* l (*range* σ)
interpret *Submetric* M d K
proof
show $K \subseteq M$
unfolding K_def **using** l *limitin_mspace* ‹$S \subseteq M$› σ **by** *blast*
qed
have *compactin mtopology* K
unfolding K_def **using** ‹$S \subseteq M$› σ
by (*force intro*: *compactin_sequence_with_limit* [*OF* l])
then have $*$: *closedin sub.mtopology* $(K \cap S)$
by (*simp add*: S *mtopology_submetric*)
have σ $n \in K \cap S$ **for** n
by (*simp add*: K_def *range_subsetD* σ)
moreover have *limitin sub.mtopology* σ l *sequentially*
using l
unfolding *sub.limit_metric_sequentially limit_metric_sequentially*
by (*force simp*: K_def)
ultimately have $l \in K \cap S$
by (*meson* $*$ σ *image_subsetI sub.metric_closedin_iff_sequentially_closed*)

then show *?thesis*
by *simp*
qed
then show *closedin mtopology* S
unfolding *metric_closedin_iff_sequentially_closed*
using ‹$S \subseteq M$› **by** *blast*
qed
qed

lemma (**in** *Metric_space*) *k_space_mtopology*: *k_space mtopology*
by (*simp add*: *metrizable_imp_k_space metrizable_space_mtopology*)

lemma *k_space_euclideanreal*: *k_space* (*euclidean* :: *'a*::*metric_space topology*)
using *metrizable_imp_k_space metrizable_space_euclidean* **by** *auto*

5.11.3 Hereditarily normal spaces

lemma *hereditarily_B*:
assumes $\bigwedge S\ T.$ *separatedin* X S T
$\Longrightarrow \exists U\ V.$ *openin* X $U \wedge$ *openin* X $V \wedge S \subseteq U \wedge T \subseteq V \wedge$ *disjnt* U V
shows *hereditarily normal_space* X
unfolding *hereditarily_def*
proof (*intro strip*)
fix W

assume $W \subseteq$ *topspace* X
show *normal_space* (*subtopology* X W)
unfolding *normal_space_def*
proof *clarify*
fix S T
assume *clo*: *closedin* (*subtopology* X W) S *closedin* (*subtopology* X W) T
and *disjnt* S T
then have *separatedin* (*subtopology* X W) S T
by (*simp add*: *separatedin_closed_sets*)
then obtain U V **where** *openin* X U $\wedge$ *openin* X V $\wedge$ $S \subseteq U \wedge T \subseteq V \wedge$ *disjnt* U V
using *assms* [*of* S T] **by** (*meson separatedin_subtopology*)
then show $\exists U\ V.$ *openin* (*subtopology* X W) $U \wedge$ *openin* (*subtopology* X W) $V \wedge S \subseteq U \wedge T \subseteq V \wedge$ *disjnt* U V
apply (*simp add*: *openin_subtopology_alt*)
by (*meson clo closedin_imp_subset disjnt_subset1 disjnt_subset2 inf_le2*)
qed
qed

lemma *hereditarily_C*:
assumes *separatedin* X S T **and** *norm*: $\bigwedge U.$ *openin* X $U \Longrightarrow$ *normal_space* (*subtopology* X U)
shows $\exists U\ V.$ *openin* X $U \wedge$ *openin* X $V \wedge S \subseteq U \wedge T \subseteq V \wedge$ *disjnt* U V
proof −
define ST **where** $ST \equiv X$ *closure_of* $S \cap X$ *closure_of* T
have *subX*: $S \subseteq$ *topspace* X $T \subseteq$ *topspace* X
by (*meson* ‹*separatedin* X S T› *separation_closedin_Un_gen*)+
have *sub*: $S \subseteq$ *topspace* $X - ST$ $T \subseteq$ *topspace* $X - ST$
unfolding *ST_def*
by (*metis Diff_mono Diff_triv* ‹*separatedin* X S T› *Int_lower1 Int_lower2 separatedin_def*)+
have *normal_space* (*subtopology* X (*topspace* $X - ST$))
by (*simp add*: *ST_def norm closedin_Int openin_diff*)
moreover have *disjnt* (*subtopology* X (*topspace* $X - ST$) *closure_of* S) (*subtopology* X (*topspace* $X - ST$) *closure_of* T)
using *Int_absorb1 ST_def sub* **by** (*fastforce simp*: *disjnt_iff closure_of_subtopology*)
ultimately show *?thesis*
using *sub subX*
apply (*simp add*: *normal_space_closures*)
by (*metis ST_def closedin_Int closedin_closure_of closedin_def openin_trans_full*)
qed

lemma *hereditarily_normal_space*:
hereditarily normal_space $X \longleftrightarrow$ ($\forall U.$ *openin* X $U \longrightarrow$ *normal_space*(*subtopology* X U))
by (*metis hereditarily_B hereditarily_C hereditarily*)

lemma *hereditarily_normal_separation*:
hereditarily normal_space $X \longleftrightarrow$

(∀ *S T*. *separatedin X S T*
⟶ (∃ *U V*. *openin X U* ∧ *openin X V* ∧ *S* ⊆ *U* ∧ *T* ⊆ *V* ∧ *disjnt U V*))
by (*metis hereditarily_B hereditarily_C hereditarily*)

lemma *metrizable_imp_hereditarily_normal_space*:
metrizable_space X ⟹ *hereditarily normal_space X*
by (*simp add*: *hereditarily metrizable_imp_normal_space metrizable_space_subtopology*)

lemma *metrizable_space_separation*:
⟦*metrizable_space X*; *separatedin X S T*⟧
⟹ ∃ *U V*. *openin X U* ∧ *openin X V* ∧ *S* ⊆ *U* ∧ *T* ⊆ *V* ∧ *disjnt U V*
by (*metis hereditarily hereditarily_C metrizable_imp_hereditarily_normal_space*)

lemma *hereditarily_normal_separation_pairwise*:
hereditarily normal_space X ⟷
(∀ *U*. *finite U* ∧ (∀ *S* ∈ *U*. *S* ⊆ *topspace X*) ∧ *pairwise* (*separatedin X*) *U*
⟶ (∃ *F*. (∀ *S* ∈ *U*. *openin X* (*F S*) ∧ *S* ⊆ *F S*) ∧
pairwise (λ*S T*. *disjnt* (*F S*) (*F T*)) *U*))
(**is** *?lhs* ⟷ *?rhs*)
proof
assume *L*: *?lhs*
show *?rhs*
proof *clarify*
fix *U*
assume *finite U* **and** *U*: ∀ *S*∈*U*. *S* ⊆ *topspace X*
and *pwU*: *pairwise* (*separatedin X*) *U*
have ∃ *V W*. *openin X V* ∧ *openin X W* ∧ *S* ⊆ *V* ∧
(∀ *T*. *T* ∈ *U* ∧ *T* ≠ *S* ⟶ *T* ⊆ *W*) ∧ *disjnt V W*
if *S* ∈ *U* **for** *S*
proof −
have *separatedin X S* (⋃ (*U* − {*S*}))
by (*metis U* ‹*finite U*› *pwU finite_Diff pairwise_alt separatedin_Union*(*1*) *that*)
with *L* **show** *?thesis*
unfolding *hereditarily_normal_separation*
by (*smt* (*verit*) *Diff_iff UnionI empty_iff insert_iff subset_iff*)
qed
then obtain *F G*
where ∗: ⋀*S*. *S* ∈ *U* ⟹ *S* ⊆ *F S* ∧ (∀ *T*. *T* ∈ *U* ∧ *T* ≠ *S* ⟶ *T* ⊆ *G S*)
and *ope*: ⋀*S*. *S* ∈ *U* ⟹ *openin X* (*F S*) ∧ *openin X* (*G S*)
and *dis*: ⋀*S*. *S* ∈ *U* ⟹ *disjnt* (*F S*) (*G S*)
by *metis*
define *H* **where** *H* ≡ λ*S*. *F S* ∩ (⋂ *T* ∈ *U* − {*S*}. *G T*)
show ∃ *F*. (∀ *S*∈*U*. *openin X* (*F S*) ∧ *S* ⊆ *F S*) ∧ *pairwise* (λ*S T*. *disjnt* (*F S*) (*F T*)) *U*
proof (*intro exI conjI strip*)
show *openin X* (*H S*) **if** *S* ∈ *U* **for** *S*

```
      unfolding ℋ_def
      by (smt (verit) ope that DiffD1 ‹finite 𝒰› finite_Diff finite_imageI imageE
openin_Int_Inter)
    show S ⊆ ℋ S if S ∈ 𝒰 for S
      unfolding ℋ_def using ∗ that by auto
  show pairwise (λS T. disjnt (ℋ S) (ℋ T)) 𝒰
    using dis by (fastforce simp: disjnt_iff pairwise_alt ℋ_def)
  qed
 qed
next
 assume R: ?rhs
 show ?lhs
  unfolding hereditarily_normal_separation
 proof (intro strip)
  fix S T
  assume separatedin X S T
  show ∃ U V. openin X U ∧ openin X V ∧ S ⊆ U ∧ T ⊆ V ∧ disjnt U V
  proof (cases T=S)
   case True
   then show ?thesis
    using ‹separatedin X S T› by force
  next
   case False
   have pairwise (separatedin X) {S, T}
    by (simp add: ‹separatedin X S T› pairwise_insert separatedin_sym)
   moreover have ∀ S∈{S, T}. S ⊆ topspace X
    by (metis ‹separatedin X S T› insertE separatedin_def singletonD)
    ultimately show ?thesis
    using R by (smt (verit) False finite.emptyI finite.insertI insertCI pairwiseD)
  qed
 qed
qed

lemma hereditarily_normal_space_perfect_map_image:
 ⟦hereditarily normal_space X; perfect_map X Y f⟧ ⟹ hereditarily normal_space
Y
 unfolding perfect_map_def proper_map_def
 by (meson hereditarily_normal_space_continuous_closed_map_image)

lemma regular_second_countable_imp_hereditarily_normal_space:
 assumes regular_space X ∧ second_countable X
 shows  hereditarily normal_space X
 unfolding hereditarily
 proof (intro regular_Lindelof_imp_normal_space strip)
 show regular_space (subtopology X S) for S
  by (simp add: assms regular_space_subtopology)
 show Lindelof_space (subtopology X S) for S
  using assms by (simp add: second_countable_imp_Lindelof_space second_countable_subtopology)
qed
```

5.11.4 Completely regular spaces

definition *completely_regular_space* **where**
completely_regular_space X ≡
∀ *S x*. *closedin X S* ∧ *x* ∈ *topspace X* − *S*
⟶ (∃ *f*::'*a*⇒*real*. *continuous_map X* (*top_of_set* {*0*..*1*}) *f* ∧
f x = *0* ∧ (*f* ‘ *S* ⊆ {*1*}))

lemma *homeomorphic_completely_regular_space_aux*:
assumes *X*: *completely_regular_space X* **and** *hom*: *X homeomorphic_space Y*
shows *completely_regular_space Y*
proof −
obtain *f g* **where** *hmf*: *homeomorphic_map X Y f* **and** *hmg*: *homeomorphic_map Y X g*
and *fg*: (∀ *x* ∈ *topspace X*. *g*(*f x*) = *x*) ∧ (∀ *y* ∈ *topspace Y*. *f*(*g y*) = *y*)
using *assms X homeomorphic_maps_map homeomorphic_space_def* **by** *fastforce*
show *?thesis*
unfolding *completely_regular_space_def*
proof *clarify*
fix *S x*
assume *A*: *closedin Y S* **and** *x*: *x* ∈ *topspace Y* **and** *x* ∉ *S*
then have *closedin X* (*g*‘*S*)
using *hmg homeomorphic_map_closedness_eq* **by** *blast*
moreover have *g x* ∉ *g*‘*S*
by (*meson A x* ‹*x* ∉ *S*› *closedin_subset hmg homeomorphic_imp_injective_map inj_on_image_mem_iff*)
ultimately obtain *φ* **where** *φ*: *continuous_map X* (*top_of_set* {*0*..*1*::*real*}) *φ* ∧ *φ* (*g x*) = *0* ∧ *φ* ‘ *g*‘*S* ⊆ {*1*}
by (*metis DiffI X completely_regular_space_def hmg homeomorphic_imp_surjective_map image_eqI x*)
then have *continuous_map Y* (*top_of_set* {*0*..*1*::*real*}) (*φ* ∘ *g*)
by (*meson continuous_map_compose hmg homeomorphic_imp_continuous_map*)
then show ∃ *ψ*. *continuous_map Y* (*top_of_set* {*0*..*1*::*real*}) *ψ* ∧ *ψ x* = *0* ∧ *ψ* ‘ *S* ⊆ {*1*}
by (*metis φ comp_apply image_comp*)
qed
qed

lemma *homeomorphic_completely_regular_space*:
assumes *X homeomorphic_space Y*
shows *completely_regular_space X* ⟷ *completely_regular_space Y*
by (*meson assms homeomorphic_completely_regular_space_aux homeomorphic_space_sym*)

lemma *completely_regular_space_alt*:
completely_regular_space X ⟷
(∀ *S x*. *closedin X S* ⟶ *x* ∈ *topspace X* − *S*
⟶ (∃ *f*. *continuous_map X euclideanreal f* ∧ *f x* = *0* ∧ *f* ‘ *S* ⊆ {*1*}))
proof −
have ∃ *f*. *continuous_map X* (*top_of_set* {*0*..*1*::*real*}) *f* ∧ *f x* = *0* ∧ *f* ‘ *S* ⊆

```
{1}
    if closedin X S x ∈ topspace X − S and f: continuous_map X euclideanreal f
∧ f x = 0 ∧ f ‘ S ⊆ {1}
    for S x f
  proof (intro exI conjI)
    show continuous_map X (top_of_set {0..1}) (λx. max 0 (min (f x) 1))
      using that
    by (auto simp: continuous_map_in_subtopology intro!: continuous_map_real_max
continuous_map_real_min)
  qed (use that in auto)
  with continuous_map_in_subtopology show ?thesis
    unfolding completely_regular_space_def by metis
qed
```

As above, but with *openin*

```
lemma completely_regular_space_alt':
   completely_regular_space X ⟷
     (∀ S x. openin X S ⟶ x ∈ S
           ⟶ (∃f. continuous_map X euclideanreal f ∧ f x = 0 ∧ f ‘ (topspace X
− S) ⊆ {1}))
  apply (simp add: completely_regular_space_alt all_closedin)
  by (meson openin_subset subsetD)

lemma completely_regular_space_gen_alt:
  fixes a b::real
  assumes a ≠ b
  shows completely_regular_space X ⟷
        (∀ S x. closedin X S ⟶ x ∈ topspace X − S
              ⟶ (∃f. continuous_map X euclidean f ∧ f x = a ∧ (f ‘ S ⊆ {b})))
proof −
  have ∃f. continuous_map X euclideanreal f ∧ f x = 0 ∧ f ‘ S ⊆ {1}
    if closedin X S x ∈ topspace X − S
       and f: continuous_map X euclidean f ∧ f x = a ∧ f ‘ S ⊆ {b}
    for S x f
  proof (intro exI conjI)
    show continuous_map X euclideanreal ((λx. inverse(b − a) ∗ (x − a)) ∘ f)
      using that by (intro continuous_intros) auto
  qed (use that assms in auto)
  moreover
  have ∃f. continuous_map X euclidean f ∧ f x = a ∧ f ‘ S ⊆ {b}
    if closedin X S x ∈ topspace X − S
       and f: continuous_map X euclideanreal f ∧ f x = 0 ∧ f ‘ S ⊆ {1}
    for S x f
  proof (intro exI conjI)
    show continuous_map X euclideanreal ((λx. a + (b − a) ∗ x) ∘ f)
      using that by (intro continuous_intros) auto
  qed (use that in auto)
  ultimately show ?thesis
    unfolding completely_regular_space_alt by meson
```

qed

As above, but with *openin*

lemma *completely_regular_space_gen_alt'*:
fixes *a b*::*real*
assumes $a \neq b$
shows *completely_regular_space X* ⟷
(∀ *S x*. *openin X S* ⟶ $x \in S$
⟶ (∃ *f*. *continuous_map X euclidean f* ∧ $f\ x = a$ ∧ (f ‘ (*topspace X* − *S*) ⊆ {*b*})))
apply (*simp add*: *completely_regular_space_gen_alt*[*OF assms*] *all_closedin*)
by (*meson openin_subset subsetD*)

lemma *completely_regular_space_gen*:
fixes *a b*::*real*
assumes $a < b$
shows *completely_regular_space X* ⟷
(∀ *S x*. *closedin X S* ∧ $x \in$ *topspace X* − *S*
⟶ (∃ *f*. *continuous_map X* (*top_of_set* {*a*..*b*}) *f* ∧ $f\ x = a$ ∧ f ‘ *S* ⊆ {*b*}))
proof −
have ∃ *f*. *continuous_map X* (*top_of_set* {*a*..*b*}) *f* ∧ $f\ x = a$ ∧ f ‘ *S* ⊆ {*b*}
if *closedin X S x* ∈ *topspace X* − *S*
and *f*: *continuous_map X euclidean f* ∧ $f\ x = a$ ∧ f ‘ *S* ⊆ {*b*}
for *S x f*
proof (*intro exI conjI*)
show *continuous_map X* (*top_of_set* {*a*..*b*}) (λ*x*. *max a* (*min* (*f x*) *b*))
using *that assms*
by (*auto simp*: *continuous_map_in_subtopology intro*!: *continuous_map_real_max continuous_map_real_min*)
qed (*use that assms* **in** *auto*)
with *continuous_map_in_subtopology assms* **show** *?thesis*
using *completely_regular_space_gen_alt* [*of a b*]
by (*smt* (*verit*) *atLeastAtMost_singleton atLeastatMost_empty singletonI*)
qed

lemma *normal_imp_completely_regular_space_A*:
assumes *normal_space X t1_space X*
shows *completely_regular_space X*
unfolding *completely_regular_space_alt*
proof *clarify*
fix *x S*
assume *A*: *closedin X S x* ∉ *S*
assume $x \in$ *topspace X*
then have *closedin X* {*x*}
by (*simp add*: ‹*t1_space X*› *closedin_t1_singleton*)
with *A* ‹*normal_space X*› **have** ∃ *f*. *continuous_map X euclideanreal f* ∧ f ‘ {*x*} ⊆ {*0*} ∧ f ‘ *S* ⊆ {*1*}
using *assms* **unfolding** *normal_space_iff_Urysohn_alt disjnt_iff* **by** *blast*

then show $\exists f.$ *continuous_map X euclideanreal* $f \wedge f\, x = 0 \wedge f\, `\, S \subseteq \{1\}$
by *auto*
qed

lemma *normal_imp_completely_regular_space_B*:
assumes *normal_space X regular_space X*
shows *completely_regular_space X*
unfolding *completely_regular_space_alt*
proof *clarify*
fix *x S*
assume *closedin X S x* $\notin$ *S x* $\in$ *topspace X*
then obtain *U C* **where** *openin X U closedin X C x* $\in$ *U U* $\subseteq$ *C C* $\subseteq$ *topspace X* $-$ *S*
using *assms*
unfolding *neighbourhood_base_of_closedin* [*symmetric*] *neighbourhood_base_of closedin_def* **by** (*metis Diff_iff*)
then obtain *f* **where** *continuous_map X euclideanreal f* $\wedge$ *f* ` *C* $\subseteq$ {*0*} $\wedge$ *f* ` *S* $\subseteq$ {*1*}
using *assms* **unfolding** *normal_space_iff_Urysohn_alt*
by (*metis Diff_iff* ‹*closedin X S*› *disjnt_iff subsetD*)
then show $\exists f.$ *continuous_map X euclideanreal* $f \wedge f\, x = 0 \wedge f\, `\, S \subseteq \{1\}$
by (*meson* ‹*U* $\subseteq$ *C*› ‹*x* $\in$ *U*› *image_subset_iff singletonD subsetD*)
qed

lemma *normal_imp_completely_regular_space_gen*:
⟦*normal_space X*; *t1_space X* $\vee$ *Hausdorff_space X* $\vee$ *regular_space X*⟧ $\Longrightarrow$ *completely_regular_space X*
using *normal_imp_completely_regular_space_A normal_imp_completely_regular_space_B t1_or_Hausdorff_space* **by** *blast*

lemma *normal_imp_completely_regular_space*:
⟦*normal_space X*; *Hausdorff_space X* $\vee$ *regular_space X*⟧ $\Longrightarrow$ *completely_regular_space X*
by (*simp add*: *normal_imp_completely_regular_space_gen*)

lemma (**in** *Metric_space*) *completely_regular_space_mtopology*:
completely_regular_space mtopology
by (*simp add*: *normal_imp_completely_regular_space normal_space_mtopology regular_space_mtopology*)

lemma *metrizable_imp_completely_regular_space*:
metrizable_space X $\Longrightarrow$ *completely_regular_space X*
by (*simp add*: *metrizable_imp_normal_space metrizable_imp_regular_space normal_imp_completely_regular_space*)

lemma *completely_regular_space_discrete_topology*:
completely_regular_space(*discrete_topology U*)
by (*simp add*: *normal_imp_completely_regular_space normal_space_discrete_topology*)

lemma *completely_regular_space_subtopology*:
assumes *completely_regular_space X*
shows *completely_regular_space* (*subtopology X S*)
unfolding *completely_regular_space_def*
proof *clarify*
fix *A x*
assume *closedin* (*subtopology X S*) *A* **and** *x*: *x* ∈ *topspace* (*subtopology X S*) **and** *x* ∉ *A*
then obtain *T* **where** *closedin X T A* = *S* ∩ *T x* ∈ *topspace X x* ∈ *S*
by (*force simp*: *closedin_subtopology_alt image_iff*)
then show ∃*f*. *continuous_map* (*subtopology X S*) (*top_of_set* {*0*::*real*..*1*}) *f* ∧ *f x* = *0* ∧ *f* ‘ *A* ⊆ {*1*}
using *assms* ‹*x* ∉ *A*›
apply (*simp add*: *completely_regular_space_def continuous_map_from_subtopology*)
using *continuous_map_from_subtopology* **by** *fastforce*
qed

lemma *completely_regular_space_retraction_map_image*:
⟦*retraction_map X Y r*; *completely_regular_space X*⟧ ⟹ *completely_regular_space Y*
using *completely_regular_space_subtopology hereditary_imp_retractive_property homeomorphic_completely_regular_space* **by** *blast*

lemma *completely_regular_imp_regular_space*:
assumes *completely_regular_space X*
shows *regular_space X*
proof −
have ∗: ∃ *U V*. *openin X U* ∧ *openin X V* ∧ *a* ∈ *U* ∧ *C* ⊆ *V* ∧ *disjnt U V*
if *contf*: *continuous_map X euclideanreal f* **and** *a*: *a* ∈ *topspace X* − *C* **and** *closedin X C*
and *fim*: *f* ‘ *topspace X* ⊆ {*0*..*1*} **and** *f0*: *f a* = *0* **and** *f1*: *f* ‘ *C* ⊆ {*1*}
for *C a f*
proof (*intro exI conjI*)
show *openin X* {*x* ∈ *topspace X*. *f x* ∈ {..<*1* / *2*}} *openin X* {*x* ∈ *topspace X*. *f x* ∈ {*1* / *2*<..}}
using *openin_continuous_map_preimage* [*OF contf*]
by (*meson open_lessThan open_greaterThan open_openin*)+
show *a* ∈ {*x* ∈ *topspace X*. *f x* ∈ {..<*1* / *2*}}
using *a f0* **by** *auto*
show *C* ⊆ {*x* ∈ *topspace X*. *f x* ∈ {*1* / *2*<..}}
using ‹*closedin X C*› *f1 closedin_subset* **by** *auto*
qed (*auto simp*: *disjnt_iff*)
show *?thesis*
using *assms*
unfolding *completely_regular_space_def regular_space_def continuous_map_in_subtopology*
by (*meson* ∗)
qed

proposition *locally_compact_regular_imp_completely_regular_space*:
assumes *locally_compact_space X Hausdorff_space X ∨ regular_space X*
shows *completely_regular_space X*
unfolding *completely_regular_space_def*
proof *clarify*
fix *S x*
assume *closedin X S* **and** *x ∈ topspace X* **and** *x ∉ S*
have *regular_space X*
using *assms locally_compact_Hausdorff_imp_regular_space* **by** *blast*
then have *nbase*: *neighbourhood_base_of* (*λC. compactin X C ∧ closedin X C*) *X*
using *assms*(*1*) *locally_compact_regular_space_neighbourhood_base* **by** *blast*
then obtain *U M* **where** *openin X U compactin X M closedin X M x ∈ U U ⊆ M M ⊆ topspace X − S*
unfolding *neighbourhood_base_of* **by** (*metis* (*no_types*, *lifting*) *Diff_iff* ‹*closedin X S*› ‹*x ∈ topspace X*› ‹*x ∉ S*› *closedin_def*)
then have *M ⊆ topspace X*
by *blast*
obtain *V K* **where** *openin X V closedin X K x ∈ V V ⊆ K K ⊆ U*
by (*metis* (*no_types*, *lifting*) ‹*openin X U*› ‹*x ∈ U*› *neighbourhood_base_of nbase*)
have *compact_space* (*subtopology X M*)
by (*simp add*: ‹*compactin X M*› *compact_space_subtopology*)
then have *normal_space* (*subtopology X M*)
by (*simp add*: ‹*regular_space X*› *compact_Hausdorff_or_regular_imp_normal_space regular_space_subtopology*)
moreover have *closedin* (*subtopology X M*) *K*
using ‹*K ⊆ U*› ‹*U ⊆ M*› ‹*closedin X K*› *closedin_subset_topspace* **by** *fastforce*
moreover have *closedin* (*subtopology X M*) (*M − U*)
by (*simp add*: ‹*closedin X M*› ‹*openin X U*› *closedin_diff closedin_subset_topspace*)
moreover have *disjnt K* (*M − U*)
by (*meson DiffD2* ‹*K ⊆ U*› *disjnt_iff subsetD*)
ultimately obtain *f*::*'a⇒real* **where** *contf*: *continuous_map* (*subtopology X M*) (*top_of_set* {*0..1*}) *f*
and *f0*: *f ‘ K ⊆ {0}* **and** *f1*: *f ‘ (M − U) ⊆ {1}*
using *Urysohn_lemma* [*of subtopology X M K M−U 0 1*] **by** *auto*
then obtain *g*::*'a⇒real* **where** *contg*: *continuous_map* (*subtopology X M*) *euclidean g* **and** *gim*: *g ‘ M ⊆ {0..1}*
and *g0*: *⋀x. x ∈ K ⟹ g x = 0* **and** *g1*: *⋀x. ⟦x ∈ M; x ∉ U⟧ ⟹ g x = 1*
using ‹*M ⊆ topspace X*› **by** (*force simp*: *continuous_map_in_subtopology image_subset_iff*)
show *∃f*::*'a⇒real. continuous_map X* (*top_of_set* {*0..1*}) *f ∧ f x = 0 ∧ f ‘ S ⊆ {1}*
proof (*intro exI conjI*)
show *continuous_map X* (*top_of_set* {*0..1*}) (*λx. if x ∈ M then g x else 1*)
unfolding *continuous_map_closedin*
proof (*intro strip conjI*)
fix *C*
assume *C*: *closedin* (*top_of_set* {*0*::*real..1*}) *C*

```
    have eq: {x ∈ topspace X. (if x ∈ M then g x else 1) ∈ C} = {x ∈ M. g x ∈
C} ∪ (if 1 ∈ C then topspace X − U else {})
      using ‹U ⊆ M› ‹M ⊆ topspace X› g1 by auto
    show closedin X {x ∈ topspace X. (if x ∈ M then g x else 1) ∈ C}
      unfolding eq
    proof (intro closedin_Un)
      have closedin euclidean C
        using C closed_closedin closedin_closed_trans by blast
      then have closedin (subtopology X M) {x ∈ M. g x ∈ C}
       using closedin_continuous_map_preimage_gen [OF contg] ‹M ⊆ topspace
X› by auto
      then show closedin X {x ∈ M. g x ∈ C}
        using ‹closedin X M› closedin_trans_full by blast
    qed (use ‹openin X U› in force)
  qed (use gim in force)
  show (if x ∈ M then g x else 1) = 0
    using ‹U ⊆ M› ‹V ⊆ K› g0 ‹x ∈ U› ‹x ∈ V› by auto
  show (λx. if x ∈ M then g x else 1) ‘ S ⊆ {1}
    using ‹M ⊆ topspace X − S› by auto
 qed
qed

lemma completely_regular_eq_regular_space:
  locally_compact_space X
      ⟹ (completely_regular_space X ⟷ regular_space X)
 using completely_regular_imp_regular_space locally_compact_regular_imp_completely_regular_space

 by blast

lemma completely_regular_space_prod_topology:
  completely_regular_space (prod_topology X Y) ⟷
    (prod_topology X Y) = trivial_topology ∨
    completely_regular_space X ∧ completely_regular_space Y (is ?lhs=?rhs)
proof
 assume ?lhs then show ?rhs
   by (rule topological_property_of_prod_component)
    (auto simp: completely_regular_space_subtopology homeomorphic_completely_regular_space)
next
 assume R: ?rhs
 show ?lhs
 proof (cases (prod_topology X Y) = trivial_topology)
   case False
   then have X: completely_regular_space X and Y: completely_regular_space
Y
     using R by blast+
   show ?thesis
     unfolding completely_regular_space_alt'
   proof clarify
     fix W x y
```

```
      assume openin (prod_topology X Y) W and (x, y) ∈ W
      then obtain U V where openin X U openin Y V x ∈ U y ∈ V U×V ⊆ W
        by (force simp: openin_prod_topology_alt)
      then have x ∈ topspace X y ∈ topspace Y
        using openin_subset by fastforce+
      obtain f where contf: continuous_map X euclideanreal f and f x = 0
        and f1: ⋀x. x ∈ topspace X ⟹ x ∉ U ⟹ f x = 1
        using X ‹openin X U› ‹x ∈ U› unfolding completely_regular_space_alt'
        by (smt (verit, best) Diff_iff image_subset_iff singletonD)
      obtain g where contg: continuous_map Y euclideanreal g and g y = 0
        and g1: ⋀y. y ∈ topspace Y ⟹ y ∉ V ⟹ g y = 1
        using Y ‹openin Y V› ‹y ∈ V› unfolding completely_regular_space_alt'
        by (smt (verit, best) Diff_iff image_subset_iff singletonD)
      define h where h ≡ λ(x,y). 1 − (1 − f x) ∗ (1 − g y)
      show ∃h. continuous_map (prod_topology X Y) euclideanreal h ∧ h (x,y) =
0 ∧ h ‘ (topspace (prod_topology X Y) − W) ⊆ {1}
      proof (intro exI conjI)
        have continuous_map (prod_topology X Y) euclideanreal (f ∘ fst)
          using contf continuous_map_of_fst by blast
        moreover
        have continuous_map (prod_topology X Y) euclideanreal (g ∘ snd)
          using contg continuous_map_of_snd by blast
        ultimately
        show continuous_map (prod_topology X Y) euclideanreal h
          unfolding o_def h_def case_prod_unfold
          by (intro continuous_intros) auto
        show h (x, y) = 0
          by (simp add: h_def ‹f x = 0› ‹g y = 0›)
        show h ‘ (topspace (prod_topology X Y) − W) ⊆ {1}
          using ‹U × V ⊆ W› f1 g1 by (force simp: h_def)
      qed
    qed
  qed (force simp: completely_regular_space_def)
qed


proposition completely_regular_space_product_topology:
   completely_regular_space (product_topology X I) ⟷
    (∃i∈I. X i = trivial_topology) ∨ (∀i ∈ I. completely_regular_space (X i))
   (is ?lhs ⟷ ?rhs)
proof
  assume ?lhs then show ?rhs
    by (rule topological_property_of_product_component)
     (auto simp: completely_regular_space_subtopology homeomorphic_completely_regular_space)
next
  assume R: ?rhs
  show ?lhs
  proof (cases ∃i∈I. X i = trivial_topology)
    case False
```

```
    show ?thesis
      unfolding completely_regular_space_alt'
    proof clarify
      fix W x
      assume W: openin (product_topology X I) W and x ∈ W
      then obtain U where finU: finite {i ∈ I. U i ≠ topspace (X i)}
            and ope: ⋀i. i∈I ⟹ openin (X i) (U i)
            and x: x ∈ Pi_E I U and Pi_E I U ⊆ W
        by (auto simp: openin_product_topology_alt)
      have ∀ i ∈ I. openin (X i) (U i) ∧ x i ∈ U i
            ⟶ (∃f. continuous_map (X i) euclideanreal f ∧
                  f (x i) = 0 ∧ (∀ x ∈ topspace(X i). x ∉ U i ⟶ f x = 1))
        using R unfolding completely_regular_space_alt'
        by (smt (verit) DiffI False image_subset_iff singletonD)
      with ope x have ⋀i. ∃f. i ∈ I ⟶ continuous_map (X i) euclideanreal f ∧
            f (x i) = 0 ∧ (∀ x ∈ topspace (X i). x ∉ U i ⟶ f x = 1)
        by auto
      then obtain f where f: ⋀i. i ∈ I ⟹ continuous_map (X i) euclideanreal
(f i) ∧
                                    f i (x i) = 0 ∧ (∀ x ∈ topspace (X i). x ∉ U i
⟶ f i x = 1)
        by metis
      define h where h ≡ λz. 1 − prod (λi. 1 − f i (z i)) {i∈I. U i ≠ topspace(X
i)}
      show ∃ h. continuous_map (product_topology X I) euclideanreal h ∧ h x = 0
∧
                  h ‘ (topspace (product_topology X I) − W) ⊆ {1}
      proof (intro conjI exI)
        have continuous_map (product_topology X I) euclidean (f i ∘ (λx. x i)) if
i∈I for i
          using f that
          by (blast intro: continuous_intros continuous_map_product_projection)
        then show continuous_map (product_topology X I) euclideanreal h
          unfolding h_def o_def by (intro continuous_intros) (auto simp: finU)
        show h x = 0
          by (simp add: h_def f)
        show h ‘ (topspace (product_topology X I) − W) ⊆ {1}
          proof −
          have ∃ i. i ∈ I ∧ U i ≠ topspace (X i) ∧ f i (x' i) = 1
            if x' ∈ (Π_E i∈I. topspace (X i)) x' ∉ W for x'
            using that ‹Pi_E I U ⊆ W› by (smt (verit, best) PiE_iff f in_mono)
          then show ?thesis
            by (auto simp: f h_def finU)
        qed
      qed
    qed
  qed (force simp: completely_regular_space_def)
qed
```

lemma *zero_dimensional_imp_completely_regular_space*:
assumes *X dim_le 0*
shows *completely_regular_space X*
proof (*clarsimp simp*: *completely_regular_space_def*)
fix *C a*
assume *closedin X C* **and** *a* ∈ *topspace X* **and** *a* ∉ *C*
then obtain *U* **where** *closedin X U openin X U a* ∈ *U* **and** *U*: *U* ⊆ *topspace X* − *C*
using *assms* **by** (*force simp add*: *closedin_def dimension_le_0_neighbourhood_base_of_clopen open_neighbourhood_base_of*)
show ∃*f*. *continuous_map X* (*top_of_set* {*0*::*real*..*1*}) *f* ∧ *f a* = *0* ∧ *f* ‘ *C* ⊆ {*1*}
proof (*intro exI conjI*)
have *continuous_map X euclideanreal* (λ*x*. *if x* ∈ *U then 0 else 1*)
using ‹*closedin X U*› ‹*openin X U*› **apply** (*clarsimp simp*: *continuous_map_def closedin_def*)
by (*smt* (*verit*) *Diff_iff mem_Collect_eq openin_subopen subset_eq*)
then show *continuous_map X* (*top_of_set* {*0*::*real*..*1*}) (λ*x*. *if x* ∈ *U then 0 else 1*)
by (*auto simp*: *continuous_map_in_subtopology*)
qed (*use U* ‹*a* ∈ *U*› **in** *auto*)
qed

lemma *zero_dimensional_imp_regular_space*: *X dim_le 0* ⟹ *regular_space X*
by (*simp add*: *completely_regular_imp_regular_space zero_dimensional_imp_completely_regular_spa*

lemma (**in** *Metric_space*) *t1_space_mtopology*:
t1_space mtopology
using *Hausdorff_space_mtopology t1_or_Hausdorff_space* **by** *blast*

5.11.5 More generally, the k-ification functor

definition *kification_open*
where *kification_open* ≡
λ*X S*. *S* ⊆ *topspace X* ∧ (∀ *K*. *compactin X K* ⟶ *openin* (*subtopology X K*) (*K* ∩ *S*))

definition *kification*
where *kification X* ≡ *topology* (*kification_open X*)

lemma *istopology_kification_open*: *istopology* (*kification_open X*)
unfolding *istopology_def*
proof (*intro conjI strip*)
show *kification_open X* (*S* ∩ *T*)
if *kification_open X S* **and** *kification_open X T* **for** *S T*
using *that* **unfolding** *kification_open_def*
by (*smt* (*verit*, *best*) *inf.idem inf_commute inf_left_commute le_infI2 openin_Int*)
show *kification_open X* (⋃ *K*) **if** ∀ *K*∈*K*. *kification_open X K* **for** *K*

using *that* **unfolding** *kification_open_def Int_Union* **by** *blast*
qed

lemma *openin_kification*:
openin (*kification X*) *U* ⟷
U ⊆ *topspace X* ∧
(∀ *K*. *compactin X K* ⟶ *openin* (*subtopology X K*) (*K* ∩ *U*))
by (*metis topology_inverse' kification_def istopology_kification_open kification_open_def*)

lemma *openin_kification_finer*:
openin X S ⟹ *openin* (*kification X*) *S*
by (*simp add*: *openin_kification openin_subset openin_subtopology_Int2*)

lemma *topspace_kification* [*simp*]:
topspace(*kification X*) = *topspace X*
by (*meson openin_kification openin_kification_finer openin_topspace subset_antisym*)

lemma *closedin_kification*:
closedin (*kification X*) *U* ⟷
U ⊆ *topspace X* ∧
(∀ *K*. *compactin X K* ⟶ *closedin* (*subtopology X K*) (*K* ∩ *U*))
proof (*cases U* ⊆ *topspace X*)
case *True*
then show *?thesis*
by (*simp add*: *closedin_def Diff_Int_distrib inf_commute le_infI2 openin_kification*)
qed (*simp add*: *closedin_def*)

lemma *closedin_kification_finer*: *closedin X S* ⟹ *closedin* (*kification X*) *S*
by (*simp add*: *closedin_def openin_kification_finer*)

lemma *kification_eq_self*: *kification X* = *X* ⟷ *k_space X*
unfolding *fun_eq_iff openin_kification k_space_alt openin_inject* [*symmetric*]
by (*metis openin_closedin_eq*)

lemma *compactin_kification* [*simp*]:
compactin (*kification X*) *K* ⟷ *compactin X K* (**is** *?lhs* ⟷ *?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*simp add*: *compactin_contractive openin_kification_finer*)
next
assume *R*: *?rhs*
show *?lhs*
unfolding *compactin_def*
proof (*intro conjI strip*)
show *K* ⊆ *topspace* (*kification X*)
by (*simp add*: *R compactin_subset_topspace*)
fix *𝒰*
assume *𝒰*: *Ball 𝒰* (*openin* (*kification X*)) ∧ *K* ⊆ ⋃ *𝒰*
then have ∗: ⋀*U*. *U* ∈ *𝒰* ⟹ *U* ⊆ *topspace X* ∧ *openin* (*subtopology X K*)

```
(K ∩ U)
      by (simp add: R openin_kification)
    have K ⊆ topspace X compact_space (subtopology X K)
      using R compactin_subspace by force+
     then have ∃ V. finite V ∧ V ⊆ (λU. K ∩ U) ` 𝒰 ∧ ⋃ V = topspace
(subtopology X K)
      unfolding compact_space
     by (smt (verit, del_insts) Int_Union 𝒰 ∗ image_iff inf.order_iff inf_commute
topspace_subtopology)
    then show ∃ℱ. finite ℱ ∧ ℱ ⊆ 𝒰 ∧ K ⊆ ⋃ ℱ
       by (metis Int_Union ‹K ⊆ topspace X› finite_subset_image inf.orderI
topspace_subtopology_subset)
  qed
qed

lemma compact_space_kification [simp]:
   compact_space(kification X) ⟷ compact_space X
  by (simp add: compact_space_def)

lemma kification_kification [simp]:
   kification(kification X) = kification X
  unfolding openin_inject [symmetric]
proof
  fix U
  show openin (kification (kification X)) U = openin (kification X) U
  proof
    show openin (kification (kification X)) U ⟹ openin (kification X) U
    by (metis compactin_kification inf_commute openin_kification openin_subtopology
topspace_kification)
  qed (simp add: openin_kification_finer)
qed

lemma k_space_kification [iff]: k_space(kification X)
  using kification_eq_self by fastforce

lemma continuous_map_into_kification:
  assumes k_space X
  shows continuous_map X (kification Y) f ⟷ continuous_map X Y f (is ?lhs
⟷ ?rhs)
proof
  assume ?lhs then show ?rhs
    by (simp add: continuous_map_def openin_kification_finer)
next
  assume R: ?rhs
  have openin X {x ∈ topspace X. f x ∈ V} if V: openin (kification Y) V for V
  proof −
    have openin (subtopology X K) (K ∩ {x ∈ topspace X. f x ∈ V})
      if compactin X K for K
    proof −
```

```
    have compactin Y (f ‘ K)
      using R image_compactin that by blast
    then have openin (subtopology Y (f ‘ K)) (f ‘ K ∩ V)
      by (meson V openin_kification)
    then obtain U where U: openin Y U f‘K ∩ V = U ∩ f‘K
      by (meson openin_subtopology)
    show ?thesis
      unfolding openin_subtopology
    proof (intro conjI exI)
      show openin X {x ∈ topspace X. f x ∈ U}
        using R U openin_continuous_map_preimage_gen by (simp add: con-
tinuous_map_def)
    qed (use U in auto)
  qed
  then show ?thesis
    by (metis (full_types) Collect_subset assms k_space_open)
 qed
 with R show ?lhs
  by (simp add: continuous_map_def)
qed

lemma continuous_map_from_kification:
  continuous_map X Y f ⟹ continuous_map (kification X) Y f
 by (simp add: continuous_map_openin_preimage_eq openin_kification_finer)

lemma continuous_map_kification:
  continuous_map X Y f ⟹ continuous_map (kification X) (kification Y) f
 by (simp add: continuous_map_from_kification continuous_map_into_kification)

lemma subtopology_kification_compact:
 assumes compactin X K
 shows subtopology (kification X) K = subtopology X K
 unfolding openin_inject [symmetric]
proof
 fix U
 show openin (subtopology (kification X) K) U = openin (subtopology X K) U
  by (metis assms inf_commute openin_kification openin_subset openin_subtopology)
qed


lemma subtopology_kification_finer:
 assumes openin (subtopology (kification X) S) U
 shows openin (kification (subtopology X S)) U
 using assms
 by (fastforce simp: openin_subtopology_alt image_iff openin_kification subtopol-
ogy_subtopology compactin_subtopology)

lemma proper_map_from_kification:
 assumes k_space Y
```

shows *proper_map (kification X) Y f ⟷ proper_map X Y f* (**is** *?lhs ⟷ ?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*simp add*: *closed_map_def closedin_kification_finer proper_map_alt*)
next
assume *R*: *?rhs*
have *compactin Y K ⟹ compactin X {x ∈ topspace X. f x ∈ K}* **for** *K*
using *R proper_map_alt* **by** *auto*
with *R* **show** *?lhs*
by (*simp add*: *assms proper_map_into_k_space_eq subtopology_kification_compact*)
qed

lemma *perfect_map_from_kification*:
⟦*k_space Y*; *perfect_map X Y f*⟧ *⟹ perfect_map(kification X) Y f*
by (*simp add*: *continuous_map_from_kification perfect_map_def proper_map_from_kification*)

lemma *k_space_perfect_map_image_eq*:
assumes *Hausdorff_space X perfect_map X Y f*
shows *k_space X ⟷ k_space Y*
proof
show *k_space X ⟹ k_space Y*
using *k_space_perfect_map_image assms* **by** *blast*
assume *k_space Y*
have *homeomorphic_map (kification X) X id*
unfolding *homeomorphic_eq_injective_perfect_map*
proof (*intro conjI perfect_map_from_composition_right* [**where** *f = id*])
show *perfect_map (kification X) Y (f ∘ id)*
by (*simp add*: ‹*k_space Y*› *assms*(*2*) *perfect_map_from_kification*)
show *continuous_map (kification X) X id*
by (*simp add*: *continuous_map_from_kification*)
qed (*use assms perfect_map_def* **in** *auto*)
then show *k_space X*
using *homeomorphic_k_space homeomorphic_space* **by** *blast*
qed

5.11.6 One-point compactifications and the Alexandroff extension construction

lemma *one_point_compactification_dense*:
⟦*compact_space X*; ¬ *compactin X (topspace X − {a})*⟧ *⟹ X closure_of (topspace X − {a}) = topspace X*
unfolding *closure_of_complement*
by (*metis Diff_empty closedin_compact_space interior_of_eq_empty openin_closedin_eq subset_singletonD*)

lemma *one_point_compactification_interior*:
⟦*compact_space X*; ¬ *compactin X (topspace X − {a})*⟧ *⟹ X interior_of {a} = {}*

by (*simp add*: *interior_of_eq_empty_complement one_point_compactification_dense*)

proposition *kc_space_one_point_compactification_gen*:
assumes *compact_space X*
shows *kc_space X* ⟷
openin X (*topspace X* − {*a*}) ∧ (∀*K*. *compactin X K* ∧ *a*∉*K* ⟶ *closedin X K*) ∧
k_space (*subtopology X* (*topspace X* − {*a*})) ∧ *kc_space* (*subtopology X* (*topspace X* − {*a*}))
(**is** *?lhs* ⟷ *?rhs*)
proof
assume *L*: *?lhs* **show** *?rhs*
proof (*intro conjI strip*)
show *openin X* (*topspace X* − {*a*})
using *L kc_imp_t1_space t1_space_openin_delete_alt* **by** *auto*
then show *k_space* (*subtopology X* (*topspace X* − {*a*}))
by (*simp add*: *L assms k_space_open_subtopology_aux*)
show *closedin X k* **if** *compactin X k* ∧ *a* ∉ *k* **for** *k* :: *'a set*
using *L kc_space_def that* **by** *blast*
show *kc_space* (*subtopology X* (*topspace X* − {*a*}))
by (*simp add*: *L kc_space_subtopology*)
qed
next
assume *R*: *?rhs*
show *?lhs*
unfolding *kc_space_def*
proof (*intro strip*)
fix *S*
assume *compactin X S*
then have *S* ⊆*topspace X*
by (*simp add*: *compactin_subset_topspace*)
show *closedin X S*
proof (*cases a* ∈ *S*)
case *True*
then have *topspace X* − *S* = *topspace X* − {*a*} − (*S* − {*a*})
by *auto*
moreover have *openin X* (*topspace X* − {*a*} − (*S* − {*a*}))
proof (*rule openin_trans_full*)
show *openin* (*subtopology X* (*topspace X* − {*a*})) (*topspace X* − {*a*} − (*S* − {*a*}))
proof
show *openin* (*subtopology X* (*topspace X* − {*a*})) (*topspace X* − {*a*})
using *R openin_open_subtopology* **by** *blast*
have *closedin* (*subtopology X* ((*topspace X* − {*a*}) ∩ *K*)) (*K* ∩ (*S* − {*a*}))
if *compactin X K K* ⊆ *topspace X* − {*a*} **for** *K*
proof (*intro closedin_subset_topspace*)
show *closedin X* (*K* ∩ (*S* − {*a*}))
using *that*
by (*metis IntD1 Int_insert_right_if0 R True* ‹*compactin X S*›

```
closed_Int_compactin insert_Diff subset_Diff_insert)
        qed (use that in auto)
        moreover have k_space (subtopology X (topspace X − {a}))
          using R by blast
        moreover have S−{a} ⊆ topspace X ∧ S−{a} ⊆ topspace X − {a}
          using ‹S ⊆ topspace X› by auto
        ultimately show closedin (subtopology X (topspace X − {a})) (S − {a})
          using ‹S ⊆ topspace X› True
       by (simp add: k_space_def compactin_subtopology subtopology_subtopology)
       qed
       show openin X (topspace X − {a})
        by (simp add: R)
     qed
     ultimately show ?thesis
      by (simp add: ‹S ⊆ topspace X› closedin_def)
   next
     case False
     then show ?thesis
      by (simp add: R ‹compactin X S›)
   qed
  qed
qed


inductive Alexandroff_open for X where
  base: openin X U ⟹ Alexandroff_open X (Some ‘ U)
| ext: ⟦compactin X C; closedin X C⟧ ⟹ Alexandroff_open X (insert None (Some
‘ (topspace X − C)))

hide_fact (open) base ext

lemma Alexandroff_open_iff: Alexandroff_open X S ⟷
   (∃ U. (S = Some ‘ U ∧ openin X U) ∨ (S = insert None (Some ‘ (topspace X
− U)) ∧ compactin X U ∧ closedin X U))
  by (meson Alexandroff_open.cases Alexandroff_open.ext Alexandroff_open.base)

lemma Alexandroff_open_Un_aux:
  assumes U: openin X U and Alexandroff_open X T
  shows  Alexandroff_open X (Some ‘ U ∪ T)
  using ‹Alexandroff_open X T›
proof (induction rule: Alexandroff_open.induct)
  case (base V)
  then show ?case
    by (metis Alexandroff_open.base U image_Un openin_Un)
next
  case (ext C)
  have U ⊆ topspace X
    by (simp add: U openin_subset)
  then have eq: Some ‘ U ∪ insert None (Some ‘ (topspace X − C)) = insert
```

None (*Some* ‘ (*topspace X* − (*C* ∩ (*topspace X* − *U*))))
by *force*
have *closedin X* (*C* ∩ (*topspace X* − *U*))
using *U ext.hyps*(*2*) **by** *blast*
moreover
have *compactin X* (*C* ∩ (*topspace X* − *U*))
using *U compact_Int_closedin ext.hyps*(*1*) **by** *blast*
ultimately show *?case*
unfolding *eq* **using** *Alexandroff_open.ext* **by** *blast*
qed

lemma *Alexandroff_open_Un*:
assumes *Alexandroff_open X S* **and** *Alexandroff_open X T*
shows *Alexandroff_open X* (*S* ∪ *T*)
using *assms*
proof (*induction rule*: *Alexandroff_open.induct*)
case (*base U*)
then show *?case*
by (*simp add*: *Alexandroff_open_Un_aux*)
next
case (*ext C*)
then show *?case*
by (*smt* (*verit*, *best*) *Alexandroff_open_Un_aux Alexandroff_open_iff Un_commute Un_insert_left closedin_def insert_absorb2*)
qed

lemma *Alexandroff_open_Int_aux*:
assumes *U*: *openin X U* **and** *Alexandroff_open X T*
shows *Alexandroff_open X* (*Some* ‘ *U* ∩ *T*)
using ‹*Alexandroff_open X T*›
proof (*induction rule*: *Alexandroff_open.induct*)
case (*base V*)
then show *?case*
by (*metis Alexandroff_open.base U image_Int inj_Some openin_Int*)
next
case (*ext C*)
have *eq*: *Some* ‘ *U* ∩ *insert None* (*Some* ‘ (*topspace X* − *C*)) = *Some* ‘ (*topspace X* − (*C* ∪ (*topspace X* − *U*)))
by *force*
have *openin X* (*topspace X* − (*C* ∪ (*topspace X* − *U*)))
using *U ext.hyps*(*2*) **by** *blast*
then show *?case*
unfolding *eq* **using** *Alexandroff_open.base* **by** *blast*
qed

proposition *istopology_Alexandroff_open*: *istopology* (*Alexandroff_open X*)
unfolding *istopology_def*
proof (*intro conjI strip*)
fix *S T*

```
  assume Alexandroff_open X S and Alexandroff_open X T
  then show Alexandroff_open X (S ∩ T)
  proof (induction rule: Alexandroff_open.induct)
    case (base U)
    then show ?case
      using Alexandroff_open_Int_aux by blast
  next
    case EC: (ext C)
    show ?case
      using ‹Alexandroff_open X T›
    proof (induction rule: Alexandroff_open.induct)
      case (base V)
      then show ?case
      by (metis Alexandroff_open.ext Alexandroff_open_Int_aux EC.hyps inf_commute)
    next
      case (ext D)
       have eq: insert None (Some ‘ (topspace X − C)) ∩ insert None (Some ‘
(topspace X − D))
            = insert None (Some ‘ (topspace X − (C ∪ D)))
        by auto
      show ?case
        unfolding eq
        by (simp add: Alexandroff_open.ext EC.hyps closedin_Un compactin_Un
ext.hyps)
    qed
  qed
next
  fix 𝒦
  assume §: ∀K∈𝒦. Alexandroff_open X K
  show Alexandroff_open X (⋃𝒦)
  proof (cases None ∈ ⋃𝒦)
    case True
    have ∀K∈𝒦. ∃U. (openin X U ∧ K = Some ‘ U) ∨ (K = insert None (Some
‘ (topspace X − U)) ∧ compactin X U ∧ closedin X U)
      by (metis § Alexandroff_open_iff)
    then obtain U where U:
      ⋀K. K ∈ 𝒦 ⟹ openin X (U K) ∧ K = Some ‘ (U K)
                 ∨ (K = insert None (Some ‘ (topspace X − U K)) ∧ compactin X
(U K) ∧ closedin X (U K))
      by metis
    define 𝒦N where 𝒦N ≡ {K ∈ 𝒦. None ∈ K}
    define A where A ≡ ⋃K∈𝒦−𝒦N. U K
    define B where B ≡ ⋂K∈𝒦N. U K
    have U1: ⋀K. K ∈ 𝒦−𝒦N ⟹ openin X (U K) ∧ K = Some ‘ (U K)
      using U 𝒦N_def by auto
    have U2: ⋀K. K ∈ 𝒦N ⟹ K = insert None (Some ‘ (topspace X − U K))
∧ compactin X (U K) ∧ closedin X (U K)
      using U 𝒦N_def by auto
    have eqA: ⋃(𝒦−𝒦N) = Some ‘ A
```

proof
show $\bigcup\ (\mathcal{K} - \mathcal{K}N) \subseteq Some\ `\ A$
by (*metis A_def Sup_le_iff U1 UN_upper subset_image_iff*)
show $Some\ `\ A \subseteq \bigcup\ (\mathcal{K} - \mathcal{K}N)$
using *A_def U1* **by** *blast*
qed
have *eqB*: $\bigcup\mathcal{K}N = insert\ None\ (Some\ `\ (topspace\ X - B))$
using *U2 True*
by (*auto simp*: *B_def image_iff* $\mathcal{K}N$*_def*)
have $\bigcup\mathcal{K} = \bigcup\mathcal{K}N \cup \bigcup(\mathcal{K}-\mathcal{K}N)$
by (*auto simp*: $\mathcal{K}N$*_def*)
then have *eq*: $\bigcup\mathcal{K} = (Some\ `\ A) \cup (insert\ None\ (Some\ `\ (topspace\ X - B)))$
by (*simp add*: *eqA eqB Un_commute*)
show *?thesis*
unfolding *eq*
proof (*intro Alexandroff_open_Un Alexandroff_open.intros*)
show *openin X A*
using *A_def U1* **by** *blast*
show *closedin X B*
unfolding *B_def* **using** *U2 True* $\mathcal{K}N$*_def* **by** *auto*
show *compactin X B*
by (*metis B_def U2 eqB Inf_lower Union_iff* ‹*closedin X B*› *closed_compactin imageI insertI1*)
qed
next
case *False*
then have $\forall\, K{\in}\mathcal{K}.\ \exists\, U.\ openin\ X\ U \wedge K = Some\ `\ U$
by (*metis Alexandroff_open.simps UnionI* § *insertCI*)
then obtain *U* **where** *U*: $\forall\, K{\in}\mathcal{K}.\ openin\ X\ (U\ K) \wedge K = Some\ `\ (U\ K)$
by *metis*
then have *eq*: $\bigcup\mathcal{K} = Some\ `\ (\bigcup\ K{\in}\mathcal{K}.\ U\ K)$
using *image_iff* **by** *fastforce*
show *?thesis*
unfolding *eq* **by** (*simp add*: *U Alexandroff_open.base openin_clauses*(*3*))
qed
qed

definition *Alexandroff_compactification* **where**
Alexandroff_compactification X ≡ *topology* (*Alexandroff_open X*)

lemma *openin_Alexandroff_compactification*:
openin(*Alexandroff_compactification X*) *V* ⟷
(∃ *U*. *openin X U* ∧ *V* = *Some* ‘ *U*) ∨
(∃ *C*. *compactin X C* ∧ *closedin X C* ∧ *V* = *insert None* (*Some* ‘ (*topspace X* − *C*)))
by (*auto simp*: *Alexandroff_compactification_def istopology_Alexandroff_open Alexandroff_open.simps*)

lemma *topspace_Alexandroff_compactification* [*simp*]:
topspace(*Alexandroff_compactification X*) = *insert None* (*Some* ‘ *topspace X*)
(**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs*
by (*force simp*: *topspace_def openin_Alexandroff_compactification*)
have *None* ∈ *topspace* (*Alexandroff_compactification X*)
by (*meson closedin_empty compactin_empty insert_subset openin_Alexandroff_compactification openin_subset*)
moreover have *Some x* ∈ *topspace* (*Alexandroff_compactification X*)
if *x* ∈ *topspace X* **for** *x*
by (*meson that imageI openin_Alexandroff_compactification openin_subset openin_topspace subsetD*)
ultimately show *?rhs* ⊆ *?lhs*
by (*auto simp*: *image_subset_iff*)
qed

lemma *closedin_Alexandroff_compactification*:
closedin (*Alexandroff_compactification X*) *C* ⟷
(∃ *K*. *compactin X K* ∧ *closedin X K* ∧ *C* = *Some* ‘ *K*) ∨
(∃ *U*. *openin X U* ∧ *C* = *topspace*(*Alexandroff_compactification X*) − *Some* ‘ *U*)
(**is** *?lhs* ⟷ *?rhs*)
proof
show *?lhs* ⟹ *?rhs*
apply (*clarsimp simp*: *closedin_def openin_Alexandroff_compactification*)
by (*smt* (*verit*) *Diff_Diff_Int None_notin_image_Some image_set_diff inf.absorb_iff2 inj_Some insert_Diff_if subset_insert*)
show *?rhs* ⟹ *?lhs*
using *openin_subset*
by (*fastforce simp*: *closedin_def openin_Alexandroff_compactification*)
qed

lemma *openin_Alexandroff_compactification_image_Some* [*simp*]:
openin(*Alexandroff_compactification X*) (*Some* ‘ *U*) ⟷ *openin X U*
by (*auto simp*: *openin_Alexandroff_compactification inj_image_eq_iff*)

lemma *closedin_Alexandroff_compactification_image_Some* [*simp*]:
closedin (*Alexandroff_compactification X*) (*Some* ‘ *K*) ⟷ *compactin X K* ∧ *closedin X K*
by (*auto simp*: *closedin_Alexandroff_compactification inj_image_eq_iff*)

lemma *open_map_Some*: *open_map X* (*Alexandroff_compactification X*) *Some*
using *open_map_def openin_Alexandroff_compactification* **by** *blast*

lemma *continuous_map_Some*: *continuous_map X* (*Alexandroff_compactification X*) *Some*
unfolding *continuous_map_def*

proof (*intro conjI strip*)
fix *U*
assume *openin* (*Alexandroff_compactification X*) *U*
then consider *V* **where** *openin X V U* = *Some* ‘ *V*
| *C* **where** *compactin X C closedin X C U* = *insert None* (*Some* ‘ (*topspace X* − *C*))
by (*auto simp*: *openin_Alexandroff_compactification*)
then show *openin X* {*x* ∈ *topspace X*. *Some x* ∈ *U*}
proof *cases*
case *1*
then show *?thesis*
using *openin_subopen openin_subset* **by** *fastforce*
next
case *2*
then show *?thesis*
by (*simp add*: *closedin_def image_iff set_diff_eq*)
qed
qed *auto*

lemma *embedding_map_Some*: *embedding_map X* (*Alexandroff_compactification X*) *Some*
by (*simp add*: *continuous_map_Some injective_open_imp_embedding_map open_map_Some*)

lemma *compact_space_Alexandroff_compactification* [*simp*]:
compact_space(*Alexandroff_compactification X*)
proof (*clarsimp simp*: *compact_space_alt image_subset_iff*)
fix $\mathcal{U}$ *U*
assume *ope* [*rule_format*]: ∀ *U*∈$\mathcal{U}$. *openin* (*Alexandroff_compactification X*) *U*
and *cover*: ∀ *x*∈*topspace X*. ∃ *X*∈$\mathcal{U}$. *Some x* ∈ *X*
and *U* ∈ $\mathcal{U}$ *None* ∈ *U*
then have *Usub*: *U* ⊆ *insert None* (*Some* ‘ *topspace X*)
by (*metis openin_subset topspace_Alexandroff_compactification*)
with *ope* [*OF* ‹*U* ∈ $\mathcal{U}$›] ‹*None* ∈ *U*›
obtain *C* **where** *C*: *compactin X C* ∧ *closedin X C* ∧
insert None (*Some* ‘ *topspace X*) − *U* = *Some* ‘ *C*
by (*auto simp*: *openin_closedin closedin_Alexandroff_compactification*)
then have *D*: *compactin* (*Alexandroff_compactification X*) (*insert None* (*Some* ‘ *topspace X*) − *U*)
by (*metis continuous_map_Some image_compactin*)
consider *V* **where** *openin X V U* = *Some* ‘ *V*
| *C* **where** *compactin X C closedin X C U* = *insert None* (*Some* ‘ (*topspace X* − *C*))
using *ope* [*OF* ‹*U* ∈ $\mathcal{U}$›] **by** (*auto simp*: *openin_Alexandroff_compactification*)
then show ∃ $\mathcal{F}$. *finite* $\mathcal{F}$ ∧ $\mathcal{F}$ ⊆ $\mathcal{U}$ ∧ (∃ *X*∈$\mathcal{F}$. *None* ∈ *X*) ∧ (∀ *x*∈*topspace X*. ∃ *X*∈$\mathcal{F}$. *Some x* ∈ *X*)
proof *cases*
case *1*
then show *?thesis*

```
      using ‹None ∈ U› by blast
  next
    case 2
   obtain ℱ where finite ℱ ℱ ⊆ 𝒰 insert None (Some ‘ topspace X) − U ⊆ ⋃ℱ
    by (smt (verit, del_insts) C D Union_iff compactinD compactin_subset_topspace
cover image_subset_iff ope subsetD)
    with ‹U ∈ 𝒰› show ?thesis
      by (rule_tac x=insert U ℱ in exI) auto
  qed
qed

lemma topspace_Alexandroff_compactification_delete:
   topspace(Alexandroff_compactification X) − {None} = Some ‘ topspace X
  by simp

lemma Alexandroff_compactification_dense:
  assumes ¬ compact_space X
  shows (Alexandroff_compactification X) closure_of (Some ‘ topspace X) =
        topspace(Alexandroff_compactification X)
  unfolding topspace_Alexandroff_compactification_delete [symmetric]
proof (intro one_point_compactification_dense)
 show ¬ compactin (Alexandroff_compactification X) (topspace (Alexandroff_compactification
X) − {None})
   using assms compact_space_proper_map_preimage compact_space_subtopology
embedding_map_Some embedding_map_def homeomorphic_imp_proper_map by
fastforce
qed auto


lemma t0_space_one_point_compactification:
  assumes compact_space X ∧ openin X (topspace X − {a})
  shows t0_space X ⟷ t0_space (subtopology X (topspace X − {a}))
   (is ?lhs ⟷ ?rhs)
proof
  show ?lhs ⟹ ?rhs
    using t0_space_subtopology by blast
  show ?rhs ⟹ ?lhs
    using assms
   unfolding t0_space_def by (bestsimp simp flip: Int_Diff dest: openin_trans_full)
qed

lemma t0_space_Alexandroff_compactification [simp]:
   t0_space (Alexandroff_compactification X) ⟷ t0_space X
  using t0_space_one_point_compactification [of Alexandroff_compactification X
None]
  using embedding_map_Some embedding_map_imp_homeomorphic_space home-
omorphic_t0_space by fastforce

lemma t1_space_one_point_compactification:
```

assumes *Xa*: *openin X* (*topspace X* − {*a*})
and §: ⋀*K*. ⟦*compactin* (*subtopology X* (*topspace X* − {*a*})) *K*; *closedin* (*subtopology X* (*topspace X* − {*a*})) *K*⟧ ⟹ *closedin X K*
shows *t1_space X* ⟷ *t1_space* (*subtopology X* (*topspace X* − {*a*})) (**is** *?lhs* ⟷ *?rhs*)
proof
show *?lhs* ⟹ *?rhs*
using *t1_space_subtopology* **by** *blast*
assume *R*: *?rhs*
show *?lhs*
unfolding *t1_space_closedin_singleton*
proof (*intro strip*)
fix *x*
assume *x* ∈ *topspace X*
show *closedin X* {*x*}
proof (*cases x*=*a*)
case *True*
then show *?thesis*
using ‹*x* ∈ *topspace X*› *Xa closedin_def* **by** *blast*
next
case *False*
show *?thesis*
by (*simp add*: § *False R* ‹*x* ∈ *topspace X*› *closedin_t1_singleton*)
qed
qed
qed

lemma *closedin_Alexandroff_I*:
assumes *compactin* (*Alexandroff_compactification X*) *K K* ⊆ *Some* ‘ *topspace X*
closedin (*Alexandroff_compactification X*) *T K* = *T* ∩ *Some* ‘ *topspace X*
shows *closedin* (*Alexandroff_compactification X*) *K*
proof −
obtain *S* **where** *S*: *S* ⊆ *topspace X K* = *Some* ‘ *S*
by (*meson* ‹*K* ⊆ *Some* ‘ *topspace X*› *subset_imageE*)
with *assms* **have** *compactin X S*
by (*metis compactin_subtopology embedding_map_Some embedding_map_def homeomorphic_map_compactness*)
moreover have *closedin X S*
using *assms S*
by (*metis closedin_subtopology embedding_map_Some embedding_map_def homeomorphic_map_closedness*)
ultimately show *?thesis*
by (*simp add*: *S*)
qed

lemma *t1_space_Alexandroff_compactification* [*simp*]:
t1_space(*Alexandroff_compactification X*) ⟷ *t1_space X*

proof −
have *openin* (*Alexandroff_compactification X*) (*topspace* (*Alexandroff_compactification X*) − {*None*})
by *auto*
then show *?thesis*
using *t1_space_one_point_compactification* [*of Alexandroff_compactification X None*]
using *embedding_map_Some embedding_map_imp_homeomorphic_space homeomorphic_t1_space*
by (*fastforce simp*: *compactin_subtopology closedin_Alexandroff_I closedin_subtopology*)
qed

lemma *kc_space_one_point_compactification*:
assumes *compact_space X*
and *ope*: *openin X* (*topspace X* − {*a*})
and §: ⋀*K*. ⟦*compactin* (*subtopology X* (*topspace X* − {*a*})) *K*; *closedin* (*subtopology X* (*topspace X* − {*a*})) *K*⟧
⟹ *closedin X K*
shows *kc_space X* ⟷
k_space (*subtopology X* (*topspace X* − {*a*})) ∧ *kc_space* (*subtopology X* (*topspace X* − {*a*}))
proof −
have *closedin X K*
if *kc_space* (*subtopology X* (*topspace X* − {*a*})) **and** *compactin X K a* ∉ *K* **for** *K*
using *that* **unfolding** *kc_space_def*
by (*metis* § *Diff_empty compactin_subspace compactin_subtopology subset_Diff_insert*)
then show *?thesis*
by (*metis* ‹*compact_space X*› *kc_space_one_point_compactification_gen ope*)
qed

lemma *kc_space_Alexandroff_compactification*:
kc_space(*Alexandroff_compactification X*) ⟷ (*k_space X* ∧ *kc_space X*) (**is** *kc_space ?Y* = _)
proof −
have *kc_space* (*Alexandroff_compactification X*) ⟷
(*k_space* (*subtopology ?Y* (*topspace ?Y* − {*None*})) ∧ *kc_space* (*subtopology ?Y* (*topspace ?Y* − {*None*})))
by (*rule kc_space_one_point_compactification*) (*auto simp*: *compactin_subtopology closedin_subtopology closedin_Alexandroff_I*)
also have ... ⟷ *k_space X* ∧ *kc_space X*
using *embedding_map_Some embedding_map_imp_homeomorphic_space homeomorphic_k_space homeomorphic_kc_space* **by** *simp blast*
finally show *?thesis* **.**
qed

proposition *regular_space_one_point_compactification*:

assumes *compact_space X* **and** *ope*: *openin X* (*topspace X* − {*a*})
and §: ⋀*K*. ⟦*compactin* (*subtopology X* (*topspace X* − {*a*})) *K*; *closedin* (*subtopology X* (*topspace X* − {*a*})) *K*⟧ ⟹ *closedin X K*
shows *regular_space X* ⟷
regular_space (*subtopology X* (*topspace X* − {*a*})) ∧ *locally_compact_space* (*subtopology X* (*topspace X* − {*a*}))
(**is** *?lhs* ⟷ *?rhs*)
proof
show *?lhs* ⟹ *?rhs*
using *assms*(*1*) *compact_imp_locally_compact_space locally_compact_space_open_subset ope regular_space_subtopology* **by** *blast*
assume *R*: *?rhs*
let *?Xa* = *subtopology X* (*topspace X* − {*a*})
show *?lhs*
unfolding *neighbourhood_base_of_closedin* [*symmetric*] *neighbourhood_base_of imp_conjL*
proof (*intro strip*)
fix *W x*
assume *openin X W* **and** *x* ∈ *W*
show ∃ *U V*. *openin X U* ∧ *closedin X V* ∧ *x* ∈ *U* ∧ *U* ⊆ *V* ∧ *V* ⊆ *W*
proof (*cases x*=*a*)
case *True*
have *compactin ?Xa* (*topspace X* − *W*)
using ‹*openin X W*› *assms*(*1*) *closedin_compact_space*
by (*metis Diff_mono True* ‹*x* ∈ *W*› *compactin_subtopology empty_subsetI insert_subset openin_closedin_eq order_refl*)
then obtain *V K* **where** *V*: *openin ?Xa V* **and** *K*: *compactin ?Xa K closedin ?Xa K* **and** *topspace X* − *W* ⊆ *V V* ⊆ *K*
by (*metis locally_compact_space_compact_closed_compact R*)
show *?thesis*
proof (*intro exI conjI*)
show *openin X* (*topspace X* − *K*)
using § *K* **by** *blast*
show *closedin X* (*topspace X* − *V*)
using *V ope openin_trans_full* **by** *blast*
show *x* ∈ *topspace X* − *K*
proof (*rule*)
show *x* ∈ *topspace X*
using ‹*openin X W*› ‹*x* ∈ *W*› *openin_subset* **by** *blast*
show *x* ∉ *K*
using *K True closedin_imp_subset* **by** *blast*
qed
show *topspace X* − *K* ⊆ *topspace X* − *V*
by (*simp add*: *Diff_mono* ‹*V* ⊆ *K*›)
show *topspace X* − *V* ⊆ *W*
using ‹*topspace X* − *W* ⊆ *V*› **by** *auto*
qed
next
case *False*

have *openin ?Xa* ((*topspace X* − {*a*}) ∩ *W*)
using ‹*openin X W*› *openin_subtopology_Int2* **by** *blast*
moreover have *x* ∈ (*topspace X* − {*a*}) ∩ *W*
using ‹*openin X W*› ‹*x* ∈ *W*› *openin_subset False* **by** *blast*
ultimately obtain *U V* **where** *openin ?Xa U compactin ?Xa V closedin ?Xa V*
x ∈ *U U* ⊆ *V V* ⊆ (*topspace X* − {*a*}) ∩ *W*
using *R locally_compact_regular_space_neighbourhood_base neighbourhood_base_of*
by (*metis* (*no_types*, *lifting*))
then show *?thesis*
by (*meson* § *le_infE ope openin_trans_full*)
qed
qed
qed

lemma *regular_space_Alexandroff_compactification*:
regular_space(*Alexandroff_compactification X*) ⟷ *regular_space X* ∧ *locally_compact_space X*
(**is** *regular_space ?Y* = *?rhs*)
proof −
have *regular_space ?Y* ⟷
regular_space (*subtopology ?Y* (*topspace ?Y* − {*None*})) ∧ *locally_compact_space* (*subtopology ?Y* (*topspace ?Y* − {*None*}))
by (*rule regular_space_one_point_compactification*) (*auto simp*: *compactin_subtopology closedin_subtopology closedin_Alexandroff_I*)
also have ... ⟷ *regular_space X* ∧ *locally_compact_space X*
by (*metis embedding_map_Some embedding_map_imp_homeomorphic_space homeomorphic_locally_compact_space*
homeomorphic_regular_space topspace_Alexandroff_compactification_delete)
finally show *?thesis* **.**
qed

lemma *Hausdorff_space_one_point_compactification*:
assumes *compact_space X* **and** *openin X* (*topspace X* − {*a*})
and ⋀*K*. ⟦*compactin* (*subtopology X* (*topspace X* − {*a*})) *K*; *closedin* (*subtopology X* (*topspace X* − {*a*})) *K*⟧ ⟹ *closedin X K*
shows *Hausdorff_space X* ⟷
Hausdorff_space (*subtopology X* (*topspace X* − {*a*})) ∧ *locally_compact_space* (*subtopology X* (*topspace X* − {*a*}))
(**is** *?lhs* ⟷ *?rhs*)
proof
show *?rhs* **if** *?lhs*
proof −
have *locally_compact_space* (*subtopology X* (*topspace X* − {*a*}))
using *assms that compact_imp_locally_compact_space locally_compact_space_open_subset*

```
    by blast
   with that show ?rhs
    by (simp add: Hausdorff_space_subtopology)
  qed
next
  show ?rhs ⟹ ?lhs
   by (metis assms locally_compact_Hausdorff_or_regular regular_space_one_point_compactification
       regular_t1_eq_Hausdorff_space t1_space_one_point_compactification)
qed

lemma Hausdorff_space_Alexandroff_compactification:
   Hausdorff_space(Alexandroff_compactification X) ⟷ Hausdorff_space X ∧
locally_compact_space X
 by (meson compact_Hausdorff_imp_regular_space compact_space_Alexandroff_compactification

    locally_compact_Hausdorff_or_regular regular_space_Alexandroff_compactification

     regular_t1_eq_Hausdorff_space t1_space_Alexandroff_compactification)

lemma completely_regular_space_Alexandroff_compactification:
  completely_regular_space(Alexandroff_compactification X) ⟷
      completely_regular_space X ∧ locally_compact_space X
 by (metis regular_space_Alexandroff_compactification completely_regular_eq_regular_space
    compact_imp_locally_compact_space compact_space_Alexandroff_compactification)

proposition Hausdorff_space_one_point_compactification_asymmetric_prod:
  assumes compact_space X
  shows Hausdorff_space X ⟷
        kc_space (prod_topology X (subtopology X (topspace X - {a}))) ∧
         k_space (prod_topology X (subtopology X (topspace X - {a})))  (is ?lhs
⟷ ?rhs)
proof (cases a ∈ topspace X)
  case True
  show ?thesis
  proof
    show ?rhs if ?lhs
    proof
      show kc_space (prod_topology X (subtopology X (topspace X - {a})))
      using Hausdorff_imp_kc_space kc_space_prod_topology_right kc_space_subtopology
that by blast
      show k_space (prod_topology X (subtopology X (topspace X - {a})))
      by (meson Hausdorff_imp_kc_space assms compact_imp_locally_compact_space
k_space_prod_topology_left
          kc_space_one_point_compactification_gen that)
    qed
  next
    assume R: ?rhs
    define D where D ≡ (λx. (x,x)) ` (topspace X - {a})
    show ?lhs
```

proof (*cases topspace X = {a}*)
case *True*
then show *?thesis*
by (*simp add*: *Hausdorff_space_def*)
next
case *False*
with *R True* **have** *kc_space X*
using *kc_space_retraction_map_image* [*of prod_topology X* (*subtopology X* (*topspace X − {a}*)) *X fst*]
by (*metis Diff_subset insert_Diff retraction_map_fst topspace_discrete_topology topspace_subtopology_subset*)
have *closedin* (*subtopology* (*prod_topology X* (*subtopology X* (*topspace X − {a}*))) *K*) (*K ∩ D*)
if *compactin* (*prod_topology X* (*subtopology X* (*topspace X − {a}*))) *K* **for** *K*
proof (*intro closedin_subtopology_Int_subset*[**where** *V=K*] *closedin_subset_topspace*)
show *fst ‘ K × snd ‘ K ∩ D ⊆ fst ‘ K × snd ‘ K K ⊆ fst ‘ K × snd ‘ K*
by *force+*
have *eq*: (*fst ‘ K × snd ‘ K ∩ D*) = ((*λx*. (*x*,*x*)) ‘ (*fst ‘ K ∩ snd ‘ K*))
using *compactin_subset_topspace that* **by** (*force simp*: *D_def image_iff*)
have *compactin* (*prod_topology X* (*subtopology X* (*topspace X − {a}*))) (*fst ‘ K × snd ‘ K ∩ D*)
unfolding *eq*
proof (*rule image_compactin* [*of subtopology X* (*topspace X − {a}*)])
have *compactin X* (*fst ‘ K*) *compactin X* (*snd ‘ K*)
by (*meson compactin_subtopology continuous_map_fst continuous_map_snd image_compactin that*)*+*
moreover have *fst ‘ K ∩ snd ‘ K ⊆ topspace X − {a}*
using *compactin_subset_topspace that* **by** *force*
ultimately
show *compactin* (*subtopology X* (*topspace X − {a}*)) (*fst ‘ K ∩ snd ‘ K*)
unfolding *compactin_subtopology*
by (*meson ‹kc_space X› closed_Int_compactin kc_space_def*)
show *continuous_map* (*subtopology X* (*topspace X − {a}*)) (*prod_topology X* (*subtopology X* (*topspace X − {a}*))) (*λx*. (*x*,*x*))
by (*simp add*: *continuous_map_paired*)
qed
then show *closedin* (*prod_topology X* (*subtopology X* (*topspace X − {a}*))) (*fst ‘ K × snd ‘ K ∩ D*)
using *R compactin_imp_closedin_gen* **by** *blast*
qed
moreover have *D ⊆ topspace X × (topspace X ∩ (topspace X − {a}))*
by (*auto simp*: *D_def*)
ultimately have ∗: *closedin* (*prod_topology X* (*subtopology X* (*topspace X − {a}*))) *D*
using *R* **by** (*auto simp*: *k_space*)
have *x=y*
if *x ∈ topspace X y ∈ topspace X*
and §: ⋀*T*. ⟦(*x*,*y*) ∈ *T*; *openin* (*prod_topology X X*) *T*⟧ ⟹ ∃ *z* ∈ *topspace*

```
X. (z,z) ∈ T for x y
    proof (cases x=a ∧ y=a)
      case False
      then consider x≠a | y≠a
        by blast
      then show ?thesis
      proof cases
        case 1
        have ∃z ∈ topspace X − {a}. (z,z) ∈ T
          if (y,x) ∈ T openin (prod_topology X (subtopology X (topspace X −
{a}))) T for T
        proof −
          have (x,y) ∈ {z ∈ topspace (prod_topology X X). (snd z,fst z) ∈ T ∩
topspace X × (topspace X − {a})}
            by (simp add: 1 ‹x ∈ topspace X› ‹y ∈ topspace X› that)
        moreover have openin (prod_topology X X) {z ∈ topspace (prod_topology
X X). (snd z,fst z) ∈ T ∩ topspace X × (topspace X − {a})}
          proof (rule openin_continuous_map_preimage)
            show continuous_map (prod_topology X X) (prod_topology X X) (λx.
(snd x, fst x))
              by (simp add: continuous_map_fst continuous_map_pairedI contin-
uous_map_snd)
            have openin (prod_topology X X) (topspace X × (topspace X − {a}))
            using ‹kc_space X› assms kc_space_one_point_compactification_gen
openin_prod_Times_iff by fastforce
            moreover have openin (prod_topology X X) T
            using kc_space_one_point_compactification_gen [OF ‹compact_space
X›] ‹kc_space X› that
            by (metis openin_prod_Times_iff openin_topspace openin_trans_full
prod_topology_subtopology(2))
            ultimately show openin (prod_topology X X) (T ∩ topspace X ×
(topspace X − {a}))
              by blast
          qed
          ultimately show ?thesis
            by (smt (verit) § Int_iff fst_conv mem_Collect_eq mem_Sigma_iff
snd_conv)
        qed
        then have (y,x) ∈ prod_topology X (subtopology X (topspace X − {a}))
closure_of D
          by (simp add: 1 D_def in_closure_of that)
        then show ?thesis
          using that ∗ D_def closure_of_closedin by fastforce
      next
        case 2
        have ∃z ∈ topspace X − {a}. (z,z) ∈ T
          if (x,y) ∈ T openin (prod_topology X (subtopology X (topspace X −
{a}))) T for T
        proof −
```

```
          have openin (prod_topology X X) (topspace X × (topspace X − {a}))
           using ‹kc_space X› assms kc_space_one_point_compactification_gen
openin_prod_Times_iff by fastforce
          moreover have XXT: openin (prod_topology X X) T
           using kc_space_one_point_compactification_gen [OF ‹compact_space
X›] ‹kc_space X› that
            by (metis openin_prod_Times_iff openin_topspace openin_trans_full
prod_topology_subtopology(2))
            ultimately have openin (prod_topology X X) (T ∩ topspace X ×
(topspace X − {a}))
            by blast
          then show ?thesis
              by (smt (verit) § Diff_subset XXT mem_Sigma_iff openin_subset
subsetD that topspace_prod_topology topspace_subtopology_subset)
        qed
        then have (x,y) ∈ prod_topology X (subtopology X (topspace X − {a}))
closure_of D
          by (simp add: 2 D_def in_closure_of that)
        then show ?thesis
          using that ∗ D_def closure_of_closedin by fastforce
      qed
    qed auto
    then show ?thesis
    unfolding Hausdorff_space_closedin_diagonal closure_of_subset_eq [symmetric]

        by (force simp: closure_of_def)
  qed
 qed
next
 case False
 then show ?thesis
    by (simp add: assms compact_imp_k_space compact_space_prod_topology
kc_space_compact_prod_topology)
qed


lemma Hausdorff_space_Alexandroff_compactification_asymmetric_prod:
  Hausdorff_space(Alexandroff_compactification X) ⟷
      kc_space(prod_topology (Alexandroff_compactification X) X) ∧
      k_space(prod_topology (Alexandroff_compactification X) X)
   (is Hausdorff_space ?Y = ?rhs)
proof −
 have ∗: subtopology (Alexandroff_compactification X)
    (topspace (Alexandroff_compactification X) −
    {None}) homeomorphic_space X
  using embedding_map_Some embedding_map_imp_homeomorphic_space homeomorphic_space_sym by fastforce
 have Hausdorff_space (Alexandroff_compactification X) ⟷
    (kc_space (prod_topology ?Y (subtopology ?Y (topspace ?Y − {None}))) ∧
```

k_space (*prod_topology ?Y* (*subtopology ?Y* (*topspace ?Y* − {*None*}))))
by (*rule Hausdorff_space_one_point_compactification_asymmetric_prod*) (*auto simp*: *compactin_subtopology closedin_subtopology closedin_Alexandroff_I*)
also have ... ⟷ *?rhs*
using *homeomorphic_k_space homeomorphic_kc_space homeomorphic_space_prod_topology*

homeomorphic_space_refl ∗ **by** *blast*
finally show *?thesis* **.**
qed

lemma *kc_space_as_compactification_unique*:
assumes *kc_space X compact_space X*
shows *openin X U* ⟷
(*if a* ∈ *U then U* ⊆ *topspace X* ∧ *compactin X* (*topspace X* − *U*)
else openin (*subtopology X* (*topspace X* − {*a*})) *U*)
proof (*cases a* ∈ *U*)
case *True*
then show *?thesis*
by (*meson assms closedin_compact_space compactin_imp_closedin_gen openin_closedin_eq*)
next
case *False*
then show *?thesis*
by (*metis Diff_empty kc_space_one_point_compactification_gen openin_open_subtopology openin_subset subset_Diff_insert assms*)
qed

lemma *kc_space_as_compactification_unique_explicit*:
assumes *kc_space X compact_space X*
shows *openin X U* ⟷
(*if a* ∈ *U then U* ⊆ *topspace X* ∧
compactin (*subtopology X* (*topspace X* − {*a*})) (*topspace X* − *U*)
∧
closedin (*subtopology X* (*topspace X* − {*a*})) (*topspace X* − *U*)
else openin (*subtopology X* (*topspace X* − {*a*})) *U*)
apply (*simp add*: *kc_space_subtopology compactin_imp_closedin_gen assms compactin_subtopology cong*: *conj_cong*)
by (*metis Diff_mono assms bot.extremum insert_subset kc_space_as_compactification_unique subset_refl*)

lemma *Alexandroff_compactification_unique*:
assumes *kc_space X compact_space X* **and** *a*: *a* ∈ *topspace X*
shows *Alexandroff_compactification* (*subtopology X* (*topspace X* − {*a*})) *homeomorphic_space X*
(**is** *?Y homeomorphic_space X*)
proof −
have [*simp*]: *topspace X* ∩ (*topspace X* − {*a*}) = *topspace X* − {*a*}
by *auto*
have [*simp*]: *insert None* (*Some* ‘ *A*) = *insert None* (*Some* ‘ *B*) ⟷ *A* = *B*
insert None (*Some* ‘ *A*) ≠ *Some* ‘ *B* **for** *A B*

```
    by auto
  have quotient_map X ?Y (λx. if x = a then None else Some x)
    unfolding quotient_map_def
  proof (intro conjI strip)
    show (λx. if x = a then None else Some x) ‘ topspace X = topspace ?Y
      using ‹a ∈ topspace X› by force
    show openin X {x ∈ topspace X. (if x = a then None else Some x) ∈ U} =
openin ?Y U (is ?L = ?R)
      if U ⊆ topspace ?Y for U
    proof (cases None ∈ U)
      case True
      then obtain T where T[simp]: U = insert None (Some ‘ T)
       by (metis Int_insert_right UNIV_I UNIV_option_conv inf.orderE inf_le2
subsetI subset_imageE)
      have Tsub: T ⊆ topspace X − {a}
        using in_these_eq that by auto
     then have {x ∈ topspace X. (if x = a then None else Some x) ∈ U} = insert
a T
        by (auto simp: a image_iff cong: conj_cong)
      then have ?L ⟷ openin X (insert a T)
        by metis
      also have ... ⟷ ?R
        using Tsub assms
      apply (simp add: openin_Alexandroff_compactification kc_space_as_compactification_unique_exp
[where a=a])
        by (smt (verit, best) Diff_insert2 Diff_subset closedin_imp_subset dou-
ble_diff)
      finally show ?thesis .
    next
      case False
      then obtain T where [simp]: U = Some ‘ T
       by (metis Int_insert_right UNIV_I UNIV_option_conv inf.orderE inf_le2
subsetI subset_imageE)
      have ∗∗: ⋀V. openin X V ⟹ openin X (V − {a})
        by (simp add: assms compactin_imp_closedin_gen openin_diff)
      have Tsub: T ⊆ topspace X − {a}
        using in_these_eq that by auto
      then have {x ∈ topspace X. (if x = a then None else Some x) ∈ U} = T
        by (auto simp: image_iff cong: conj_cong)
      then show ?thesis
        by (simp add: ∗∗ Tsub openin_open_subtopology)
    qed
  qed
  moreover have inj_on (λx. if x = a then None else Some x) (topspace X)
    by (auto simp: inj_on_def)
  ultimately show ?thesis
   using homeomorphic_space_sym homeomorphic_space homeomorphic_map_def
by blast
qed
```

5.11.7 Extending continuous maps "pointwise" in a regular space

lemma *continuous_map_on_intermediate_closure_of*:
assumes *Y*: *regular_space Y*
and *T*: $T \subseteq X$ *closure_of S*
and *f*: $\bigwedge t.\ t \in T \Longrightarrow$ *limitin Y f* (*f t*) (*atin_within X t S*)
shows *continuous_map* (*subtopology X T*) *Y f*
proof (*clarsimp simp add*: *continuous_map_atin*)
fix *a*
assume $a \in$ *topspace X* **and** $a \in T$
have *f* ' $T \subseteq$ *topspace Y*
by (*metis f image_subsetI limitin_topspace*)
have $\forall_F\ x$ *in atin_within X a T*. *f x* $\in W$
if *W*: *openin Y W f a* $\in W$ **for** *W*
proof −
obtain *V C* **where** *openin Y V closedin Y C f a* $\in V$ $V \subseteq C$ $C \subseteq W$
by (*metis Y W neighbourhood_base_of neighbourhood_base_of_closedin*)
have $\forall_F\ x$ *in atin_within X a S*. *f x* $\in V$
by (*metis* ‹$a \in T$› ‹*f a* $\in V$› ‹*openin Y V*› *f limitin_def*)
then obtain *U* **where** *openin X U a* $\in U$ **and** *U*: $\forall x \in U - \{a\}.\ x \in S \longrightarrow f\ x \in V$
by (*smt* (*verit*) *Diff_iff* ‹$a \in$ *topspace X*› *eventually_atin_within insert_iff*)
moreover have *f z* $\in W$ **if** $z \in U$ $z \neq a$ $z \in T$ **for** *z*
proof −
have $z \in$ *topspace X*
using ‹*openin X U*› *openin_subset* ‹$z \in U$› **by** *blast*
then have *f z* $\in$ *topspace Y*
using ‹*f* ' $T \subseteq$ *topspace Y*› ‹$z \in T$› **by** *blast*
{ assume *f z* $\in$ *topspace Y f z* $\notin C$
then have $\forall_F\ x$ *in atin_within X z S*. *f x* $\in$ *topspace Y* − *C*
by (*metis Diff_iff* ‹*closedin Y C*› *closedin_def f limitinD* ‹$z \in T$›)
then obtain *U'* **where** *U'*: *openin X U' z* $\in U'$
$\bigwedge x.\ x \in U' - \{z\} \Longrightarrow x \in S \Longrightarrow f\ x \notin C$
by (*smt* (*verit*) *Diff_iff* ‹$z \in$ *topspace X*› *eventually_atin_within insertCI*)
then have $*$: $\bigwedge D.\ z \in D \wedge$ *openin X D* $\Longrightarrow \exists y.\ y \in S \wedge y \in D$
by (*meson T in_closure_of subsetD* ‹$z \in T$›)
have *False*
using $*$ [*of U* $\cap$ *U'*] *U' U* ‹$V \subseteq C$› ‹*f a* $\in V$› ‹*f z* $\notin C$› ‹*openin X U*› *that*
by *blast*
}
then show *?thesis*
using ‹$C \subseteq W$› ‹*f z* $\in$ *topspace Y*› **by** *auto*
qed
ultimately have $\exists U.$ *openin X U* $\wedge$ $a \in U \wedge (\forall x \in U - \{a\}.\ x \in T \longrightarrow f\ x \in W)$
by *blast*
then show *?thesis*
using *eventually_atin_within* **by** *fastforce*

```
  qed
  then show limitin Y f (f a) (atin (subtopology X T) a)
    by (metis ‹a ∈ T› atin_subtopology_within f limitin_def)
qed


lemma continuous_map_on_intermediate_closure_of_eq:
  assumes regular_space Y S ⊆ T and Tsub: T ⊆ X closure_of S
  shows continuous_map (subtopology X T) Y f ⟷ (∀ t ∈ T. limitin Y f (f t)
(atin_within X t S))
       (is ?lhs ⟷ ?rhs)
proof
  assume L: ?lhs
  show ?rhs
  proof (clarsimp simp add: continuous_map_atin)
    fix x
    assume x ∈ T
    with L Tsub closure_of_subset_topspace
    have limitin Y f (f x) (atin (subtopology X T) x)
      by (fastforce simp: continuous_map_atin)
    then show limitin Y f (f x) (atin_within X x S)
      using ‹x ∈ T› ‹S ⊆ T›
     by (force simp: limitin_def atin_subtopology_within eventually_atin_within)
  qed
next
  show ?rhs ⟹ ?lhs
    using assms continuous_map_on_intermediate_closure_of by blast
qed

lemma continuous_map_extension_pointwise_alt:
  assumes §: regular_space Y S ⊆ T T ⊆ X closure_of S
    and f: continuous_map (subtopology X S) Y f
    and lim: ⋀t. t ∈ T−S ⟹ ∃ l. limitin Y f l (atin_within X t S)
  obtains g where continuous_map (subtopology X T) Y g ⋀x. x ∈ S ⟹ g x =
f x
proof −
  obtain g where g: ⋀t. t ∈ T ∧ t ∉ S ⟹ limitin Y f (g t) (atin_within X t S)
    by (metis Diff_iff lim)
  let ?h = λx. if x ∈ S then f x else g x
  show thesis
  proof
    have T: T ⊆ topspace X
      using § closure_of_subset_topspace by fastforce
    have limitin Y ?h (f t) (atin_within X t S) if t ∈ T t ∈ S for t
    proof −
      have limitin Y f (f t) (atin_within X t S)
        by (meson T f limit_continuous_map_within subset_eq that)
      then show ?thesis
        by (simp add: eventually_atin_within limitin_def)
```

qed
moreover have *limitin Y ?h (g t) (atin_within X t S)* **if** $t \in T$ $t \notin S$ **for** *t*
by (*smt* (*verit*, *del_insts*) *eventually_atin_within g limitin_def that*)
ultimately show *continuous_map* (*subtopology X T*) *Y ?h*
unfolding *continuous_map_on_intermediate_closure_of_eq* [*OF* §]
by (*auto simp*: § *atin_subtopology_within*)
qed *auto*
qed

lemma *continuous_map_extension_pointwise*:
assumes *regular_space Y S* $S \subseteq T$ **and** *Tsub*: $T \subseteq X$ *closure_of S*
and *ex*: $\bigwedge x.\ x \in T \Longrightarrow \exists g.$ *continuous_map* (*subtopology X* (*insert x S*)) *Y g* $\wedge$
$(\forall x \in S.\ g\ x = f\ x)$
obtains *g* **where** *continuous_map* (*subtopology X T*) *Y g* $\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
proof (*rule continuous_map_extension_pointwise_alt*)
show *continuous_map* (*subtopology X S*) *Y f*
proof (*clarsimp simp add*: *continuous_map_atin*)
fix *t*
assume $t \in$ *topspace X* **and** $t \in S$
then obtain *g* **where** *g*: *limitin Y g* (*g t*) (*atin* (*subtopology X* (*insert t S*)) *t*) **and** *gf*: $\forall x \in S.\ g\ x = f\ x$
by (*metis Int_iff* ‹$S \subseteq T$› *continuous_map_atin ex inf.orderE insert_absorb topspace_subtopology*)
with ‹$t \in S$› **show** *limitin Y f* (*f t*) (*atin* (*subtopology X S*) *t*)
by (*simp add*: *limitin_def atin_subtopology_within_if eventually_atin_within gf insert_absorb*)
qed
show $\exists l.$ *limitin Y f l* (*atin_within X t S*) **if** $t \in T - S$ **for** *t*
proof −
obtain *g* **where** *g*: *continuous_map* (*subtopology X* (*insert t S*)) *Y g* **and** *gf*: $\forall x \in S.\ g\ x = f\ x$
using ‹$S \subseteq T$› *ex* ‹$t \in T - S$› **by** *force*
then have *limitin Y g* (*g t*) (*atin_within X t* (*insert t S*))
using *Tsub in_closure_of limit_continuous_map_within that* **by** *fastforce*
then show *?thesis*
unfolding *limitin_def*
by (*smt* (*verit*) *eventually_atin_within gf subsetD subset_insertI*)
qed
qed (*use assms* **in** *auto*)

5.11.8 Extending Cauchy continuous functions to the closure

lemma *Cauchy_continuous_map_extends_to_continuous_closure_of_aux*:
assumes *m2*: *mcomplete_of m2* **and** *f*: *Cauchy_continuous_map* (*submetric m1 S*) *m2 f*
and $S \subseteq$ *mspace m1*

obtains *g*
where *continuous_map* (*subtopology* (*mtopology_of m1*) (*mtopology_of m1 closure_of S*))
(*mtopology_of m2*) *g* ⋀*x*. *x* ∈ *S* ⟹ *g x* = *f x*
proof (*rule continuous_map_extension_pointwise_alt*)
interpret *L*: *Metric_space12 mspace m1 mdist m1 mspace m2 mdist m2*
by (*simp add*: *Metric_space12_mspace_mdist*)
interpret *S*: *Metric_space S* ∩ *mspace m1 mdist m1*
by (*simp add*: *L.M1.subspace*)
show *regular_space* (*mtopology_of m2*)
by (*simp add*: *Metric_space.regular_space_mtopology mtopology_of_def*)
show *S* ⊆ *mtopology_of m1 closure_of S*
by (*simp add*: *assms*(*3*) *closure_of_subset*)
show *continuous_map* (*subtopology* (*mtopology_of m1*) *S*) (*mtopology_of m2*) *f*
by (*metis Cauchy_continuous_imp_continuous_map f mtopology_of_submetric*)
fix *a*
assume *a*: *a* ∈ *mtopology_of m1 closure_of S* − *S*
then obtain *σ* **where** *ranσ*: *range σ* ⊆ *S range σ* ⊆ *mspace m1*
and *limσ*: *limitin L.M1.mtopology σ a sequentially*
by (*force simp*: *mtopology_of_def L.M1.closure_of_sequentially*)
then have *L.M1.MCauchy σ*
by (*simp add*: *L.M1.convergent_imp_MCauchy mtopology_of_def*)
then have *L.M2.MCauchy* (*f* ∘ *σ*)
using *f ranσ* **by** (*simp add*: *Cauchy_continuous_map_def L.M1.subspace Metric_space.MCauchy_def*)
then obtain *l* **where** *l*: *limitin L.M2.mtopology* (*f* ∘ *σ*) *l sequentially*
by (*meson L.M2.mcomplete_def m2 mcomplete_of_def*)
have *limitin L.M2.mtopology f l* (*atin_within L.M1.mtopology a S*)
unfolding *L.limit_atin_sequentially_within imp_conjL*
proof (*intro conjI strip*)
show *l* ∈ *mspace m2*
using *L.M2.limitin_mspace l* **by** *blast*
fix *ϱ*
assume *range ϱ* ⊆ *S* ∩ *mspace m1* − {*a*} **and** *limϱ*: *limitin L.M1.mtopology ϱ a sequentially*
then have *ranϱ*: *range ϱ* ⊆ *S range ϱ* ⊆ *mspace m1* ⋀*n*. *ϱ n* ≠ *a*
by *auto*
have *a* ∈ *mspace m1*
using *L.M1.limitin_mspace limϱ* **by** *auto*
have *S.MCauchy σ S.MCauchy ϱ*
using *L.M1.convergent_imp_MCauchy L.M1.MCauchy_def S.MCauchy_def limσ ranσ limϱ ranϱ* **by** *force*+
then have *L.M2.MCauchy* (*f* ∘ *ϱ*) *L.M2.MCauchy* (*f* ∘ *σ*)
using *f* **by** (*auto simp*: *Cauchy_continuous_map_def*)
then have *ran_f*: *range* (λ*x*. *f* (*ϱ x*)) ⊆ *mspace m2 range* (λ*x*. *f* (*σ x*)) ⊆ *mspace m2*
by (*auto simp*: *L.M2.MCauchy_def*)
have (λ*n*. *mdist m2* (*f* (*ϱ n*)) *l*) ⟶ *0*
proof (*rule Lim_null_comparison*)

have *mdist m2* (*f* (*ϱ n*)) *l* ≤ *mdist m2* (*f* (*σ n*)) *l* + *mdist m2* (*f* (*σ n*)) (*f* (*ϱ n*)) **for** *n*
using ‹*l* ∈ *mspace m2*› *ran_f L.M2.triangle''* **by** (*smt* (*verit*, *best*) *range_subsetD*)
then show ∀$_F$ *n in sequentially*. *norm* (*mdist m2* (*f* (*ϱ n*)) *l*) ≤ *mdist m2* (*f* (*σ n*)) *l* + *mdist m2* (*f* (*σ n*)) (*f* (*ϱ n*))
by *force*
define *ψ* **where** *ψ* ≡ λ*n*. *if even n then σ* (*n div 2*) *else ϱ* (*n div 2*)
have (λ*n*. *mdist m1* (*σ n*) (*ϱ n*)) ⟶ *0*
proof (*rule Lim_null_comparison*)
show ∀$_F$ *n in sequentially*. *norm* (*mdist m1* (*σ n*) (*ϱ n*)) ≤ *mdist m1* (*σ n*) *a* + *mdist m1* (*ϱ n*) *a*
using *L.M1.triangle'* [*of* _ *a*] *ranσ ranϱ* ‹*a* ∈ *mspace m1*› **by** (*simp add*: *range_subsetD*)
have (λ*n*. *mdist m1* (*σ n*) *a*) ⟶ *0*
using *L.M1.limitin_metric_dist_null limσ* **by** *blast*
moreover have (λ*n*. *mdist m1* (*ϱ n*) *a*) ⟶ *0*
using *L.M1.limitin_metric_dist_null limϱ* **by** *blast*
ultimately show (λ*n*. *mdist m1* (*σ n*) *a* + *mdist m1* (*ϱ n*) *a*) ⟶ *0*
by (*simp add*: *tendsto_add_zero*)
qed
with ‹*S.MCauchy σ*› ‹*S.MCauchy ϱ*› **have** *S.MCauchy ψ*
by (*simp add*: *S.MCauchy_interleaving_gen ψ_def*)
then have *L.M2.MCauchy* (*f* ∘ *ψ*)
by (*metis Cauchy_continuous_map_def f mdist_submetric mspace_submetric*)
then have (λ*n*. *mdist m2* (*f* (*σ n*)) (*f* (*ϱ n*))) ⟶ *0*
using *L.M2.MCauchy_interleaving_gen* [*of f* ∘ *σ f* ∘ *ϱ*]
by (*simp add*: *if_distrib ψ_def o_def cong*: *if_cong*)
moreover have ∀$_F$ *n in sequentially*. *f* (*σ n*) ∈ *mspace m2* ∧ (λ*x*. *mdist m2* (*f* (*σ x*)) *l*) ⟶ *0*
using *l* **by** (*auto simp*: *L.M2.limitin_metric_dist_null* ‹*l* ∈ *mspace m2*›)
ultimately show (λ*n*. *mdist m2* (*f* (*σ n*)) *l* + *mdist m2* (*f* (*σ n*)) (*f* (*ϱ n*))) ⟶ *0*
by (*metis* (*mono_tags*) *tendsto_add_zero eventually_sequentially order_refl*)
qed
with *ran_f* **show** *limitin L.M2.mtopology* (*f* ∘ *ϱ*) *l sequentially*
by (*auto simp*: *L.M2.limitin_metric_dist_null eventually_sequentially* ‹*l* ∈ *mspace m2*›)
qed
then show ∃*l*. *limitin* (*mtopology_of m2*) *f l* (*atin_within* (*mtopology_of m1*) *a S*)
by (*force simp*: *mtopology_of_def*)
qed *auto*

lemma *Cauchy_continuous_map_extends_to_continuous_closure_of*:
assumes *mcomplete_of m2*
and *f*: *Cauchy_continuous_map* (*submetric m1 S*) *m2 f*
obtains *g*

where *continuous_map* (*subtopology* (*mtopology_of m1*) ((*mtopology_of m1*) *closure_of S*))
(*mtopology_of m2*) *g* ⋀*x*. *x* ∈ *S* ⟹ *g x* = *f x*
proof −
obtain *g* **where** *cmg*:
continuous_map (*subtopology* (*mtopology_of m1*) ((*mtopology_of m1*) *closure_of* (*mspace m1* ∩ *S*)))
(*mtopology_of m2*) *g*
and *gf*: (∀ *x* ∈ *mspace m1* ∩ *S*. *g x* = *f x*)
using *Cauchy_continuous_map_extends_to_continuous_closure_of_aux assms*
by (*metis inf_commute inf_le2 submetric_restrict*)
define *h* **where** *h* ≡ λ*x*. *if x* ∈ *topspace*(*mtopology_of m1*) *then g x else f x*
show *thesis*
proof
show *continuous_map* (*subtopology* (*mtopology_of m1*) ((*mtopology_of m1*) *closure_of S*))
(*mtopology_of m2*) *h*
unfolding *h_def*
proof (*rule continuous_map_eq*)
show *continuous_map* (*subtopology* (*mtopology_of m1*) (*mtopology_of m1 closure_of S*)) (*mtopology_of m2*) *g*
by (*metis closure_of_restrict cmg topspace_mtopology_of*)
qed *auto*
qed (*auto simp*: *gf h_def*)
qed

lemma *Cauchy_continuous_map_extends_to_continuous_intermediate_closure_of*:
assumes *mcomplete_of m2*
and *f*: *Cauchy_continuous_map* (*submetric m1 S*) *m2 f*
and *T*: *T* ⊆ *mtopology_of m1 closure_of S*
obtains *g*
where *continuous_map* (*subtopology* (*mtopology_of m1*) *T*) (*mtopology_of m2*) *g*
(∀ *x* ∈ *S*. *g x* = *f x*)
by (*metis Cauchy_continuous_map_extends_to_continuous_closure_of T assms*(1) *continuous_map_from_subtopology_mono f*)

Technical lemma helpful for porting particularly ugly HOL Light proofs

lemma *all_in_closure_of*:
assumes *P*: ∀ *x* ∈ *S*. *P x* **and** *clo*: *closedin X* {*x* ∈ *topspace X*. *P x*}
shows ∀ *x* ∈ *X closure_of S*. *P x*
proof −
have ∗: *topspace X* ∩ *S* ⊆ {*x* ∈ *topspace X*. *P x*}
using *P* **by** *auto*
show *?thesis*
using *closure_of_minimal* [*OF* ∗ *clo*] *closure_of_restrict* **by** *fastforce*
qed

lemma *Lipschitz_continuous_map_on_intermediate_closure_aux*:
assumes *lcf*: *Lipschitz_continuous_map* (*submetric m1 S*) *m2 f*
and $S \subseteq T$ **and** *Tsub*: $T \subseteq$ (*mtopology_of m1*) *closure_of S*
and *cmf*: *continuous_map* (*subtopology* (*mtopology_of m1*) *T*) (*mtopology_of m2*) *f*
and $S \subseteq$ *mspace m1*
shows *Lipschitz_continuous_map* (*submetric m1 T*) *m2 f*
proof −
interpret *L*: *Metric_space12 mspace m1 mdist m1 mspace m2 mdist m2*
by (*simp add*: *Metric_space12_mspace_mdist*)
interpret *S*: *Metric_space* $S \cap$ *mspace m1 mdist m1*
by (*simp add*: *L.M1.subspace*)
have $T \subseteq$ *mspace m1*
using *Tsub* **by** (*auto simp*: *mtopology_of_def closure_of_def*)
show *?thesis*
unfolding *Lipschitz_continuous_map_pos*
proof
show $f \in$ *mspace* (*submetric m1 T*) $\to$ *mspace m2*
by (*metis cmf Metric_space.metric_continuous_map Metric_space_mspace_mdist mtopology_of_def*
mtopology_of_submetric image_subset_iff_funcset)
define *X* **where** *X* ≡ *prod_topology* (*subtopology L.M1.mtopology T*) (*subtopology L.M1.mtopology T*)
obtain *B*::*real* **where** $B > 0$ **and** *B*: $\forall (x,y) \in S \times S$. *mdist m2* (*f x*) (*f y*) ≤ $B *$ *mdist m1 x y*
using *lcf* ‹$S \subseteq$ *mspace m1*› **by** (*force simp*: *Lipschitz_continuous_map_pos*)
have *eq*: {*z* ∈ *A*. *case z of* (*x*,*y*) ⇒ *p x y* ≤ *B* ∗ *q x y*} = {*z* ∈ *A*. ((λ(*x*,*y*). *B* ∗ *q x y* − *p x y*)*z*) ∈ {*0*..}}
for *p q* **and** *A*::(*'a*∗*'a*)*set*
by *auto*
have *clo*: *closedin X* {*z* ∈ *topspace X*. *case z of* (*x*, *y*) ⇒ *mdist m2* (*f x*) (*f y*) ≤ *B* ∗ *mdist m1 x y*}
unfolding *eq*
proof (*rule closedin_continuous_map_preimage*)
have ∗: *continuous_map X L.M2.mtopology* (*f* ∘ *fst*) *continuous_map X L.M2.mtopology* (*f* ∘ *snd*)
using *cmf* **by** (*auto simp*: *mtopology_of_def X_def intro*: *continuous_map_compose continuous_map_fst continuous_map_snd*)
then show *continuous_map X euclidean* (λ*x*. *case x of* (*x*, *y*) ⇒ *B* ∗ *mdist m1 x y* − *mdist m2* (*f x*) (*f y*))
unfolding *case_prod_unfold*
proof (*intro continuous_intros*; *simp add*: *mtopology_of_def o_def*)
show *continuous_map X L.M1.mtopology fst continuous_map X L.M1.mtopology snd*
by (*simp_all add*: *X_def continuous_map_subtopology_fst continuous_map_subtopology_snd flip*: *subtopology_Times*)
qed
qed *auto*
have *mdist m2* (*f x*) (*f y*) ≤ *B* ∗ *mdist m1 x y* **if** $x \in T$ $y \in T$ **for** *x y*

using *all_in_closure_of* [*OF B clo*] ‹$S \subseteq T$› *Tsub*
by (*fastforce simp*: *X_def subset_iff closure_of_Times closure_of_subtopology inf.absorb2*
mtopology_of_def that)
then show $\exists B{>}0.\ \forall x{\in}mspace\ (submetric\ m1\ T).$
$\forall y{\in}mspace\ (submetric\ m1\ T).$
$mdist\ m2\ (f\ x)\ (f\ y) \leq B * mdist\ (submetric\ m1\ T)\ x\ y$
using ‹$0 < B$› **by** *auto*
qed
qed

lemma *Lipschitz_continuous_map_on_intermediate_closure*:
assumes *Lipschitz_continuous_map* (*submetric m1 S*) *m2 f*
and $S \subseteq T\ T \subseteq$ (*mtopology_of m1*) *closure_of S*
and *continuous_map* (*subtopology* (*mtopology_of m1*) *T*) (*mtopology_of m2*) *f*
shows *Lipschitz_continuous_map* (*submetric m1 T*) *m2 f*
by (*metis Lipschitz_continuous_map_on_intermediate_closure_aux assms closure_of_subset_topspace subset_trans topspace_mtopology_of*)

lemma *Lipschitz_continuous_map_extends_to_closure_of*:
assumes *m2*: *mcomplete_of m2*
and *f*: *Lipschitz_continuous_map* (*submetric m1 S*) *m2 f*
obtains *g*
where *Lipschitz_continuous_map* (*submetric m1* (*mtopology_of m1 closure_of S*)) *m2 g*
$\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
proof −
obtain *g*
where *g*: *continuous_map* (*subtopology* (*mtopology_of m1*) ((*mtopology_of m1*) *closure_of S*))
(*mtopology_of m2*) *g* $(\forall x \in S.\ g\ x = f\ x)$
by (*metis Cauchy_continuous_map_extends_to_continuous_closure_of Lipschitz_imp_Cauchy_continuous_map f m2*)
have *Lipschitz_continuous_map* (*submetric m1* (*mtopology_of m1 closure_of S*)) *m2 g*
proof (*rule Lipschitz_continuous_map_on_intermediate_closure*)
show *Lipschitz_continuous_map* (*submetric m1* (*mspace m1* ∩ *S*)) *m2 g*
by (*smt* (*verit*, *best*) *IntD2 Lipschitz_continuous_map_eq f g*(*2*) *inf_commute mspace_submetric submetric_restrict*)
show *mspace m1* ∩ *S* ⊆ *mtopology_of m1 closure_of S*
using *closure_of_subset_Int* **by** *force*
show *mtopology_of m1 closure_of S* ⊆ *mtopology_of m1 closure_of* (*mspace m1* ∩ *S*)
by (*metis closure_of_restrict subset_refl topspace_mtopology_of*)
show *continuous_map* (*subtopology* (*mtopology_of m1*) (*mtopology_of m1 closure_of S*)) (*mtopology_of m2*) *g*

by (*simp add*: *g*)
qed
with *g that* **show** *thesis*
by *metis*
qed

lemma *Lipschitz_continuous_map_extends_to_intermediate_closure_of*:
assumes *mcomplete_of m2*
and *Lipschitz_continuous_map* (*submetric m1 S*) *m2 f*
and *T* ⊆ *mtopology_of m1 closure_of S*
obtains *g*
where *Lipschitz_continuous_map* (*submetric m1 T*) *m2 g* ⋀*x*. *x* ∈ *S* ⟹ *g x* = *f x*
by (*metis Lipschitz_continuous_map_extends_to_closure_of Lipschitz_continuous_map_from_submetric_mono assms*)

This proof uses the same trick to extend the function's domain to its closure

lemma *uniformly_continuous_map_on_intermediate_closure_aux*:
assumes *ucf*: *uniformly_continuous_map* (*submetric m1 S*) *m2 f*
and *S* ⊆ *T* **and** *Tsub*: *T* ⊆ (*mtopology_of m1*) *closure_of S*
and *cmf*: *continuous_map* (*subtopology* (*mtopology_of m1*) *T*) (*mtopology_of m2*) *f*
and *S* ⊆ *mspace m1*
shows *uniformly_continuous_map* (*submetric m1 T*) *m2 f*
proof −
interpret *L*: *Metric_space12 mspace m1 mdist m1 mspace m2 mdist m2*
by (*simp add*: *Metric_space12_mspace_mdist*)
interpret *S*: *Metric_space S* ∩ *mspace m1 mdist m1*
by (*simp add*: *L.M1.subspace*)
have *T* ⊆ *mspace m1*
using *Tsub* **by** (*auto simp*: *mtopology_of_def closure_of_def*)
show *?thesis*
unfolding *uniformly_continuous_map_def*
proof (*intro conjI strip*)
show *f* ∈ *mspace* (*submetric m1 T*) → *mspace m2*
by (*metis cmf Metric_space.metric_continuous_map Metric_space_mspace_mdist mtopology_of_def mtopology_of_submetric image_subset_iff_funcset*)
fix *ε*::*real*
assume *ε* > *0*
then obtain *δ* **where** *δ*>*0* **and** *δ*: ∀ (*x*,*y*) ∈ *S*×*S*. *mdist m1 x y* < *δ* ⟶ *mdist m2* (*f x*) (*f y*) ≤ *ε*/*2*
using *ucf* ‹*S* ⊆ *mspace m1*› **unfolding** *uniformly_continuous_map_def mspace_submetric*
apply (*simp add*: *Ball_def del*: *divide_const_simps*)
by (*metis IntD2 half_gt_zero inf.orderE less_eq_real_def*)
define *X* **where** *X* ≡ *prod_topology* (*subtopology L.M1.mtopology T*) (*subtopology L.M1.mtopology T*)

A clever construction involving the union of two closed sets

have *eq*: $\{z \in A.\ case\ z\ of\ (x,y) \Rightarrow p\ x\ y < d \longrightarrow q\ x\ y \leq e\}$
$= \{z \in A.\ ((\lambda(x,y).\ p\ x\ y - d)z) \in \{0..\}\} \cup \{z \in A.\ ((\lambda(x,y).\ e - q\ x\ y)z) \in \{0..\}\}$
for *p q* **and** *d e*::*real* **and** *A*::(*'a*∗*'a*)*set*
by *auto*
have *clo*: *closedin X* $\{z \in$ *topspace X. case z of* $(x,\ y) \Rightarrow$ *mdist m1 x y* $< \delta \longrightarrow$ *mdist m2* $(f\ x)\ (f\ y) \leq \varepsilon/2\}$
unfolding *eq*
proof (*intro closedin_Un closedin_continuous_map_preimage*)
have ∗: *continuous_map X L.M1.mtopology fst continuous_map X L.M1.mtopology snd*
by (*metis X_def continuous_map_subtopology_fst subtopology_Times continuous_map_subtopology_snd*)+
show *continuous_map X euclidean* $(\lambda x.$ *case x of* $(x,\ y) \Rightarrow$ *mdist m1 x y* $- \delta)$
unfolding *case_prod_unfold*
by (*intro continuous_intros*; *simp add*: *mtopology_of_def* ∗)
have ∗: *continuous_map X L.M2.mtopology* $(f \circ fst)$ *continuous_map X L.M2.mtopology* $(f \circ snd)$
using *cmf* **by** (*auto simp*: *mtopology_of_def X_def intro*: *continuous_map_compose continuous_map_fst continuous_map_snd*)
then show *continuous_map X euclidean* $(\lambda x.$ *case x of* $(x,\ y) \Rightarrow \varepsilon\ /\ 2 -$ *mdist m2* $(f\ x)\ (f\ y)$)
unfolding *case_prod_unfold*
by (*intro continuous_intros*; *simp add*: *mtopology_of_def o_def*)
qed *auto*
have *mdist m2* $(f\ x)\ (f\ y) \leq \varepsilon/2$ **if** $x \in T\ y \in T$ *mdist m1 x y* $< \delta$ **for** *x y*
using *all_in_closure_of* [*OF δ clo*] ‹$S \subseteq T$› *Tsub*
by (*fastforce simp*: *X_def subset_iff closure_of_Times closure_of_subtopology inf.absorb2*
mtopology_of_def that)
then show $\exists \delta{>}0.\ \forall x{\in}$*mspace* (*submetric m1 T*). $\forall y{\in}$*mspace* (*submetric m1 T*). *mdist* (*submetric m1 T*) *y x* $< \delta \longrightarrow$ *mdist m2* $(f\ y)\ (f\ x) < \varepsilon$
using ‹$0 < \delta$› ‹$0 < \varepsilon$› **by** *fastforce*
qed
qed

lemma *uniformly_continuous_map_on_intermediate_closure*:
assumes *uniformly_continuous_map* (*submetric m1 S*) *m2 f*
and $S \subseteq T$ **and** $T \subseteq$ (*mtopology_of m1*) *closure_of S*
and *continuous_map* (*subtopology* (*mtopology_of m1*) *T*) (*mtopology_of m2*) *f*
shows *uniformly_continuous_map* (*submetric m1 T*) *m2 f*
by (*metis assms closure_of_subset_topspace subset_trans topspace_mtopology_of*
uniformly_continuous_map_on_intermediate_closure_aux)

lemma *uniformly_continuous_map_extends_to_closure_of*:

assumes *m2*: *mcomplete_of m2*
and *f*: *uniformly_continuous_map* (*submetric m1 S*) *m2 f*
obtains *g*
where *uniformly_continuous_map* (*submetric m1* (*mtopology_of m1 closure_of S*)) *m2 g*
⋀*x*. *x* ∈ *S* ⟹ *g x* = *f x*
proof −
obtain *g*
where *g*: *continuous_map* (*subtopology* (*mtopology_of m1*) ((*mtopology_of m1*) *closure_of S*))
(*mtopology_of m2*) *g* (∀ *x* ∈ *S*. *g x* = *f x*)
by (*metis Cauchy_continuous_map_extends_to_continuous_closure_of uniformly_imp_Cauchy_continuous_map f m2*)
have *uniformly_continuous_map* (*submetric m1* (*mtopology_of m1 closure_of S*)) *m2 g*
proof (*rule uniformly_continuous_map_on_intermediate_closure*)
show *uniformly_continuous_map* (*submetric m1* (*mspace m1* ∩ *S*)) *m2 g*
by (*smt* (*verit*, *best*) *IntD2 uniformly_continuous_map_eq f g*(*2*) *inf_commute mspace_submetric submetric_restrict*)
show *mspace m1* ∩ *S* ⊆ *mtopology_of m1 closure_of S*
using *closure_of_subset_Int* **by** *force*
show *mtopology_of m1 closure_of S* ⊆ *mtopology_of m1 closure_of* (*mspace m1* ∩ *S*)
by (*metis closure_of_restrict subset_refl topspace_mtopology_of*)
show *continuous_map* (*subtopology* (*mtopology_of m1*) (*mtopology_of m1 closure_of S*)) (*mtopology_of m2*) *g*
by (*simp add*: *g*)
qed
with *g that* **show** *thesis*
by *metis*
qed

lemma *uniformly_continuous_map_extends_to_intermediate_closure_of*:
assumes *mcomplete_of m2*
and *uniformly_continuous_map* (*submetric m1 S*) *m2 f*
and *T* ⊆ *mtopology_of m1 closure_of S*
obtains *g*
where *uniformly_continuous_map* (*submetric m1 T*) *m2 g* ⋀*x*. *x* ∈ *S* ⟹ *g x* = *f x*
by (*metis uniformly_continuous_map_extends_to_closure_of uniformly_continuous_map_from_submetric_mo assms*)

lemma *Cauchy_continuous_map_on_intermediate_closure_aux*:
assumes *ucf*: *Cauchy_continuous_map* (*submetric m1 S*) *m2 f*
and *S* ⊆ *T* **and** *Tsub*: *T* ⊆ (*mtopology_of m1*) *closure_of S*
and *cmf*: *continuous_map* (*subtopology* (*mtopology_of m1*) *T*) (*mtopology_of m2*) *f*

```
    and S ⊆ mspace m1
  shows Cauchy_continuous_map (submetric m1 T) m2 f
proof −
  interpret L: Metric_space12 mspace m1 mdist m1 mspace m2 mdist m2
    by (simp add: Metric_space12_mspace_mdist)
  interpret S: Metric_space S ∩ mspace m1 mdist m1
    by (simp add: L.M1.subspace)
  interpret T: Metric_space T mdist m1
     by (metis L.M1.subspace Tsub closure_of_subset_topspace dual_order.trans
topspace_mtopology_of)
  have T ⊆ mspace m1
    using Tsub by (auto simp: mtopology_of_def closure_of_def)
  then show ?thesis
  proof (clarsimp simp: Cauchy_continuous_map_def Int_absorb2)
    fix σ
    assume σ: T.MCauchy σ
    have ∃ y∈S. mdist m1 (σ n) y < inverse (Suc n) ∧ mdist m2 (f (σ n)) (f y)
< inverse (Suc n) for n
    proof −
      have σ n ∈ T
        using σ by (force simp: T.MCauchy_def)
     moreover have continuous_map (mtopology_of (submetric m1 T)) L.M2.mtopology
f
        by (metis cmf mtopology_of_def mtopology_of_submetric)
       ultimately obtain δ where δ>0 and δ: ∀ x ∈ T. mdist m1 (σ n) x < δ
⟶ mdist m2 (f(σ n)) (f x) < inverse (Suc n)
        using ‹T ⊆ mspace m1›
       apply (simp add: mtopology_of_def Metric_space.metric_continuous_map
L.M1.subspace Int_absorb2)
        by (metis inverse_Suc of_nat_Suc)
      have ∃ y ∈ S. mdist m1 (σ n) y < min δ (inverse (Suc n))
        using ‹σ n ∈ T› Tsub ‹δ>0›
      unfolding mtopology_of_def L.M1.metric_closure_of subset_iff mem_Collect_eq
L.M1.in_mball
        by (smt (verit, del_insts) inverse_Suc )
      with δ ‹S ⊆ T› show ?thesis
        by auto
    qed
    then obtain ϱ where ϱS: ⋀n. ϱ n ∈ S and ϱ1: ⋀n. mdist m1 (σ n) (ϱ n) <
inverse (Suc n)
                    and ϱ2: ⋀n. mdist m2 (f (σ n)) (f (ϱ n)) < inverse (Suc n)
      by metis
    have S.MCauchy ϱ
      unfolding S.MCauchy_def
    proof (intro conjI strip)
      show range ϱ ⊆ S ∩ mspace m1
        using ‹S ⊆ mspace m1› by (auto simp: ϱS)
      fix ε :: real
      assume ε>0
```

then obtain M **where** M: $\bigwedge n\ n'.\ M \leq n \Longrightarrow M \leq n' \Longrightarrow mdist\ m1\ (\sigma\ n)\ (\sigma\ n') < \varepsilon/2$
using σ **unfolding** *T.MCauchy_def* **by** (*meson half_gt_zero*)
have $\forall_F\ n$ *in sequentially. inverse* (*Suc n*) $< \varepsilon/4$
using *Archimedean_eventually_inverse* ‹$0 < \varepsilon$› *divide_pos_pos zero_less_numeral* **by** *blast*
then obtain N **where** N: $\bigwedge n.\ N \leq n \Longrightarrow inverse\ (Suc\ n) < \varepsilon/4$
by (*meson eventually_sequentially*)
have $mdist\ m1\ (\varrho\ n)\ (\varrho\ n') < \varepsilon$ **if** $n \geq max\ M\ N$ $n' \geq max\ M\ N$ **for** $n\ n'$
proof −
have $mdist\ m1\ (\varrho\ n)\ (\varrho\ n') \leq mdist\ m1\ (\varrho\ n)\ (\sigma\ n) + mdist\ m1\ (\sigma\ n)\ (\varrho\ n')$
by (*meson T.MCauchy_def T.triangle ϱS σ* ‹$S \subseteq T$› *rangeI subset_iff*)
also have $\ldots \leq mdist\ m1\ (\varrho\ n)\ (\sigma\ n) + mdist\ m1\ (\sigma\ n)\ (\sigma\ n') + mdist\ m1\ (\sigma\ n')\ (\varrho\ n')$
by (*smt* (*verit*, *best*) *T.MCauchy_def T.triangle ϱS σ* ‹$S \subseteq T$› *in_mono rangeI*)
also have $\ldots < \varepsilon/4 + \varepsilon/2 + \varepsilon/4$
using *ϱ1*[*of n*] *ϱ1*[*of n'*] *N*[*of n*] *N*[*of n'*] *that M*[*of n n'*] **by** (*simp add*: *T.commute*)
also have ... $\leq \varepsilon$
by *simp*
finally show *?thesis* **.**
qed
then show $\exists N.\ \forall n\ n'.\ N \leq n \longrightarrow N \leq n' \longrightarrow mdist\ m1\ (\varrho\ n)\ (\varrho\ n') < \varepsilon$
by *blast*
qed
then have *fϱ*: *L.M2.MCauchy* ($f \circ \varrho$)
using *ucf* **by** (*simp add*: *Cauchy_continuous_map_def*)
show *L.M2.MCauchy* ($f \circ \sigma$)
unfolding *L.M2.MCauchy_def*
proof (*intro conjI strip*)
show *range* ($f \circ \sigma$) $\subseteq$ *mspace m2*
using ‹$T \subseteq$ *mspace m1*› *σ cmf*
apply (*auto simp*:)
by (*metis Metric_space.metric_continuous_map Metric_space_mspace_mdist T.MCauchy_def image_eqI inf.absorb1 mspace_submetric mtopology_of_def mtopology_of_submetric range_subsetD subset_iff*)
fix ε :: *real*
assume $\varepsilon > 0$
then obtain M **where** M: $\bigwedge n\ n'.\ M \leq n \Longrightarrow M \leq n' \Longrightarrow mdist\ m2\ ((f \circ \varrho)\ n)\ ((f \circ \varrho)\ n') < \varepsilon/2$
using *fϱ* **unfolding** *L.M2.MCauchy_def* **by** (*meson half_gt_zero*)
have $\forall_F\ n$ *in sequentially. inverse* (*Suc n*) $< \varepsilon/4$
using *Archimedean_eventually_inverse* ‹$0 < \varepsilon$› *divide_pos_pos zero_less_numeral* **by** *blast*
then obtain N **where** N: $\bigwedge n.\ N \leq n \Longrightarrow inverse\ (Suc\ n) < \varepsilon/4$
by (*meson eventually_sequentially*)
have $mdist\ m2\ ((f \circ \sigma)\ n)\ ((f \circ \sigma)\ n') < \varepsilon$ **if** $n \geq max\ M\ N$ $n' \geq max\ M\ N$

for n n'
proof −
have *mdist m2* ((*f* ∘ σ) *n*) ((*f* ∘ σ) *n'*) ≤ *mdist m2* ((*f* ∘ σ) *n*) ((*f* ∘ ϱ) *n*) + *mdist m2* ((*f* ∘ ϱ) *n*) ((*f* ∘ σ) *n'*)
by (*meson L.M2.MCauchy_def* ‹*range* (*f* ∘ σ) ⊆ *mspace m2*› *f*ϱ *mdist_triangle rangeI subset_eq*)
also have ... ≤ *mdist m2* ((*f* ∘ σ) *n*) ((*f* ∘ ϱ) *n*) + *mdist m2* ((*f* ∘ ϱ) *n*) ((*f* ∘ ϱ) *n'*) + *mdist m2* ((*f* ∘ ϱ) *n'*) ((*f* ∘ σ) *n'*)
by (*smt* (*verit*) *L.M2.MCauchy_def L.M2.triangle* ‹*range* (*f* ∘ σ) ⊆ *mspace m2*› *f*ϱ *range_subsetD*)
also have ... < ε/4 + ε/2 + ε/4
using ϱ*2*[*of n*] ϱ*2*[*of n'*] *N*[*of n*] *N*[*of n'*] *that M*[*of n n'*] **by** (*simp add*: *L.M2.commute*)
also have ... ≤ ε
by *simp*
finally show *?thesis* **.**
qed
then show ∃ *N*. ∀ *n n'*. *N* ≤ *n* ⟶ *N* ≤ *n'* ⟶ *mdist m2* ((*f* ∘ σ) *n*) ((*f* ∘ σ) *n'*) < ε
by *blast*
qed
qed
qed

lemma *Cauchy_continuous_map_on_intermediate_closure*:
assumes *Cauchy_continuous_map* (*submetric m1 S*) *m2 f*
and *S* ⊆ *T* **and** *T* ⊆ (*mtopology_of m1*) *closure_of S*
and *continuous_map* (*subtopology* (*mtopology_of m1*) *T*) (*mtopology_of m2*) *f*
shows *Cauchy_continuous_map* (*submetric m1 T*) *m2 f*
by (*metis Cauchy_continuous_map_on_intermediate_closure_aux assms closure_of_subset_topspace order.trans topspace_mtopology_of*)

lemma *Cauchy_continuous_map_extends_to_closure_of*:
assumes *m2*: *mcomplete_of m2*
and *f*: *Cauchy_continuous_map* (*submetric m1 S*) *m2 f*
obtains *g*
where *Cauchy_continuous_map* (*submetric m1* (*mtopology_of m1 closure_of S*)) *m2 g*
⋀*x*. *x* ∈ *S* ⟹ *g x* = *f x*
proof −
obtain *g*
where *g*: *continuous_map* (*subtopology* (*mtopology_of m1*) ((*mtopology_of m1*) *closure_of S*))
(*mtopology_of m2*) *g* (∀ *x* ∈ *S*. *g x* = *f x*)
by (*metis Cauchy_continuous_map_extends_to_continuous_closure_of f m2*)
have *Cauchy_continuous_map* (*submetric m1* (*mtopology_of m1 closure_of S*)) *m2 g*

proof (*rule Cauchy_continuous_map_on_intermediate_closure*)
show *Cauchy_continuous_map* (*submetric m1* (*mspace m1* ∩ *S*)) *m2 g*
by (*smt* (*verit*, *best*) *IntD2 Cauchy_continuous_map_eq f g*(*2*) *inf_commute mspace_submetric submetric_restrict*)
show *mspace m1* ∩ *S* ⊆ *mtopology_of m1 closure_of S*
using *closure_of_subset_Int* **by** *force*
show *mtopology_of m1 closure_of S* ⊆ *mtopology_of m1 closure_of* (*mspace m1* ∩ *S*)
by (*metis closure_of_restrict subset_refl topspace_mtopology_of*)
show *continuous_map* (*subtopology* (*mtopology_of m1*) (*mtopology_of m1 closure_of S*)) (*mtopology_of m2*) *g*
by (*simp add*: *g*)
qed
with *g that* **show** *thesis*
by *metis*
qed

lemma *Cauchy_continuous_map_extends_to_intermediate_closure_of*:
assumes *mcomplete_of m2*
and *Cauchy_continuous_map* (*submetric m1 S*) *m2 f*
and *T* ⊆ *mtopology_of m1 closure_of S*
obtains *g*
where *Cauchy_continuous_map* (*submetric m1 T*) *m2 g* ⋀*x*. *x* ∈ *S* ⟹ *g x* = *f x*
by (*metis Cauchy_continuous_map_extends_to_closure_of Cauchy_continuous_map_from_submetric_mono assms*)

5.11.9 Metric space of bounded functions

context *Metric_space*
begin

definition *fspace* :: *'b set* ⇒ (*'b* ⇒ *'a*) *set* **where**
fspace ≡ λ*S*. {*f*. *f*‘*S* ⊆ *M* ∧ *f* ∈ *extensional S* ∧ *mbounded* (*f*‘*S*)}

definition *fdist* :: [*'b set*, *'b* ⇒ *'a*, *'b* ⇒ *'a*] ⇒ *real* **where**
fdist ≡ λ*S f g*. *if f* ∈ *fspace S* ∧ *g* ∈ *fspace S* ∧ *S* ≠ {}
then Sup ((λ*x*. *d* (*f x*) (*g x*)) ‘ *S*) *else 0*

lemma *fspace_empty* [*simp*]: *fspace* {} = {λ*x*. *undefined*}
by (*auto simp*: *fspace_def*)

lemma *fdist_empty* [*simp*]: *fdist* {} = (λ*x y*. *0*)
by (*auto simp*: *fdist_def*)

lemma *fspace_in_M*: ⟦*f* ∈ *fspace S*; *x* ∈ *S*⟧ ⟹ *f x* ∈ *M*
by (*auto simp*: *fspace_def*)

```
lemma bdd_above_dist:
  assumes f: f ∈ fspace S and g: g ∈ fspace S and S ≠ {}
  shows bdd_above ((λu. d (f u) (g u)) ‘ S)
proof −
  obtain a where a ∈ S
    using ‹S ≠ {}› by blast
  obtain B x C y where B>0  and B: f‘S ⊆ mcball x B
    and C>0 and C: g‘S ⊆ mcball y C
    using f g mbounded_pos by (auto simp: fspace_def)
  have d (f u) (g u) ≤ B + d x y + C if u∈S for u
  proof −
    have f u ∈ M
     by (meson B image_eqI mbounded_mcball mbounded_subset_mspace subsetD
that)
    have g u ∈ M
     by (meson C image_eqI mbounded_mcball mbounded_subset_mspace subsetD
that)
    have x ∈ M y ∈ M
      using B C that by auto
    have d (f u) (g u) ≤ d (f u) x + d x (g u)
      by (simp add: ‹f u ∈ M› ‹g u ∈ M› ‹x ∈ M› triangle)
    also have ... ≤ d (f u) x + d x y + d y (g u)
      by (simp add: ‹f u ∈ M› ‹g u ∈ M› ‹x ∈ M› ‹y ∈ M› triangle)
    also have ... ≤ B + d x y + C
      using B C commute that by fastforce
    finally show ?thesis .
  qed
  then show ?thesis
    by (meson bdd_above.I2)
qed


lemma Metric_space_funspace: Metric_space (fspace S) (fdist S)
proof
  show ∗: 0 ≤ fdist S f g for f g
    by (auto simp: fdist_def intro: cSUP_upper2 [OF bdd_above_dist])
  show fdist S f g = fdist S g f for f g
    by (auto simp: fdist_def commute)
  show (fdist S f g = 0) = (f = g)
    if fg: f ∈ fspace S g ∈ fspace S for f g
  proof
    assume 0: fdist S f g = 0
    show f = g
    proof (cases S={})
      case True
      with 0 that show ?thesis
        by (simp add: fdist_def fspace_def)
    next
      case False
```

with *0 fg* **have** *Sup0*: $(SUP\ x{\in}S.\ d\ (f\ x)\ (g\ x)) = 0$
by (*simp add*: *fdist_def*)
have $d\ (f\ x)\ (g\ x) = 0$ **if** $x{\in}S$ **for** x
by (*smt* (*verit*) *False Sup0* ‹$x{\in}S$› *bdd_above_dist* [*OF fg*] *less_cSUP_iff nonneg*)
with *fg* **show** $f{=}g$
by (*simp add*: *fspace_def extensionalityI image_subset_iff*)
qed
next
show $f = g \Longrightarrow fdist\ S\ f\ g = 0$
using *fspace_in_M* [*OF* ‹$g \in fspace\ S$›] **by** (*auto simp*: *fdist_def*)
qed
show $fdist\ S\ f\ h \leq fdist\ S\ f\ g + fdist\ S\ g\ h$
if *fgh*: $f \in fspace\ S\ g \in fspace\ S\ h \in fspace\ S$ **for** $f\ g\ h$
proof (*clarsimp simp add*: *fdist_def that*)
assume $S \neq \{\}$
have *dfh*: $d\ (f\ x)\ (h\ x) \leq d\ (f\ x)\ (g\ x) + d\ (g\ x)\ (h\ x)$ **if** $x{\in}S$ **for** x
by (*meson fgh fspace_in_M that triangle*)
have *bdd_fgh*: *bdd_above* $((\lambda x.\ d\ (f\ x)\ (g\ x))\ `\ S)$ *bdd_above* $((\lambda x.\ d\ (g\ x)\ (h\ x))\ `\ S)$
by (*simp_all add*: ‹$S \neq \{\}$› *bdd_above_dist that*)
then obtain $B\ C$ **where** B: $\bigwedge x.\ x{\in}S \Longrightarrow d\ (f\ x)\ (g\ x) \leq B$ **and** C: $\bigwedge x.\ x{\in}S \Longrightarrow d\ (g\ x)\ (h\ x) \leq C$
by (*auto simp*: *bdd_above_def*)
then have $\bigwedge x.\ x{\in}S \Longrightarrow d\ (f\ x)\ (g\ x) + d\ (g\ x)\ (h\ x) \leq B{+}C$
by *force*
then have *bdd*: *bdd_above* $((\lambda x.\ d\ (f\ x)\ (g\ x) + d\ (g\ x)\ (h\ x))\ `\ S)$
by (*auto simp*: *bdd_above_def*)
then have $(SUP\ x{\in}S.\ d\ (f\ x)\ (h\ x)) \leq (SUP\ x{\in}S.\ d\ (f\ x)\ (g\ x) + d\ (g\ x)\ (h\ x))$
by (*metis* (*mono_tags*, *lifting*) *cSUP_mono* ‹$S \neq \{\}$› *dfh*)
also have $\ldots \leq (SUP\ x{\in}S.\ d\ (f\ x)\ (g\ x)) + (SUP\ x{\in}S.\ d\ (g\ x)\ (h\ x))$
by (*simp add*: ‹$S \neq \{\}$› *bdd cSUP_le_iff bdd_fgh add_mono cSup_upper*)
finally show $(SUP\ x{\in}S.\ d\ (f\ x)\ (h\ x)) \leq (SUP\ x{\in}S.\ d\ (f\ x)\ (g\ x)) + (SUP\ x{\in}S.\ d\ (g\ x)\ (h\ x))$ **.**
qed
qed

end

definition *funspace* **where**
$funspace\ S\ m \equiv metric\ (Metric\_space.fspace\ (mspace\ m)\ (mdist\ m)\ S,$
$Metric\_space.fdist\ (mspace\ m)\ (mdist\ m)\ S)$

lemma *mspace_funspace* [*simp*]:
$mspace\ (funspace\ S\ m) = Metric\_space.fspace\ (mspace\ m)\ (mdist\ m)\ S$
by (*simp add*: *Metric_space.Metric_space_funspace Metric_space.mspace_metric funspace_def*)

lemma *mdist_funspace* [*simp*]:
mdist (*funspace S m*) = *Metric_space.fdist* (*mspace m*) (*mdist m*) *S*
by (*simp add*: *Metric_space.Metric_space_funspace Metric_space.mdist_metric funspace_def*)

lemma *funspace_imp_welldefined*:
⟦*f* ∈ *mspace* (*funspace S m*); *x* ∈ *S*⟧ ⟹ *f x* ∈ *mspace m*
by (*simp add*: *Metric_space.fspace_def subset_iff*)

lemma *funspace_imp_extensional*:
f ∈ *mspace* (*funspace S m*) ⟹ *f* ∈ *extensional S*
by (*simp add*: *Metric_space.fspace_def*)

lemma *funspace_imp_bounded_image*:
f ∈ *mspace* (*funspace S m*) ⟹ *Metric_space.mbounded* (*mspace m*) (*mdist m*) (*f* ‘ *S*)
by (*simp add*: *Metric_space.fspace_def*)

lemma *funspace_imp_bounded*:
f ∈ *mspace* (*funspace S m*) ⟹ *S* = {} ∨ (∃ *c B*. ∀ *x* ∈ *S*. *mdist m c* (*f x*) ≤ *B*)
by (*auto simp*: *Metric_space.fspace_def Metric_space.mbounded*)

lemma (**in** *Metric_space*) *funspace_imp_bounded2*:
assumes *f* ∈ *fspace S g* ∈ *fspace S*
obtains *B* **where** ⋀*x*. *x* ∈ *S* ⟹ *d* (*f x*) (*g x*) ≤ *B*
proof −
have *mbounded* (*f* ‘ *S* ∪ *g* ‘ *S*)
using *mbounded_Un assms* **by** (*force simp*: *fspace_def*)
then show *thesis*
by (*metis UnCI imageI mbounded_alt that*)
qed

lemma *funspace_imp_bounded2*:
assumes *f* ∈ *mspace* (*funspace S m*) *g* ∈ *mspace* (*funspace S m*)
obtains *B* **where** ⋀*x*. *x* ∈ *S* ⟹ *mdist m* (*f x*) (*g x*) ≤ *B*
by (*metis Metric_space_mspace_mdist assms mspace_funspace Metric_space.funspace_imp_bounded2*

lemma (**in** *Metric_space*) *funspace_mdist_le*:
assumes *fg*: *f* ∈ *fspace S g* ∈ *fspace S* **and** *S* ≠ {}
shows *fdist S f g* ≤ *a* ⟷ (∀ *x* ∈ *S*. *d* (*f x*) (*g x*) ≤ *a*)
using *assms bdd_above_dist* [*OF fg*] **by** (*simp add*: *fdist_def cSUP_le_iff*)

lemma *funspace_mdist_le*:
assumes *f* ∈ *mspace* (*funspace S m*) *g* ∈ *mspace* (*funspace S m*) **and** *S* ≠ {}
shows *mdist* (*funspace S m*) *f g* ≤ *a* ⟷ (∀ *x* ∈ *S*. *mdist m* (*f x*) (*g x*) ≤ *a*)
using *assms* **by** (*simp add*: *Metric_space.funspace_mdist_le*)

lemma (**in** *Metric_space*) *mcomplete_funspace*:
assumes *mcomplete*
shows *mcomplete_of* (*funspace S Self*)
proof −
interpret *F*: *Metric_space fspace S fdist S*
by (*simp add*: *Metric_space_funspace*)
show *?thesis*
proof (*cases S={}*)
case *True*
then show *?thesis*
by (*simp add*: *mcomplete_of_def mcomplete_trivial_singleton*)
next
case *False*
show *?thesis*
proof (*clarsimp simp*: *mcomplete_of_def Metric_space.mcomplete_def*)
fix *σ*
assume *σ*: *F.MCauchy σ*
then have *σM*: ⋀*n x*. *x* ∈ *S* ⟹ *σ n x* ∈ *M*
by (*auto simp*: *F.MCauchy_def intro*: *fspace_in_M*)
have *fdist_less*: ∃ *N*. ∀ *n n′*. *N* ≤ *n* ⟶ *N* ≤ *n′* ⟶ *fdist S* (*σ n*) (*σ n′*) < *ε* **if** *ε*>*0* **for** *ε*
using *σ that* **by** (*auto simp*: *F.MCauchy_def*)
have *σext*: ⋀*n*. *σ n* ∈ *extensional S*
using *σ* **unfolding** *F.MCauchy_def* **by** (*auto simp*: *fspace_def*)
have *σbd*: ⋀*n*. *mbounded* (*σ n* ‘ *S*)
using *σ* **unfolding** *F.MCauchy_def* **by** (*simp add*: *fspace_def image_subset_iff*)
have *σin*[*simp*]: *σ n* ∈ *fspace S* **for** *n*
using *F.MCauchy_def σ* **by** *blast*
have *bd2*: ⋀*n n′*. ∃ *B*. ∀ *x* ∈ *S*. *d* (*σ n x*) (*σ n′ x*) ≤ *B*
using *σ* **unfolding** *F.MCauchy_def* **by** (*metis range_subsetD funspace_imp_bounded2*)
have *sup*: ⋀*n n′ x0*. *x0* ∈ *S* ⟹ *d* (*σ n x0*) (*σ n′ x0*) ≤ *Sup* ((λ*x*. *d* (*σ n x*) (*σ n′ x*)) ‘ *S*)
proof (*rule cSup_upper*)
show *bdd_above* ((λ*x*. *d* (*σ n x*) (*σ n′ x*)) ‘ *S*) **if** *x0* ∈ *S* **for** *n n′ x0*
using *that bd2* **by** (*meson bdd_above.I2*)
qed *auto*
have *pcy*: *MCauchy* (λ*n*. *σ n x*) **if** *x* ∈ *S* **for** *x*
unfolding *MCauchy_def*
proof (*intro conjI strip*)
show *range* (λ*n*. *σ n x*) ⊆ *M*
using *σM that* **by** *blast*
fix *ε* :: *real*
assume *ε* > *0*
then obtain *N* **where** *N*: ⋀*n n′*. *N* ≤ *n* ⟶ *N* ≤ *n′* ⟶ *fdist S* (*σ n*) (*σ n′*) < *ε*
using *σ* **by** (*force simp*: *F.MCauchy_def*)
{ fix *n n′*
assume *n*: *N* ≤ *n N* ≤ *n′*

```
      have d (σ n x) (σ n' x) ≤ (SUP x∈S. d (σ n x) (σ n' x))
        using that sup by presburger
      then have d (σ n x) (σ n' x) ≤ fdist S (σ n) (σ n')
        by (simp add: fdist_def ‹S ≠ {}›)
      with N n have d (σ n x) (σ n' x) < ε
        by fastforce
    } then show ∃N. ∀n n'. N ≤ n ⟶ N ≤ n' ⟶ d (σ n x) (σ n' x) < ε
      by blast
  qed
  have ∃l. limitin mtopology (λn. σ n x) l sequentially if x ∈ S for x
    using assms mcomplete_def pcy ‹x ∈ S› by presburger
  then obtain g0 where g0: ⋀x. x ∈ S ⟹ limitin mtopology (λn. σ n x) (g0
x) sequentially
    by metis
  define g where g ≡ restrict g0 S
  have gext: g ∈ extensional S
   and glim: ⋀x. x ∈ S ⟹ limitin mtopology (λn. σ n x) (g x) sequentially
    by (auto simp: g_def g0)
  have gwd: g x ∈ M if x ∈ S for x
    using glim limitin_metric that by blast
  have unif: ∃N. ∀x n. x ∈ S ⟶ N ≤ n ⟶ d (σ n x) (g x) < ε if ε>0 for
ε
  proof −
    obtain N where N: ⋀n n'. N ≤ n ∧ N ≤ n' ⟹ Sup ((λx. d (σ n x) (σ
n' x)) ` S) < ε/2
      using ‹S≠{}› ‹ε>0› fdist_less [of ε/2]
      by (metis (mono_tags) σin fdist_def half_gt_zero)
    show ?thesis
    proof (intro exI strip)
      fix x n
      assume x ∈ S and N ≤ n
      obtain N' where N': ⋀n. N' ≤ n ⟹ σ n x ∈ M ∧ d (σ n x) (g x) <
ε/2
        by (metis ‹0 < ε› ‹x ∈ S› glim half_gt_zero limit_metric_sequentially)
      have d (σ n x) (g x) ≤ d (σ n x) (σ (max N N') x) + d (σ (max N N')
x) (g x)
        using ‹x ∈ S› σM gwd triangle by presburger
      also have ... < ε/2 + ε/2
        by (smt (verit) N N' ‹N ≤ n› ‹x ∈ S› max.cobounded1 max.cobounded2
sup)
      finally show d (σ n x) (g x) < ε by simp
    qed
  qed
  have limitin F.mtopology σ g sequentially
    unfolding F.limit_metric_sequentially
  proof (intro conjI strip)
    obtain N where N: ⋀n n'. N ≤ n ∧ N ≤ n' ⟹ Sup ((λx. d (σ n x) (σ
n' x)) ` S) < 1
      using fdist_less [of 1] ‹S≠{}› by (auto simp: fdist_def)
```

```
    have ⋀x. x ∈ σ N ‘ S ⟹ x ∈ M
      using σM by blast
    obtain a B where a ∈ M and B: ⋀x. x ∈ (σ N) ‘ S ⟹ d a x ≤ B
      by (metis False σM σbd ex_in_conv imageI mbounded_alt_pos)
    have d a (g x) ≤ B+1 if x∈S for x
    proof −
      have d a (g x) ≤ d a (σ N x) + d (σ N x) (g x)
        by (simp add: ‹a ∈ M› σM gwd that triangle)
      also have ... ≤ B+1
      proof −
        have d a (σ N x) ≤ B
          by (simp add: B that)
        moreover
        have False if 1: d (σ N x) (g x) > 1
        proof −
          obtain r where 1 < r and r: r < d (σ N x) (g x)
            using 1 dense by blast
          then obtain N′ where N′: ⋀n. N′ ≤ n ⟹ σ n x ∈ M ∧ d (σ n x)
(g x) < r−1
            using glim [OF ‹x∈S›] by (fastforce simp: limit_metric_sequentially)
            have d (σ N x) (g x) ≤ d (σ N x) (σ (max N N′) x) + d (σ (max N
N′) x) (g x)
              by (metis ‹x ∈ S› σM commute gwd triangle′)
            also have ... < 1 + (r−1)
                 by (smt (verit) N N′ ‹x ∈ S› max.cobounded1 max.cobounded2
max.idem sup)
            finally have d (σ N x) (g x) < r
              by simp
            with r show False
              by linarith
          qed
          ultimately show ?thesis
            by force
        qed
        finally show ?thesis .
      qed
      with gwd ‹a ∈ M› have mbounded (g ‘ S)
        unfolding mbounded by blast
      with gwd gext show g ∈ fspace S
        by (auto simp: fspace_def)
      fix ε::real
      assume ε>0
      then obtain N where ⋀x n. x ∈ S ⟹ N ≤ n ⟹ d (σ n x) (g x) < ε/2
        by (meson unif half_gt_zero)
      then have fdist S (σ n) g ≤ ε/2 if N ≤ n for n
        using ‹g ∈ fspace S› False that
        by (force simp: funspace_mdist_le simp del: divide_const_simps)
      then show ∃N. ∀n≥N. σ n ∈ fspace S ∧ fdist S (σ n) g < ε
           by (metis ‹0 < ε› σin add_strict_increasing field_sum_of_halves
```

half_gt_zero)
qed
then show $\exists x.$ *limitin F.mtopology* σ *x sequentially*
by *blast*
qed
qed
qed

5.11.10 Metric space of continuous bounded functions

definition *cfunspace* **where**
cfunspace X m $\equiv$ *submetric* (*funspace* (*topspace X*) *m*) {*f*. *continuous_map X* (*mtopology_of m*) *f*}

lemma *mspace_cfunspace* [*simp*]:
mspace (*cfunspace X m*) =
{*f*. *f* $\in$ *topspace X* $\to$ *mspace m* $\wedge$ *f* $\in$ *extensional* (*topspace X*) $\wedge$
Metric_space.mbounded (*mspace m*) (*mdist m*) (*f* ‘ (*topspace X*)) $\wedge$
continuous_map X (*mtopology_of m*) *f*}
by (*auto simp*: *cfunspace_def Metric_space.fspace_def*)

lemma *mdist_cfunspace_eq_mdist_funspace*:
mdist (*cfunspace X m*) = *mdist* (*funspace* (*topspace X*) *m*)
by (*auto simp*: *cfunspace_def*)

lemma *cfunspace_subset_funspace*:
mspace (*cfunspace X m*) $\subseteq$ *mspace* (*funspace* (*topspace X*) *m*)
by (*simp add*: *cfunspace_def*)

lemma *cfunspace_mdist_le*:
⟦*f* $\in$ *mspace* (*cfunspace X m*); *g* $\in$ *mspace* (*cfunspace X m*); *topspace X* $\neq$ {}⟧
$\Longrightarrow$ *mdist* (*cfunspace X m*) *f g* $\leq$ *a* $\longleftrightarrow$ ($\forall$ *x* $\in$ *topspace X*. *mdist m* (*f x*) (*g x*) $\leq$ *a*)
by (*simp add*: *cfunspace_def Metric_space.funspace_mdist_le*)

lemma *cfunspace_imp_bounded2*:
assumes *f* $\in$ *mspace* (*cfunspace X m*) *g* $\in$ *mspace* (*cfunspace X m*)
obtains *B* **where** $\bigwedge$*x*. *x* $\in$ *topspace X* $\Longrightarrow$ *mdist m* (*f x*) (*g x*) $\leq$ *B*
by (*metis assms all_not_in_conv cfunspace_mdist_le nle_le*)

lemma *cfunspace_mdist_lt*:
⟦*compactin X* (*topspace X*); *f* $\in$ *mspace* (*cfunspace X m*);
g $\in$ *mspace* (*cfunspace X m*); *mdist* (*cfunspace X m*) *f g* $<$ *a*;
x $\in$ *topspace X*⟧
$\Longrightarrow$ *mdist m* (*f x*) (*g x*) $<$ *a*
by (*metis* (*full_types*) *cfunspace_mdist_le empty_iff less_eq_real_def less_le_not_le*)

lemma *mdist_cfunspace_le*:
assumes *0* $\leq$ *B* **and** *B*: $\bigwedge$*x*. *x* $\in$ *topspace X* $\Longrightarrow$ *mdist m* (*f x*) (*g x*) $\leq$ *B*

shows *mdist* (*cfunspace X m*) *f g* ≤ *B*
proof (*cases X* = *trivial_topology*)
case *True*
then show *?thesis*
by (*simp add*: *Metric_space.fdist_empty* ‹*B*≥*0*› *cfunspace_def*)
next
case *False*
have *bdd*: *bdd_above* ((λ*u*. *mdist m* (*f u*) (*g u*)) ‘ *topspace X*)
by (*meson B bdd_above.I2*)
with *assms bdd* **show** *?thesis*
by (*simp add*: *mdist_cfunspace_eq_mdist_funspace Metric_space.fdist_def cSUP_le_iff*)
qed

lemma *mdist_cfunspace_imp_mdist_le*:
⟦*f* ∈ *mspace* (*cfunspace X m*); *g* ∈ *mspace* (*cfunspace X m*);
mdist (*cfunspace X m*) *f g* ≤ *a*; *x* ∈ *topspace X*⟧ ⟹ *mdist m* (*f x*) (*g x*) ≤ *a*
using *cfunspace_mdist_le* **by** *blast*

lemma *compactin_mspace_cfunspace*:
compactin X (*topspace X*)
⟹ *mspace* (*cfunspace X m*) =
{*f*. (∀ *x* ∈ *topspace X*. *f x* ∈ *mspace m*) ∧
f ∈ *extensional* (*topspace X*) ∧
continuous_map X (*mtopology_of m*) *f*}
by (*auto simp*: *Metric_space.compactin_imp_mbounded image_compactin mtopology_of_def*)

lemma (**in** *Metric_space*) *mcomplete_cfunspace*:
assumes *mcomplete*
shows *mcomplete_of* (*cfunspace X Self*)
proof −
interpret *F*: *Metric_space fspace* (*topspace X*) *fdist* (*topspace X*)
by (*simp add*: *Metric_space_funspace*)
interpret *S*: *Submetric fspace* (*topspace X*) *fdist* (*topspace X*) *mspace* (*cfunspace X Self*)
proof
show *mspace* (*cfunspace X Self*) ⊆ *fspace* (*topspace X*)
by (*metis cfunspace_subset_funspace mdist_Self mspace_Self mspace_funspace*)
qed
show *?thesis*
proof (*cases X* = *trivial_topology*)
case *True*
then show *?thesis*
by (*simp add*: *mcomplete_of_def mcomplete_trivial_singleton mdist_cfunspace_eq_mdist_funspace cong*: *conj_cong*)
next
case *False*

```
    have ∗: continuous_map X mtopology g
      if range σ ⊆ mspace (cfunspace X Self)
        and g: limitin F.mtopology σ g sequentially for σ g
      unfolding continuous_map_to_metric
    proof (intro strip)
      have σ: ⋀n. continuous_map X mtopology (σ n)
        using that by (auto simp: mtopology_of_def)
      fix x and ε::real
      assume x ∈ topspace X and 0 < ε
     then obtain N where N: ⋀n. N ≤ n ⟹ σ n ∈ fspace (topspace X) ∧ fdist
(topspace X) (σ n) g < ε/3
        unfolding mtopology_of_def F.limitin_metric
      by (metis F.limit_metric_sequentially divide_pos_pos g zero_less_numeral)

      then obtain U where openin X U x ∈ U
        and U: ⋀y. y ∈ U ⟹ σ N y ∈ mball (σ N x) (ε/3)
        by (metis Metric_space.continuous_map_to_metric Metric_space_axioms
‹0 < ε› ‹x ∈ topspace X› σ divide_pos_pos zero_less_numeral)
      moreover
      have g y ∈ mball (g x) ε if y∈U for y
      proof −
        have U ⊆ topspace X
          using ‹openin X U› by (simp add: openin_subset)
        have gx: g x ∈ M
          by (meson F.limitin_mspace ‹x ∈ topspace X› fspace_in_M g)
        have y ∈ topspace X
          using ‹U ⊆ topspace X› that by auto
        have gy: g y ∈ M
        by (meson F.limitin_mspace[OF g] ‹U ⊆ topspace X› fspace_in_M subsetD
that)
        have d (g x) (g y) < ε
        proof −
          have ∗: d (σ N x0) (g x0) ≤ ε/3 if x0 ∈ topspace X for x0
          proof −
            have g ∈ fspace (topspace X)
              using F.limit_metric_sequentially g by blast
            with N that have bdd_above ((λx. d (σ N x) (g x)) ‘ topspace X)
              by (force intro: bdd_above_dist)
           then have d (σ N x0) (g x0) ≤ Sup ((λx. d (σ N x) (g x)) ‘ topspace X)
              by (simp add: cSup_upper that)
            also have ... ≤ ε/3
              using g False N ‹g ∈ fspace (topspace X)›
                by (fastforce simp: F.limit_metric_sequentially fdist_def)
            finally show ?thesis .
          qed
          have d (g x) (g y) ≤ d (g x) (σ N x) + d (σ N x) (g y)
            using U gx gy that triangle by force
          also have ... < ε/3 + ε/3 + ε/3
              by (smt (verit) ∗ U gy ‹x ∈ topspace X› ‹y ∈ topspace X› commute
```

in_mball that triangle)
finally show *?thesis* **by** *simp*
qed
with *gx gy* **show** *?thesis* **by** *simp*
qed
ultimately show $\exists U.$ *openin X U* $\wedge$ $x \in U \wedge (\forall y{\in}U.\ g\ y \in$ *mball* $(g\ x)\ \varepsilon)$
by *blast*
qed

have *S.sub.mcomplete*
proof (*rule S.sequentially_closedin_mcomplete_imp_mcomplete*)
show *F.mcomplete*
by (*metis assms mcomplete_funspace mcomplete_of_def mdist_Self mdist_funspace mspace_Self mspace_funspace*)
fix σ *g*
assume *g*: *range* $\sigma \subseteq$ *mspace* (*cfunspace X Self*) $\wedge$ *limitin F.mtopology* σ *g sequentially*
show $g \in$ *mspace* (*cfunspace X Self*)
proof (*simp add*: *mtopology_of_def*, *intro conjI*)
show $g \in$ *topspace* $X \to M$ $g \in$ *extensional* (*topspace X*) *mbounded* (*g* ‘ *topspace X*)
using *g F.limitin_mspace* **by** (*force simp*: *fspace_def*)+
show *continuous_map X mtopology g*
using $*$ *g* **by** *blast*
qed
qed
then show *?thesis*
by (*simp add*: *mcomplete_of_def mdist_cfunspace_eq_mdist_funspace*)
qed
qed

5.11.11 Existence of completion for any metric space M as a subspace of M=>R

lemma (**in** *Metric_space*) *metric_completion_explicit*:
obtains $f :: ['a,'a] \Rightarrow$ *real* **and** *S* **where**
$S \subseteq$ *mspace*(*funspace M euclidean_metric*)
mcomplete_of (*submetric* (*funspace M euclidean_metric*) *S*)
$f \in M \to S$
mtopology_of(*funspace M euclidean_metric*) *closure_of f* ‘ *M* $= S$
$\bigwedge x\ y.\ [\![x \in M;\ y \in M]\!]$
$\Longrightarrow$ *mdist* (*funspace M euclidean_metric*) (*f x*) (*f y*) $=$ *d x y*
proof −
define m':: ('*a*⇒*real*) *metric* **where** $m' \equiv$ *funspace M euclidean_metric*
show *thesis*
proof (*cases M* $= \{\}$)
case *True*
then show *?thesis*
using *that* **by** (*simp add*: *mcomplete_of_def mcomplete_trivial*)

```
  next
    case False
    then obtain a where a ∈ M
      by auto
    define f where f ≡ λx. (λu ∈ M. d x u − d a u)
    define S where S ≡ mtopology_of(funspace M euclidean_metric) closure_of
(f ‘ M)
   interpret S: Submetric Met_TC.fspace M Met_TC.fdist M S ∩ Met_TC.fspace
M
       by (simp add: Met_TC.Metric_space_funspace Submetric.intro Submet-
ric_axioms_def)

    have fim: f ‘ M ⊆ mspace m′
    proof (clarsimp simp: m′_def Met_TC.fspace_def)
      fix b
      assume b ∈ M
      then have ⋀c. ⟦c ∈ M⟧ ⟹ |d b c − d a c| ≤ d a b
        by (smt (verit, best) ‹a ∈ M› commute triangle′′)
      then have (λx. d b x − d a x) ‘ M ⊆ cball 0 (d a b)
        by force
      then show f b ∈ extensional M ∧ bounded (f b ‘ M)
          by (metis bounded_cball bounded_subset f_def image_restrict_eq re-
strict_extensional_sub set_eq_subset)
    qed
    show thesis
    proof
      show S ⊆ mspace (funspace M euclidean_metric)
        by (simp add: S_def in_closure_of subset_iff)
      have closedin S.mtopology (S ∩ Met_TC.fspace M)
        by (simp add: S_def closedin_Int funspace_def)
      moreover have S.mcomplete
        using Metric_space.mcomplete_funspace Met_TC.Metric_space_axioms
by (fastforce simp: mcomplete_of_def)
      ultimately show mcomplete_of (submetric (funspace M euclidean_metric)
S)
        by (simp add: S.closedin_eq_mcomplete mcomplete_of_def)
      show f ∈ M → S
        using S_def fim in_closure_of m′_def by fastforce
      show mtopology_of (funspace M euclidean_metric) closure_of f ‘ M = S
        by (auto simp: f_def S_def mtopology_of_def)
      show mdist (funspace M euclidean_metric) (f x) (f y) = d x y
        if x ∈ M y ∈ M for x y
      proof −
        have ∀ c∈M. dist (f x c) (f y c) ≤ r ⟹ d x y ≤ r for r
          using that by (auto simp: f_def dist_real_def)
        moreover have dist (f x z) (f y z) ≤ r if d x y ≤ r and z ∈ M for r z
          using that ‹x ∈ M› ‹y ∈ M›
          apply (simp add: f_def Met_TC.fdist_def dist_real_def)
          by (smt (verit, best) commute triangle′)
```

```
      ultimately have (SUP c ∈ M. dist (f x c) (f y c)) = d x y
        by (intro cSup_unique) auto
      with that fim show ?thesis
          using that fim by (simp add: Met_TC.fdist_def False m'_def im-
age_subset_iff)
    qed
  qed
 qed
qed


lemma (in Metric_space) metric_completion:
 obtains f :: ['a,'a] ⇒ real and m' where
   mcomplete_of m'
   f ∈ M → mspace m'
   mtopology_of m' closure_of f ‘ M = mspace m'
   ⋀x y. ⟦x ∈ M; y ∈ M⟧ ⟹ mdist m' (f x) (f y) = d x y
proof −
 obtain f :: ['a,'a] ⇒ real and S where
   Ssub: S ⊆ mspace(funspace M euclidean_metric)
   and mcom: mcomplete_of (submetric (funspace M euclidean_metric) S)
   and fim: f ∈ M → S
   and eqS: mtopology_of(funspace M euclidean_metric) closure_of f ‘ M = S
   and eqd: ⋀x y. ⟦x ∈ M; y ∈ M⟧ ⟹ mdist (funspace M euclidean_metric) (f
x) (f y) = d x y
   using metric_completion_explicit by metis
 define m' where m' ≡ submetric (funspace M euclidean_metric) S
 show thesis
 proof
   show mcomplete_of m'
     by (simp add: mcom m'_def)
   show f ∈ M → mspace m'
     using Ssub fim m'_def by auto
   show mtopology_of m' closure_of f ‘ M = mspace m'
     using eqS fim Ssub
       by (force simp: m'_def mtopology_of_submetric closure_of_subtopology
Int_absorb1 image_subset_iff_funcset)
   show mdist m' (f x) (f y) = d x y if x ∈ M and y ∈ M for x y
     using that eqd m'_def by force
 qed
qed

lemma metrizable_space_completion:
 assumes metrizable_space X
 obtains f :: ['a,'a] ⇒ real and Y where
      completely_metrizable_space Y embedding_map X Y f
           Y closure_of (f ‘ (topspace X)) = topspace Y
proof −
 obtain M d where Metric_space M d and Xeq: X = Metric_space.mtopology
```

M d
  using *assms metrizable_space_def* **by** *blast*
 then interpret *Metric_space M d* **by** *simp*
 obtain *f* :: ['*a*,'*a*] ⇒ *real* **and** *m'* **where**
  mcomplete_of m'
  and *fim*: *f* ∈ *M* → *mspace m'*
  and *m'*: *mtopology_of m' closure_of f ' M = mspace m'*
  and *eqd*: ⋀*x y*. ⟦*x* ∈ *M*; *y* ∈ *M*⟧ ⟹ *mdist m'* (*f x*) (*f y*) = *d x y*
  by (*metis metric_completion*)
 show *thesis*
 proof
  show *completely_metrizable_space* (*mtopology_of m'*)
   using ‹*mcomplete_of m'*›
  unfolding *completely_metrizable_space_def mcomplete_of_def mtopology_of_def*
   by (*metis Metric_space_mspace_mdist*)
  show *embedding_map X* (*mtopology_of m'*) *f*
   using *Metric_space12.isometry_imp_embedding_map*
  by (*metis Metric_space12_def Metric_space_axioms Metric_space_mspace_mdist Xeq eqd fim*
        mtopology_of_def)
  show (*mtopology_of m'*) *closure_of f ' topspace X = topspace* (*mtopology_of m'*)
   by (*simp add*: *Xeq m'*)
 qed
qed

5.11.12 Contractions

lemma (**in** *Metric_space*) *contraction_imp_unique_fixpoint*:
 assumes *f x = x f y = y*
  and *f* ∈ *M* → *M*
  and *k* < *1*
  and ⋀*x y*. ⟦*x* ∈ *M*; *y* ∈ *M*⟧ ⟹ *d* (*f x*) (*f y*) ≤ *k* ∗ *d x y*
  and *x* ∈ *M y* ∈ *M*
 shows *x = y*
 by (*smt* (*verit*, *ccfv_SIG*) *mdist_pos_less mult_le_cancel_right1 assms*)

Banach Fixed-Point Theorem (aka, Contraction Mapping Principle)

lemma (**in** *Metric_space*) *Banach_fixedpoint_thm*:
 assumes *mcomplete* **and** *M* ≠ {} **and** *fim*: *f* ∈ *M* → *M*
  and *k* < *1*
  and *con*: ⋀*x y*. ⟦*x* ∈ *M*; *y* ∈ *M*⟧ ⟹ *d* (*f x*) (*f y*) ≤ *k* ∗ *d x y*
 obtains *x* **where** *x* ∈ *M f x = x*
proof −
 obtain *a* **where** *a* ∈ *M*
  using ‹*M* ≠ {}› **by** *blast*
 show *thesis*
 proof (*cases* ∀ *x* ∈ *M*. *f x = f a*)
  case *True*

```
    then show ?thesis
      by (metis ‹a ∈ M› fim image_subset_iff image_subset_iff_funcset that)
  next
    case False
    then obtain b where b ∈ M and b: f b ≠ f a
      by blast
    have k>0
      using Lipschitz_coefficient_pos [where f=f]
      by (metis False ‹a ∈ M› con fim mdist_Self mspace_Self)
    define σ where σ ≡ λn. (f^^n) a
    have f_iter: σ n ∈ M for n
      unfolding σ_def by (induction n) (use ‹a ∈ M› fim in auto)
    show ?thesis
    proof (cases f a = a)
      case True
      then show ?thesis
        using ‹a ∈ M› that by blast
    next
      case False
      have MCauchy σ
      proof −
        show ?thesis
          unfolding MCauchy_def
        proof (intro conjI strip)
          show range σ ⊆ M
            using f_iter by blast
          fix ε::real
          assume ε>0
          with ‹k < 1› ‹f a ≠ a› ‹a ∈ M› fim have gt0: ((1 − k) ∗ ε) / d a (f a)
> 0
            by (fastforce simp: divide_simps Pi_iff)
          obtain N where k^N < ((1−k) ∗ ε) / d a (f a)
            using real_arch_pow_inv [OF gt0 ‹k < 1›] by blast
          then have N: ⋀n. n ≥ N ⟹ k^n < ((1−k) ∗ ε) / d a (f a)
            by (smt (verit) ‹0 < k› assms(4) power_decreasing)
          have ∀ n n'. n<n' ⟶ N ≤ n ⟶ N ≤ n' ⟶ d (σ n) (σ n') < ε
          proof (intro exI strip)
            fix n n'
            assume n<n' N ≤ n N ≤ n'
            have d (σ n) (σ n') ≤ (∑ i=n..<n'. d (σ i) (σ (Suc i)))
            proof −
              have n < m ⟹ d (σ n) (σ m) ≤ (∑ i=n..<m. d (σ i) (σ (Suc i)))
for m
              proof (induction m)
                case 0
                then show ?case
                  by simp
              next
                case (Suc m)
```

```
      then consider n<m | m=n
        by linarith
      then show ?case
      proof cases
        case 1
       have d (σ n) (σ (Suc m)) ≤ d (σ n) (σ m) + d (σ m) (σ (Suc m))
          by (simp add: f_iter triangle)
         also have ... ≤ (∑ i=n..<m. d (σ i) (σ (Suc i))) + d (σ m) (σ
(Suc m))
          using Suc 1 by linarith
        also have ... = (∑ i = n..<Suc m. d (σ i) (σ (Suc i)))
          using 1 by force
        finally show ?thesis .
      qed auto
    qed
    with ‹n < n'› show ?thesis by blast
  qed
  also have ... ≤ (∑ i=n..<n'. d a (f a) * k^i)
  proof (rule sum_mono)
    fix i
    assume i ∈ {n..<n'}
    show d (σ i) (σ (Suc i)) ≤ d a (f a) * k ^ i
    proof (induction i)
      case 0
      then show ?case
        by (auto simp: σ_def)
    next
      case (Suc i)
      have d (σ (Suc i)) (σ (Suc (Suc i))) ≤ k * d (σ i) (σ (Suc i))
        using con σ_def f_iter fim by fastforce
      also have ... ≤ d a (f a) * k ^ Suc i
        using Suc ‹0 < k› by auto
      finally show ?case .
    qed
  qed
  also have ... = d a (f a) * (∑ i=n..<n'. k^i)
    by (simp add: sum_distrib_left)
  also have ... = d a (f a) * (∑ i=0..<n'−n. k^(i+n))
     using sum.shift_bounds_nat_ivl [of power k 0 n n'−n] ‹n < n'› by
simp
  also have ... = d a (f a) * k^n * (∑ i<n'−n. k^i)
    by (simp add: power_add lessThan_atLeast0 flip: sum_distrib_right)
  also have ... = d a (f a) * (k ^ n − k ^ n') / (1 − k)
    using ‹k < 1› ‹n < n'› apply (simp add: sum_gp_strict)
    by (simp add: algebra_simps flip: power_add)
  also have ... < ε
    using N ‹k < 1› ‹0 < ε› ‹0 < k› ‹N ≤ n›
    apply (simp add: field_simps)
        by (smt (verit) nonneg pos_less_divide_eq zero_less_divide_iff
```

zero_less_power)
finally show $d\ (\sigma\ n)\ (\sigma\ n') < \varepsilon$ **.**
qed
then show $\exists N.\ \forall n\ n'.\ N \leq n \longrightarrow N \leq n' \longrightarrow d\ (\sigma\ n)\ (\sigma\ n') < \varepsilon$
by (*metis* ‹*0* < *ε*› *commute f_iter linorder_not_le local.mdist_zero nat_less_le*)
qed
qed
then obtain *l* **where** *l*: *limitin mtopology σ l sequentially*
using ‹*mcomplete*› *mcomplete_def* **by** *blast*
show *?thesis*
proof
show $l \in M$
using *l limitin_mspace* **by** *blast*
show $f\ l = l$
proof (*rule limitin_metric_unique*)
have *limitin mtopology* $(f \circ \sigma)$ $(f\ l)$ *sequentially*
proof (*rule continuous_map_limit*)
have *Lipschitz_continuous_map Self Self f*
using *con* **by** (*auto simp*: *Lipschitz_continuous_map_def fim*)
then show *continuous_map mtopology mtopology f*
using *Lipschitz_continuous_imp_continuous_map Self_def* **by** *force*
qed (*use l* **in** *auto*)
moreover have $(f \circ \sigma) = (\lambda i.\ \sigma(i+1))$
by (*auto simp*: *σ_def*)
ultimately show *limitin mtopology* $(\lambda n.\ (f \widehat{\ } n)a)$ $(f\ l)$ *sequentially*
using *limitin_sequentially_offset_rev* [*of mtopology σ 1*]
by (*simp add*: *σ_def*)
qed (*use l* **in** ‹*auto simp*: *σ_def*›)
qed
qed
qed
qed

5.11.13 The Baire Category Theorem

Possibly relevant to the theorem "Baire" in Elementary Normed Spaces

lemma (**in** *Metric_space*) *metric_Baire_category*:
assumes *mcomplete countable* $\mathcal{G}$
and $\bigwedge T.\ T \in \mathcal{G} \Longrightarrow$ *openin mtopology T* $\wedge$ *mtopology closure_of T* $= M$
shows *mtopology closure_of* $\bigcap \mathcal{G} = M$
proof (*cases* $\mathcal{G}$={})
case *False*
then obtain $U :: nat \Rightarrow$ *'a set* **where** *U*: *range U* $= \mathcal{G}$
by (*metis* ‹*countable* $\mathcal{G}$› *uncountable_def*)
with *assms* **have** *u_open*: $\bigwedge n.$ *openin mtopology* $(U\ n)$ **and** *u_dense*: $\bigwedge n.$ *mtopology closure_of* $(U\ n) = M$
by *auto*
have $\bigcap(\textit{range } U) \cap W \neq \{\}$ **if** *W*: *openin mtopology W* $W \neq \{\}$ **for** *W*

proof −
have $W \subseteq M$
using *openin_mtopology W* **by** *blast*
have $\exists r'\ x'.\ 0 < r' \wedge r' < r/2 \wedge x' \in M \wedge mcball\ x'\ r' \subseteq mball\ x\ r \cap U\ n$
if $r{>}0\ x \in M$ **for** $x\ r\ n$
proof −
obtain z **where** $z{:}\ z \in U\ n \cap mball\ x\ r$
using *u_dense* [*of n*] ‹$r{>}0$› ‹$x \in M$›
by (*metis dense_intersects_open centre_in_mball_iff empty_iff openin_mball topspace_mtopology equals0I*)
then have $z \in M$ **by** *auto*
have *openin mtopology* $(U\ n \cap mball\ x\ r)$
by (*simp add*: *openin_Int u_open*)
with ‹$z \in M$› z **obtain** e **where** $e{>}0$ **and** $e{:}\ mcball\ z\ e \subseteq U\ n \cap mball\ x\ r$
by (*meson openin_mtopology_mcball*)
define r' **where** $r' \equiv min\ e\ (r/4)$
show *?thesis*
proof (*intro exI conjI*)
show $0 < r'\ r' < r\ /\ 2\ z \in M$
using ‹$e{>}0$› ‹$r{>}0$› ‹$z \in M$› **by** (*auto simp*: *r'_def*)
show $mcball\ z\ r' \subseteq mball\ x\ r \cap U\ n$
using *Metric_space.mcball_subset_concentric e r'_def* **by** *auto*
qed
qed
then obtain *nextr nextx*
where *nextr*: $\bigwedge r\ x\ n.\ [\![r{>}0;\ x{\in}M]\!] \Longrightarrow 0 < nextr\ r\ x\ n \wedge nextr\ r\ x\ n < r/2$
and *nextx*: $\bigwedge r\ x\ n.\ [\![r{>}0;\ x{\in}M]\!] \Longrightarrow nextx\ r\ x\ n \in M$
and *nextsub*: $\bigwedge r\ x\ n.\ [\![r{>}0;\ x{\in}M]\!] \Longrightarrow mcball\ (nextx\ r\ x\ n)\ (nextr\ r\ x\ n) \subseteq mball\ x\ r \cap U\ n$
by *metis*
obtain *x0* **where** *x0*: $x0 \in U\ 0 \cap W$
by (*metis W dense_intersects_open topspace_mtopology all_not_in_conv u_dense*)
then have $x0 \in M$
using ‹$W \subseteq M$› **by** *fastforce*
obtain *r0* **where** $0 < r0\ r0 < 1$ **and** *sub*: $mcball\ x0\ r0 \subseteq U\ 0 \cap W$
proof −
have *openin mtopology* $(U\ 0 \cap W)$
using *W u_open* **by** *blast*
then obtain r **where** $r{>}0$ **and** $r{:}\ mball\ x0\ r \subseteq U\ 0\ mball\ x0\ r \subseteq W$
by (*meson Int_subset_iff openin_mtopology x0*)
define *r0* **where** $r0 \equiv (min\ r\ 1)\ /\ 2$
show *thesis*
proof
show $0 < r0\ r0 < 1$
using ‹$r{>}0$› **by** (*auto simp*: *r0_def*)
show $mcball\ x0\ r0 \subseteq U\ 0 \cap W$
using r ‹$0 < r0$› *r0_def* **by** *auto*
qed

```
    qed
    define b where b ≡ rec_nat (x0,r0) (λn (x,r). (nextx r x n, nextr r x n))
    have b0[simp]: b 0 = (x0,r0)
      by (simp add: b_def)
     have bSuc[simp]: b (Suc n) = (let (x,r) = b n in (nextx r x n, nextr r x n))
for n
      by (simp add: b_def)
    define xf where xf ≡ fst ∘ b
    define rf where rf ≡ snd ∘ b
    have rfxf: 0 < rf n ∧ xf n ∈ M for n
    proof (induction n)
      case 0
      with ‹0 < r0› ‹x0 ∈ M› show ?case
        by (auto simp: rf_def xf_def)
    next
      case (Suc n)
      then show ?case
        by (auto simp: rf_def xf_def case_prod_unfold nextr nextx Let_def)
    qed
    have mcball_sub: mcball (xf (Suc n)) (rf (Suc n)) ⊆ mball (xf n) (rf n) ∩ U
n for n
      using rfxf nextsub by (auto simp: xf_def rf_def case_prod_unfold Let_def)
    have half: rf (Suc n) < rf n / 2 for n
      using rfxf nextr by (auto simp: xf_def rf_def case_prod_unfold Let_def)
    then have decseq rf
     using rfxf by (smt (verit, ccfv_threshold) decseq_SucI field_sum_of_halves)
    have nested: mball (xf n) (rf n) ⊆ mball (xf m) (rf m) if m ≤ n for m n
      using that
    proof (induction n)
      case (Suc n)
      then show ?case
      by (metis mcball_sub order.trans inf.boundedE le_Suc_eq mball_subset_mcball
order.refl)
    qed auto
    have MCauchy xf
      unfolding MCauchy_def
    proof (intro conjI strip)
      show range xf ⊆ M
        using rfxf by blast
      fix ε :: real
      assume ε>0
      then obtain N where N: inverse (2^N) < ε
        using real_arch_pow_inv by (force simp flip: power_inverse)
      have d (xf n) (xf n') < ε if n ≤ n' N ≤ n N ≤ n' for n n'
      proof −
        have ∗: rf n < inverse (2 ^ n) for n
        proof (induction n)
          case 0
          then show ?case
```

```
        by (simp add: ‹r0 < 1› rf_def)
      next
        case (Suc n)
        with half show ?case
          by simp (smt (verit))
      qed
      have rf n ≤ rf N
        using ‹decseq rf› ‹N ≤ n› by (simp add: decseqD)
      moreover
      have xf n' ∈ mball (xf n) (rf n)
        using nested rfxf ‹n ≤ n'› by blast
      ultimately have d (xf n) (xf n') < rf N
        by auto
      also have ... < ε
        using ∗ N order.strict_trans by blast
      finally show ?thesis .
    qed
    then show ∃N. ∀n n'. N ≤ n ⟶ N ≤ n' ⟶ d (xf n) (xf n') < ε
      by (metis commute linorder_le_cases)
  qed
  then obtain l where l: limitin mtopology xf l sequentially
    using ‹mcomplete› mcomplete_alt by blast
  have l_in: l ∈ mcball (xf n) (rf n) for n
  proof −
    have ∀F m in sequentially. xf m ∈ mcball (xf n) (rf n)
      unfolding eventually_sequentially
     by (meson nested rfxf centre_in_mball_iff mball_subset_mcball subset_iff)
    with l limitin_closedin show ?thesis
      by (metis closedin_mcball trivial_limit_sequentially)
  qed
  then have ⋀n. l ∈ U n
    using mcball_sub by blast
  moreover have l ∈ W
    using l_in[of 0] sub by (auto simp: xf_def rf_def)
  ultimately show ?thesis by auto
 qed
 with U show ?thesis
  by (metis dense_intersects_open topspace_mtopology)
qed auto



lemma (in Metric_space) metric_Baire_category_alt:
 assumes mcomplete countable 𝒢
  and empty: ⋀T. T ∈ 𝒢 ⟹ closedin mtopology T ∧ mtopology interior_of T
= {}
 shows mtopology interior_of ⋃𝒢 = {}
proof −
 have ∗: mtopology closure_of ⋂((−)M ‘ 𝒢) = M
```

```
  proof (intro metric_Baire_category conjI ‹mcomplete›)
    show countable ((−) M ‘ 𝒢)
      using ‹countable 𝒢› by blast
    fix T
    assume T ∈ (−) M ‘ 𝒢
    then obtain U where U: U ∈ 𝒢 T = M−U U ⊆ M
      using empty metric_closedin_iff_sequentially_closed by force
    with empty show openin mtopology T by blast
    show mtopology closure_of T = M
      using U by (simp add: closure_of_interior_of double_diff empty)
  qed
  with closure_of_eq show ?thesis
    by (fastforce simp: interior_of_closure_of split: if_split_asm)
qed
```

Since all locally compact Hausdorff spaces are regular, the disjunction in the HOL Light version is redundant.

```
lemma Baire_category_aux:
  assumes locally_compact_space X regular_space X
    and countable 𝒢
    and empty: ⋀G. G ∈ 𝒢 ⟹ closedin X G ∧ X interior_of G = {}
  shows X interior_of ⋃𝒢 = {}
proof (cases 𝒢 = {})
  case True
  then show ?thesis
    by simp
next
  case False
  then obtain T :: nat ⇒ 'a set where T: 𝒢 = range T
    by (metis ‹countable 𝒢› uncountable_def)
  with empty have Tempty: ⋀n. X interior_of (T n) = {}
    by auto
  show ?thesis
  proof (clarsimp simp: T interior_of_def)
    fix z U
    assume z ∈ U and openA: openin X U and Asub: U ⊆ ⋃ (range T)
    with openin_subset have z ∈ topspace X
      by blast
    have neighbourhood_base_of (λC. compactin X C ∧ closedin X C) X
      using assms locally_compact_regular_space_neighbourhood_base by auto
    then obtain V K where openin X V compactin X K closedin X K z ∈ V V
⊆ K K ⊆ U
      by (metis (no_types, lifting) ‹z ∈ U› neighbourhood_base_of openA)
    have nb_closedin: neighbourhood_base_of (closedin X) X
      using ‹regular_space X› neighbourhood_base_of_closedin by auto
    have ∃Φ. ∀n. (Φ n ⊆ K ∧ closedin X (Φ n) ∧ X interior_of Φ n ≠ {} ∧
disjnt (Φ n) (T n)) ∧
          Φ (Suc n) ⊆ Φ n
    proof (rule dependent_nat_choice)
```

show $\exists x{\subseteq}K.$ *closedin X x* $\wedge$ *X interior_of x* $\neq \{\}$ $\wedge$ *disjnt x* (*T 0*)
proof −
have *False* **if** $V \subseteq T\ 0$
using *Tempty* ‹*openin X V*› ‹$z \in V$› *interior_of_maximal that* **by** *fastforce*
then obtain *x* **where** *openin X* ($V - T\ 0$) $\wedge$ $x \in V - T\ 0$
using *T* ‹*openin X V*› *empty* **by** *blast*
with *nb_closedin*
obtain *N C* **where** *openin X N closedin X C x* $\in$ *N N* $\subseteq$ *C C* $\subseteq$ *V* − *T 0*
unfolding *neighbourhood_base_of* **by** *metis*
show *?thesis*
proof (*intro exI conjI*)
show *C* $\subseteq$ *K*
using ‹*C* $\subseteq$ *V* − *T 0*› ‹*V* $\subseteq$ *K*› **by** *auto*
show *X interior_of C* $\neq \{\}$
by (*metis* ‹*N* $\subseteq$ *C*› ‹*openin X N*› ‹*x* $\in$ *N*› *empty_iff interior_of_eq_empty*)
show *disjnt C* (*T 0*)
using ‹*C* $\subseteq$ *V* − *T 0*› *disjnt_iff* **by** *fastforce*
qed (*use* ‹*closedin X C*› **in** *auto*)
qed
show $\exists L.$ (*L* $\subseteq$ *K* $\wedge$ *closedin X L* $\wedge$ *X interior_of L* $\neq \{\}$ $\wedge$ *disjnt L* (*T* (*Suc n*))) $\wedge$ *L* $\subseteq$ *C*
if §: *C* $\subseteq$ *K* $\wedge$ *closedin X C* $\wedge$ *X interior_of C* $\neq \{\}$ $\wedge$ *disjnt C* (*T n*)
for *C n*
proof −
have *False* **if** *X interior_of C* $\subseteq$ *T* (*Suc n*)
by (*metis Tempty interior_of_eq_empty* § *openin_interior_of that*)
then obtain *x* **where** *openin X* (*X interior_of C* − *T* (*Suc n*)) $\wedge$ *x* $\in$ *X interior_of C* − *T* (*Suc n*)
using *T empty* **by** *fastforce*
with *nb_closedin*
obtain *N D* **where** *openin X N closedin X D x* $\in$ *N N* $\subseteq$ *D* **and** *D*: *D* $\subseteq$ *X interior_of C* − *T*(*Suc n*)
unfolding *neighbourhood_base_of* **by** *metis*
show *?thesis*
proof (*intro conjI exI*)
show *D* $\subseteq$ *K*
using *D interior_of_subset* § **by** *fastforce*
show *X interior_of D* $\neq \{\}$
by (*metis* ‹*N* $\subseteq$ *D*› ‹*openin X N*› ‹*x* $\in$ *N*› *empty_iff interior_of_eq_empty*)
show *disjnt D* (*T* (*Suc n*))
using *D disjnt_iff* **by** *fastforce*
show *D* $\subseteq$ *C*
using *interior_of_subset* [*of X C*] *D* **by** *blast*
qed (*use* ‹*closedin X D*› **in** *auto*)
qed
qed
then obtain Φ **where** Φ: $\bigwedge n.$ Φ *n* $\subseteq$ *K* $\wedge$ *closedin X* (Φ *n*) $\wedge$ *X interior_of* Φ *n* $\neq \{\}$ $\wedge$ *disjnt* (Φ *n*) (*T n*)
and $\bigwedge n.$ Φ (*Suc n*) $\subseteq$ Φ *n* **by** *metis*

then have *decseq* Φ
by (*simp add*: *decseq_SucI*)
moreover have $\bigwedge n.\ \Phi\ n \neq \{\}$
by (*metis* Φ *bot.extremum_uniqueI interior_of_subset*)
ultimately have $\bigcap(\mathit{range}\ \Phi) \neq \{\}$
by (*metis* Φ *compact_space_imp_nest* ‹*compactin X K*› *compactin_subspace closedin_subset_topspace*)
moreover have $U \subseteq \{y.\ \exists x.\ y \in T\ x\}$
using *Asub* **by** *auto*
with T **have** $\{a.\ \forall n.\ \ a \in \Phi\ n\} \subseteq \{\}$
by (*smt* (*verit*) *Asub* Φ *Collect_empty_eq UN_iff* ‹$K \subseteq U$› *disjnt_iff subset_iff*)
ultimately show *False*
by *blast*
qed
qed

lemma *Baire_category_alt*:
assumes *completely_metrizable_space X* ∨ *locally_compact_space X* ∧ *regular_space X*
and *countable* $\mathcal{G}$
and $\bigwedge T.\ T \in \mathcal{G} \Longrightarrow \mathit{closedin}\ X\ T \wedge X\ \mathit{interior\_of}\ T = \{\}$
shows *X interior_of* $\bigcup\mathcal{G} = \{\}$
using *Baire_category_aux* [*of X* $\mathcal{G}$] *Metric_space.metric_Baire_category_alt*
by (*metis assms completely_metrizable_space_def*)

lemma *Baire_category*:
assumes *completely_metrizable_space X* ∨ *locally_compact_space X* ∧ *regular_space X*
and *countable* $\mathcal{G}$
and *top*: $\bigwedge T.\ T \in \mathcal{G} \Longrightarrow \mathit{openin}\ X\ T \wedge X\ \mathit{closure\_of}\ T = \mathit{topspace}\ X$
shows *X closure_of* $\bigcap\mathcal{G} = \mathit{topspace}\ X$
proof (*cases* $\mathcal{G}$={})
case *False*
have ∗: *X interior_of* $\bigcup((-)(\mathit{topspace}\ X)\ `\ \mathcal{G}) = \{\}$
proof (*intro Baire_category_alt conjI assms*)
show *countable* $((-)\ (\mathit{topspace}\ X)\ `\ \mathcal{G})$
using *assms* **by** *blast*
fix T
assume $T \in (-)\ (\mathit{topspace}\ X)\ `\ \mathcal{G}$
then obtain U **where** U: $U \in \mathcal{G}\ T = (\mathit{topspace}\ X) - U\ U \subseteq (\mathit{topspace}\ X)$
by (*meson top image_iff openin_subset*)
then show *closedin X T*
by (*simp add*: *closedin_diff top*)
show *X interior_of T* = {}
using *U top* **by** (*simp add*: *interior_of_closure_of double_diff*)
qed

then show *?thesis*
by (*simp add*: *closure_of_eq_topspace interior_of_complement*)
qed *auto*

5.11.14 Sierpinski-Hausdorff type results about countable closed unions

lemma *locally_connected_not_countable_closed_union*:
assumes *topspace* $X \neq \{\}$ **and** *csX*: *connected_space* X
and *lcX*: *locally_connected_space* X
and X: *completely_metrizable_space* $X \vee$ *locally_compact_space* $X \wedge$ *Hausdorff_space* X
and *countable* $\mathcal{U}$ **and** *pwU*: *pairwise disjnt* $\mathcal{U}$
and *clo*: $\bigwedge C.\ C \in \mathcal{U} \Longrightarrow$ *closedin* $X\ C \wedge C \neq \{\}$
and *UU_eq*: $\bigcup \mathcal{U} =$ *topspace* X
shows $\mathcal{U} = \{$*topspace* $X\}$
proof −
define $\mathcal{V}$ **where** $\mathcal{V} \equiv$ (*frontier_of*) X ‘ $\mathcal{U}$
define B **where** $B \equiv \bigcup \mathcal{V}$
then have *Bsub*: $B \subseteq$ *topspace* X
by (*simp add*: *Sup_le_iff* $\mathcal{V}$*_def closedin_frontier_of closedin_subset*)
have *allsub*: $A \subseteq$ *topspace* X **if** $A \in \mathcal{U}$ **for** A
by (*meson clo closedin_def that*)
show *?thesis*
proof (*rule ccontr*)
assume $\mathcal{U} \neq \{$*topspace* $X\}$
with *assms* **have** $\exists A{\in}\mathcal{U}.\ \neg$ (*closedin* $X\ A \wedge$ *openin* $X\ A$)
by (*metis Union_empty connected_space_clopen_in singletonI subsetI subset_singleton_iff*)
then have $B \neq \{\}$
by (*auto simp*: B*_def* $\mathcal{V}$*_def frontier_of_eq_empty allsub*)
moreover
have *subtopology* $X\ B$ *interior_of* $B = B$
by (*simp add*: *Bsub interior_of_openin openin_subtopology_refl*)
ultimately have *int_B_nonempty*: *subtopology* $X\ B$ *interior_of* $B \neq \{\}$
by *auto*
have *subtopology* $X\ B$ *interior_of* $\bigcup \mathcal{V} = \{\}$
proof (*intro Baire_category_alt conjI*)
have $\bigcup \mathcal{U} \subseteq B \cup \bigcup$((*interior_of*) X ‘ $\mathcal{U}$)
using *clo closure_of_closedin* **by** (*fastforce simp*: B*_def* $\mathcal{V}$*_def frontier_of_def*)
moreover have $B \cup \bigcup$((*interior_of*) X ‘ $\mathcal{U}$) $\subseteq \bigcup \mathcal{U}$
using *allsub clo frontier_of_subset_eq interior_of_subset* **by** (*fastforce simp*: B*_def* $\mathcal{V}$*_def*)
moreover have *disjnt* B ($\bigcup$((*interior_of*) X ‘ $\mathcal{U}$))
using *pwU*
apply (*clarsimp simp*: B*_def* $\mathcal{V}$*_def frontier_of_def pairwise_def disjnt_iff*)
by (*metis clo closure_of_eq interior_of_subset subsetD*)
ultimately have $B =$ *topspace* $X - \bigcup$ ((*interior_of*) X ‘ $\mathcal{U}$)

by (*auto simp*: *UU_eq disjnt_iff*)
then have *closedin X B*
by *fastforce*
with *X* **show** *completely_metrizable_space* (*subtopology X B*) ∨ *locally_compact_space* (*subtopology X B*) ∧ *regular_space* (*subtopology X B*)
by (*metis completely_metrizable_space_closedin locally_compact_Hausdorff_or_regular*
locally_compact_space_closed_subset regular_space_subtopology)
show *countable* $\mathcal{V}$
by (*simp add*: $\mathcal{V}$*_def* ‹*countable* $\mathcal{U}$›)
fix *V*
assume $V \in \mathcal{V}$
then obtain *S* **where** *S*: $S \in \mathcal{U}$ $V = X$ *frontier_of S*
by (*auto simp*: $\mathcal{V}$*_def*)
show *closedin* (*subtopology X B*) *V*
by (*metis B_def Sup_upper* $\mathcal{V}$*_def* ‹$V \in \mathcal{V}$› *closedin_frontier_of closedin_subset_topspace image_iff*)
have *subtopology X B interior_of* (*X frontier_of S*) = {}
proof (*clarsimp simp*: *interior_of_def openin_subtopology_alt*)
fix *a U*
assume $a \in B$ $a \in U$ **and** *openU*: *openin X U* **and** *BUsub*: $B \cap U \subseteq X$ *frontier_of S*
then have $a \in S$
by (*meson IntI* ‹$S \in \mathcal{U}$› *clo frontier_of_subset_closedin subsetD*)
then obtain *W C* **where** *openin X W connectedin X C* $a \in W$ $W \subseteq C$ $C \subseteq U$
by (*metis* ‹$a \in U$› *lcX locally_connected_space openU*)
have $W \cap X$ *frontier_of S* ≠ {}
using ‹$B \cap U \subseteq X$ *frontier_of S*› ‹$a \in B$› ‹$a \in U$› ‹$a \in W$› **by** *auto*
with *frontier_of_openin_straddle_Int*
obtain $W \cap S \neq \{\}$ $W - S \neq \{\}$ $W \subseteq$ *topspace X*
using ‹*openin X W*› **by** (*metis openin_subset*)
then obtain *b* **where** $b \in$ *topspace X* $b \in W - S$
by *blast*
with *UU_eq* **obtain** *T* **where** $T \in \mathcal{U}$ $T \neq S$ $W \cap T \neq \{\}$
by *auto*
then have *disjnt S T*
by (*metis* ‹$S \in \mathcal{U}$› *pairwise_def pwU*)
then have $C - T \neq \{\}$
by (*meson Diff_eq_empty_iff* ‹$W \subseteq C$› ‹$a \in S$› ‹$a \in W$› *disjnt_iff subsetD*)
then have $C \cap X$ *frontier_of T* ≠ {}
using ‹$W \cap T \neq \{\}$› ‹$W \subseteq C$› ‹*connectedin X C*› *connectedin_Int_frontier_of*
by *blast*
moreover have $C \cap X$ *frontier_of T* = {}
proof −
have *X frontier_of S* ⊆ *S X frontier_of T* ⊆ *T*
using *frontier_of_subset_closedin* ‹$S \in \mathcal{U}$› ‹$T \in \mathcal{U}$› *clo* **by** *blast*+
moreover have *X frontier_of T* ∪ *B* = *B*

```
          using B_def V_def ‹T ∈ U› by blast
        ultimately show ?thesis
          using BUsub ‹C ⊆ U› ‹disjnt S T› unfolding disjnt_def by blast
      qed
      ultimately show False
        by simp
    qed
    with S show subtopology X B interior_of V = {}
      by meson
  qed
  then show False
    using B_def int_B_nonempty by blast
 qed
qed

lemma real_Sierpinski_lemma:
 fixes a b::real
 assumes a ≤ b
   and countable U and pwU: pairwise disjnt U
   and clo: ⋀C. C ∈ U ⟹ closed C ∧ C ≠ {}
   and ⋃U = {a..b}
 shows U = {{a..b}}
proof −
 have locally_connected_space (top_of_set {a..b})
   by (simp add: locally_connected_real_interval)
 moreover
 have completely_metrizable_space (top_of_set {a..b})
   by (metis box_real(2) completely_metrizable_space_cbox)
 ultimately
 show ?thesis
  using locally_connected_not_countable_closed_union [of subtopology euclidean
{a..b}] assms
   apply (simp add: closedin_subtopology)
   by (metis Union_upper inf.orderE)
qed
```

5.11.15 The Tychonoff embedding

```
lemma completely_regular_space_cube_embedding_explicit:
 assumes completely_regular_space X Hausdorff_space X
 shows embedding_map X
          (product_topology (λf. top_of_set {0..1::real})
            (mspace (submetric (cfunspace X euclidean_metric)
              {f. f ∈ topspace X → {0..1}})) )
          (λx. λf ∈ mspace (submetric (cfunspace X euclidean_metric) {f. f ∈
topspace X → {0..1}}).
           f x)
proof −
 define K where K ≡ mspace(submetric (cfunspace X euclidean_metric) {f. f
```

$\in$ *topspace X* $\to$ {*0..1*::*real*}})
define *e* **where** *e* $\equiv$ $\lambda x.$ $\lambda f{\in}K.$ *f x*
have *e x* $\neq$ *e y* **if** *xy*: *x* $\neq$ *y x* $\in$ *topspace X y* $\in$ *topspace X* **for** *x y*
proof $-$
have *closedin X* {*x*}
by (*simp add*: ‹*Hausdorff_space X*› *closedin_Hausdorff_singleton* ‹*x* $\in$ *topspace X*›)
then obtain *f* **where** *contf*: *continuous_map X euclideanreal f*
and *f01*: $\bigwedge x.$ *x* $\in$ *topspace X* $\Longrightarrow$ *f x* $\in$ {*0..1*} **and** *fxy*: *f y* = *0 f x* = *1*
using ‹*completely_regular_space X*› *xy* **unfolding** *completely_regular_space_def*
by (*smt* (*verit*, *ccfv_threshold*) *Diff_iff continuous_map_in_subtopology image_subset_iff singleton_iff*)
then have *bounded* (*f* ‘ *topspace X*)
by (*meson bounded_closed_interval bounded_subset image_subset_iff*)
with *contf f01* **have** *restrict f* (*topspace X*) $\in$ *K*
by (*auto simp*: *K_def*)
with *fxy xy* **show** *?thesis*
unfolding *e_def* **by** (*metis restrict_apply′ zero_neq_one*)
qed
then have *inj_on e* (*topspace X*)
by (*meson inj_onI*)
then obtain *e′* **where** *e′*: $\bigwedge x.$ *x* $\in$ *topspace X* $\Longrightarrow$ *e′* (*e x*) = *x*
by (*metis inv_into_f_f*)
have *continuous_map* (*subtopology* (*product_topology* ($\lambda f.$ *top_of_set* {*0..1*}) *K*) (*e* ‘ *topspace X*)) *X e′*
proof (*clarsimp simp add*: *continuous_map_atin limitin_atin openin_subtopology_alt e′*)
fix *x U*
assume *e x* $\in$ *K* $\to_E$ {*0..1*} **and** *x* $\in$ *topspace X* **and** *openin X U* **and** *x* $\in$ *U*
then obtain *g* **where** *contg*: *continuous_map X* (*top_of_set* {*0..1*}) *g* **and** *g x* = *0*
and *gim*: *g* ‘ (*topspace X* $-$ *U*) $\subseteq$ {*1*::*real*}
using ‹*completely_regular_space X*› **unfolding** *completely_regular_space_def*

by (*metis Diff_iff openin_closedin_eq*)
then have *bounded* (*g* ‘ *topspace X*)
by (*meson bounded_closed_interval bounded_subset continuous_map_in_subtopology*)
moreover have *g* $\in$ *topspace X* $\to$ {*0..1*}
using *contg* **by** (*simp add*: *continuous_map_def*)
ultimately have *g_in_K*: *restrict g* (*topspace X*) $\in$ *K*
using *contg* **by** (*force simp add*: *K_def continuous_map_in_subtopology*)
have *openin* (*top_of_set* {*0..1*}) {*0..<1*::*real*}
using *open_real_greaterThanLessThan*[*of* $-$*1 1*] **by** (*force simp*: *openin_open*)
moreover have *e x* $\in$ (Π_E *f*$\in$*K*. *if f* = *restrict g* (*topspace X*) *then* {*0..<1*} *else* {*0..1*})
using ‹*e x* $\in$ *K* $\to_E$ {*0..1*}› **by** (*simp add*: *e_def* ‹*g x* = *0*› ‹*x* $\in$ *topspace X*› *PiE_iff*)
moreover have *e y* = *e x*
if *y* $\notin$ *U* **and** *ey*: *e y* $\in$ (Π_E *f*$\in$*K*. *if f* = *restrict g* (*topspace X*) *then* {*0..<1*}

else $\{0..1\}$)
        and *y*: $y \in$ *topspace X* **for** *y*
  proof −
    have *e y* (*restrict g* (*topspace X*)) $\in \{0..<1\}$
      using *ey* **by** (*smt* (*verit*, *ccfv_SIG*) *PiE_mem g_in_K*)
  with *gim g_in_K y* ‹$y \notin U$› **show** *?thesis*
    by (*fastforce simp*: *e_def*)
  qed
  ultimately
  show $\exists W$. *openin* (*product_topology* (λf. *top_of_set* $\{0..1\}$) *K*) *W* $\wedge$ *e x* $\in$ *W* $\wedge$ *e′* ‘ (*e* ‘ *topspace X* $\cap$ *W* − $\{$*e x*$\}$) $\subseteq$ *U*
    apply (*rule_tac x*=*PiE K* (λf. *if f* = *restrict g* (*topspace X*) *then* $\{0..<1\}$ *else* $\{0..1\}$) **in** *exI*)
    by (*auto simp*: *openin_PiE_gen e′*)
 qed
 with *e′* **have** *embedding_map X* (*product_topology* (λf. *top_of_set* $\{0..1\}$) *K*) *e*
  unfolding *embedding_map_def homeomorphic_map_maps homeomorphic_maps_def*
    by (*fastforce simp*: *e_def K_def continuous_map_in_subtopology continuous_map_componentwise*)
 then show *?thesis*
  by (*simp add*: *K_def e_def*)
qed

lemma *completely_regular_space_cube_embedding*:
 fixes *X* :: *′a topology*
 assumes *completely_regular_space X Hausdorff_space X*
 obtains *K*:: (*′a*⇒*real*)*set* **and** *e*
  where *embedding_map X* (*product_topology* (λf. *top_of_set* $\{0..1$::*real*$\}$) *K*) *e*
 using *completely_regular_space_cube_embedding_explicit* [*OF assms*] **by** *metis*

5.11.16 Urysohn and Tietze analogs for completely regular spaces

"Urysohn and Tietze analogs for completely regular spaces if (()) set is assumed compact instead of closed. Note that Hausdorffness is *not* required: inside () proof we factor through the Kolmogorov quotient." – John Harrison

lemma *Urysohn_completely_regular_closed_compact*:
 fixes *a b*::*real*
 assumes $a \le b$ *completely_regular_space X closedin X S compactin X T disjnt S T*
 obtains *f* **where** *continuous_map X* (*subtopology euclidean* $\{a..b\}$) *f f* ‘ *T* $\subseteq$ $\{a\}$ *f* ‘ *S* $\subseteq$ $\{b\}$
proof −
 obtain *f* **where** *contf*: *continuous_map X* (*subtopology euclideanreal* $\{0..1\}$) *f*
  and *f0*: *f* ‘ *T* $\subseteq$ $\{0\}$ **and** *f1*: *f* ‘ *S* $\subseteq$ $\{1\}$

```
  proof (cases T={})
    case True
    show thesis
    proof
      show continuous_map X (top_of_set {0..1}) (λx. 1::real) (λx. 1::real) ‘ T
⊆ {0} (λx. 1::real) ‘ S ⊆ {1}
        using True by auto
    qed
  next
    case False
   have ⋀t. t ∈ T ⟹ ∃f. continuous_map X (subtopology euclideanreal ({0..1}))
f ∧ f t = 0 ∧ f ‘ S ⊆ {1}
      using assms unfolding completely_regular_space_def
      by (meson DiffI compactin_subset_topspace disjnt_iff subset_eq)
    then obtain g where contg: ⋀t. t ∈ T ⟹ continuous_map X (subtopology
euclideanreal {0..1}) (g t)
                  and g0: ⋀t. t ∈ T ⟹ g t t = 0
                  and g1: ⋀t. t ∈ T ⟹ g t ‘ S ⊆ {1}
      by metis
    then have g01: ⋀t. t ∈ T ⟹ g t ‘ topspace X ⊆ {0..1}
      by (meson continuous_map_in_subtopology)
    define G where G ≡ λt. {x ∈ topspace X. g t x ∈ {..<1/2}}
    have Ball (G‘T) (openin X)
      using contg unfolding G_def continuous_map_in_subtopology
       by (smt (verit, best) Collect_cong openin_continuous_map_preimage im-
age_iff open_lessThan open_openin)
    moreover have T ⊆ ⋃(G‘T)
     using ‹compactin X T› g0 compactin_subset_topspace by (force simp: G_def)
    ultimately have ∃ℱ. finite ℱ ∧ ℱ ⊆ G‘T ∧ T ⊆ ⋃ ℱ
      using ‹compactin X T› unfolding compactin_def by blast
    then obtain K where K: finite K K ⊆ T T ⊆ ⋃(G‘K)
      by (metis finite_subset_image)
    with False have K ≠ {}
      by fastforce
    define f where f ≡ λx. 2 * max 0 (Inf ((λt. g t x) ‘ K) - 1/2)
    have [simp]: max 0 (x - 1/2) = 0 ⟷ x ≤ 1/2 for x::real
      by force
    have [simp]: 2 * max 0 (x - 1/2) = 1 ⟷ x = 1 for x::real
      by (simp add: max_def_raw)
    show thesis
    proof
      have g t s = 1 if s ∈ S t ∈ K for s t
        using ‹K ⊆ T› g1 that by auto
      then show f ‘ S ⊆ {1}
        using ‹K ≠ {}› by (simp add: f_def image_subset_iff)
      have (INF t∈K. g t x) ≤ 1/2 if x ∈ T for x
      proof -
        obtain k where k ∈ K g k x < 1/2
          using K ‹x ∈ T› by (auto simp: G_def)
```

```
      then show ?thesis
       by (meson ‹finite K› cInf_le_finite dual_order.trans finite_imageI imageI
less_le_not_le)
    qed
    then show f ‘ T ⊆ {0}
      by (force simp: f_def)
    have ⋀t. t ∈ K ⟹ continuous_map X euclideanreal (g t)
      using ‹K ⊆ T› contg continuous_map_in_subtopology by blast
    moreover have 2 * max 0 ((INF t∈K. g t x) − 1/2) ≤ 1 if x ∈ topspace
X for x
    proof −
      obtain k where k ∈ K g k x ≤ 1
        using K ‹x ∈ topspace X› ‹K ≠ {}› g01 by (fastforce simp: G_def)
      then have (INF t∈K. g t x) ≤ 1
      by (meson ‹finite K› cInf_le_finite dual_order.trans finite_imageI imageI)
      then show ?thesis
        by (simp add: max_def_raw)
    qed
    ultimately show continuous_map X (top_of_set {0..1}) f
      by (force simp: f_def continuous_map_in_subtopology intro!: ‹finite K›
continuous_intros)
  qed
 qed
 define g where g ≡ λx. a + (b − a) * f x
 show thesis
 proof
  have ∀ x∈topspace X. a + (b − a) * f x ≤ b
    using contf ‹a ≤ b› apply (simp add: continuous_map_in_subtopology
image_subset_iff)
    by (smt (verit, best) mult_right_le_one_le)
  then show continuous_map X (top_of_set {a..b}) g
    using contf ‹a ≤ b› unfolding g_def continuous_map_in_subtopology im-
age_subset_iff
    by (intro conjI continuous_intros; simp)
  show g ‘ T ⊆ {a} g ‘ S ⊆ {b}
    using f0 f1 by (auto simp: g_def)
 qed
qed


lemma Urysohn_completely_regular_compact_closed:
 fixes a b::real
 assumes a ≤ b completely_regular_space X compactin X S closedin X T disjnt
S T
 obtains f where continuous_map X (subtopology euclidean {a..b}) f f ‘ T ⊆
{a} f ‘ S ⊆ {b}
proof −
 obtain f where contf: continuous_map X (subtopology euclidean {−b..−a}) f
and fim: f ‘ T ⊆ {−a} f ‘ S ⊆ {−b}
```

by (*meson Urysohn_completely_regular_closed_compact assms disjnt_sym neg_le_iff_le*)
show *thesis*
proof
show *continuous_map X* (*top_of_set* {*a*..*b*}) (*uminus* ∘ *f*)
using *contf* **by** (*auto simp*: *continuous_map_in_subtopology o_def*)
show (*uminus o f*) ‘ *T* ⊆ {*a*} (*uminus o f*) ‘ *S* ⊆ {*b*}
using *fim* **by** *fastforce+*
qed
qed

lemma *Urysohn_completely_regular_compact_closed_alt*:
fixes *a b*::*real*
assumes *completely_regular_space X compactin X S closedin X T disjnt S T*
obtains *f* **where** *continuous_map X euclideanreal f f* ‘ *T* ⊆ {*a*} *f* ‘ *S* ⊆ {*b*}
proof (*cases a b rule*: *le_cases*)
case *le*
then show *?thesis*
by (*meson Urysohn_completely_regular_compact_closed assms continuous_map_into_fulltopology that*)
next
case *ge*
then show *?thesis*
using *Urysohn_completely_regular_closed_compact assms*
by (*metis Urysohn_completely_regular_closed_compact assms continuous_map_into_fulltopology disjnt_sym that*)
qed

lemma *Tietze_extension_comp_reg_aux*:
fixes *T* :: *real set*
assumes *completely_regular_space X Hausdorff_space X compactin X S*
and *T*: *is_interval T T*≠{}
and *contf*: *continuous_map* (*subtopology X S*) *euclidean f* **and** *fim*: *f*‘*S* ⊆ *T*
obtains *g* **where** *continuous_map X euclidean g g* ‘ *topspace X* ⊆ *T* ⋀*x*. *x* ∈ *S* ⟹ *g x* = *f x*
proof −
obtain *K*:: (*'a*⇒*real*)*set* **and** *e*
where *e0*: *embedding_map X* (*product_topology* (λ*f*. *top_of_set* {*0*..*1*::*real*}) *K*) *e*
using *assms completely_regular_space_cube_embedding* **by** *blast*
define *cube* **where** *cube* ≡ *product_topology* (λ*f*. *top_of_set* {*0*..*1*::*real*}) *K*
have *e*: *embedding_map X cube e*
using *e0* **by** (*simp add*: *cube_def*)
obtain *e'* **where** *e'*: *homeomorphic_maps X* (*subtopology cube* (*e* ‘ *topspace X*)) *e e'*
using *e* **by** (*force simp*: *cube_def embedding_map_def homeomorphic_map_maps*)
then have *conte*: *continuous_map X* (*subtopology cube* (*e* ‘ *topspace X*)) *e*
and *conte'*: *continuous_map* (*subtopology cube* (*e* ‘ *topspace X*)) *X e'*

```
    and e'e: ∀x∈topspace X. e' (e x) = x
   by (auto simp: homeomorphic_maps_def)
  have Hausdorff_space cube
   unfolding cube_def
    using Hausdorff_space_euclidean Hausdorff_space_product_topology Haus-
dorff_space_subtopology by blast
  have normal_space cube
  proof (rule compact_Hausdorff_or_regular_imp_normal_space)
   show compact_space cube
    unfolding cube_def
     using compact_space_product_topology compact_space_subtopology com-
pactin_euclidean_iff by blast
  qed (use ‹Hausdorff_space cube› in auto)
  moreover
  have comp: compactin cube (e ‘ S)
   by (meson ‹compactin X S› conte continuous_map_in_subtopology image_compactin)
  then have closedin cube (e ‘ S)
   by (intro compactin_imp_closedin ‹Hausdorff_space cube›)
  moreover
  have continuous_map (subtopology cube (e ‘ S)) euclideanreal (f ∘ e')
  proof (intro continuous_map_compose)
   show continuous_map (subtopology cube (e ‘ S)) (subtopology X S) e'
    unfolding continuous_map_in_subtopology
   proof
    show continuous_map (subtopology cube (e ‘ S)) X e'
     by (meson ‹compactin X S› compactin_subset_topspace conte' continu-
ous_map_from_subtopology_mono image_mono)
    show e' ‘ topspace (subtopology cube (e ‘ S)) ⊆ S
     using ‹compactin X S› compactin_subset_topspace e'e by fastforce
   qed
  qed (simp add: contf)
  moreover
  have (f ∘ e') ‘ e ‘ S ⊆ T
   using ‹compactin X S› compactin_subset_topspace e'e fim by fastforce
  ultimately
  obtain g where contg: continuous_map cube euclidean g and gsub: g ‘ topspace
cube ⊆ T
          and gf: ⋀x. x ∈ e‘S ⟹ g x = (f ∘ e') x
   using Tietze_extension_realinterval T by metis
  show thesis
  proof
   show continuous_map X euclideanreal (g ∘ e)
    by (meson contg conte continuous_map_compose continuous_map_in_subtopology)
   show (g ∘ e) ‘ topspace X ⊆ T
    using gsub conte continuous_map_image_subset_topspace by fastforce
   fix x
   assume x ∈ S
   then show (g ∘ e) x = f x
    using gf ‹compactin X S› compactin_subset_topspace e'e by fastforce
```

qed
qed

lemma *Tietze_extension_completely_regular*:
assumes *completely_regular_space X compactin X S is_interval T T* ≠ {}
and *contf*: *continuous_map* (*subtopology X S*) *euclidean f* **and** *fim*: *f*‘*S* ⊆ *T*
obtains *g* **where** *continuous_map X euclideanreal g g* ‘ *topspace X* ⊆ *T*
⋀*x*. *x* ∈ *S* ⟹ *g x* = *f x*
proof −
define *Q* **where** *Q* ≡ *Kolmogorov_quotient X* ‘ (*topspace X*)
obtain *g* **where** *contg*: *continuous_map* (*subtopology X* (*Kolmogorov_quotient X* ‘ *S*)) *euclidean g*
and *gf*: ⋀*x*. *x* ∈ *S* ⟹ *g*(*Kolmogorov_quotient X x*) = *f x*
using *Kolmogorov_quotient_lift_exists*
by (*metis* ‹*compactin X S*› *contf compactin_subset_topspace open_openin t0_space_def t1_space*)
have *S* ⊆ *topspace X*
by (*simp add*: ‹*compactin X S*› *compactin_subset_topspace*)
then have [*simp*]: *Q* ∩ *Kolmogorov_quotient X* ‘ *S* = *Kolmogorov_quotient X* ‘ *S*
using *Q_def* **by** *blast*
have *creg*: *completely_regular_space* (*subtopology X Q*)
by (*simp add*: ‹*completely_regular_space X*› *completely_regular_space_subtopology*)
then have *regular_space* (*subtopology X Q*)
by (*simp add*: *completely_regular_imp_regular_space*)
then have *Hausdorff_space* (*subtopology X Q*)
using *Q_def regular_t0_eq_Hausdorff_space t0_space_Kolmogorov_quotient* **by** *blast*
moreover
have *compactin* (*subtopology X Q*) (*Kolmogorov_quotient X* ‘ *S*)
by (*metis Q_def* ‹*compactin X S*› *image_compactin quotient_imp_continuous_map quotient_map_Kolmogorov_quotient*)
ultimately obtain *h* **where** *conth*: *continuous_map* (*subtopology X Q*) *euclidean h*
and *him*: *h* ‘ *topspace* (*subtopology X Q*) ⊆ *T*
and *hg*: ⋀*x*. *x* ∈ *Kolmogorov_quotient X* ‘ *S* ⟹ *h x* = *g x*
using *Tietze_extension_comp_reg_aux* [*of subtopology X Q Kolmogorov_quotient X* ‘ *S T g*]
apply (*simp add*: *subtopology_subtopology creg contg assms*)
using *fim gf* **by** *blast*
show *thesis*
proof
show *continuous_map X euclideanreal* (*h* ∘ *Kolmogorov_quotient X*)
by (*metis Q_def conth continuous_map_compose quotient_imp_continuous_map quotient_map_Kolmogorov_quotient*)
show (*h* ∘ *Kolmogorov_quotient X*) ‘ *topspace X* ⊆ *T*
using *Q_def continuous_map_Kolmogorov_quotient continuous_map_image_subset_topspace him* **by** *fastforce*

```
    fix x
    assume x ∈ S then show (h ∘ Kolmogorov_quotient X) x = f x
      by (simp add: gf hg)
  qed
qed
```

5.11.17 Size bounds on connected or path-connected spaces

```
lemma connected_space_imp_card_ge_alt:
  assumes connected_space X completely_regular_space X closedin X S S ≠ {} S
≠ topspace X
  shows (UNIV::real set) ≲ topspace X
proof −
  have S ⊆ topspace X
    using ‹closedin X S› closedin_subset by blast
  then obtain a where a ∈ topspace X a ∉ S
    using ‹S ≠ topspace X› by blast
  have (UNIV::real set) ≲ {0..1::real}
    using eqpoll_real_subset
    by (meson atLeastAtMost_iff eqpoll_imp_lepoll eqpoll_sym less_eq_real_def
zero_less_one)
  also have ... ≲ topspace X
  proof −
    obtain f where contf: continuous_map X euclidean f
      and fim: f ∈ (topspace X) → {0..1::real}
      and f0: f a = 0 and f1: f ‘ S ⊆ {1}
      using ‹completely_regular_space X›
      unfolding completely_regular_space_def
        by (metis Diff_iff ‹a ∈ topspace X› ‹a ∉ S› ‹closedin X S› continu-
ous_map_in_subtopology image_subset_iff_funcset)
    have ∃y∈topspace X. x = f y if 0 ≤ x and x ≤ 1 for x
    proof −
      have connectedin euclidean (f ‘ topspace X)
        using ‹connected_space X› connectedin_continuous_map_image connecte-
din_topspace contf by blast
      moreover have ∃y. 0 = f y ∧ y ∈ topspace X
        using ‹a ∈ topspace X› f0 by auto
      moreover have ∃y. 1 = f y ∧ y ∈ topspace X
        using ‹S ⊆ topspace X› ‹S ≠ {}› f1 by fastforce
      ultimately show ?thesis
      using that by (fastforce simp: is_interval_1 simp flip: is_interval_connected_1)
    qed
    then show ?thesis
      unfolding lepoll_iff using atLeastAtMost_iff by blast
  qed
  finally show ?thesis .
qed
```

lemma *connected_space_imp_card_ge_gen*:
assumes *connected_space X normal_space X closedin X S closedin X T S* $\neq \{\}$ *T* $\neq \{\}$ *disjnt S T*
shows (*UNIV*::*real set*) $\lesssim$ *topspace X*
proof −
have (*UNIV*::*real set*) $\lesssim \{0..1::real\}$
by (*metis atLeastAtMost_iff eqpoll_real_subset eqpoll_imp_lepoll eqpoll_sym less_le_not_le zero_less_one*)
also have ... $\lesssim$ *topspace X*
proof −
obtain *f* **where** *contf*: *continuous_map X euclidean f*
and *fim*: *f* $\in$ (*topspace X*) $\to \{0..1::real\}$
and *f0*: *f* ‘ *S* $\subseteq \{0\}$ **and** *f1*: *f* ‘ *T* $\subseteq \{1\}$
using *assms* **by** (*metis continuous_map_in_subtopology normal_space_iff_Urysohn image_subset_iff_funcset*)
have $\exists y{\in}$*topspace X*. *x* = *f y* **if** *0* $\le$ *x* **and** *x* $\le$ *1* **for** *x*
proof −
have *connectedin euclidean* (*f* ‘ *topspace X*)
using ‹*connected_space X*› *connectedin_continuous_map_image connectedin_topspace contf* **by** *blast*
moreover have $\exists y.$ *0* = *f y* $\wedge$ *y* $\in$ *topspace X*
using ‹*closedin X S*› ‹*S* $\neq \{\}$› *closedin_subset f0* **by** *fastforce*
moreover have $\exists y.$ *1* = *f y* $\wedge$ *y* $\in$ *topspace X*
using ‹*closedin X T*› ‹*T* $\neq \{\}$› *closedin_subset f1* **by** *fastforce*
ultimately show *?thesis*
using *that* **by** (*fastforce simp*: *is_interval_1 simp flip*: *is_interval_connected_1*)
qed
then show *?thesis*
unfolding *lepoll_iff* **using** *atLeastAtMost_iff* **by** *blast*
qed
finally show *?thesis* **.**
qed

lemma *connected_space_imp_card_ge*:
assumes *connected_space X normal_space X t1_space X* **and** *nosing*: ¬ ($\exists a.$ *topspace X* $\subseteq \{a\}$)
shows (*UNIV*::*real set*) $\lesssim$ *topspace X*
proof −
obtain *a b* **where** *a* $\in$ *topspace X b* $\in$ *topspace X a* $\neq$ *b*
by (*metis nosing singletonI subset_iff*)
then have {*a*} $\neq$ *topspace X*
by *force*
with *connected_space_imp_card_ge_alt assms* **show** *?thesis*
by (*metis* ‹*a* $\in$ *topspace X*› *closedin_t1_singleton insert_not_empty normal_imp_completely_regular_space_A*)
qed

lemma *connected_space_imp_infinite_gen*:
⟦*connected_space X*; *t1_space X*; $\nexists a.$ *topspace X* $\subseteq \{a\}$⟧ $\Longrightarrow$ *infinite*(*topspace*

X)
by (*metis connected_space_discrete_topology finite_t1_space_imp_discrete_topology*)

lemma *connected_space_imp_infinite*:
⟦*connected_space X*; *Hausdorff_space X*; ∄*a*. *topspace X* ⊆ {*a*}⟧ ⟹ *infinite*(*topspace X*)
by (*simp add*: *Hausdorff_imp_t1_space connected_space_imp_infinite_gen*)

lemma *connected_space_imp_infinite_alt*:
assumes *connected_space X regular_space X closedin X S S* ≠ {} *S* ≠ *topspace X*
shows *infinite*(*topspace X*)
proof −
have *S* ⊆ *topspace X*
using ‹*closedin X S*› *closedin_subset* **by** *blast*
then obtain *a* **where** *a*: *a* ∈ *topspace X a* ∉ *S*
using ‹*S* ≠ *topspace X*› **by** *blast*
have ∃Φ. ∀*n*. (*disjnt* (Φ *n*) *S* ∧ *a* ∈ Φ *n* ∧ *openin X* (Φ *n*)) ∧ Φ(*Suc n*) ⊂ Φ *n*
proof (*rule dependent_nat_choice*)
show ∃ *T*. *disjnt T S* ∧ *a* ∈ *T* ∧ *openin X T*
by (*metis Diff_iff a* ‹*closedin X S*› *closedin_def disjnt_iff*)
fix *V n*
assume §: *disjnt V S* ∧ *a* ∈ *V* ∧ *openin X V*
then obtain *U C* **where** *U*: *openin X U closedin X C a* ∈ *U U* ⊆ *C C* ⊆ *V*
using ‹*regular_space X*› **by** (*metis neighbourhood_base_of neighbourhood_base_of_closedin*)
with *assms* **have** *U* ⊂ *V*
by (*metis* § ‹*S* ⊆ *topspace X*› *connected_space_clopen_in disjnt_def empty_iff inf.absorb_iff2 inf.orderE psubsetI subset_trans*)
with *U* **show** ∃ *U*. (*disjnt U S* ∧ *a* ∈ *U* ∧ *openin X U*) ∧ *U* ⊂ *V*
using § *disjnt_subset1* **by** *blast*
qed
then obtain Φ **where** Φ: ⋀*n*. *disjnt* (Φ *n*) *S* ∧ *a* ∈ Φ *n* ∧ *openin X* (Φ *n*)
and Φ*sub*: ⋀*n*. Φ (*Suc n*) ⊂ Φ *n* **by** *metis*
then have *decseq* Φ
by (*simp add*: *decseq_SucI psubset_eq*)
have ∀*n*. ∃*x*. *x* ∈ Φ *n* ∧ *x* ∉ Φ(*Suc n*)
by (*meson* Φ*sub psubsetE subsetI*)
then obtain *f* **where** *fin*: ⋀*n*. *f n* ∈ Φ *n* **and** *fout*: ⋀*n*. *f n* ∉ Φ(*Suc n*)
by *metis*
have *range f* ⊆ *topspace X*
by (*meson* Φ *fin image_subset_iff openin_subset subset_iff*)
moreover have *inj f*
by (*metis Suc_le_eq* ‹*decseq* Φ› *decseq_def fin fout linorder_injI subsetD*)
ultimately show *?thesis*
using *infinite_iff_countable_subset* **by** *blast*
qed

lemma *path_connected_space_imp_card_ge*:
assumes *path_connected_space X Hausdorff_space X* **and** *nosing*: ¬ (∃*x*. *topspace*

$X \subseteq \{x\}$)
shows (*UNIV*::*real set*) $\lesssim$ *topspace X*
proof −
obtain *a b* **where** *a* ∈ *topspace X b* ∈ *topspace X a* ≠ *b*
by (*metis nosing singletonI subset_iff*)
then obtain γ **where** γ: *pathin X* γ γ *0* = *a* γ *1* = *b*
by (*meson* ‹*a* ∈ *topspace X*› ‹*b* ∈ *topspace X*› ‹*path_connected_space X*›
path_connected_space_def)
let *?Y* = *subtopology X* (γ ‘ (*topspace* (*subtopology euclidean* {*0..1*})))
have (*UNIV*::*real set*) $\lesssim$ *topspace ?Y*
proof (*intro compact_Hausdorff_or_regular_imp_normal_space connected_space_imp_card_ge*)
show *connected_space ?Y*
using ‹*pathin X* γ› *connectedin_def connectedin_path_image* **by** *auto*
show *Hausdorff_space ?Y* ∨ *regular_space ?Y*
using *Hausdorff_space_subtopology* ‹*Hausdorff_space X*› **by** *blast*
show *t1_space ?Y*
using *Hausdorff_imp_t1_space* ‹*Hausdorff_space X*› *t1_space_subtopology*
by *blast*
show *compact_space ?Y*
by (*simp add*: ‹*pathin X* γ› *compact_space_subtopology compactin_path_image*)
have *a* ∈ *topspace ?Y b* ∈ *topspace ?Y*
using γ *pathin_subtopology* **by** *fastforce+*
with ‹*a* ≠ *b*› **show** ∄*x*. *topspace ?Y* ⊆ {*x*}
by *blast*
qed
also have ... $\lesssim$ γ ‘ {*0..1*}
by (*simp add*: *subset_imp_lepoll*)
also have ... $\lesssim$ *topspace X*
by (*meson* γ *path_image_subset_topspace subset_imp_lepoll image_subset_iff_funcset*)
finally show *?thesis* **.**
qed

lemma *connected_space_imp_uncountable*:
assumes *connected_space X regular_space X Hausdorff_space X* ¬ (∃ *a*. *topspace*
X ⊆ {*a*})
shows ¬ *countable*(*topspace X*)
proof
assume *coX*: *countable* (*topspace X*)
with ‹*regular_space X*› **have** *normal_space X*
using *countable_imp_Lindelof_space regular_Lindelof_imp_normal_space* **by**
blast
then have (*UNIV*::*real set*) $\lesssim$ *topspace X*
by (*simp add*: *Hausdorff_imp_t1_space assms connected_space_imp_card_ge*)
with *coX* **show** *False*
using *countable_lepoll uncountable_UNIV_real* **by** *blast*
qed

lemma *path_connected_space_imp_uncountable*:
assumes *path_connected_space X t1_space X* **and** *nosing*: ¬ (∃ *a*. *topspace X*

$\subseteq \{a\}$)
shows $\neg$ *countable*(*topspace X*)
proof
assume *coX*: *countable* (*topspace X*)
obtain *a b* **where** $a \in$ *topspace X* $b \in$ *topspace X* $a \neq b$
by (*metis nosing singletonI subset_iff*)
then obtain γ **where** *pathin X* γ γ *0* $= a$ γ *1* $= b$
by (*meson* ‹$a \in$ *topspace X*› ‹$b \in$ *topspace X*› ‹*path_connected_space X*› *path_connected_space_def*)
then have γ ‘ $\{0..1\} \lesssim$ *topspace X*
by (*meson path_image_subset_topspace subset_imp_lepoll image_subset_iff_funcset*)
define $\mathcal{A}$ **where** $\mathcal{A} \equiv ((\lambda a.\ \{x \in \{0..1\}.\ \gamma\ x \in \{a\}\})$ ‘ *topspace X*$) - \{\{\}\}$
have *A01*: $\mathcal{A} = \{\{0..1\}\}$
proof (*rule real_Sierpinski_lemma*)
show *countable* $\mathcal{A}$
using $\mathcal{A}$*_def coX* **by** *blast*
show *disjoint* $\mathcal{A}$
by (*auto simp*: $\mathcal{A}$*_def disjnt_iff pairwise_def*)
show $\bigcup \mathcal{A} = \{0..1\}$
using ‹*pathin X* γ› *path_image_subset_topspace* **by** (*fastforce simp*: $\mathcal{A}$*_def Bex_def*)
fix *C*
assume $C \in \mathcal{A}$
then obtain *a* **where** $a \in$ *topspace X* **and** *C*: $C = \{x \in \{0..1\}.\ \gamma\ x \in \{a\}\}$ $C \neq \{\}$
by (*auto simp*: $\mathcal{A}$*_def*)
then have *closedin X* $\{a\}$
by (*meson* ‹*t1_space X*› *closedin_t1_singleton*)
then have *closedin* (*top_of_set* $\{0..1\}$) *C*
using *C* ‹*pathin X* γ› *closedin_continuous_map_preimage pathin_def* **by** *fastforce*
then show *closed C* $\wedge$ $C \neq \{\}$
using *C closedin_closed_trans* **by** *blast*
qed *auto*
then have $\{0..1\} \in \mathcal{A}$
by *blast*
then have $\exists a \in$ *topspace X*. $\{0..1\} \subseteq \{x.\ \gamma\ x = a\}$
using $\mathcal{A}$*_def image_iff* **by** *auto*
then show *False*
using ‹γ *0* $= a$› ‹γ *1* $= b$› ‹$a \neq b$› *atLeastAtMost_iff zero_less_one_class.zero_le_one* **by** *blast*
qed

5.11.18 Lavrentiev extension etc

lemma (**in** *Metric_space*) *convergent_eq_zero_oscillation_gen*:
assumes *mcomplete* **and** *fim*: $f \in$ (*topspace X* $\cap$ *S*) $\to$ *M*
shows ($\exists$ *l*. *limitin mtopology f l* (*atin_within X a S*)) $\longleftrightarrow$
M $\neq \{\}$ $\wedge$

```
        (a ∈ topspace X
          ⟶ (∀ε>0. ∃ U. openin X U ∧ a ∈ U ∧
                    (∀ x ∈ (S ∩ U) − {a}. ∀ y ∈ (S ∩ U) − {a}. d (f x) (f y) <
ε)))
proof (cases M = {})
  case True
  with limitin_mspace show ?thesis
    by blast
next
  case False
  show ?thesis
  proof (cases a ∈ topspace X)
    case True
    let ?R = ∀ε>0. ∃ U. openin X U ∧ a ∈ U ∧ (∀x∈S ∩ U − {a}. ∀y∈S ∩ U
− {a}. d (f x) (f y) < ε)
    show ?thesis
    proof (cases a ∈ X derived_set_of S)
      case True
      have ?R
        if limitin mtopology f l (atin_within X a S) for l
      proof (intro strip)
        fix ε::real
        assume ε>0
        with that ‹a ∈ topspace X›
        obtain U where U: openin X U a ∈ U l ∈ M
          and Uless: ∀x∈U − {a}. x ∈ S ⟶ f x ∈ M ∧ d (f x) l < ε/2
          unfolding limitin_metric eventually_within_imp eventually_atin
          using half_gt_zero by blast
        show ∃ U. openin X U ∧ a ∈ U ∧ (∀x∈S ∩ U − {a}. ∀y∈S ∩ U − {a}.
d (f x) (f y) < ε)
        proof (intro exI strip conjI)
          fix x y
          assume x: x ∈ S ∩ U − {a} and y: y ∈ S ∩ U − {a}
          then have d (f x) l < ε/2 d (f y) l < ε/2 f x ∈ M f y ∈ M
            using Uless by auto
          then show d (f x) (f y) < ε
            using triangle' ‹l ∈ M› by fastforce
        qed (auto simp add: U)
      qed
      moreover have ∃ l. limitin mtopology f l (atin_within X a S)
        if R [rule_format]: ?R
      proof −
        define F where F ≡ λU. mtopology closure_of f ‘ (S ∩ U − {a})
        define C where C ≡ F ‘ {U. openin X U ∧ a ∈ U}
        have C_clo: ∀ C ∈ C. closedin mtopology C
          by (force simp add: C_def F_def)
        moreover have sub_mcball: ∃ C a. C ∈ C ∧ C ⊆ mcball a ε if ε>0 for ε
        proof −
          obtain U where U: openin X U a ∈ U
```

```
          and Uless: ∀ x∈S ∩ U − {a}. ∀ y∈S ∩ U − {a}. d (f x) (f y) < ε
          using R [OF ‹ε>0›] by blast
        then obtain b where b: b ≠ a b ∈ S b ∈ U
          using True by (auto simp add: in_derived_set_of)
        have U ⊆ topspace X
          by (simp add: U(1) openin_subset)
        have f b ∈ M
          using b ‹openin X U› by (metis image_subset_iff_funcset Int_iff fim
image_eqI openin_subset subsetD)
        moreover
        have mtopology closure_of f ‘ ((S ∩ U) − {a}) ⊆ mcball (f b) ε
        proof (rule closure_of_minimal)
          have f y ∈ M if y ∈ S and y ∈ U for y
            using ‹U ⊆ topspace X› fim that by (auto simp: Pi_iff)
          moreover
          have d (f b) (f y) ≤ ε if y ∈ S y ∈ U y ≠ a for y
            using that Uless b by force
          ultimately show f ‘ (S ∩ U − {a}) ⊆ mcball (f b) ε
            by (force simp: ‹f b ∈ M›)
        qed auto
        ultimately show ?thesis
          using U by (auto simp add: C_def F_def)
      qed
      moreover have ⋂ℱ ≠ {} if finite ℱ ℱ ⊆ 𝒞 for ℱ
      proof −
        obtain 𝒢 where sub: 𝒢 ⊆ {U. openin X U ∧ a ∈ U} and eq: ℱ = F ‘ 𝒢
and finite 𝒢
        by (metis (no_types, lifting) C_def ‹ℱ ⊆ 𝒞› ‹finite ℱ› finite_subset_image)
        then have U ⊆ topspace X if U ∈ 𝒢 for U
          using openin_subset that by auto
        then have T ⊆ mtopology closure_of T
          if T ∈ (λU. f ‘ (S ∩ U − {a})) ‘ 𝒢 for T
          using that fim by (fastforce simp add: intro!: closure_of_subset)
        moreover
        have ain: a ∈ ⋂ (insert (topspace X) 𝒢) openin X (⋂ (insert (topspace
X) 𝒢))
          using True in_derived_set_of sub ‹finite 𝒢› by (fastforce intro!:
openin_Inter)+
        then obtain y where y ≠ a y ∈ S and y: y ∈ ⋂ (insert (topspace X) 𝒢)
          by (meson ‹a ∈ X derived_set_of S› sub in_derived_set_of)
        then have f y ∈ ⋂ℱ
          using eq that ain fim by (auto simp add: F_def image_subset_iff
in_closure_of)
        then show ?thesis by blast
      qed
      ultimately have ⋂𝒞 ≠ {}
        using ‹mcomplete› mcomplete_fip by metis
      then obtain b where b ∈ ⋂𝒞
        by auto
```

then have $b \in M$
using *sub_mcball C_clo mbounded_alt_pos mbounded_empty metric_closedin_iff_sequentially_closed* **by** *force*
have *limitin mtopology f b (atin_within X a S)*
proof (*clarsimp simp*: *limitin_metric* ‹$b \in M$›)
fix ε :: *real*
assume $\varepsilon > 0$
then obtain U **where** U: *openin X U a* $\in U$ **and** *subU*: $U \subseteq$ *topspace X*
and *Uless*: $\forall x{\in}S \cap U - \{a\}.\ \forall y{\in}S \cap U - \{a\}.\ d\ (f\ x)\ (f\ y) < \varepsilon/2$
by (*metis R half_gt_zero openin_subset*)
then obtain x **where** x: $x \in S$ $x \in U$ $x \neq a$ **and** *fx*: $f\ x \in$ *mball b* $(\varepsilon/2)$
using ‹$b \in \bigcap \mathcal{C}$›
apply (*simp add*: *C_def F_def closure_of_def del*: *divide_const_simps*)
by (*metis Diff_iff Int_iff centre_in_mball_iff in_mball openin_mball singletonI zero_less_numeral*)
moreover
have $d\ (f\ y)\ b < \varepsilon$ **if** $y \in U$ $y \neq a$ $y \in S$ **for** y
proof −
have $d\ (f\ x)\ (f\ y) < \varepsilon/2$
using *Uless that x* **by** *force*
moreover have $d\ b\ (f\ x)\ < \varepsilon/2$
using *fx* **by** *simp*
ultimately show *?thesis*
using *triangle* [*of b f x f y*] *subU that* ‹$b \in M$› *commute fim fx* **by** *fastforce*
qed
ultimately show $\forall_F\ x$ *in atin_within X a S*. $f\ x \in M \wedge d\ (f\ x)\ b < \varepsilon$
using *fim U*
apply (*simp add*: *eventually_atin eventually_within_imp del*: *divide_const_simps flip*: *image_subset_iff_funcset*)
by (*smt* (*verit*, *del_insts*) *Diff_iff Int_iff imageI insertI1 openin_subset subsetD*)
qed
then show *?thesis* **..**
qed
ultimately
show *?thesis*
by (*meson True* ‹$M \neq \{\}$› *in_derived_set_of*)
next
case *False*
have $(\exists\, l.$ *limitin mtopology f l (atin_within X a S))*
by (*metis* ‹$M \neq \{\}$› *False derived_set_of_trivial_limit equals0I limitin_trivial topspace_mtopology*)
moreover have $\forall\, e{>}0.\ \exists\, U.$ *openin X U* $\wedge$ $a \in U$ $\wedge$ $(\forall x{\in}S \cap U - \{a\}.\ \forall y{\in}S \cap U - \{a\}.\ d\ (f\ x)\ (f\ y) < e)$
by (*metis Diff_iff False IntE True in_derived_set_of insert_iff*)
ultimately show *?thesis*
using *limitin_mspace* **by** *blast*
qed

```
  next
    case False
    then show ?thesis
      by (metis derived_set_of_trivial_limit ex_in_conv in_derived_set_of limitin_mspace limitin_trivial topspace_mtopology)
  qed
qed
```

The HOL Light proof uses some ugly tricks to share common parts of what are two separate proofs for the two cases

```
lemma (in Metric_space) gdelta_in_points_of_convergence_within:
  assumes mcomplete
    and f: continuous_map (subtopology X S) mtopology f ∨ t1_space X ∧ f ∈ S → M
  shows gdelta_in X {x ∈ topspace X. ∃ l. limitin mtopology f l (atin_within X x S)}
proof −
  have fim: f ∈ (topspace X ∩ S) → M
    using continuous_map_image_subset_topspace f by force
  show ?thesis
  proof (cases M={})
    case True
    then show ?thesis
      by (smt (verit) Collect_cong empty_def empty_iff gdelta_in_empty limitin_mspace)
  next
    case False
    define A where A ≡ {a ∈ topspace X. ∀ε>0. ∃ U. openin X U ∧ a ∈ U ∧ (∀ x∈S ∩ U − {a}. ∀ y∈S ∩ U − {a}. d (f x) (f y) < ε)}
    have gdelta_in X A
      using f
    proof (elim disjE conjE)
      assume cm: continuous_map (subtopology X S) mtopology f
      define C where C ≡ λr. ⋃{U. openin X U ∧ (∀ x ∈ S∩U. ∀ y ∈ S∩U. d (f x) (f y) < r)}
      define B where B ≡ (⋂ n. C(inverse(Suc n)))
      define D where D ≡ (⋂ (C ` {0<..}))
      have D=B
        unfolding B_def C_def D_def
        apply (intro Inter_eq_Inter_inverse_Suc Sup_subset_mono)
        by (smt (verit, ccfv_threshold) Collect_mono_iff)
      have B ⊆ topspace X
        using openin_subset by (force simp add: B_def C_def)
      have (countable intersection_of openin X) B
        unfolding B_def C_def
      by (intro relative_to_inc countable_intersection_of_Inter countable_intersection_of_inc) auto
      then have gdelta_in X B
        unfolding gdelta_in_def by (intro relative_to_subset_inc ‹B ⊆ topspace
```

```
X›)
    moreover have A=D
    proof (intro equalityI subsetI)
      fix a
      assume x: a ∈ A
      then have a ∈ topspace X
        using A_def by blast
      show a ∈ D
      proof (clarsimp simp: D_def C_def ‹a ∈ topspace X›)
        fix ε::real assume ε > 0
        then obtain U where openin X U a ∈ U and U: (∀x∈S ∩ U − {a}.
∀y∈S ∩ U − {a}. d (f x) (f y) < ε)
          using x by (force simp: A_def)
        show ∃T. openin X T ∧ (∀x∈S ∩ T. ∀y∈S ∩ T. d (f x) (f y) < ε) ∧ a
∈ T
        proof (cases a ∈ S)
          case True
          then obtain V where openin X V a ∈ V and V: ∀x. x ∈ S ∧ x ∈ V
⟶ f a ∈ M ∧ f x ∈ M ∧ d (f a) (f x) < ε
            using ‹a ∈ topspace X› ‹ε > 0› cm
          by (force simp add: continuous_map_to_metric openin_subtopology_alt
Ball_def)
          show ?thesis
          proof (intro exI conjI strip)
            show openin X (U ∩ V)
              using ‹openin X U› ‹openin X V› by blast
            show a ∈ U ∩ V
              using ‹a ∈ U› ‹a ∈ V› by blast
           show ⋀x y. ⟦x ∈ S ∩ (U ∩ V); y ∈ S ∩ (U ∩ V)⟧ ⟹ d (f x) (f y) <
ε
              by (metis DiffI Int_iff U V commute singletonD)
          qed
        next
          case False then show ?thesis
            using U ‹a ∈ U› ‹openin X U› by auto
        qed
      qed
    next
      fix x
      assume x: x ∈ D
      then have x ∈ topspace X
        using ‹B ⊆ topspace X› ‹D=B› by blast
      with x show x ∈ A
        apply (clarsimp simp: D_def C_def A_def)
        by (meson DiffD1 greaterThan_iff)
    qed
    ultimately show ?thesis
      by (simp add: ‹D=B›)
  next
```

assume *t1_space X f* ∈ *S* → *M*
define *C* **where** *C* ≡ λ*r*. ⋃{*U*. *openin X U* ∧
(∃ *b* ∈ *topspace X*. ∀ *x* ∈ *S*∩*U* − {*b*}. ∀ *y* ∈ *S*∩*U* − {*b*}. *d* (*f* *x*) (*f y*) < *r*)}
define *B* **where** *B* ≡ (⋂ *n*. *C*(*inverse*(*Suc n*)))
define *D* **where** *D* ≡ (⋂ (*C* ‘ {*0*<..}))
have *D*=*B*
unfolding *B_def C_def D_def*
apply (*intro Inter_eq_Inter_inverse_Suc Sup_subset_mono*)
by (*smt* (*verit*, *ccfv_threshold*) *Collect_mono_iff*)
have *B* ⊆ *topspace X*
using *openin_subset* **by** (*force simp add*: *B_def C_def*)
have (*countable intersection_of openin X*) *B*
unfolding *B_def C_def*
by (*intro relative_to_inc countable_intersection_of_Inter countable_intersection_of_inc*) *auto*
then have *gdelta_in X B*
unfolding *gdelta_in_def* **by** (*intro relative_to_subset_inc* ‹*B* ⊆ *topspace X*›)
moreover have *A*=*D*
proof (*intro equalityI subsetI*)
fix *x*
assume *x*: *x* ∈ *D*
then have *x* ∈ *topspace X*
using ‹*B* ⊆ *topspace X*› ‹*D*=*B*› **by** *blast*
show *x* ∈ *A*
proof (*clarsimp simp*: *A_def* ‹*x* ∈ *topspace X*›)
fix ε :: *real*
assume ε>*0*
then obtain *U b* **where** *openin X U b* ∈ *topspace X*
and *U*: ∀ *x*∈*S* ∩ *U* − {*b*}. ∀ *y*∈*S* ∩ *U* − {*b*}. *d* (*f x*) (*f y*) < ε **and** *x* ∈ *U*
using *x* **by** (*auto simp*: *D_def C_def A_def Ball_def*)
then have *openin X* (*U*−{*b*})
by (*meson* ‹*t1_space X*› *t1_space_openin_delete_alt*)
then show ∃ *U*. *openin X U* ∧ *x* ∈ *U* ∧ (∀ *xa*∈*S* ∩ *U* − {*x*}. ∀ *y*∈*S* ∩ *U* − {*x*}. *d* (*f xa*) (*f y*) < ε)
using *U* ‹*openin X U*› ‹*x* ∈ *U*› **by** *auto*
qed
next
show ⋀*x*. *x* ∈ *A* ⟹ *x* ∈ *D*
unfolding *A_def D_def C_def*
by *clarsimp meson*
qed
ultimately show *?thesis*
by (*simp add*: ‹*D*=*B*›)
qed
then show *?thesis*
by (*simp add*: *A_def convergent_eq_zero_oscillation_gen False fim* ‹*mcom-*

plete› *cong*: *conj_cong*)
qed
qed

lemma *gdelta_in_points_of_convergence_within*:
assumes *Y*: *completely_metrizable_space Y*
and *f*: *continuous_map* (*subtopology X S*) *Y f* ∨ *t1_space X* ∧ *f* ∈ *S* →
topspace Y
shows *gdelta_in X* {*x* ∈ *topspace X*. ∃ *l*. *limitin Y f l* (*atin_within X x S*)}
using *assms*
unfolding *completely_metrizable_space_def*
using *Metric_space.gdelta_in_points_of_convergence_within Metric_space.topspace_mtopology*
by *fastforce*

lemma *Lavrentiev_extension_gen*:
assumes *S* ⊆ *topspace X* **and** *Y*: *completely_metrizable_space Y*
and *contf*: *continuous_map* (*subtopology X S*) *Y f*
obtains *U g* **where** *gdelta_in X U S* ⊆ *U*
continuous_map (*subtopology X* (*X closure_of S* ∩ *U*)) *Y g*
⋀*x*. *x* ∈ *S* ⟹ *g x* = *f x*
proof −
define *U* **where** *U* ≡ {*x* ∈ *topspace X*. ∃ *l*. *limitin Y f l* (*atin_within X x S*)}
have *S* ⊆ *U*
using *that contf limit_continuous_map_within subsetD* [*OF* ‹*S* ⊆ *topspace X*›]

by (*fastforce simp*: *U_def*)
then have *S* ⊆ *X closure_of S* ∩ *U*
by (*simp add*: ‹*S* ⊆ *topspace X*› *closure_of_subset*)
moreover
have ⋀*t*. *t* ∈ *X closure_of S* ∩ *U* − *S* ⟹ ∃ *l*. *limitin Y f l* (*atin_within X t S*)
using *U_def* **by** *blast*
moreover have *regular_space Y*
by (*simp add*: *Y completely_metrizable_imp_metrizable_space metrizable_imp_regular_space*)
ultimately
obtain *g* **where** *g*: *continuous_map* (*subtopology X* (*X closure_of S* ∩ *U*)) *Y g*
and *gf*: ⋀*x*. *x* ∈ *S* ⟹ *g x* = *f x*
using *continuous_map_extension_pointwise_alt assms* **by** *blast*
show *thesis*
proof
show *gdelta_in X U*
by (*simp add*: *U_def Y contf gdelta_in_points_of_convergence_within*)
show *continuous_map* (*subtopology X* (*X closure_of S* ∩ *U*)) *Y g*
by (*simp add*: *g*)
qed (*use* ‹*S* ⊆ *U*› *gf* **in** *auto*)
qed

lemma *Lavrentiev_extension*:

assumes $S \subseteq$ *topspace* X
and X: *metrizable_space* $X \vee$ *topspace* $X \subseteq X$ *closure_of* S
and Y: *completely_metrizable_space* Y
and *contf*: *continuous_map* (*subtopology* X S) Y f
obtains U g **where** *gdelta_in* X U $S \subseteq U$ $U \subseteq X$ *closure_of* S
continuous_map (*subtopology* X U) Y g $\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
proof −
obtain U g **where** *gdelta_in* X U $S \subseteq U$
and *contg*: *continuous_map* (*subtopology* X (X *closure_of* $S \cap U$)) Y g
and *gf*: $\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
using *Lavrentiev_extension_gen* Y *assms*(*1*) *contf* **by** *blast*
define V **where** $V \equiv X$ *closure_of* $S \cap U$
show *thesis*
proof
show *gdelta_in* X V
by (*metis V_def X* ‹*gdelta_in X U*› *closed_imp_gdelta_in closedin_closure_of closure_of_subset_topspace gdelta_in_Int gdelta_in_topspace subset_antisym*)
show $S \subseteq V$
by (*simp add*: *V_def* ‹$S \subseteq U$› *assms*(*1*) *closure_of_subset*)
show *continuous_map* (*subtopology* X V) Y g
by (*simp add*: *V_def contg*)
qed (*auto simp*: *gf V_def*)
qed

5.11.19 Embedding in products and hence more about completely metrizable spaces

lemma (**in** *Metric_space*) *gdelta_homeomorphic_space_closedin_product*:
assumes S: $\bigwedge i.\ i \in I \Longrightarrow$ *openin mtopology* (S i)
obtains T **where** *closedin* (*prod_topology mtopology* (*powertop_real* I)) T
subtopology mtopology ($\bigcap i \in I.\ S\ i$) *homeomorphic_space*
subtopology (*prod_topology mtopology* (*powertop_real* I)) T
proof (*cases* I={})
case *True*
then have *top*: *topspace* (*prod_topology mtopology* (*powertop_real* I)) = (M × {($\lambda x.$ *undefined*)})
by *simp*
show *?thesis*
proof
show *closedin* (*prod_topology mtopology* (*powertop_real* I)) (M × {($\lambda x.$ *undefined*)})
by (*metis top closedin_topspace*)
have *subtopology mtopology* ($\bigcap$ (S ‘ I)) *homeomorphic_space mtopology*
by (*simp add*: *True product_topology_empty_discrete*)
also have ... *homeomorphic_space* (*prod_topology mtopology* (*discrete_topology* {$\lambda x.$ *undefined*}))
by (*meson homeomorphic_space_sym prod_topology_homeomorphic_space_left*)
finally
show *subtopology mtopology* ($\bigcap$ (S ‘ I)) *homeomorphic_space subtopology* (*prod_topology*

mtopology (*powertop_real I*)) (*M* × {(λ*x*. *undefined*)})
by (*smt* (*verit*, *ccfv_SIG*) *True product_topology_empty_discrete subtopology_topspace top*)
qed
next
case *False*
have *SM*: ⋀*i*. *i* ∈ *I* ⟹ *S i* ⊆ *M*
using *assms openin_mtopology* **by** *blast*
then have (⋂ *i* ∈ *I*. *S i*) ⊆ *M*
using *False* **by** *blast*
define *dd* **where** *dd* ≡ λ*i*. *if i*∉*I* ∨ *S i* = *M then* λ*u*. *1 else* (λ*u*. *INF x* ∈ *M* − *S i*. *d u x*)
have [*simp*]: *bdd_below* (*d u* ‘ *A*) **for** *u A*
by (*meson bdd_belowI2 nonneg*)
have *cont_dd*: *continuous_map* (*subtopology mtopology* (*S i*)) *euclidean* (*dd i*) **if** *i* ∈ *I* **for** *i*
proof −
have *dist* (*Inf* (*d x* ‘ (*M* − *S i*))) (*Inf* (*d y* ‘ (*M* − *S i*))) ≤ *d x y*
if *x* ∈ *S i x* ∈ *M y* ∈ *S i y* ∈ *M S i* ≠ *M* **for** *x y*
proof −
have [*simp*]: ¬ *M* ⊆ *S i*
using *SM* ‹*S i* ≠ *M*› ‹*i* ∈ *I*› **by** *auto*
have ⋀*u*. ⟦*u* ∈ *M*; *u* ∉ *S i*⟧ ⟹ *Inf* (*d x* ‘ (*M* − *S i*)) ≤ *d x y* + *d y u*
apply (*clarsimp simp add*: *cInf_le_iff_less*)
by (*smt* (*verit*) *DiffI triangle* ‹*x* ∈ *M*› ‹*y* ∈ *M*›)
then have *Inf* (*d x* ‘ (*M* − *S i*)) − *d x y* ≤ *Inf* (*d y* ‘ (*M* − *S i*))
by (*force simp add*: *le_cInf_iff*)
moreover
have ⋀*u*. ⟦*u* ∈ *M*; *u* ∉ *S i*⟧ ⟹ *Inf* (*d y* ‘ (*M* − *S i*)) ≤ *d x u* + *d x y*
apply (*clarsimp simp add*: *cInf_le_iff_less*)
by (*smt* (*verit*) *DiffI triangle''* ‹*x* ∈ *M*› ‹*y* ∈ *M*›)
then have *Inf* (*d y* ‘ (*M* − *S i*)) − *d x y* ≤ *Inf* (*d x* ‘ (*M* − *S i*))
by (*force simp add*: *le_cInf_iff*)
ultimately show *?thesis*
by (*simp add*: *dist_real_def abs_le_iff*)
qed
then have ∗: *Lipschitz_continuous_map* (*submetric Self* (*S i*)) *euclidean_metric* (λ*u*. *Inf* (*d u* ‘ (*M* − *S i*)))
unfolding *Lipschitz_continuous_map_def* **by** (*force intro*!: *exI* [**where** *x*=*1*])
then show *?thesis*
using *Lipschitz_continuous_imp_continuous_map* [*OF* ∗]
by (*simp add*: *dd_def Self_def mtopology_of_submetric*)
qed
have *dd_pos*: *0* < *dd i x* **if** *x* ∈ *S i* **for** *i x*
proof (*clarsimp simp add*: *dd_def*)
assume *i* ∈ *I* **and** *S i* ≠ *M*
have *opeS*: *openin mtopology* (*S i*)
by (*simp add*: ‹*i* ∈ *I*› *assms*)
then obtain *r* **where** *r*>*0* **and** *r*: ⋀*y*. ⟦*y* ∈ *M*; *d x y* < *r*⟧ ⟹ *y* ∈ *S i*

```
    by (meson ‹x ∈ S i› in_mball openin_mtopology subsetD)
  then have ⋀y. y ∈ M − S i ⟹ d x y ≥ r
    by (meson Diff_iff linorder_not_le)
  then have Inf (d x ‘ (M − S i)) ≥ r
  by (meson Diff_eq_empty_iff SM ‹S i ≠ M› ‹i ∈ I› cINF_greatest set_eq_subset)
  with ‹r>0› show 0 < Inf (d x ‘ (M − S i)) by simp
 qed
 define f where f ≡ λx. (x, λi∈I. inverse(dd i x))
 define T where T ≡ f ‘ (⋂ i ∈ I. S i)
 show ?thesis
 proof
  show closedin (prod_topology mtopology (powertop_real I)) T
   unfolding closure_of_subset_eq [symmetric]
  proof
   show T ⊆ topspace (prod_topology mtopology (powertop_real I))
    using False SM by (auto simp: T_def f_def)

   have (x,ds) ∈ T
    if §: ⋀U. ⟦(x, ds) ∈ U; openin (prod_topology mtopology (powertop_real I))
U⟧ ⟹ ∃y∈T. y ∈ U
      and x ∈ M and ds: ds ∈ I →E UNIV for x ds
   proof −
    have ope: ∃x. x ∈ ⋂(S ‘ I) ∧ f x ∈ U × V
     if x ∈ U and ds ∈ V and openin mtopology U and openin (powertop_real
I) V for U V
    using § [of U×V] that by (force simp add: T_def openin_prod_Times_iff)
    have x_in_INT: x ∈ ⋂(S ‘ I)
    proof clarify
     fix i
     assume i ∈ I
     show x ∈ S i
     proof (rule ccontr)
      assume x ∉ S i
       have openin (powertop_real I) {z ∈ topspace (powertop_real I). z i ∈
{ds i − 1 <..< ds i + 1}}
       proof (rule openin_continuous_map_preimage)
        show continuous_map (powertop_real I) euclidean (λx. x i)
         by (metis ‹i ∈ I› continuous_map_product_projection)
       qed auto
       then obtain y where y ∈ S i y ∈ M and dxy: d x y < inverse (|ds i|
+ 1)
                    and intvl: inverse (dd i y) ∈ {ds i − 1 <..< ds i + 1}
      using ope [of mball x (inverse(abs(ds i) + 1)) {z ∈ topspace(powertop_real
I). z i ∈ {ds i − 1<..<ds i + 1}}]
            ‹x ∈ M› ‹ds ∈ I →E UNIV› ‹i ∈ I›
       by (fastforce simp add: f_def)
      have ¬ M ⊆ S i
       using ‹x ∉ S i› ‹x ∈ M› by blast
      have inverse (|ds i| + 1) ≤ dd i y
```

```
            using intvl ‹y ∈ S i› dd_pos [of y i]
          by (smt (verit, ccfv_threshold) greaterThanLessThan_iff inverse_inverse_eq
le_imp_inverse_le)
           also have ... ≤ d x y
            using ‹i ∈ I› ‹¬ M ⊆ S i› ‹x ∉ S i› ‹x ∈ M›
            apply (simp add: dd_def cInf_le_iff_less)
            using commute by force
           finally show False
            using dxy by linarith
         qed
        qed
        moreover have ds = (λi∈I. inverse (dd i x))
        proof (rule PiE_ext [OF ds])
         fix i
         assume i ∈ I
         define e where e ≡ |ds i − inverse (dd i x)|
         { assume con: e > 0
         have continuous_map (subtopology mtopology (S i)) euclidean (λx. inverse
(dd i x))
           using dd_pos cont_dd ‹i ∈ I›
           by (fastforce simp: intro!: continuous_map_real_inverse)
          then have openin (subtopology mtopology (S i))
                    {z ∈ topspace (subtopology mtopology (S i)).
                     inverse (dd i z) ∈ {inverse(dd i x) − e/2<..<inverse(dd i x)
+ e/2}}
           using openin_continuous_map_preimage open_greaterThanLessThan
open_openin by blast
          then obtain U where openin mtopology U
              and U: {z ∈ S i. inverse (dd i x) − e/2 < inverse (dd i z) ∧
                       inverse (dd i z) < inverse (dd i x) + e/2}
                     = U ∩ S i
            using SM ‹i ∈ I› by (auto simp: openin_subtopology)
          have x ∈ U
            using U x_in_INT ‹i ∈ I› con by fastforce
          have ds ∈ {z ∈ topspace (powertop_real I). z i ∈ {ds i − e / 2<..<ds
i + e/2}}
            by (simp add: con ds)
          moreover
          have openin (powertop_real I) {z ∈ topspace (powertop_real I). z i ∈
{ds i − e / 2<..<ds i + e/2}}
          proof (rule openin_continuous_map_preimage)
            show continuous_map (powertop_real I) euclidean (λx. x i)
              by (metis ‹i ∈ I› continuous_map_product_projection)
          qed auto
          ultimately obtain y where y ∈ ⋂(S ‘ I) ∧ f y ∈ U × {z ∈ topspace
(powertop_real I). z i ∈ {ds i − e / 2<..<ds i + e/2}}
            using ope ‹x ∈ U› ‹openin mtopology U› ‹x ∈ U›
            by presburger
          with ‹i ∈ I› U
```

```
            have False unfolding set_eq_iff f_def e_def by simp (smt (verit)
field_sum_of_halves)
        }
        then show ds i = (λi∈I. inverse (dd i x)) i
          using ‹i ∈ I› by (force simp: e_def)
      qed auto
      ultimately show ?thesis
        by (auto simp: T_def f_def)
    qed
    then show prod_topology mtopology (powertop_real I) closure_of T ⊆ T
      by (auto simp: closure_of_def)
  qed
  have eq: (⋂(S ‘ I) × (I →E UNIV) ∩ f ‘ (M ∩ ⋂(S ‘ I))) = (f ‘ ⋂(S ‘ I))
    using False SM by (force simp: f_def image_iff)
  have continuous_map (subtopology mtopology (⋂(S ‘ I))) euclidean (dd i) if i
∈ I for i
    by (meson INT_lower cont_dd continuous_map_from_subtopology_mono
that)
  then have continuous_map (subtopology mtopology (⋂(S ‘ I))) (powertop_real
I) (λx. λi∈I. inverse (dd i x))
      using dd_pos by (fastforce simp: continuous_map_componentwise_intro!:
continuous_map_real_inverse)
  then have embedding_map (subtopology mtopology (⋂(S ‘ I))) (prod_topology
(subtopology mtopology (⋂(S ‘ I))) (powertop_real I)) f
    by (simp add: embedding_map_graph f_def)
  moreover have subtopology (prod_topology (subtopology mtopology (⋂(S ‘ I)))
(powertop_real I))
                  (f ‘ topspace (subtopology mtopology (⋂(S ‘ I)))) =
                subtopology (prod_topology mtopology (powertop_real I)) T
    by (simp add: prod_topology_subtopology subtopology_subtopology T_def eq)
  ultimately
 show subtopology mtopology (⋂(S ‘ I)) homeomorphic_space subtopology (prod_topology
mtopology (powertop_real I)) T
    by (metis embedding_map_imp_homeomorphic_space)
 qed
qed


lemma gdelta_homeomorphic_space_closedin_product:
 assumes metrizable_space X and ⋀i. i ∈ I ⟹ openin X (S i)
 obtains T where closedin (prod_topology X (powertop_real I)) T
                 subtopology X (⋂i ∈ I. S i) homeomorphic_space
                  subtopology (prod_topology X (powertop_real I)) T
 using Metric_space.gdelta_homeomorphic_space_closedin_product
 by (metis assms metrizable_space_def)

lemma open_homeomorphic_space_closedin_product:
 assumes metrizable_space X and openin X S
 obtains T where closedin (prod_topology X euclideanreal) T
```

subtopology X S homeomorphic_space
subtopology (prod_topology X euclideanreal) T
proof −
obtain *T* **where** *cloT*: *closedin* (*prod_topology X* (*powertop_real* {()})) *T*
and *homT*: *subtopology X S homeomorphic_space*
subtopology (*prod_topology X* (*powertop_real* {()})) *T*
using *gdelta_homeomorphic_space_closedin_product* [*of X* {()} *λi. S*] *assms*
by *auto*
have *prod_topology X* (*powertop_real* {()}) *homeomorphic_space prod_topology X euclideanreal*
by (*meson homeomorphic_space_prod_topology homeomorphic_space_refl homeomorphic_space_singleton_product*)
then obtain *f* **where** *f*: *homeomorphic_map* (*prod_topology X* (*powertop_real* {()})) (*prod_topology X euclideanreal*) *f*
unfolding *homeomorphic_space* **by** *metis*
show *thesis*
proof
show *closedin* (*prod_topology X euclideanreal*) (*f ‘ T*)
using *cloT f homeomorphic_map_closedness_eq* **by** *blast*
moreover have *T* = *topspace* (*subtopology* (*prod_topology X* (*powertop_real* {()})) *T*)
by (*metis cloT closedin_subset topspace_subtopology_subset*)
ultimately show *subtopology X S homeomorphic_space subtopology* (*prod_topology X euclideanreal*) (*f ‘ T*)
by (*smt* (*verit*, *best*) *closedin_subset f homT homeomorphic_map_subtopologies homeomorphic_space*
homeomorphic_space_trans topspace_subtopology topspace_subtopology_subset)
qed
qed

lemma *completely_metrizable_space_gdelta_in_alt*:
assumes *X*: *completely_metrizable_space X*
and *S*: (*countable intersection_of openin X*) *S*
shows *completely_metrizable_space* (*subtopology X S*)
proof −
obtain $\mathcal{U}$ **where** *countable* $\mathcal{U}$ *S* = $\bigcap\mathcal{U}$ **and** *ope*: $\bigwedge U.\ U \in \mathcal{U} \Longrightarrow$ *openin X U*
using *S* **by** (*force simp add*: *intersection_of_def*)
then have $\mathcal{U}$: *completely_metrizable_space* (*powertop_real* $\mathcal{U}$)
by (*simp add*: *completely_metrizable_space_euclidean completely_metrizable_space_product_topology*)
obtain *C* **where** *closedin* (*prod_topology X* (*powertop_real* $\mathcal{U}$)) *C*
and *sub*: *subtopology X* ($\bigcap\mathcal{U}$) *homeomorphic_space*
subtopology (*prod_topology X* (*powertop_real* $\mathcal{U}$)) *C*
by (*metis gdelta_homeomorphic_space_closedin_product X completely_metrizable_imp_metrizable_space ope INF_identity_eq*)
moreover have *completely_metrizable_space* (*prod_topology X* (*powertop_real* $\mathcal{U}$))
by (*simp add*: *completely_metrizable_space_prod_topology X* $\mathcal{U}$)
ultimately have *completely_metrizable_space* (*subtopology* (*prod_topology X* (*powertop_real* $\mathcal{U}$)) *C*)

using *completely_metrizable_space_closedin* **by** *blast*
then show *?thesis*
using ‹$S = \bigcap \mathcal{U}$› *sub homeomorphic_completely_metrizable_space* **by** *blast*
qed

lemma *completely_metrizable_space_gdelta_in*:
⟦*completely_metrizable_space X*; *gdelta_in X S*⟧
⟹ *completely_metrizable_space* (*subtopology X S*)
by (*simp add*: *completely_metrizable_space_gdelta_in_alt gdelta_in_alt*)

lemma *completely_metrizable_space_openin*:
⟦*completely_metrizable_space X*; *openin X S*⟧
⟹ *completely_metrizable_space* (*subtopology X S*)
by (*simp add*: *completely_metrizable_space_gdelta_in open_imp_gdelta_in*)

lemma (**in** *Metric_space*) *locally_compact_imp_completely_metrizable_space*:
assumes *locally_compact_space mtopology*
shows *completely_metrizable_space mtopology*
proof −
obtain $f :: ['a,'a] \Rightarrow real$ **and** m' **where**
mcomplete_of m′ **and** *fim*: $f \in M \to$ *mspace m′*
and *clo*: *mtopology_of m′ closure_of f ‘ M* = *mspace m′*
and *d*: $\bigwedge x\ y.\ ⟦x \in M;\ y \in M⟧ \Longrightarrow$ *mdist m′* (*f x*) (*f y*) = *d x y*
by (*metis metric_completion*)
then have *embedding_map mtopology* (*mtopology_of m′*) *f*
unfolding *mtopology_of_def*
by (*metis Metric_space12.isometry_imp_embedding_map Metric_space12_mspace_mdist mdist_metric mspace_metric*)
then have *hom*: *mtopology homeomorphic_space subtopology* (*mtopology_of m′*) (*f ‘ M*)
by (*metis embedding_map_imp_homeomorphic_space topspace_mtopology*)
have *locally_compact_space* (*subtopology* (*mtopology_of m′*) (*f ‘ M*))
using *assms hom homeomorphic_locally_compact_space* **by** *blast*
moreover have *Hausdorff_space* (*mtopology_of m′*)
by (*simp add*: *Metric_space.Hausdorff_space_mtopology mtopology_of_def*)
ultimately have *openin* (*mtopology_of m′*) (*f ‘ M*)
by (*simp add*: *clo dense_locally_compact_openin_Hausdorff_space fim image_subset_iff_funcset*)
then
have *completely_metrizable_space* (*subtopology* (*mtopology_of m′*) (*f ‘ M*))
using ‹*mcomplete_of m′*› **unfolding** *mcomplete_of_def mtopology_of_def*
by (*metis Metric_space.completely_metrizable_space_mtopology Metric_space_mspace_mdist completely_metrizable_space_openin*)
then show *?thesis*
using *hom homeomorphic_completely_metrizable_space* **by** *blast*
qed

lemma *locally_compact_imp_completely_metrizable_space*:

assumes *metrizable_space X* **and** *locally_compact_space X*
shows *completely_metrizable_space X*
by (*metis Metric_space.locally_compact_imp_completely_metrizable_space assms metrizable_space_def*)

lemma *completely_metrizable_space_imp_gdelta_in*:
assumes *X*: *metrizable_space X* **and** *S* ⊆ *topspace X*
and *XS*: *completely_metrizable_space* (*subtopology X S*)
shows *gdelta_in X S*
proof −
obtain *U f* **where** *gdelta_in X U S* ⊆ *U* **and** *U*: *U* ⊆ *X closure_of S*
and *contf*: *continuous_map* (*subtopology X U*) (*subtopology X S*) *f*
and *fid*: ⋀*x*. *x* ∈ *S* ⟹ *f x* = *x*
using *Lavrentiev_extension*[*of S X subtopology X S id*] *assms* **by** *auto*
then have *f* ‘ *topspace* (*subtopology X U*) ⊆ *topspace* (*subtopology X S*)
using *continuous_map_image_subset_topspace* **by** *blast*
then have *f*‘*U* ⊆ *S*
by (*metis* ‹*gdelta_in X U*› ‹*S* ⊆ *topspace X*› *gdelta_in_subset topspace_subtopology_subset*)
moreover
have *Hausdorff_space* (*subtopology X U*)
by (*simp add*: *Hausdorff_space_subtopology X metrizable_imp_Hausdorff_space*)
then have ⋀*x*. *x* ∈ *U* ⟹ *f x* = *x*
using *U fid contf forall_in_closure_of_eq* [*of _ subtopology X U S subtopology X U f id*]
by (*metis* ‹*S* ⊆ *U*› *closure_of_subtopology_open continuous_map_id continuous_map_in_subtopology id_apply inf.orderE subtopology_subtopology*)
ultimately have *U* ⊆ *S*
by *auto*
then show *?thesis*
using ‹*S* ⊆ *U*› ‹*gdelta_in X U*› **by** *auto*
qed

lemma *completely_metrizable_space_eq_gdelta_in*:
⟦*completely_metrizable_space X*; *S* ⊆ *topspace X*⟧
⟹ *completely_metrizable_space* (*subtopology X S*) ⟷ *gdelta_in X S*
using *completely_metrizable_imp_metrizable_space completely_metrizable_space_gdelta_in completely_metrizable_space_imp_gdelta_in* **by** *blast*

lemma *gdelta_in_eq_completely_metrizable_space*:
completely_metrizable_space X
⟹ *gdelta_in X S* ⟷ *S* ⊆ *topspace X* ∧ *completely_metrizable_space* (*subtopology X S*)
by (*metis completely_metrizable_space_eq_gdelta_in gdelta_in_alt*)

5.11.20 Theorems from Kuratowski

Kuratowski, Remark on an Invariance Theorem, *Fundamenta Mathematicae* **37** (1950), pp. 251-252. The idea is that in suitable spaces, to show "number

of components of the complement" (without distinguishing orders of infinity) is a homeomorphic invariant, it suffices to show it for closed subsets. Kuratowski states the main result for a "locally connected continuum", and seems clearly to be implicitly assuming that means metrizable. We call out the general topological hypotheses more explicitly, which do not however include connectedness.

lemma *separation_by_closed_intermediates_count*:
 assumes X: *hereditarily normal_space X*
  and *finite* $\mathcal{U}$
  and *pwU*: *pairwise* (*separatedin X*) $\mathcal{U}$
  and *nonempty*: $\{\} \notin \mathcal{U}$
  and *UU*: $\bigcup\mathcal{U} =$ *topspace X* $-$ *S*
 obtains *C* **where** *closedin X C C* $\subseteq$ *S*
 $\bigwedge D.$ $[\![$*closedin X D*; *C* $\subseteq$ *D*; *D* $\subseteq$ *S*$]\!]$
 $\Longrightarrow \exists\mathcal{V}.$ $\mathcal{V} \approx \mathcal{U} \wedge$ *pairwise* (*separatedin X*) $\mathcal{V} \wedge \{\} \notin \mathcal{V} \wedge \bigcup\mathcal{V} =$ *topspace X* $-$ *D*
proof $-$
 obtain *F* **where** *F*: $\bigwedge S.$ $S \in \mathcal{U} \Longrightarrow$ *openin X* (*F S*) $\wedge$ *S* $\subseteq$ *F S*
  and *pwF*: *pairwise* ($\lambda S\ T.$ *disjnt* (*F S*) (*F T*)) $\mathcal{U}$
  using *assms* **by** (*smt* (*verit*, *best*) *Diff_subset Sup_le_iff hereditarily_normal_separation_pairwise*)
 show *thesis*
 proof
  show *closedin X* (*topspace X* $-$ $\bigcup$(*F* ‘ $\mathcal{U}$))
   using *F* **by** *blast*
  show *topspace X* $-$ $\bigcup$(*F* ‘ $\mathcal{U}$) $\subseteq$ *S*
   using *UU F* **by** *auto*
  show $\exists\mathcal{V}.$ $\mathcal{V} \approx \mathcal{U} \wedge$ *pairwise* (*separatedin X*) $\mathcal{V} \wedge \{\} \notin \mathcal{V} \wedge \bigcup\mathcal{V} =$ *topspace X* $-$ *C*
   if *closedin X C C* $\subseteq$ *S* **and** *C*: *topspace X* $-$ $\bigcup$(*F* ‘ $\mathcal{U}$) $\subseteq$ *C* **for** *C*
  proof (*intro exI conjI strip*)
   have *inj_on* ($\lambda S.$ *F S* $-$ *C*) $\mathcal{U}$
    using *pwF F*
    unfolding *inj_on_def pairwise_def disjnt_iff*
    by (*metis Diff_iff UU UnionI nonempty subset_empty subset_eq* ‹*C* $\subseteq$ *S*›)
   then show ($\lambda S.$ *F S* $-$ *C*) ‘ $\mathcal{U} \approx \mathcal{U}$
    by *simp*
   show *pairwise* (*separatedin X*) (($\lambda S.$ *F S* $-$ *C*) ‘ $\mathcal{U}$)
     using ‹*closedin X C*› *F pwF* **by** (*force simp*: *pairwise_def openin_diff separatedin_open_sets disjnt_iff*)
   show $\{\} \notin$ ($\lambda S.$ *F S* $-$ *C*) ‘ $\mathcal{U}$
    using *nonempty UU* ‹*C* $\subseteq$ *S*› *F*
    by *clarify* (*metis DiffD2 Diff_eq_empty_iff F UnionI subset_empty subset_eq*)
   show ($\bigcup S{\in}\mathcal{U}.$ *F S* $-$ *C*) = *topspace X* $-$ *C*
    using *UU F C openin_subset* **by** *fastforce*
  qed
 qed
qed

lemma *separation_by_closed_intermediates_gen*:
assumes *X*: *hereditarily normal_space X*
and *discon*: ¬ *connectedin X* (*topspace X* − *S*)
obtains *C* **where** *closedin X C C* ⊆ *S*
⋀*D*. ⟦*closedin X D*; *C* ⊆ *D*; *D* ⊆ *S*⟧ ⟹ ¬ *connectedin X* (*topspace X* − *D*)
proof −
obtain *C1 C2* **where** *Ueq*: *C1* ∪ *C2* = *topspace X* − *S* **and** *C1* ≠ {} *C2* ≠ {}
and *sep*: *separatedin X C1 C2* **and** *C1* ≠ *C2*
by (*metis Diff_subset connectedin_eq_not_separated discon separatedin_refl*)
then obtain *C* **where** *closedin X C C* ⊆ *S*
and *C*: ⋀*D*. ⟦*closedin X D*; *C* ⊆ *D*; *D* ⊆ *S*⟧
⟹ ∃*V*. *V* ≈ {*C1*,*C2*} ∧ *pairwise* (*separatedin X*) *V* ∧ {} ∉ *V* ∧ ⋃*V* = *topspace X* − *D*
using *separation_by_closed_intermediates_count* [*of X* {*C1*,*C2*} *S*] *X*
apply (*simp add*: *pairwise_insert separatedin_sym*)
by *metis*
have ¬ *connectedin X* (*topspace X* − *D*)
if *D*: *closedin X D C* ⊆ *D D* ⊆ *S* **for** *D*
proof −
obtain *V1 V2* **where** ∗: *pairwise* (*separatedin X*) {*V1*,*V2*} {} ∉ {*V1*,*V2*}
⋃{*V1*,*V2*} = *topspace X* − *D V1*≠*V2*
by (*metis C* [*OF D*] ‹*C1* ≠ *C2*› *eqpoll_doubleton_iff*)
then have *disjnt V1 V2*
by (*metis pairwise_insert separatedin_imp_disjoint singleton_iff*)
with ∗ **show** *?thesis*
by (*auto simp add*: *connectedin_eq_not_separated pairwise_insert*)
qed
then show *thesis*
using ‹*C* ⊆ *S*› ‹*closedin X C*› *that* **by** *auto*
qed

lemma *separation_by_closed_intermediates_eq_count*:
fixes *n*::*nat*
assumes *lcX*: *locally_connected_space X* **and** *hnX*: *hereditarily normal_space X*
shows (∃*U*. *U* ≈ {..<*n*} ∧ *pairwise* (*separatedin X*) *U* ∧ {} ∉ *U* ∧ ⋃*U* = *topspace X* − *S*) ⟷
(∃ *C*. *closedin X C* ∧ *C* ⊆ *S* ∧
(∀ *D*. *closedin X D* ∧ *C* ⊆ *D* ∧ *D* ⊆ *S*
⟶ (∃*U*. *U* ≈ {..<*n*} ∧ *pairwise* (*separatedin X*) *U* ∧ {} ∉ *U* ∧ ⋃*U* = *topspace X* − *D*)))
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*smt* (*verit*, *best*) *hnX separation_by_closed_intermediates_count eqpoll_iff_finite_card eqpoll_trans*)
next

assume *R*: *?rhs*
show *?lhs*
proof (*cases n=0*)
case *True*
with *R* **show** *?thesis*
by *fastforce*
next
case *False*
obtain *C* **where** *closedin X C C* ⊆ *S*
and *C*: ⋀*D*. ⟦*closedin X D*; *C* ⊆ *D*; *D* ⊆ *S*⟧
⟹ ∃*U*. *U* ≈ {..<*n*} ∧ *pairwise* (*separatedin X*) *U* ∧ {} ∉ *U* ∧
⋃*U* = *topspace X* − *D*
using *R* **by** *force*
then have *C* ⊆ *topspace X*
by (*simp add*: *closedin_subset*)
define *U* **where** *U* ≡ {*D* ∈ *connected_components_of* (*subtopology X* (*topspace*
X − *C*)). *D*−*S* ≠ {}}
have *opeU*: *openin X U* **if** *U* ∈ *U* **for** *U*
using *that* ‹*closedin X C*› *lcX locally_connected_space_open_connected_components*

by (*fastforce simp add*: *closedin_def U_def*)
have {} ∉ *U*
by (*auto simp*: *U_def*)
have *pairwise disjnt U*
using *connected_components_of_disjoint* **by** (*fastforce simp add*: *pairwise_def*
U_def)
show *?lhs*
proof (*rule ccontr*)
assume *con*: ∄*U*. *U* ≈ {..<*n*} ∧ *pairwise* (*separatedin X*) *U* ∧ {} ∉ *U* ∧ ⋃*U*
= *topspace X* − *S*
have *cardU*: *finite U* ∧ *card U* < *n*
proof (*rule ccontr*)
assume ¬ (*finite U* ∧ *card U* < *n*)
then obtain *V* **where** *V* ⊆ *U finite V card V* = *n*
by (*metis infinite_arbitrarily_large linorder_not_less obtain_subset_with_card_n*)
then obtain *T* **where** *T* ∈ *V*
using *False* **by** *force*
define *W* **where** *W* ≡ *insert* (*topspace X* − *S* − ⋃(*V* − {*T*})) ((λ*D*. *D*
− *S*) ‘ (*V* − {*T*}))
have ⋃*W* = *topspace X* − *S*
using ‹⋀*U*. *U* ∈ *U* ⟹ *openin X U*› ‹*V* ⊆ *U*› *topspace_def* **by** (*fastforce*
simp: *W_def*)
moreover have {} ∉ *W*
proof −
obtain *a* **where** *a* ∈ *T a* ∉ *S*
using *U_def* ‹*T* ∈ *V*› ‹*V* ⊆ *U*› **by** *blast*
then have *a* ∈ *topspace X*
using ‹*T* ∈ *V*› *opeU* ‹*V* ⊆ *U*› *openin_subset* **by** *blast*
moreover have *a* ∉ ⋃(*V* − {*T*})

```
            using diff_Union_pairwise_disjoint [of 𝒱 {T}] ‹disjoint 𝒰› pairwise_subset ‹T ∈ 𝒱› ‹𝒱 ⊆ 𝒰› ‹a ∈ T›
            by auto
          ultimately have topspace X - S - ⋃(𝒱 - {T}) ≠ {}
            using ‹a ∉ S› by blast
          moreover have ⋀V. V ∈ 𝒱 - {T} ⟹ V - S ≠ {}
            using 𝒰_def ‹𝒱 ⊆ 𝒰› by blast
          ultimately show ?thesis
            by (metis (no_types, lifting) 𝒲_def image_iff insert_iff)
        qed
        moreover have disjoint 𝒱
          using ‹𝒱 ⊆ 𝒰› ‹disjoint 𝒰› pairwise_subset by blast
        then have inj: inj_on (λD. D - S) (𝒱 - {T})
          unfolding inj_on_def using ‹𝒱 ⊆ 𝒰› disjointD 𝒰_def inf_commute by blast
        have finite 𝒲 card 𝒲 = n
          using ‹{} ∉ 𝒲› ‹n ≠ 0› ‹T ∈ 𝒱›
         by (auto simp add: 𝒲_def ‹finite 𝒱› card_insert_if card_image inj ‹card 𝒱 = n›)
        moreover have pairwise (separatedin X) 𝒲
        proof -
          have disjoint 𝒲
            using ‹disjoint 𝒱› by (auto simp: 𝒲_def pairwise_def disjnt_iff)
          have pairwise (separatedin (subtopology X (topspace X - S))) 𝒲
          proof (intro pairwiseI)
            fix A B
            assume §: A ∈ 𝒲 B ∈ 𝒲 A ≠ B
            then have disjnt A B
              by (meson ‹disjoint 𝒲› pairwiseD)
            have closedin (subtopology X (topspace X - C)) (⋃(𝒱 - {T}))
              using 𝒰_def ‹𝒱 ⊆ 𝒰› closedin_connected_components_of ‹finite 𝒱›
              by (force simp add: intro!: closedin_Union)
            with ‹C ⊆ S› have openin (subtopology X (topspace X - S)) (topspace X - S - ⋃(𝒱 - {T}))
                  by (fastforce simp add: openin_closedin_eq closedin_subtopology Int_absorb1)
          moreover have ⋀V. V ∈ 𝒱 ∧ V≠T ⟹ openin (subtopology X (topspace X - S)) (V - S)
            using ‹𝒱 ⊆ 𝒰› opeU
            by (metis IntD2 Int_Diff inf.orderE openin_subset openin_subtopology)

           ultimately have openin (subtopology X (topspace X - S)) A openin (subtopology X (topspace X - S)) B
            using § 𝒲_def by blast+
           with ‹disjnt A B› show separatedin (subtopology X (topspace X - S)) A B
            using separatedin_open_sets by blast
          qed
          then show ?thesis
```

```
          by (simp add: pairwise_def separatedin_subtopology)
        qed
        ultimately show False
          by (metis con eqpoll_iff_finite_card)
      qed
      obtain 𝒱 where 𝒱 ≈ {..<n}  {} ∉ 𝒱
              and pw𝒱: pairwise (separatedin X) 𝒱 and UV: ⋃𝒱 = topspace X −
(topspace X − ⋃𝒰)
      proof −
        have closedin X (topspace X − ⋃𝒰)
          using ope𝒰 by blast
        moreover have C ⊆ topspace X − ⋃𝒰
          using ‹C ⊆ topspace X› connected_components_of_subset by (fastforce
simp: 𝒰_def)
        moreover have topspace X − ⋃𝒰 ⊆ S
          using Union_connected_components_of [of subtopology X (topspace X −
C)] ‹C ⊆ S›
          by (auto simp: 𝒰_def)
        ultimately show thesis
          by (metis C that)
      qed
      have 𝒱 ≲ 𝒰
      proof (rule lepoll_relational_full)
        have ⋃𝒱 = ⋃𝒰
          by (simp add: Sup_le_iff UV double_diff ope𝒰 openin_subset)
        then show ∃U. U ∈ 𝒰 ∧ ¬ disjnt U V if V ∈ 𝒱 for V
          using that
          by (metis ‹{} ∉ 𝒱› disjnt_Union1 disjnt_self_iff_empty)
        show C1 = C2
          if T ∈ 𝒰 and C1 ∈ 𝒱 and C2 ∈ 𝒱 and ¬ disjnt T C1 and ¬ disjnt T
C2 for T C1 C2
        proof (cases C1=C2)
          case False
          then have connectedin X T
         using 𝒰_def connectedin_connected_components_of connectedin_subtopology
‹T ∈ 𝒰› by blast
          have T ⊆ C1 ∪ ⋃(𝒱 − {C1})
            using ‹⋃𝒱 = ⋃𝒰› ‹T ∈ 𝒰› by auto
          with ‹connectedin X T›
          have ¬ separatedin X C1 (⋃(𝒱 − {C1}))
            unfolding connectedin_eq_not_separated_subset
          by (smt (verit) that False disjnt_def UnionI disjnt_iff insertE insert_Diff)
          with that show ?thesis
                by (metis (no_types, lifting) ‹𝒱 ≈ {..<n}› eqpoll_iff_finite_card
finite_Diff pairwiseD pairwise_alt pw𝒱 separatedin_Union(1) separatedin_def)
        qed auto
      qed
      then show False
       by (metis ‹𝒱 ≈ {..<n}› card𝒰 eqpoll_iff_finite_card leD lepoll_iff_card_le)
```

```
    qed
  qed
qed

lemma separation_by_closed_intermediates_eq_gen:
  assumes locally_connected_space X hereditarily normal_space X
  shows ¬ connectedin X (topspace X − S) ⟷
        (∃ C. closedin X C ∧ C ⊆ S ∧
            (∀ D. closedin X D ∧ C ⊆ D ∧ D ⊆ S ⟶ ¬ connectedin X (topspace
X − D)))
    (is ?lhs = ?rhs)
proof −
  have ∗: (∃𝒰::'a set set. 𝒰 ≈ {..<Suc (Suc 0)} ∧ P 𝒰) ⟷ (∃ A B. A≠B ∧
P{A,B}) for P
    by (metis One_nat_def eqpoll_doubleton_iff lessThan_Suc lessThan_empty_iff
zero_neq_one)
  have ∗: (∃ C1 C2. separatedin X C1 C2 ∧ C1≠C2 ∧ C1≠{} ∧ C2≠{} ∧ C1 ∪
C2 = topspace X − S) ⟷
        (∃ C. closedin X C ∧ C ⊆ S ∧
            (∀ D. closedin X D ∧ C ⊆ D ∧ D ⊆ S
                ⟶ (∃ C1 C2. separatedin X C1 C2 ∧ C1≠C2 ∧ C1≠{} ∧ C2≠{}
∧ C1 ∪ C2 = topspace X − D)))
    using separation_by_closed_intermediates_eq_count [OF assms, of Suc(Suc
0) S]
    apply (simp add: ∗ pairwise_insert separatedin_sym cong: conj_cong)
    apply (simp add: eq_sym_conv conj_ac)
    done
  with separatedin_refl
  show ?thesis
    apply (simp add: connectedin_eq_not_separated)
    by (smt (verit, best) separatedin_refl)
qed



lemma lepoll_connnected_components_connectedin:
  assumes ⋀C. C ∈ 𝒰 ⟹ connectedin X C  ⋃𝒰 = topspace X
  shows connected_components_of X ≲ 𝒰
proof −
  have connected_components_of X ≲ 𝒰 − {{}}
  proof (rule lepoll_relational_full)
    show ∃ U. U ∈ 𝒰 − {{}} ∧ U ⊆ V
      if V ∈ connected_components_of X for V
      using that unfolding connected_components_of_def image_iff
        by (metis Union_iff assms connected_component_of_maximal empty_iff
insert_Diff_single insert_iff)
    show V = V'
    if U ∈ 𝒰 − {{}} V ∈ connected_components_of X V' ∈ connected_components_of
X U ⊆ V U ⊆ V'
```

```
    for U V V'
   by (metis DiffD2 disjointD insertCI le_inf_iff pairwise_disjoint_connected_components_of
subset_empty that)
 qed
 also have ... ≲ 𝒰
   by (simp add: subset_imp_lepoll)
 finally show ?thesis .
qed

lemma lepoll_connected_components_alt:
 {..<n::nat} ≲ connected_components_of X ⟷
   n = 0 ∨ (∃𝒰. 𝒰 ≈ {..<n} ∧ pairwise (separatedin X) 𝒰 ∧ {} ∉ 𝒰 ∧ ⋃𝒰 =
topspace X)
 (is ?lhs ⟷ ?rhs)
proof (cases n=0)
next
 case False
 show ?thesis
 proof
   assume L: ?lhs
   with False show ?rhs
   proof (induction n rule: less_induct)
     case (less n)
     show ?case
     proof (cases n≤1)
       case True
       with less.prems have topspace X ≠ {} n=1
         by (fastforce simp add: connected_components_of_def)+
       then have {} ∉ {topspace X}
         by blast
       with ‹n=1› show ?thesis
         by (simp add: eqpoll_iff_finite_card card_Suc_eq flip: ex_simps)
     next
       case False
       then have n−1 ≠ 0
         by linarith
       have n1_lesspoll: {..<n−1} ≺ {..<n}
         using False lesspoll_iff_finite_card by fastforce
       also have ... ≲ connected_components_of X
         using less by blast
       finally have {..<n−1} ≲ connected_components_of X
         using lesspoll_imp_lepoll by blast
       then obtain 𝒰 where Ueq: 𝒰 ≈ {..<n−1} and {} ∉ 𝒰
         and pwU: pairwise (separatedin X) 𝒰 and UU: ⋃𝒰 = topspace X
         by (meson ‹n − 1 ≠ 0› diff_less gr0I less zero_less_one)
       show ?thesis
       proof (cases ∀ C ∈ 𝒰. connectedin X C)
         case True
         then show ?thesis
```

```
          using lepoll_connnected_components_connectedin [of 𝒰 X] less.prems
          by (metis UU Ueq lepoll_antisym lepoll_trans lepoll_trans2 lesspoll_def
n1_lesspoll)
        next
          case False
          with UU obtain C A B where ABC: C ∈ 𝒰 A ∪ B = C A ≠ {} B ≠
{} and sep: separatedin X A B
            by (fastforce simp add: connectedin_eq_not_separated)
          define 𝒱 where 𝒱 ≡ insert A (insert B (𝒰 − {C}))
          have 𝒱 ≈ {..<n}
          proof −
            have A ≠ B
              using ‹B ≠ {}› sep by auto
            moreover obtain A ∉ 𝒰 B ∉ 𝒰
              using pwU unfolding pairwise_def
                by (metis ABC sep separatedin_Un(1) separatedin_refl separate-
din_sym)
            moreover have card 𝒰 = n−1 finite 𝒰
              using Ueq eqpoll_iff_finite_card by blast+
            ultimately
            have card (insert A (insert B (𝒰 − {C}))) = n
              using ‹C ∈ 𝒰› by (auto simp add: card_insert_if)
            then show ?thesis
              using 𝒱_def ‹finite 𝒰› eqpoll_iff_finite_card by blast
          qed
          moreover have {} ∉ 𝒱
            using ABC 𝒱_def ‹{} ∉ 𝒰› by blast
          moreover have ⋃𝒱 = topspace X
            using ABC UU 𝒱_def by auto
          moreover have pairwise (separatedin X) 𝒱
            using pwU sep ABC unfolding 𝒱_def
            apply (simp add: separatedin_sym pairwise_def)
            by (metis member_remove remove_def separatedin_Un(1))
          ultimately show ?thesis
            by blast
        qed
      qed
    qed
  next
    assume ?rhs
    then obtain 𝒰 where 𝒰 ≈ {..<n} {} ∉ 𝒰 and pwU: pairwise (separatedin X)
𝒰 and UU: ⋃𝒰 = topspace X
      using False by force
    have card (connected_components_of X) ≥ n if finite (connected_components_of
X)
    proof −
      have 𝒰 ≲ connected_components_of X
      proof (rule lepoll_relational_full)
        show ∃ T. T ∈ connected_components_of X ∧ ¬ disjnt T C if C ∈ 𝒰 for
```

C
by (*metis that UU Union_connected_components_of Union_iff* ‹{} ∉ 𝒰› *disjnt_iff equals0I*)
show (*C1*::'*a set*) = *C2*
if *T* ∈ *connected_components_of X* **and** *C1* ∈ 𝒰 *C2* ∈ 𝒰 ¬ *disjnt T C1* ¬ *disjnt T C2* **for** *T C1 C2*
proof (*rule ccontr*)
assume *C1* ≠ *C2*
then have *connectedin X T*
by (*simp add*: *connectedin_connected_components_of that*(*1*))
moreover have ¬ *separatedin X C1* (⋃(𝒰 − {*C1*}))
using ‹*connectedin X T*› *pwU* **unfolding** *pairwise_def*
by (*smt* (*verit*) *Sup_upper UU Union_connected_components_of* ‹*C1* ≠ *C2*› *complete_lattice_class.Sup_insert connectedin_subset_separated_union disjnt_subset2 disjnt_sym insert_Diff separatedin_imp_disjoint that*)
ultimately show *False*
using ‹𝒰 ≈ {..<*n*}›
apply (*simp add*: *connectedin_eq_not_separated_subset eqpoll_iff_finite_card*)
by (*metis Sup_upper UU finite_Diff pairwise_alt pwU separatedin_Union*(*1*) *that*(*2*))
qed
qed
then show *?thesis*
by (*metis* ‹𝒰 ≈ {..<*n*}› *eqpoll_iff_finite_card lepoll_iff_card_le that*)
qed
then show *?lhs*
by (*metis card_lessThan finite_lepoll_infinite finite_lessThan lepoll_iff_card_le*)
qed
qed *auto*

5.11.21 A perfect set in common cases must have at least the cardinality of the continuum

lemma (**in** *Metric_space*) *lepoll_perfect_set*:
assumes *mcomplete*
and *mtopology derived_set_of S* = *S S* ≠ {}
shows (*UNIV*::*real set*) ≲ *S*
proof −
have *S* ⊆ *M*
using *assms*(*2*) *derived_set_of_infinite_mball* **by** *blast*
have (*UNIV*::*real set*) ≲ (*UNIV*::*nat set set*)
using *eqpoll_imp_lepoll eqpoll_sym nat_sets_eqpoll_reals* **by** *blast*
also have ... ≲ *S*
proof −
have ∃ *y z* δ. *y* ∈ *S* ∧ *z* ∈ *S* ∧ 0 < δ ∧ δ < ε/2 ∧
mcball y δ ⊆ *mcball x* ε ∧ *mcball z* δ ⊆ *mcball x* ε ∧ *disjnt* (*mcball y* δ) (*mcball z* δ)
if *x* ∈ *S* 0 < ε **for** *x* ε
proof −

define S' **where** $S' \equiv S \cap mball\ x\ (\varepsilon/4)$
have *infinite* S'
using *derived_set_of_infinite_mball* [*of* S] *assms that* S'*_def*
by (*smt* (*verit*, *ccfv_SIG*) *mem_Collect_eq zero_less_divide_iff*)
then have $\bigwedge x\ y\ z.\ \neg\ (S' \subseteq \{x,y,z\})$
using *finite_subset* **by** *auto*
then obtain $l\ r$ **where** *lr*: $l \in S'\ r \in S'\ l{\neq}r\ l{\neq}x\ r{\neq}x$
by (*metis insert_iff subsetI*)
show *?thesis*
proof (*intro exI conjI*)
show $l \in S\ r \in S\ d\ l\ r\ /\ 3 > 0$
using *lr* **by** (*auto simp*: S'*_def*)
show $d\ l\ r\ /\ 3 < \varepsilon/2\ mcball\ l\ (d\ l\ r\ /\ 3) \subseteq mcball\ x\ \varepsilon\ mcball\ r\ (d\ l\ r\ /\ 3) \subseteq mcball\ x\ \varepsilon$
using *lr* **by** (*clarsimp simp*: S'*_def*, *smt* (*verit*) *commute triangle''*)+
show *disjnt* (*mcball l* (*d l r / 3*)) (*mcball r* (*d l r / 3*))
using *lr* **by** (*simp add*: S'*_def disjnt_iff*) (*smt* (*verit*, *best*) *mdist_pos_less triangle'*)
qed
qed
then obtain $l\ r\ \delta$
where *lrS*: $\bigwedge x\ \varepsilon.\ [\![x \in S;\ 0 < \varepsilon]\!] \Longrightarrow l\ x\ \varepsilon \in S \wedge r\ x\ \varepsilon \in S$
and δ: $\bigwedge x\ \varepsilon.\ [\![x \in S;\ 0 < \varepsilon]\!] \Longrightarrow 0 < \delta\ x\ \varepsilon \wedge \delta\ x\ \varepsilon < \varepsilon/2$
and $\bigwedge x\ \varepsilon.\ [\![x \in S;\ 0 < \varepsilon]\!] \Longrightarrow$ *mcball* (*l x* ε) (δ *x* ε) $\subseteq$ *mcball x* $\varepsilon \wedge$ *mcball* (*r x* ε) (δ *x* ε) $\subseteq$ *mcball x* $\varepsilon \wedge$ *disjnt* (*mcball* (*l x* ε) (δ *x* ε)) (*mcball* (*r x* ε) (δ *x* ε))
by *metis*
then have *lr_mcball*: $\bigwedge x\ \varepsilon.\ [\![x \in S;\ 0 < \varepsilon]\!] \Longrightarrow mcball\ (l\ x\ \varepsilon)\ (\delta\ x\ \varepsilon) \subseteq mcball\ x\ \varepsilon \wedge mcball\ (r\ x\ \varepsilon)\ (\delta\ x\ \varepsilon) \subseteq mcball\ x\ \varepsilon$
and *lr_disjnt*: $\bigwedge x\ \varepsilon.\ [\![x \in S;\ 0 < \varepsilon]\!] \Longrightarrow disjnt\ (mcball\ (l\ x\ \varepsilon)\ (\delta\ x\ \varepsilon))\ (mcball\ (r\ x\ \varepsilon)\ (\delta\ x\ \varepsilon))$
by *metis*+
obtain a **where** $a \in S$
using ‹$S \neq \{\}$› **by** *blast*
define *xe* **where** *xe* $\equiv$
$\lambda B.$ *rec_nat* $(a,1)$ $(\lambda n\ (x,\gamma).\ ((if\ n{\in}B\ then\ r\ else\ l)\ x\ \gamma,\ \delta\ x\ \gamma))$
have [*simp*]: *xe b 0* $= (a,1)$ **for** *b*
by (*simp add*: *xe_def*)
have *xe B* (*Suc n*) = (*let* (x,γ) = *xe B n in* $((if\ n{\in}B\ then\ r\ else\ l)\ x\ \gamma,\ \delta\ x\ \gamma))$ **for** *B n*
by (*simp add*: *xe_def*)
define x **where** $x \equiv \lambda B\ n.$ *fst* (*xe B n*)
define γ **where** $\gamma \equiv \lambda B\ n.$ *snd* (*xe B n*)
have [*simp*]: $x\ B\ 0 = a\ \gamma\ B\ 0 = 1$ **for** B
by (*simp_all add*: *x_def* γ*_def xe_def*)
have *x_Suc*[*simp*]: *x B* (*Suc n*) = ((*if* $n{\in}B$ *then r else l*) (*x B n*) (γ *B n*))
and γ*_Suc*[*simp*]: γ *B* (*Suc n*) = δ (*x B n*) (γ *B n*) **for** *B n*
by (*simp_all add*: *x_def* γ*_def xe_def split*: *prod.split*)
interpret *Submetric M d S*

```
  proof qed (use ‹S ⊆ M› in metis)
  have closedin mtopology S
    by (metis assms(2) closure_of closure_of_eq inf.absorb_iff2 subset sub-
set_Un_eq subset_refl topspace_mtopology)
  with ‹mcomplete›
  have sub.mcomplete
   by (metis closedin_mcomplete_imp_mcomplete)
  have ∗: x B n ∈ S ∧ γ B n > 0 for B n
   by (induction n) (auto simp: ‹a ∈ S› lrS δ)
  with subset have E: x B n ∈ M for B n
   by blast
  have γ_le: γ B n ≤ (1/2)^n for B n
  proof(induction n)
   case 0 then show ?case by auto
  next
   case (Suc n)
   then show ?case
    by simp (smt (verit) ∗ δ field_sum_of_halves)
  qed
  { fix B
   have ⋀n. sub.mcball (x B (Suc n)) (γ B (Suc n)) ⊆ sub.mcball (x B n) (γ B
n)
    by (smt (verit, best) ∗ Int_iff γ_Suc x_Suc in_mono lr_mcball mcball_submetric_eq
subsetI)
    then have mon: monotone (≤) (λx y. y ⊆ x) (λn. sub.mcball (x B n) (γ B
n))
     by (simp add: decseq_SucI)
    have ∃ n a. sub.mcball (x B n) (γ B n) ⊆ sub.mcball a ε if ε>0 for ε
    proof −
     obtain n where (1/2)^n < ε
      using ‹0 < ε› real_arch_pow_inv by force
     with γ_le have ε: γ B n ≤ ε
      by (smt (verit))
     show ?thesis
     proof (intro exI)
      show sub.mcball (x B n) (γ B n) ⊆ sub.mcball (x B n) ε
       by (simp add: ε sub.mcball_subset_concentric)
     qed
    qed
    then have ∃ l. l ∈ S ∧ (⋂n. sub.mcball (x B n) (γ B n)) = {l}
     using ‹sub.mcomplete› mon
     unfolding sub.mcomplete_nest_sing
     apply (drule_tac x=λn. sub.mcball (x B n) (γ B n) in spec)
     by (meson ∗ order.asym sub.closedin_mcball sub.mcball_eq_empty)
  }
  then obtain z where z: ⋀B. z B ∈ S ∧ (⋂n. sub.mcball (x B n) (γ B n)) =
{z B}
   by metis
  show ?thesis
```

```
    unfolding lepoll_def
  proof (intro exI conjI)
    show inj z
    proof (rule inj_onCI)
      fix B C
      assume eq: z B = z C and B ≠ C
      then have ne: sym_diff B C ≠ {}
        by blast
      define n where n ≡ LEAST k. k ∈ (sym_diff B C)
      with ne have n: n ∈ sym_diff B C
        by (metis Inf_nat_def1 LeastI)
      then have non: n ∈ B ⟷ n ∉ C
        by blast
      have H: z C ∈ sub.mcball (x B (Suc n)) (γ B (Suc n)) ∧ z C ∈ sub.mcball
(x C (Suc n)) (γ C (Suc n))
        using z [of B] z [of C] apply (simp add: lrS set_eq_iff non ∗)
        by (smt (verit, best) γ_Suc eq non x_Suc)
      have k ∈ B ⟷ k ∈ C if k<n for k
        using that unfolding n_def by (meson DiffI UnCI not_less_Least)
      moreover have (∀ m. m < p ⟶ (m ∈ B ⟷ m ∈ C)) ⟹ x B p = x C
p ∧ γ B p = γ C p for p
        by (induction p) auto
      ultimately have x B n = x C n γ B n = γ C n
        by blast+
      then show False
        using lr_disjnt ∗ H non
        by (smt (verit) IntD2 γ_Suc disjnt_iff mcball_submetric_eq x_Suc)
    qed
    show range z ⊆ S
      using z by blast
  qed
 qed
 finally show ?thesis .
qed

lemma lepoll_perfect_set_aux:
 assumes lcX: locally_compact_space X and hsX: Hausdorff_space X
   and eq: X derived_set_of topspace X = topspace X and topspace X ≠ {}
 shows (UNIV::real set) ≲ topspace X
proof −
 have (UNIV::real set) ≲ (UNIV::nat set set)
   using eqpoll_imp_lepoll eqpoll_sym nat_sets_eqpoll_reals by blast
 also have ... ≲ topspace X
 proof −
   obtain z where z: z ∈ topspace X
     using assms by blast
   then obtain U K where openin X U compactin X K U ≠ {} U ⊆ K
     by (metis emptyE lcX locally_compact_space_def)
   then have closedin X K
```

by (*simp add*: *compactin_imp_closedin hsX*)
have *intK_ne*: *X interior_of K* ≠ {}
using ‹*U* ≠ {}› ‹*U* ⊆ *K*› ‹*openin X U*› *interior_of_eq_empty* **by** *blast*
have ∃ *D E*. *closedin X D* ∧ *D* ⊆ *K* ∧ *X interior_of D* ≠ {} ∧
closedin X E ∧ *E* ⊆ *K* ∧ *X interior_of E* ≠ {} ∧
disjnt D E ∧ *D* ⊆ *C* ∧ *E* ⊆ *C*
if *closedin X C C* ⊆ *K* **and** *C*: *X interior_of C* ≠ {} **for** *C*
proof −
obtain *z* **where** *z*: *z* ∈ *X interior_of C z* ∈ *topspace X*
using *C interior_of_subset_topspace* **by** *fastforce*
obtain *x y* **where** *x* ∈ *X interior_of C y* ∈ *X interior_of C x*≠*y*
by (*metis z eq in_derived_set_of openin_interior_of*)
then have *x* ∈ *topspace X y* ∈ *topspace X*
using *interior_of_subset_topspace* **by** *force*+
with *hsX* **obtain** *V W* **where** *openin X V openin X W x* ∈ *V y* ∈ *W disjnt V W*
by (*metis Hausdorff_space_def* ‹*x* ≠ *y*›)
have ∗: ⋀*W x*. *openin X W* ∧ *x* ∈ *W*
⟹ ∃ *U V*. *openin X U* ∧ *closedin X V* ∧ *x* ∈ *U* ∧ *U* ⊆ *V* ∧ *V* ⊆ *W*
using *lcX hsX locally_compact_Hausdorff_imp_regular_space neighbourhood_base_of_closedin neighbourhood_base_of*
by *metis*
obtain *M D* **where** *MD*: *openin X M closedin X D y* ∈ *M M* ⊆ *D D* ⊆ *X interior_of C* ∩ *W*
using ∗ [*of X interior_of C* ∩ *W y*]
using ‹*openin X W*› ‹*y* ∈ *W*› ‹*y* ∈ *X interior_of C*› **by** *fastforce*
obtain *N E* **where** *NE*: *openin X N closedin X E x* ∈ *N N* ⊆ *E E* ⊆ *X interior_of C* ∩ *V*
using ∗ [*of X interior_of C* ∩ *V x*]
using ‹*openin X V*› ‹*x* ∈ *V*› ‹*x* ∈ *X interior_of C*› **by** *fastforce*
show *?thesis*
proof (*intro exI conjI*)
show *X interior_of D* ≠ {} *X interior_of E* ≠ {}
using *MD NE* **by** (*fastforce simp*: *interior_of_def*)+
show *disjnt D E*
by (*meson MD*(5) *NE*(5) ‹*disjnt V W*› *disjnt_subset1 disjnt_sym le_inf_iff*)
qed (*use MD NE* ‹*C* ⊆ *K*› *interior_of_subset* **in** *force*)+
qed
then obtain *L R* **where**
LR: ⋀*C*. ⟦*closedin X C*; *C* ⊆ *K*; *X interior_of C* ≠ {}⟧
⟹ *closedin X* (*L C*) ∧ (*L C*) ⊆ *K* ∧ *X interior_of* (*L C*) ≠ {} ∧
closedin X (*R C*) ∧ (*R C*) ⊆ *K* ∧ *X interior_of* (*R C*) ≠ {}
and *disjLR*: ⋀*C*. ⟦*closedin X C*; *C* ⊆ *K*; *X interior_of C* ≠ {}⟧
⟹ *disjnt* (*L C*) (*R C*) ∧ (*L C*) ⊆ *C* ∧ (*R C*) ⊆ *C*
by *metis*
define *d* **where** *d* ≡ λ*B*. *rec_nat K* (λ*n*. *if n* ∈ *B then R else L*)
have *d0*[*simp*]: *d B* 0 = *K* **for** *B*
by (*simp add*: *d_def*)

```
    have [simp]: d B (Suc n) = (if n ∈ B then R else L) (d B n) for B n
      by (simp add: d_def)
    have d_correct: closedin X (d B n) ∧ d B n ⊆ K ∧ X interior_of (d B n) ≠
{} for B n
    proof (induction n)
      case 0
      then show ?case by (auto simp: ‹closedin X K› intK_ne)
    next
      case (Suc n) with LR show ?case by auto
    qed
    have (⋂n. d B n) ≠ {} for B
    proof (rule compact_space_imp_nest)
      show compact_space (subtopology X K)
        by (simp add: ‹compactin X K› compact_space_subtopology)
      show closedin (subtopology X K) (d B n) for n :: nat
        by (simp add: closedin_subset_topspace d_correct)
      show d B n ≠ {} for n :: nat
        by (metis d_correct interior_of_empty)
      show antimono (d B)
      proof (rule antimonoI [OF transitive_stepwise_le])
        fix n
        show d B (Suc n) ⊆ d B n
        by (simp add: d_correct disjLR)
      qed auto
    qed
    then obtain x where x: ⋀B. x B ∈ (⋂n. d B n)
      unfolding set_eq_iff by (metis empty_iff)
    show ?thesis
      unfolding lepoll_def
    proof (intro exI conjI)
      show inj x
      proof (rule inj_onCI)
        fix B C
        assume eq: x B = x C and B≠C
        then have ne: sym_diff B C ≠ {}
          by blast
        define n where n ≡ LEAST k. k ∈ (sym_diff B C)
        with ne have n: n ∈ sym_diff B C
          by (metis Inf_nat_def1 LeastI)
        then have non: n ∈ B ⟷ n ∉ C
          by blast
        have k ∈ B ⟷ k ∈ C if k<n for k
          using that unfolding n_def by (meson DiffI UnCI not_less_Least)
        moreover have (∀m. m < p ⟶ (m ∈ B ⟷ m ∈ C)) ⟹ d B p = d C
p for p
          by (induction p) auto
        ultimately have d B n = d C n
          by blast
        then have disjnt (d B (Suc n)) (d C (Suc n))
```

by (*simp add*: *d_correct disjLR disjnt_sym non*)
then show *False*
by (*metis InterE disjnt_iff eq rangeI x*)
qed
show *range x* ⊆ *topspace X*
using *x d0* ‹*compactin X K*› *compactin_subset_topspace d_correct* **by** *fastforce*
qed
qed
finally show *?thesis* **.**
qed

lemma *lepoll_perfect_set*:
assumes *X*: *completely_metrizable_space X* ∨ *locally_compact_space X* ∧ *Hausdorff_space X*
and *S*: *X derived_set_of S* = *S S* ≠ {}
shows (*UNIV*::*real set*) ≲ *S*
using *X*
proof
assume *completely_metrizable_space X*
with *assms* **show** (*UNIV*::*real set*) ≲ *S*
by (*metis Metric_space.lepoll_perfect_set completely_metrizable_space_def*)
next
assume *locally_compact_space X* ∧ *Hausdorff_space X*
then show (*UNIV*::*real set*) ≲ *S*
using *lepoll_perfect_set_aux* [*of subtopology X S*]
by (*metis Hausdorff_space_subtopology S closedin_derived_set_of closedin_subset derived_set_of_subtopology*
locally_compact_space_closed_subset subtopology_topspace topspace_subtopology topspace_subtopology_subset)
qed

lemma *Kuratowski_aux1*:
assumes ⋀*S T*. *R S T* ⟹ *R T S*
shows (∀ *S T n*. *R S T* ⟶ (*f S* ≈ {..<*n*::*nat*} ⟷ *f T* ≈ {..<*n*::*nat*})) ⟷
(∀ *n S T*. *R S T* ⟶ {..<*n*::*nat*} ≲ *f S* ⟶ {..<*n*::*nat*} ≲ *f T*)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*meson eqpoll_iff_finite_card eqpoll_sym finite_lepoll_infinite finite_lessThan lepoll_trans2*)
next
assume *?rhs* **then show** *?lhs*
by (*smt* (*verit*, *best*) *lepoll_iff_finite_card assms eqpoll_iff_finite_card finite_lepoll_infinite*
finite_lessThan le_Suc_eq lepoll_antisym lepoll_iff_card_le not_less_eq_eq)

qed

lemma *Kuratowski_aux2*:
pairwise (*separatedin* (*subtopology X* (*topspace X* − *S*))) $\mathcal{U}$ ∧ {} ∉ $\mathcal{U}$ ∧
$\bigcup\mathcal{U}$ = *topspace*(*subtopology X* (*topspace X* − *S*)) ⟷
pairwise (*separatedin X*) $\mathcal{U}$ ∧ {} ∉ $\mathcal{U}$ ∧ $\bigcup\mathcal{U}$ = *topspace X* − *S*
by (*auto simp*: *pairwise_def separatedin_subtopology*)

proposition *Kuratowski_component_number_invariance_aux*:
assumes *compact_space X* **and** *HsX*: *Hausdorff_space X*
and *lcX*: *locally_connected_space X* **and** *hnX*: *hereditarily normal_space X*
and *hom*: (*subtopology X S*) *homeomorphic_space* (*subtopology X T*)
and *leXS*: {..<*n*::*nat*} ≲ *connected_components_of* (*subtopology X* (*topspace X* − *S*))
assumes §: ⋀*S T*.
⟦*closedin X S*; *closedin X T*; (*subtopology X S*) *homeomorphic_space* (*subtopology X T*);
{..<*n*::*nat*} ≲ *connected_components_of* (*subtopology X* (*topspace X* − *S*))⟧
⟹ {..<*n*::*nat*} ≲ *connected_components_of* (*subtopology X* (*topspace X* − *T*))
shows {..<*n*::*nat*} ≲ *connected_components_of* (*subtopology X* (*topspace X* − *T*))
proof (*cases n=0*)
case *False*
obtain *f g* **where** *homf*: *homeomorphic_map* (*subtopology X S*) (*subtopology X T*) *f*
and *homg*: *homeomorphic_map* (*subtopology X T*) (*subtopology X S*) *g*
and *gf*: ⋀*x*. *x* ∈ *topspace* (*subtopology X S*) ⟹ *g*(*f x*) = *x*
and *fg*: ⋀*y*. *y* ∈ *topspace* (*subtopology X T*) ⟹ *f*(*g y*) = *y*
and *f*: *f* ∈ *topspace* (*subtopology X S*) → *topspace* (*subtopology X T*)
and *g*: *g* ∈ *topspace* (*subtopology X T*) → *topspace* (*subtopology X S*)
using *homeomorphic_space_unfold hom* **by** *metis*
obtain *C* **where** *closedin X C C* ⊆ *S*
and *C*: ⋀*D*. ⟦*closedin X D*; *C* ⊆ *D*; *D* ⊆ *S*⟧
⟹ ∃$\mathcal{U}$. $\mathcal{U}$ ≈ {..<*n*} ∧ *pairwise* (*separatedin X*) $\mathcal{U}$ ∧ {} ∉ $\mathcal{U}$ ∧ $\bigcup\mathcal{U}$ = *topspace X* − *D*
using *separation_by_closed_intermediates_eq_count* [*of X n S*] *assms*
by (*smt* (*verit*, *ccfv_threshold*) *False Kuratowski_aux2 lepoll_connected_components_alt*)
have ∃ *C*. *closedin X C* ∧ *C* ⊆ *T* ∧
(∀ *D*. *closedin X D* ∧ *C* ⊆ *D* ∧ *D* ⊆ *T*
⟶ (∃$\mathcal{U}$. $\mathcal{U}$ ≈ {..<*n*} ∧ *pairwise* (*separatedin X*) $\mathcal{U}$ ∧
{} ∉ $\mathcal{U}$ ∧ $\bigcup\mathcal{U}$ = *topspace X* − *D*))
proof (*intro exI*, *intro conjI strip*)
have *compactin X* (*f* ‘ *C*)
by (*meson* ‹*C* ⊆ *S*› ‹*closedin X C*› *assms*(*1*) *closedin_compact_space compactin_subtopology homeomorphic_map_compactness_eq homf*)
then show *closedin X* (*f* ‘ *C*)
using ‹*Hausdorff_space X*› *compactin_imp_closedin* **by** *blast*

show f ‘ $C \subseteq T$
by (*meson* ‹$C \subseteq S$› ‹*closedin X C*› *closedin_imp_subset closedin_subset_topspace homeomorphic_map_closedness_eq homf*)
fix D'
assume D': *closedin X* $D' \wedge f$ ‘ $C \subseteq D' \wedge D' \subseteq T$
define D **where** $D \equiv g$ ‘ D'
have *compactin X D*
unfolding *D_def*
by (*meson* D' ‹*compact_space X*› *closedin_compact_space compactin_subtopology homeomorphic_map_compactness_eq homg*)
then have *closedin X D*
by (*simp add*: *assms*(*2*) *compactin_imp_closedin*)
moreover have $C \subseteq D$
using D' *D_def* ‹$C \subseteq S$› ‹*closedin X C*› *closedin_subset gf image_iff* **by** *fastforce*
moreover have $D \subseteq S$
by (*metis* D' *D_def assms*(*1*) *closedin_compact_space compactin_subtopology homeomorphic_map_compactness_eq homg*)
ultimately obtain $\mathcal{U}$ **where** $\mathcal{U} \approx \{..<n\}$ *pairwise* (*separatedin X*) $\mathcal{U}$ $\{\} \notin \mathcal{U}$ $\bigcup\mathcal{U}$ = *topspace X* − *D*
using C **by** *meson*
moreover have (*subtopology X D*) *homeomorphic_space* (*subtopology X* D')
unfolding *homeomorphic_space_def*
proof (*intro exI*)
have *subtopology X D* = *subtopology* (*subtopology X S*) *D*
by (*simp add*: ‹$D \subseteq S$› *inf.absorb2 subtopology_subtopology*)
moreover have *subtopology X* D' = *subtopology* (*subtopology X T*) D'
by (*simp add*: D' *inf.absorb2 subtopology_subtopology*)
moreover have *homeomorphic_maps* (*subtopology X T*) (*subtopology X S*) *g f*
by (*simp add*: *fg gf homeomorphic_maps_map homf homg*)
ultimately
have *homeomorphic_maps* (*subtopology X* D') (*subtopology X D*) *g f*
by (*metis* D' *D_def* ‹*closedin X D*› *closedin_subset homeomorphic_maps_subtopologies topspace_subtopology Int_absorb1*)
then show *homeomorphic_maps* (*subtopology X D*) (*subtopology X* D') *f g*
using *homeomorphic_maps_sym* **by** *blast*
qed
ultimately show $\exists\mathcal{U}.\ \mathcal{U} \approx \{..<n\} \wedge$ *pairwise* (*separatedin X*) $\mathcal{U} \wedge \{\} \notin \mathcal{U} \wedge \bigcup\ \mathcal{U}$ = *topspace X* − D'
by (*smt* (*verit*, *ccfv_SIG*) § D' *False* ‹*closedin X D*› *Kuratowski_aux2 lepoll_connected_components_alt*)
qed
then have $\exists\mathcal{U}.\ \mathcal{U} \approx \{..<n\} \wedge$
pairwise (*separatedin* (*subtopology X* (*topspace X* − *T*))) $\mathcal{U} \wedge \{\} \notin \mathcal{U} \wedge \bigcup\mathcal{U}$ = *topspace X* − *T*
using *separation_by_closed_intermediates_eq_count* [*of X n T*] *Kuratowski_aux2 lcX hnX* **by** *auto*
with *False* **show** *?thesis*

using *lepoll_connected_components_alt* **by** *fastforce*
qed *auto*

theorem *Kuratowski_component_number_invariance*:
assumes *compact_space X Hausdorff_space X locally_connected_space X hereditarily normal_space X*
shows ((∀ *S T n*.
closedin X S ∧ *closedin X T* ∧
(*subtopology X S*) *homeomorphic_space* (*subtopology X T*)
⟶ (*connected_components_of*
(*subtopology X* (*topspace X* − *S*)) ≈ {..<*n*::*nat*} ⟷
connected_components_of
(*subtopology X* (*topspace X* − *T*)) ≈ {..<*n*::*nat*})) ⟷
(∀ *S T n*.
(*subtopology X S*) *homeomorphic_space* (*subtopology X T*)
⟶ (*connected_components_of*
(*subtopology X* (*topspace X* − *S*)) ≈ {..<*n*::*nat*} ⟷
connected_components_of
(*subtopology X* (*topspace X* − *T*)) ≈ {..<*n*::*nat*})))
(**is** *?lhs* = *?rhs*)
proof
assume *L*: *?lhs*
then show *?rhs*
apply (*subst* (*asm*) *Kuratowski_aux1*, *use homeomorphic_space_sym* **in** *blast*)
apply (*subst Kuratowski_aux1*, *use homeomorphic_space_sym* **in** *blast*)
apply (*blast intro*: *Kuratowski_component_number_invariance_aux assms*)
done
qed *blast*

end
theory *Isolated*
imports *HOL−Analysis.Elementary_Metric_Spaces*

begin

5.11.22 Isolate and discrete

definition (**in** *topological_space*) *isolated_in*:: *'a* ⇒ *'a set* ⇒ *bool* (**infixr** *isolated'_in 60*)
where *x isolated_in S* ⟷ (*x*∈*S* ∧ (∃ *T*. *open T* ∧ *T* ∩ *S* = {*x*}))

definition (**in** *topological_space*) *discrete*:: *'a set* ⇒ *bool*
where *discrete S* ⟷ (∀ *x*∈*S*. *x isolated_in S*)

definition (**in** *metric_space*) *uniform_discrete* :: *'a set* ⇒ *bool* **where**
uniform_discrete S ⟷ (∃ *e*>0. ∀ *x*∈*S*. ∀ *y*∈*S*. *dist x y* < *e* ⟶ *x* = *y*)

lemma *discreteI*: (⋀*x*. *x* ∈ *X* ⟹ *x isolated_in X*) ⟹ *discrete X*

unfolding *discrete_def* **by** *auto*

lemma *discreteD*: *discrete X* $\Longrightarrow$ $x \in X$ $\Longrightarrow$ *x isolated_in X*
unfolding *discrete_def* **by** *auto*

lemma *uniformI1*:
assumes *e>0* $\bigwedge x\ y.$ $[\![x{\in}S;y{\in}S;dist\ x\ y{<}e]\!] \Longrightarrow x = y$
shows *uniform_discrete S*
unfolding *uniform_discrete_def* **using** *assms* **by** *auto*

lemma *uniformI2*:
assumes *e>0* $\bigwedge x\ y.$ $[\![x{\in}S;y{\in}S;x{\neq}y]\!] \Longrightarrow dist\ x\ y{\geq}e$
shows *uniform_discrete S*
unfolding *uniform_discrete_def* **using** *assms not_less* **by** *blast*

lemma *isolated_in_islimpt_iff*:$(x\ isolated\_in\ S) \longleftrightarrow (\neg\ (x\ islimpt\ S) \wedge x{\in}S)$
unfolding *isolated_in_def islimpt_def* **by** *auto*

lemma *isolated_in_dist_Ex_iff*:
fixes *x*::*'a*::*metric_space*
shows *x isolated_in S* $\longleftrightarrow$ $(x{\in}S \wedge (\exists\, e{>}0.\ \forall\, y{\in}S.\ dist\ x\ y < e \longrightarrow y{=}x))$
unfolding *isolated_in_islimpt_iff islimpt_approachable* **by** (*metis dist_commute*)

lemma *discrete_empty*[*simp*]: *discrete* {}
unfolding *discrete_def* **by** *auto*

lemma *uniform_discrete_empty*[*simp*]: *uniform_discrete* {}
unfolding *uniform_discrete_def* **by** (*simp add*: *gt_ex*)

lemma *isolated_in_insert*:
fixes *x* :: *'a*::*t1_space*
shows *x isolated_in* (*insert a S*) $\longleftrightarrow$ *x isolated_in S* $\vee$ $(x{=}a \wedge \neg\ (x\ islimpt\ S))$
by (*meson insert_iff islimpt_insert isolated_in_islimpt_iff*)

lemma *isolated_inI*:
assumes *x*$\in$*S open T* $T \cap S = \{x\}$
shows *x isolated_in S*
using *assms* **unfolding** *isolated_in_def* **by** *auto*

lemma *isolated_inE*:
assumes *x isolated_in S*
obtains *T* **where** $x \in S$ *open T* $T \cap S = \{x\}$
using *assms that* **unfolding** *isolated_in_def* **by** *force*

lemma *isolated_inE_dist*:
assumes *x isolated_in S*
obtains *d* **where** $d > 0$ $\bigwedge y.\ y \in S \Longrightarrow dist\ x\ y < d \Longrightarrow y = x$
by (*meson assms isolated_in_dist_Ex_iff*)

lemma *isolated_in_altdef*:
x isolated_in S ⟷ (*x*∈*S* ∧ *eventually* (λ*y*. *y* ∉ *S*) (*at x*))
proof
assume *x isolated_in S*
from *isolated_inE*[*OF this*]
obtain *T* **where** *x* ∈ *S* **and** *T*:*open T T* ∩ *S* = {*x*}
by *metis*
have ∀ *F y in nhds x*. *y* ∈ *T*
apply (*rule eventually_nhds_in_open*)
using *T* **by** *auto*
then have *eventually* (λ*y*. *y* ∈ *T* − {*x*}) (*at x*)
unfolding *eventually_at_filter* **by** *eventually_elim auto*
then have *eventually* (λ*y*. *y* ∉ *S*) (*at x*)
by *eventually_elim* (*use T* **in** *auto*)
then show *x* ∈ *S* ∧ (∀ *F y in at x*. *y* ∉ *S*) **using** ‹*x* ∈ *S*› **by** *auto*
next
assume *x* ∈ *S* ∧ (∀ *F y in at x*. *y* ∉ *S*)
then have ∀ *F y in at x*. *y* ∉ *S x*∈*S* **by** *auto*
from *this*(*1*) **have** *eventually* (λ*y*. *y* ∉ *S* ∨ *y* = *x*) (*nhds x*)
unfolding *eventually_at_filter* **by** *eventually_elim auto*
then obtain *T* **where** *T*:*open T x* ∈ *T* (∀ *y*∈*T*. *y* ∉ *S* ∨ *y* = *x*)
unfolding *eventually_nhds* **by** *auto*
with ‹*x* ∈ *S*› **have** *T* ∩ *S* = {*x*}
by *fastforce*
with ‹*x*∈*S*› ‹*open T*›
show *x isolated_in S*
unfolding *isolated_in_def* **by** *auto*
qed

lemma *discrete_altdef*:
discrete S ⟷ (∀ *x*∈*S*. ∀ *F y in at x*. *y* ∉ *S*)
unfolding *discrete_def isolated_in_altdef* **by** *auto*

lemma *uniform_discrete_imp_closed*:
uniform_discrete S ⟹ *closed S*
by (*meson discrete_imp_closed uniform_discrete_def*)

lemma *uniform_discrete_imp_discrete*:
uniform_discrete S ⟹ *discrete S*
by (*metis discrete_def isolated_in_dist_Ex_iff uniform_discrete_def*)

lemma *isolated_in_subset*:*x isolated_in S* ⟹ *T* ⊆ *S* ⟹ *x*∈*T* ⟹ *x isolated_in T*
unfolding *isolated_in_def* **by** *fastforce*

lemma *discrete_subset*[*elim*]: *discrete S* ⟹ *T* ⊆ *S* ⟹ *discrete T*
unfolding *discrete_def* **using** *islimpt_subset isolated_in_islimpt_iff* **by** *blast*

lemma *uniform_discrete_subset*[*elim*]: *uniform_discrete S* ⟹ *T* ⊆ *S* ⟹ *uniform_discrete T*
by (*meson subsetD uniform_discrete_def*)

lemma *continuous_on_discrete*: *discrete S* ⟹ *continuous_on S f*
unfolding *continuous_on_topological* **by** (*metis discrete_def islimptI isolated_in_islimpt_iff*)

lemma *uniform_discrete_insert*: *uniform_discrete* (*insert a S*) ⟷ *uniform_discrete S*
proof
assume *asm*:*uniform_discrete S*
let *?thesis* = *uniform_discrete* (*insert a S*)
have *?thesis* **when** *a*∈*S* **using** *that asm* **by** (*simp add*: *insert_absorb*)
moreover have *?thesis* **when** *S*={} **using** *that asm* **by** (*simp add*: *uniform_discrete_def*)
moreover have *?thesis* **when** *a*∉*S S*≠{}
proof −
obtain *e1* **where** *e1*>*0* **and** *e1_dist*:∀ *x*∈*S*. ∀ *y*∈*S*. *dist y x* < *e1* ⟶ *y* = *x*
using *asm* **unfolding** *uniform_discrete_def* **by** *auto*
define *e2* **where** *e2* ≡ *min* (*setdist* {*a*} *S*) *e1*
have *closed S* **using** *asm uniform_discrete_imp_closed* **by** *auto*
then have *e2*>*0*
by (*smt* (*verit*) ‹*0* < *e1*› *e2_def infdist_eq_setdist infdist_pos_not_in_closed that*)
moreover have *x* = *y* **if** *x*∈*insert a S y*∈*insert a S dist x y* < *e2* **for** *x y*
proof (*cases x*=*a* ∨ *y*=*a*)
case *True* **then show** *?thesis*
by (*smt* (*verit*, *best*) *dist_commute e2_def infdist_eq_setdist infdist_le insertE that*)
next
case *False* **then show** *?thesis*
using *e1_dist e2_def that* **by** *force*
qed
ultimately show *?thesis* **unfolding** *uniform_discrete_def* **by** *meson*
qed
ultimately show *?thesis* **by** *auto*
qed (*simp add*: *subset_insertI uniform_discrete_subset*)

lemma *discrete_compact_finite_iff*:
fixes *S* :: *'a*::*t1_space set*
shows *discrete S* ∧ *compact S* ⟷ *finite S*
proof
assume *finite S*
then have *compact S* **using** *finite_imp_compact* **by** *auto*
moreover have *discrete S*
unfolding *discrete_def* **using** *isolated_in_islimpt_iff islimpt_finite*[*OF* ‹*finite S*›] **by** *auto*
ultimately show *discrete S* ∧ *compact S* **by** *auto*
next

assume *discrete S* ∧ *compact S*
then show *finite S*
by (*meson discrete_def Heine_Borel_imp_Bolzano_Weierstrass isolated_in_islimpt_iff order_refl*)
qed

lemma *uniform_discrete_finite_iff*:
fixes *S* :: *'a*::*heine_borel set*
shows *uniform_discrete S* ∧ *bounded S* ⟷ *finite S*
proof
assume *uniform_discrete S* ∧ *bounded S*
then have *discrete S compact S*
using *uniform_discrete_imp_discrete uniform_discrete_imp_closed compact_eq_bounded_closed*
by *auto*
then show *finite S* **using** *discrete_compact_finite_iff* **by** *auto*
next
assume *asm*:*finite S*
let *?thesis* = *uniform_discrete S* ∧ *bounded S*
have *?thesis* **when** *S*={} **using** *that* **by** *auto*
moreover have *?thesis* **when** *S*≠{}
proof −
have ∀ *x*. ∃ *d*>*0*. ∀ *y*∈*S*. *y* ≠ *x* ⟶ *d* ≤ *dist x y*
using *finite_set_avoid*[*OF* ‹*finite S*›] **by** *auto*
then obtain *f* **where** *f_pos*:*f x*>*0*
and *f_dist*: ∀ *y*∈*S*. *y* ≠ *x* ⟶ *f x* ≤ *dist x y*
if *x*∈*S* **for** *x*
by *metis*
define *f_min* **where** *f_min* ≡ *Min* (*f* ‘ *S*)
have *f_min* > *0*
unfolding *f_min_def*
by (*simp add*: *asm f_pos that*)
moreover have ∀ *x*∈*S*. ∀ *y*∈*S*. *f_min* > *dist x y* ⟶ *x*=*y*
using *f_dist* **unfolding** *f_min_def*
by (*metis Min_le asm finite_imageI imageI le_less_trans linorder_not_less*)
ultimately have *uniform_discrete S*
unfolding *uniform_discrete_def* **by** *auto*
moreover have *bounded S* **using** ‹*finite S*› **by** *auto*
ultimately show *?thesis* **by** *auto*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *uniform_discrete_image_scale*:
assumes *uniform_discrete S* **and** *dist*:∀ *x*∈*S*. ∀ *y*∈*S*. *dist x y* = *c* ∗ *dist* (*f x*) (*f y*)
shows *uniform_discrete* (*f* ‘ *S*)
proof −
have *?thesis* **when** *S*={} **using** *that* **by** *auto*
moreover have *?thesis* **when** *S*≠{} *c*≤*0*

```
  proof −
    obtain x1 where x1∈S using ‹S≠{}› by auto
    have ?thesis when S−{x1} = {}
      using ‹x1 ∈ S› subset_antisym that uniform_discrete_insert by fastforce
    moreover have ?thesis when S−{x1} ≠ {}
    proof −
      obtain x2 where x2∈S−{x1} using ‹S−{x1} ≠ {}› by auto
      then have x2∈S x1≠x2 by auto
      then have dist x1 x2 > 0 by auto
      moreover have dist x1 x2 = c * dist (f x1) (f x2)
        by (simp add: ‹x1 ∈ S› ‹x2 ∈ S› dist)
      moreover have dist (f x2) (f x2) ≥ 0 by auto
      ultimately have False using ‹c≤0› by (simp add: zero_less_mult_iff)
      then show ?thesis by auto
    qed
    ultimately show ?thesis by auto
  qed
  moreover have ?thesis when S≠{} c>0
  proof −
    obtain e1 where e1>0 and e1_dist:∀ x∈S. ∀ y∈S. dist y x < e1 ⟶ y = x
      using ‹uniform_discrete S› unfolding uniform_discrete_def by auto
    define e where e ≡ e1/c
    have x1 = x2 when x1 ∈ f ‘ S x2 ∈ f ‘ S and d: dist x1 x2 < e for x1 x2
      by (smt (verit) ‹0 < c› d dist divide_right_mono e1_dist e_def imageE
nonzero_mult_div_cancel_left that)
    moreover have e>0 using ‹e1>0› ‹c>0› unfolding e_def by auto
    ultimately show ?thesis unfolding uniform_discrete_def by meson
  qed
  ultimately show ?thesis by fastforce
qed

definition sparse :: real ⇒ 'a :: metric_space set ⇒ bool
  where sparse ε X ⟷ (∀ x∈X. ∀ y∈X−{x}. dist x y > ε)

lemma sparse_empty [simp, intro]: sparse ε {}
  by (auto simp: sparse_def)

lemma sparseI [intro?]:
  (⋀x y. x ∈ X ⟹ y ∈ X ⟹ x ≠ y ⟹ dist x y > ε) ⟹ sparse ε X
  unfolding sparse_def by auto

lemma sparseD:
  sparse ε X ⟹ x ∈ X ⟹ y ∈ X ⟹ x ≠ y ⟹ dist x y > ε
  unfolding sparse_def by auto

lemma sparseD':
  sparse ε X ⟹ x ∈ X ⟹ y ∈ X ⟹ dist x y ≤ ε ⟹ x = y
  unfolding sparse_def by force
```

lemma *sparse_singleton* [*simp*, *intro*]: *sparse* ε $\{x\}$
by (*auto simp*: *sparse_def*)

definition *setdist_gt* **where** *setdist_gt* ε X Y $\longleftrightarrow$ $(\forall\, x{\in}X.\ \forall\, y{\in}Y.\ dist\ x\ y > \varepsilon)$

lemma *setdist_gt_empty* [*simp*]: *setdist_gt* ε $\{\}$ Y *setdist_gt* ε X $\{\}$
by (*auto simp*: *setdist_gt_def*)

lemma *setdist_gtI*: $(\bigwedge x\ y.\ x \in X \Longrightarrow y \in Y \Longrightarrow dist\ x\ y > \varepsilon) \Longrightarrow$ *setdist_gt* ε X Y
unfolding *setdist_gt_def* **by** *auto*

lemma *setdist_gtD*: *setdist_gt* ε X $Y \Longrightarrow x \in X \Longrightarrow y \in Y \Longrightarrow dist\ x\ y > \varepsilon$
unfolding *setdist_gt_def* **by** *auto*

lemma *setdist_gt_setdist*: $\varepsilon <$ *setdist* A $B \Longrightarrow$ *setdist_gt* ε A B
unfolding *setdist_gt_def* **using** *setdist_le_dist* **by** *fastforce*

lemma *setdist_gt_mono*: *setdist_gt* ε' A $B \Longrightarrow \varepsilon \le \varepsilon' \Longrightarrow A' \subseteq A \Longrightarrow B' \subseteq B \Longrightarrow$ *setdist_gt* ε A' B'
by (*force simp*: *setdist_gt_def*)

lemma *setdist_gt_Un_left*: *setdist_gt* ε $(A \cup B)$ $C \longleftrightarrow$ *setdist_gt* ε A $C \wedge$ *setdist_gt* ε B C
by (*auto simp*: *setdist_gt_def*)

lemma *setdist_gt_Un_right*: *setdist_gt* ε C $(A \cup B) \longleftrightarrow$ *setdist_gt* ε C $A \wedge$ *setdist_gt* ε C B
by (*auto simp*: *setdist_gt_def*)

lemma *compact_closed_imp_eventually_setdist_gt_at_right_0*:
assumes *compact* A *closed* B $A \cap B = \{\}$
shows *eventually* $(\lambda\varepsilon.$ *setdist_gt* ε A $B)$ (*at_right* 0)
proof (*cases* $A = \{\} \vee B = \{\}$)
case *False*
hence *setdist* A $B > 0$
by (*metis IntI assms empty_iff in_closed_iff_infdist_zero order_less_le setdist_attains_inf setdist_pos_le setdist_sym*)
hence *eventually* $(\lambda\varepsilon.\ \varepsilon <$ *setdist* A $B)$ (*at_right* 0)
using *eventually_at_right_field* **by** *blast*
thus *?thesis*
by *eventually_elim* (*auto intro*: *setdist_gt_setdist*)
qed *auto*

lemma *setdist_gt_symI*: *setdist_gt* ε A $B \Longrightarrow$ *setdist_gt* ε B A
by (*force simp*: *setdist_gt_def dist_commute*)

lemma *setdist_gt_sym*: *setdist_gt* ε A $B \longleftrightarrow$ *setdist_gt* ε B A
by (*force simp*: *setdist_gt_def dist_commute*)

lemma *eventually_setdist_gt_at_right_0_mult_iff*:
assumes $c > 0$
shows *eventually* $(\lambda\varepsilon.\ setdist\_gt\ (c * \varepsilon)\ A\ B)\ (at\_right\ 0) \longleftrightarrow$
eventually $(\lambda\varepsilon.\ setdist\_gt\ \varepsilon\ A\ B)\ (at\_right\ 0)$
proof −
have *eventually* $(\lambda\varepsilon.\ setdist\_gt\ (c * \varepsilon)\ A\ B)\ (at\_right\ 0) \longleftrightarrow$
eventually $(\lambda\varepsilon.\ setdist\_gt\ \varepsilon\ A\ B)\ (filtermap\ ((*)\ c)\ (at\_right\ 0))$
by (*simp add*: *eventually_filtermap*)
also have *filtermap* $((*)\ c)\ (at\_right\ 0) = at\_right\ 0$
by (*subst filtermap_times_pos_at_right*) (*use assms* **in** *auto*)
finally show *?thesis* **.**
qed

end

5.12 Operator Norm

theory *Operator_Norm*
imports *Complex_Main*
begin

This formulation yields zero if $'a$ is the trivial vector space.

definition
onorm :: $('a$::*real_normed_vector* $\Rightarrow$ $'b$::*real_normed_vector*$) \Rightarrow$ *real* **where**
onorm $f = (SUP\ x.\ norm\ (f\ x)\ /\ norm\ x)$

proposition *onorm_bound*:
assumes $0 \le b$ **and** $\bigwedge x.\ norm\ (f\ x) \le b * norm\ x$
shows *onorm* $f \le b$
unfolding *onorm_def*
proof (*rule cSUP_least*)
fix x
show $norm\ (f\ x)\ /\ norm\ x \le b$
using *assms* **by** (*cases* $x = 0$) (*simp_all add*: *pos_divide_le_eq*)
qed *simp*

In non-trivial vector spaces, the first assumption is redundant.

lemma *onorm_le*:
fixes $f :: 'a$::{*real_normed_vector*, *perfect_space*}$ \Rightarrow 'b$::*real_normed_vector*
assumes $\bigwedge x.\ norm\ (f\ x) \le b * norm\ x$
shows *onorm* $f \le b$
proof (*rule onorm_bound* [*OF* _ *assms*])
have $\{0::'a\} \neq UNIV$ **by** (*metis not_open_singleton open_UNIV*)
then obtain $a :: 'a$ **where** $a \neq 0$ **by** *fast*
have $0 \le b * norm\ a$
by (*rule order_trans* [*OF norm_ge_zero assms*])
with ‹$a \neq 0$› **show** $0 \le b$

```
    by (simp add: zero_le_mult_iff)
qed

lemma le_onorm:
  assumes bounded_linear f
  shows norm (f x) / norm x ≤ onorm f
proof −
  interpret f: bounded_linear f by fact
  obtain b where 0 ≤ b and ∀ x. norm (f x) ≤ norm x ∗ b
    using f.nonneg_bounded by auto
  then have ∀ x. norm (f x) / norm x ≤ b
    by (clarify, case_tac x = 0,
      simp_all add: f.zero pos_divide_le_eq mult.commute)
  then have bdd_above (range (λx. norm (f x) / norm x))
    unfolding bdd_above_def by fast
  with UNIV_I show ?thesis
    unfolding onorm_def by (rule cSUP_upper)
qed

lemma onorm:
  assumes bounded_linear f
  shows norm (f x) ≤ onorm f ∗ norm x
proof −
  interpret f: bounded_linear f by fact
  show ?thesis
  proof (cases)
    assume x = 0
    then show ?thesis by (simp add: f.zero)
  next
    assume x ≠ 0
    have norm (f x) / norm x ≤ onorm f
      by (rule le_onorm [OF assms])
    then show norm (f x) ≤ onorm f ∗ norm x
      by (simp add: pos_divide_le_eq ‹x ≠ 0›)
  qed
qed

lemma onorm_pos_le:
  assumes f: bounded_linear f
  shows 0 ≤ onorm f
  using le_onorm [OF f, where x=0] by simp

lemma onorm_zero: onorm (λx. 0) = 0
proof (rule order_antisym)
  show onorm (λx. 0) ≤ 0
    by (simp add: onorm_bound)
  show 0 ≤ onorm (λx. 0)
    using bounded_linear_zero by (rule onorm_pos_le)
qed
```

lemma *onorm_eq_0*:
assumes *f*: *bounded_linear f*
shows *onorm f = 0* $\longleftrightarrow$ *(∀ x. f x = 0)*
using *onorm* [*OF f*] **by** (*auto simp*: *fun_eq_iff* [*symmetric*] *onorm_zero*)

lemma *onorm_pos_lt*:
assumes *f*: *bounded_linear f*
shows *0 < onorm f* $\longleftrightarrow$ *¬ (∀ x. f x = 0)*
by (*simp add*: *less_le onorm_pos_le* [*OF f*] *onorm_eq_0* [*OF f*])

lemma *onorm_id_le*: *onorm (λx. x) ≤ 1*
by (*rule onorm_bound*) *simp_all*

lemma *onorm_id*: *onorm (λx. x::'a::{real_normed_vector, perfect_space}) = 1*
proof (*rule antisym*[*OF onorm_id_le*])
have *{0::'a} ≠ UNIV* **by** (*metis not_open_singleton open_UNIV*)
then obtain *x :: 'a* **where** *x ≠ 0* **by** *fast*
hence *1 ≤ norm x / norm x*
by *simp*
also have *... ≤ onorm (λx::'a. x)*
by (*rule le_onorm*) (*rule bounded_linear_ident*)
finally show *1 ≤ onorm (λx::'a. x)* **.**
qed

lemma *onorm_compose*:
assumes *f*: *bounded_linear f*
assumes *g*: *bounded_linear g*
shows *onorm (f ∘ g) ≤ onorm f * onorm g*
proof (*rule onorm_bound*)
show *0 ≤ onorm f * onorm g*
by (*intro mult_nonneg_nonneg onorm_pos_le f g*)
next
fix *x*
have *norm (f (g x)) ≤ onorm f * norm (g x)*
by (*rule onorm* [*OF f*])
also have *onorm f * norm (g x) ≤ onorm f * (onorm g * norm x)*
by (*rule mult_left_mono* [*OF onorm* [*OF g*] *onorm_pos_le* [*OF f*]])
finally show *norm ((f ∘ g) x) ≤ onorm f * onorm g * norm x*
by (*simp add*: *mult.assoc*)
qed

lemma *onorm_scaleR_lemma*:
assumes *f*: *bounded_linear f*
shows *onorm (λx. r *$_R$ *f x) ≤ |r| * onorm f*
proof (*rule onorm_bound*)
show *0 ≤ |r| * onorm f*
by (*intro mult_nonneg_nonneg onorm_pos_le abs_ge_zero f*)
next

```
  fix x
  have |r| * norm (f x) ≤ |r| * (onorm f * norm x)
    by (intro mult_left_mono onorm abs_ge_zero f)
  then show norm (r *R f x) ≤ |r| * onorm f * norm x
    by (simp only: norm_scaleR mult.assoc)
qed

lemma onorm_scaleR:
  assumes f: bounded_linear f
  shows onorm (λx. r *R f x) = |r| * onorm f
proof (cases r = 0)
  assume r ≠ 0
  show ?thesis
  proof (rule order_antisym)
    show onorm (λx. r *R f x) ≤ |r| * onorm f
      using f by (rule onorm_scaleR_lemma)
  next
    have bounded_linear (λx. r *R f x)
      using bounded_linear_scaleR_right f by (rule bounded_linear_compose)
    then have onorm (λx. inverse r *R r *R f x) ≤ |inverse r| * onorm (λx. r *R
f x)
      by (rule onorm_scaleR_lemma)
    with ‹r ≠ 0› show |r| * onorm f ≤ onorm (λx. r *R f x)
      by (simp add: inverse_eq_divide pos_le_divide_eq mult.commute)
  qed
qed (simp add: onorm_zero)

lemma onorm_scaleR_left_lemma:
  assumes r: bounded_linear r
  shows onorm (λx. r x *R f) ≤ onorm r * norm f
proof (rule onorm_bound)
  fix x
  have norm (r x *R f) = norm (r x) * norm f
    by simp
  also have ... ≤ onorm r * norm x * norm f
    by (intro mult_right_mono onorm r norm_ge_zero)
  finally show norm (r x *R f) ≤ onorm r * norm f * norm x
    by (simp add: ac_simps)
qed (intro mult_nonneg_nonneg norm_ge_zero onorm_pos_le r)

lemma onorm_scaleR_left:
  assumes f: bounded_linear r
  shows onorm (λx. r x *R f) = onorm r * norm f
proof (cases f = 0)
  assume f ≠ 0
  show ?thesis
  proof (rule order_antisym)
    show onorm (λx. r x *R f) ≤ onorm r * norm f
      using f by (rule onorm_scaleR_left_lemma)
```

```
  next
    have bl1: bounded_linear (λx. r x *R f)
      by (metis bounded_linear_scaleR_const f)
    have bounded_linear (λx. r x * norm f)
      by (metis bounded_linear_mult_const f)
    from onorm_scaleR_left_lemma[OF this, of inverse (norm f)]
    have onorm r ≤ onorm (λx. r x * norm f) * inverse (norm f)
      using ‹f ≠ 0›
      by (simp add: inverse_eq_divide)
    also have onorm (λx. r x * norm f) ≤ onorm (λx. r x *R f)
      by (rule onorm_bound)
        (auto simp: abs_mult bl1 onorm_pos_le intro!: order_trans[OF _ onorm])
    finally show onorm r * norm f ≤ onorm (λx. r x *R f)
      using ‹f ≠ 0›
      by (simp add: inverse_eq_divide pos_le_divide_eq mult.commute)
  qed
qed (simp add: onorm_zero)

lemma onorm_neg:
  shows onorm (λx. − f x) = onorm f
  unfolding onorm_def by simp

lemma onorm_triangle:
  assumes f: bounded_linear f
  assumes g: bounded_linear g
  shows onorm (λx. f x + g x) ≤ onorm f + onorm g
proof (rule onorm_bound)
  show 0 ≤ onorm f + onorm g
    by (intro add_nonneg_nonneg onorm_pos_le f g)
next
  fix x
  have norm (f x + g x) ≤ norm (f x) + norm (g x)
    by (rule norm_triangle_ineq)
  also have norm (f x) + norm (g x) ≤ onorm f * norm x + onorm g * norm x
    by (intro add_mono onorm f g)
  finally show norm (f x + g x) ≤ (onorm f + onorm g) * norm x
    by (simp only: distrib_right)
qed

lemma onorm_triangle_le:
  assumes bounded_linear f
  assumes bounded_linear g
  assumes onorm f + onorm g ≤ e
  shows onorm (λx. f x + g x) ≤ e
  using assms by (rule onorm_triangle [THEN order_trans])

lemma onorm_triangle_lt:
  assumes bounded_linear f
  assumes bounded_linear g
```

assumes *onorm f* + *onorm g* < *e*
shows *onorm* (λ*x*. *f x* + *g x*) < *e*
using *assms* **by** (*rule onorm_triangle* [*THEN order_le_less_trans*])

lemma *onorm_sum*:
assumes *finite S*
assumes ⋀*s*. *s* ∈ *S* ⟹ *bounded_linear* (*f s*)
shows *onorm* (λ*x*. *sum* (λ*s*. *f s x*) *S*) ≤ *sum* (λ*s*. *onorm* (*f s*)) *S*
using *assms*
by (*induction*) (*auto simp*: *onorm_zero intro*!: *onorm_triangle_le bounded_linear_sum*)

lemmas *onorm_sum_le* = *onorm_sum*[*THEN order_trans*]

end

5.13 Limits on the Extended Real Number Line

theory *Extended_Real_Limits*
imports
Topology_Euclidean_Space
HOL−Library.Extended_Real
HOL−Library.Extended_Nonnegative_Real
HOL−Library.Indicator_Function
begin

lemma *compact_UNIV*:
compact (*UNIV* :: *'a*::{*complete_linorder*,*linorder_topology*,*second_countable_topology*}
set)
using *compact_complete_linorder*
by (*auto simp*: *seq_compact_eq_compact*[*symmetric*] *seq_compact_def*)

lemma *compact_eq_closed*:
fixes *S* :: *'a*::{*complete_linorder*,*linorder_topology*,*second_countable_topology*}
set
shows *compact S* ⟷ *closed S*
using *closed_Int_compact*[*of S*, *OF* _ *compact_UNIV*] *compact_imp_closed*
by *auto*

lemma *closed_contains_Sup_cl*:
fixes *S* :: *'a*::{*complete_linorder*,*linorder_topology*,*second_countable_topology*}
set
assumes *closed S*
and *S* ≠ {}
shows *Sup S* ∈ *S*
proof −
from *compact_eq_closed*[*of S*] *compact_attains_sup*[*of S*] *assms*
obtain *s* **where** *S*: *s* ∈ *S* ∀*t*∈*S*. *t* ≤ *s*
by *auto*
then have *Sup S* = *s*

```
    by (auto intro!: Sup_eqI)
  with S show ?thesis
    by simp
qed

lemma closed_contains_Inf_cl:
  fixes S :: 'a::{complete_linorder,linorder_topology,second_countable_topology}
set
  assumes closed S
    and S ≠ {}
  shows Inf S ∈ S
proof −
  from compact_eq_closed[of S] compact_attains_inf[of S] assms
  obtain s where S: s ∈ S ∀ t∈S. s ≤ t
    by auto
  then have Inf S = s
    by (auto intro!: Inf_eqI)
  with S show ?thesis
    by simp
qed

instance enat :: second_countable_topology
proof
  show ∃ B::enat set set. countable B ∧ open = generate_topology B
  proof (intro exI conjI)
    show countable (range lessThan ∪ range greaterThan::enat set set)
      by auto
  qed (simp add: open_enat_def)
qed

instance ereal :: second_countable_topology
proof (standard, intro exI conjI)
  let ?B = (⋃ r∈ℚ. {{..< r}, {r <..}} :: ereal set set)
  show countable ?B
    by (auto intro: countable_rat)
  show open = generate_topology ?B
  proof (intro ext iffI)
    fix S :: ereal set
    assume open S
    then show generate_topology ?B S
      unfolding open_generated_order
    proof induct
      case (Basis b)
      then obtain e where b = {..<e} ∨ b = {e<..}
        by auto
     moreover have {..<e} = ⋃{{..<x}|x. x ∈ ℚ ∧ x < e} {e<..} = ⋃{{x<..}|x.
x ∈ ℚ ∧ e < x}
        by (auto dest: ereal_dense3
                 simp del: ex_simps
```

```
            simp add: ex_simps[symmetric] conj_commute Rats_def image_iff)
    ultimately show ?case
      by (auto intro: generate_topology.intros)
  qed (auto intro: generate_topology.intros)
 next
  fix S
  assume generate_topology ?B S
  then show open S
    by induct auto
 qed
qed
```

This is a copy from *ereal* :: *second_countable_topology*. Maybe find a common super class of topological spaces where the rational numbers are densely embedded ?

```
instance ennreal :: second_countable_topology
proof (standard, intro exI conjI)
 let ?B = (⋃r∈ℚ. {{..< r}, {r <..}} :: ennreal set set)
 show countable ?B
   by (auto intro: countable_rat)
 show open = generate_topology ?B
 proof (intro ext iffI)
   fix S :: ennreal set
   assume open S
   then show generate_topology ?B S
     unfolding open_generated_order
   proof induct
     case (Basis b)
     then obtain e where b = {..<e} ∨ b = {e<..}
       by auto
    moreover have {..<e} = ⋃{{..<x}|x. x ∈ ℚ ∧ x < e} {e<..} = ⋃{{x<..}|x.
x ∈ ℚ ∧ e < x}
       by (auto dest: ennreal_rat_dense
             simp del: ex_simps
             simp add: ex_simps[symmetric] conj_commute Rats_def image_iff)
     ultimately show ?case
       by (auto intro: generate_topology.intros)
   qed (auto intro: generate_topology.intros)
 next
   fix S
   assume generate_topology ?B S
   then show open S
     by induct auto
 qed
qed

lemma ereal_open_closed_aux:
 fixes S :: ereal set
 assumes open S
```

```
    and closed S
    and S: (−∞) ∉ S
  shows S = {}
proof (rule ccontr)
  assume ¬ ?thesis
  then have ∗: Inf S ∈ S

    by (metis assms(2) closed_contains_Inf_cl)
  {
    assume Inf S = −∞
    then have False
      using ∗ assms(3) by auto
  }
  moreover
  {
    assume Inf S = ∞
    then have S = {∞}
      by (metis Inf_eq_PInfty ‹S ≠ {}›)
    then have False
      by (metis assms(1) not_open_singleton)
  }
  moreover
  {
    assume fin: |Inf S| ≠ ∞
    from ereal_open_cont_interval[OF assms(1) ∗ fin]
    obtain e where e: e > 0 {Inf S − e<..<Inf S + e} ⊆ S .
    then obtain b where b: Inf S − e < b b < Inf S
      using fin ereal_between[of Inf S e] dense[of Inf S − e]
      by auto
    then have b ∈ {Inf S − e <..< Inf S + e}
      using e fin ereal_between[of Inf S e]
      by auto
    then have b ∈ S
      using e by auto
    then have False
      using b by (metis complete_lattice_class.Inf_lower leD)
  }
  ultimately show False
    by auto
qed

lemma ereal_open_closed:
  fixes S :: ereal set
  shows open S ∧ closed S ⟷ S = {} ∨ S = UNIV
  using ereal_open_closed_aux open_closed by auto

lemma ereal_open_atLeast:
  fixes x :: ereal
  shows open {x..} ⟷ x = −∞
```

by (*metis atLeast_eq_UNIV_iff bot_ereal_def closed_atLeast ereal_open_closed not_Ici_eq_empty*)

lemma *mono_closed_real*:
fixes S :: *real set*
assumes *mono*: $\forall\, y\ z.\ y \in S \wedge y \leq z \longrightarrow z \in S$
and *closed S*
shows $S = \{\} \vee S = UNIV \vee (\exists\, a.\ S = \{a..\})$
proof −
{
assume $S \neq \{\}$
{ assume *ex*: $\exists\, B.\ \forall\, x{\in}S.\ B \leq x$
then have $*$: $\forall\, x{\in}S.\ Inf\ S \leq x$
using *cInf_lower*[*of* _ *S*] *ex* **by** (*metis bdd_below_def*)
then have $Inf\ S \in S$
by (*meson* ‹$S \neq \{\}$› *assms*(*2*) *bdd_belowI closed_contains_Inf*)
then have $\forall\, x.\ Inf\ S \leq x \longleftrightarrow x \in S$
using *mono*[*rule_format*, *of Inf S*] $*$
by *auto*
then have $S = \{Inf\ S\ ..\}$
by *auto*
then have $\exists\, a.\ S = \{a\ ..\}$
by *auto*
}
moreover
{
assume $\neg\ (\exists\, B.\ \forall\, x{\in}S.\ B \leq x)$
then have *nex*: $\forall\, B.\ \exists\, x{\in}S.\ x < B$
by (*simp add*: *not_le*)
{
fix y
obtain x **where** $x{\in}S$ **and** $x < y$
using *nex* **by** *auto*
then have $y \in S$
using *mono*[*rule_format*, *of x y*] **by** *auto*
}
then have $S = UNIV$
by *auto*
}
ultimately have $S = UNIV \vee (\exists\, a.\ S = \{a\ ..\})$
by *blast*
}
then show *?thesis*
by *blast*
qed

lemma *mono_closed_ereal*:
fixes S :: *real set*
assumes *mono*: $\forall\, y\ z.\ y \in S \wedge y \leq z \longrightarrow z \in S$

```
    and closed S
  shows ∃ a. S = {x. a ≤ ereal x}
proof −
  consider S = {} ∨ S = UNIV | (∃ a. S = {a..})
    using assms(2) mono mono_closed_real by blast
  then show ?thesis
  proof cases
    case 1
    then show ?thesis
     by (meson PInfty_neq_ereal(1) UNIV_eq_I bot.extremum empty_Collect_eq
ereal_infty_less_eq(1) mem_Collect_eq)
  next
    case 2
    then show ?thesis
    by (metis atLeast_iff ereal_less_eq(3) mem_Collect_eq subsetI subset_antisym)
  qed
qed

lemma Liminf_within:
  fixes f :: 'a::metric_space ⇒ 'b::complete_lattice
  shows Liminf (at x within S) f = (SUP e∈{0<..}. INF y∈(S ∩ ball x e − {x}).
f y)
  unfolding Liminf_def eventually_at
proof (rule SUP_eq, simp_all add: Ball_def Bex_def, safe)
  fix P d
  assume 0 < d and ∀ y. y ∈ S ⟶ y ≠ x ∧ dist y x < d ⟶ P y
  then have S ∩ ball x d − {x} ⊆ {x. P x}
    by (auto simp: dist_commute)
  then show ∃ r>0. Inf (f ‘ (Collect P)) ≤ Inf (f ‘ (S ∩ ball x r − {x}))
    by (intro exI[of _ d] INF_mono conjI ‹0 < d›) auto
next
  fix d :: real
  assume 0 < d
  then show ∃ P. (∃ d>0. ∀ xa. xa ∈ S ⟶ xa ≠ x ∧ dist xa x < d ⟶ P xa) ∧
    Inf (f ‘ (S ∩ ball x d − {x})) ≤ Inf (f ‘ (Collect P))
    by (intro exI[of _ λy. y ∈ S ∩ ball x d − {x}])
       (auto intro!: INF_mono exI[of _ d] simp: dist_commute)
qed

lemma Limsup_within:
  fixes f :: 'a::metric_space ⇒ 'b::complete_lattice
  shows Limsup (at x within S) f = (INF e∈{0<..}. SUP y∈(S ∩ ball x e − {x}).
f y)
  unfolding Limsup_def eventually_at
proof (rule INF_eq, simp_all add: Ball_def Bex_def, safe)
  fix P d
  assume 0 < d and ∀ y. y ∈ S ⟶ y ≠ x ∧ dist y x < d ⟶ P y
  then have S ∩ ball x d − {x} ⊆ {x. P x}
    by (auto simp: dist_commute)
```

then show $\exists r{>}0.\ Sup\ (f\ `\ (S \cap ball\ x\ r - \{x\})) \leq Sup\ (f\ `\ (Collect\ P))$
by (*intro exI*[*of* _ *d*] *SUP_mono conjI* ‹0 < *d*›) *auto*
next
fix *d* :: *real*
assume 0 < *d*
then show $\exists P.\ (\exists d{>}0.\ \forall xa.\ xa \in S \longrightarrow xa \neq x \wedge dist\ xa\ x < d \longrightarrow P\ xa) \wedge$
Sup (*f* ` (*Collect P*)) ≤ *Sup* (*f* ` (*S* ∩ *ball x d* − {*x*}))
by (*intro exI*[*of* _ λ*y*. *y* ∈ *S* ∩ *ball x d* − {*x*}])
(*auto intro*!: *SUP_mono exI*[*of* _ *d*] *simp*: *dist_commute*)
qed

lemma *Liminf_at*:
fixes *f* :: 'a::*metric_space* ⇒ 'b::*complete_lattice*
shows *Liminf* (*at x*) *f* = (*SUP e*∈{0<..}. *INF y*∈(*ball x e* − {*x*}). *f y*)
using *Liminf_within*[*of x UNIV f*] **by** *simp*

lemma *Limsup_at*:
fixes *f* :: 'a::*metric_space* ⇒ 'b::*complete_lattice*
shows *Limsup* (*at x*) *f* = (*INF e*∈{0<..}. *SUP y*∈(*ball x e* − {*x*}). *f y*)
using *Limsup_within*[*of x UNIV f*] **by** *simp*

lemma *min_Liminf_at*:
fixes *f* :: 'a::*metric_space* ⇒ 'b::*complete_linorder*
shows *min* (*f x*) (*Liminf* (*at x*) *f*) = (*SUP e*∈{0<..}. *INF y*∈*ball x e*. *f y*)
apply (*simp add*: *inf_min* [*symmetric*] *Liminf_at inf_commute* [*of f x*] *SUP_inf*)
apply (*metis* (*no_types*, *lifting*) *INF_insert centre_in_ball greaterThan_iff image_cong inf_commute insert_Diff*)
done

5.13.1 Extended-Real.thy

lemma *sum_constant_ereal*:
fixes *a*::*ereal*
shows $(\sum i{\in}I.\ a) = a * card\ I$
proof (*induction I rule*: *infinite_finite_induct*)
case (*insert i I*)
then show *?case*
by (*simp add*: *ereal_right_distrib flip*: *plus_ereal.simps*)
qed *auto*

lemma *real_lim_then_eventually_real*:
assumes (*u* ⟶ *ereal l*) *F*
shows *eventually* (λ*n*. *u n* = *ereal*(*real_of_ereal*(*u n*))) *F*
proof −
have *ereal l* ∈ {−∞<..<(∞::*ereal*)} **by** *simp*
moreover have *open* {−∞<..<(∞::*ereal*)} **by** *simp*
ultimately have *eventually* (λ*n*. *u n* ∈ {−∞<..<(∞::*ereal*)}) *F* **using** *assms tendsto_def* **by** *blast*
moreover have $\bigwedge x.\ x \in \{-\infty{<}..{<}(\infty{::}ereal)\} \Longrightarrow x = ereal(real\_of\_ereal\ x)$

using *ereal_real* **by** *auto*
ultimately show *?thesis* **by** (*metis* (*mono_tags*, *lifting*) *eventually_mono*)
qed

lemma *ereal_Inf_cmult*:
assumes *c*>(*0*::*real*)
shows *Inf* {*ereal c* ∗ *x* |*x*. *P x*} = *ereal c* ∗ *Inf* {*x*. *P x*}
proof −
have *bij* ((∗) (*ereal c*))
apply (*rule bij_betw_byWitness*[*of* _ λ*x*. (*x*::*ereal*) / *c*], *auto simp*: *assms ereal_mult_divide*)
using *assms ereal_divide_eq* **by** *auto*
then have *ereal c* ∗ *Inf* {*x*. *P x*} = *Inf* ((∗) (*ereal c*) ‘ {*x*. *P x*})
by (*simp add*: *assms ereal_mult_left_mono less_imp_le mono_def mono_bij_Inf*)
then show *?thesis*
by (*simp add*: *setcompr_eq_image*)
qed

Continuity of addition

The next few lemmas remove an unnecessary assumption in *tendsto_add_ereal*, culminating in *tendsto_add_ereal_general* which essentially says that the addition is continuous on ereal times ereal, except at $(-\infty, \infty)$ and $(\infty, -\infty)$. It is much more convenient in many situations, see for instance the proof of *tendsto_sum_ereal* below.

lemma *tendsto_add_ereal_PInf*:
fixes *y* :: *ereal*
assumes *y*: *y* ≠ −∞
assumes *f*: (*f* ⟶ ∞) *F* **and** *g*: (*g* ⟶ *y*) *F*
shows ((λ*x*. *f x* + *g x*) ⟶ ∞) *F*
proof −
have ∃ *C*. *eventually* (λ*x*. *g x* > *ereal C*) *F*
proof (*cases y*)
case (*real r*)
have *y* > *y*−*1* **using** *y real* **by** (*simp add*: *ereal_between*(*1*))
then have *eventually* (λ*x*. *g x* > *y* − *1*) *F* **using** *g y order_tendsto_iff* **by** *auto*
moreover have *y*−*1* = *ereal*(*real_of_ereal*(*y*−*1*))
by (*metis real ereal_eq_1*(*1*) *ereal_minus*(*1*) *real_of_ereal.simps*(*1*))
ultimately have *eventually* (λ*x*. *g x* > *ereal*(*real_of_ereal*(*y* − *1*))) *F* **by** *simp*
then show *?thesis* **by** *auto*
next
case (*PInf*)
have *eventually* (λ*x*. *g x* > *ereal 0*) *F* **using** *g PInf* **by** (*simp add*: *tendsto_PInfty*)
then show *?thesis* **by** *auto*
qed (*simp add*: *y*)

```
  then obtain C::real where ge: eventually (λx. g x > ereal C) F by auto

  {
    fix M::real
    have eventually (λx. f x > ereal(M − C)) F using f by (simp add: tendsto_PInfty)
    then have eventually (λx. (f x > ereal (M−C)) ∧ (g x > ereal C)) F
      by (auto simp: ge eventually_conj_iff)
    moreover have ⋀x. ((f x > ereal (M−C)) ∧ (g x > ereal C)) ⟹ (f x + g x > ereal M)
      using ereal_add_strict_mono2 by fastforce
    ultimately have eventually (λx. f x + g x > ereal M) F using eventually_mono by force
  }
  then show ?thesis by (simp add: tendsto_PInfty)
qed
```

One would like to deduce the next lemma from the previous one, but the fact that $-(x+y)$ is in general different from $(-x)+(-y)$ in ereal creates difficulties, so it is more efficient to copy the previous proof.

```
lemma tendsto_add_ereal_MInf:
  fixes y :: ereal
  assumes y: y ≠ ∞
  assumes f: (f ⟶ −∞) F and g: (g ⟶ y) F
  shows ((λx. f x + g x) ⟶ −∞) F
proof −
  have ∃ C. eventually (λx. g x < ereal C) F
  proof (cases y)
    case (real r)
    have y < y+1 using y real by (simp add: ereal_between(1))
    then have eventually (λx. g x < y + 1) F using g y order_tendsto_iff by force
    moreover have y+1 = ereal(real_of_ereal (y+1)) by (simp add: real)
    ultimately have eventually (λx. g x < ereal(real_of_ereal(y + 1))) F by simp
    then show ?thesis by auto
  next
    case (MInf)
    have eventually (λx. g x < ereal 0) F using g MInf by (simp add: tendsto_MInfty)
    then show ?thesis by auto
  qed (simp add: y)
  then obtain C::real where ge: eventually (λx. g x < ereal C) F by auto

  {
    fix M::real
    have eventually (λx. f x < ereal(M − C)) F using f by (simp add: tendsto_MInfty)
    then have eventually (λx. (f x < ereal (M− C)) ∧ (g x < ereal C)) F
```

```
    by (auto simp: ge eventually_conj_iff)
   moreover have ⋀x. ((f x < ereal (M-C)) ∧ (g x < ereal C)) ⟹ (f x + g x
< ereal M)
    using ereal_add_strict_mono2 by fastforce
   ultimately have eventually (λx. f x + g x < ereal M) F using eventu-
ally_mono by force
 }
 then show ?thesis by (simp add: tendsto_MInfty)
qed

lemma tendsto_add_ereal_general1:
 fixes x y :: ereal
 assumes y: |y| ≠ ∞
 assumes f: (f ⟶ x) F and g: (g ⟶ y) F
 shows ((λx. f x + g x) ⟶ x + y) F
proof (cases x)
 case (real r)
 have a: |x| ≠ ∞ by (simp add: real)
 show ?thesis by (rule tendsto_add_ereal[OF a, OF y, OF f, OF g])
next
 case PInf
 then show ?thesis using tendsto_add_ereal_PInf assms by force
next
 case MInf
 then show ?thesis using tendsto_add_ereal_MInf assms
  by (metis abs_ereal.simps(3) ereal_MInfty_eq_plus)
qed

lemma tendsto_add_ereal_general2:
 fixes x y :: ereal
 assumes x: |x| ≠ ∞
    and f: (f ⟶ x) F and g: (g ⟶ y) F
 shows ((λx. f x + g x) ⟶ x + y) F
proof −
 have ((λx. g x + f x) ⟶ x + y) F
  by (metis (full_types) add.commute f g tendsto_add_ereal_general1 x)
 moreover have ⋀x. g x + f x = f x + g x using add.commute by auto
 ultimately show ?thesis by simp
qed
```

The next lemma says that the addition is continuous on *ereal*, except at the pairs $(-\infty, \infty)$ and $(\infty, -\infty)$.

```
lemma tendsto_add_ereal_general [tendsto_intros]:
 fixes x y :: ereal
 assumes ¬((x=∞ ∧ y=−∞) ∨ (x=−∞ ∧ y=∞))
    and f: (f ⟶ x) F and g: (g ⟶ y) F
 shows ((λx. f x + g x) ⟶ x + y) F
proof (cases x)
 case (real r)
```

```
  show ?thesis
    using real assms
    by (intro tendsto_add_ereal_general2; auto)
next
  case (PInf)
  then have y ≠ -∞ using assms by simp
  then show ?thesis using tendsto_add_ereal_PInf PInf assms by auto
next
  case (MInf)
  then have y ≠ ∞ using assms by simp
  then show ?thesis
    by (metis ereal_MInfty_eq_plus tendsto_add_ereal_MInf MInf f g)
qed
```

Continuity of multiplication

In the same way as for addition, we prove that the multiplication is continuous on ereal times ereal, except at $(\infty, 0)$ and $(-\infty, 0)$ and $(0, \infty)$ and $(0, -\infty)$, starting with specific situations.

```
lemma tendsto_mult_real_ereal:
  assumes (u ⟶ ereal l) F (v ⟶ ereal m) F
  shows ((λn. u n * v n) ⟶ ereal l * ereal m) F
proof -
  have ureal: eventually (λn. u n = ereal(real_of_ereal(u n))) F by (rule real_lim_then_eventually_real[OF
assms(1)])
   then have ((λn. ereal(real_of_ereal(u n))) ⟶ ereal l) F using assms by
auto
  then have limu: ((λn. real_of_ereal(u n)) ⟶ l) F by auto
  have vreal: eventually (λn. v n = ereal(real_of_ereal(v n))) F by (rule real_lim_then_eventually_real[OF
assms(2)])
   then have ((λn. ereal(real_of_ereal(v n))) ⟶ ereal m) F using assms by
auto
  then have limv: ((λn. real_of_ereal(v n)) ⟶ m) F by auto

  {
    fix n assume u n = ereal(real_of_ereal(u n)) v n = ereal(real_of_ereal(v n))
    then have ereal(real_of_ereal(u n) * real_of_ereal(v n)) = u n * v n
      by (metis times_ereal.simps(1))
  }
  then have *: eventually (λn. ereal(real_of_ereal(u n) * real_of_ereal(v n)) =
u n * v n) F
    using eventually_elim2[OF ureal vreal] by auto

  have ((λn. real_of_ereal(u n) * real_of_ereal(v n)) ⟶ l * m) F
    using tendsto_mult[OF limu limv] by auto
  then have ((λn. ereal(real_of_ereal(u n)) * real_of_ereal(v n)) ⟶ ereal(l *
m)) F
    by auto
  then show ?thesis using * filterlim_cong by fastforce
```

qed

lemma *tendsto_mult_ereal_PInf*:
fixes *f g*::_ ⇒ *ereal*
assumes (*f* ⟶ *l*) *F l*>*0* (*g* ⟶ ∞) *F*
shows ((λ*x*. *f x* ∗ *g x*) ⟶ ∞) *F*
proof −
obtain *a*::*real* **where** *0* < *ereal a a* < *l*
using *assms*(*2*) **using** *ereal_dense2* **by** *blast*
have ∗: *eventually* (λ*x*. *f x* > *a*) *F*
using ‹*a* < *l*› *assms*(*1*) **by** (*simp add*: *order_tendsto_iff*)
{
fix *K*::*real*
define *M* **where** *M* = *max K 1*
then have *M* > *0* **by** *simp*
then have *ereal*(*M*/*a*) > *0* **using** ‹*ereal a* > *0*› **by** *simp*
then have ⋀*x*. ((*f x* > *a*) ∧ (*g x* > *M*/*a*)) ⟹ (*f x* ∗ *g x* > *ereal a* ∗ *ereal*(*M*/*a*))
using *ereal_mult_mono_strict'*[**where** *?c* = *M*/*a*, *OF* ‹*0* < *ereal a*›] **by** *auto*
moreover have *ereal a* ∗ *ereal*(*M*/*a*) = *M* **using** ‹*ereal a* > *0*› **by** *simp*
ultimately have ⋀*x*. ((*f x* > *a*) ∧ (*g x* > *M*/*a*)) ⟹ (*f x* ∗ *g x* > *M*) **by** *simp*
moreover have *M* ≥ *K* **unfolding** *M_def* **by** *simp*
ultimately have *Imp*: ⋀*x*. ((*f x* > *a*) ∧ (*g x* > *M*/*a*)) ⟹ (*f x* ∗ *g x* > *K*)
using *ereal_less_eq*(*3*) *le_less_trans* **by** *blast*

have *eventually* (λ*x*. *g x* > *M*/*a*) *F* **using** *assms*(*3*) **by** (*simp add*: *tendsto_PInfty*)
then have *eventually* (λ*x*. (*f x* > *a*) ∧ (*g x* > *M*/*a*)) *F*
using ∗ **by** (*auto simp*: *eventually_conj_iff*)
then have *eventually* (λ*x*. *f x* ∗ *g x* > *K*) *F* **using** *eventually_mono Imp* **by** *force*
}
then show *?thesis* **by** (*auto simp*: *tendsto_PInfty*)
qed

lemma *tendsto_mult_ereal_pos*:
fixes *f g*::_ ⇒ *ereal*
assumes (*f* ⟶ *l*) *F* (*g* ⟶ *m*) *F l*>*0 m*>*0*
shows ((λ*x*. *f x* ∗ *g x*) ⟶ *l* ∗ *m*) *F*
proof (*cases*)
assume ∗: *l* = ∞ ∨ *m* = ∞
then show *?thesis*
proof (*cases*)
assume *m* = ∞
then show *?thesis* **using** *tendsto_mult_ereal_PInf assms* **by** *auto*
next
assume ¬(*m* = ∞)
then have *l* = ∞ **using** ∗ **by** *simp*
then have ((λ*x*. *g x* ∗ *f x*) ⟶ *l* ∗ *m*) *F* **using** *tendsto_mult_ereal_PInf*

assms **by** *auto*
moreover have $\bigwedge x.\ g\ x * f\ x = f\ x * g\ x$ **using** *mult.commute* **by** *auto*
ultimately show *?thesis* **by** *simp*
qed
next
assume $\neg(l = \infty \vee m = \infty)$
then have $l < \infty\ m < \infty$ **by** *auto*
then obtain *lr mr* **where** $l = ereal\ lr\ m = ereal\ mr$
using ‹$l>0$› ‹$m>0$› **by** (*metis ereal_cases ereal_less*(6) *not_less_iff_gr_or_eq*)
then show *?thesis* **using** *tendsto_mult_real_ereal assms* **by** *auto*
qed

We reduce the general situation to the positive case by multiplying by suitable signs. Unfortunately, as ereal is not a ring, all the neat sign lemmas are not available there. We give the bare minimum we need.

lemma *ereal_sgn_abs*:
fixes *l*::*ereal*
shows $sgn(l) * l = abs(l)$
by (*cases l*, *auto simp*: *sgn_if ereal_less_uminus_reorder*)

lemma *sgn_squared_ereal*:
assumes $l \neq (0::ereal)$
shows $sgn(l) * sgn(l) = 1$
using *assms* **by** (*cases l*, *auto simp*: *one_ereal_def sgn_if*)

lemma *tendsto_mult_ereal* [*tendsto_intros*]:
fixes *f g*::*_* ⇒ *ereal*
assumes $(f \longrightarrow l)\ F\ (g \longrightarrow m)\ F\ \neg((l=0 \wedge abs(m) = \infty) \vee (m=0 \wedge abs(l) = \infty))$
shows $((\lambda x.\ f\ x * g\ x) \longrightarrow l * m)\ F$
proof (*cases*)
assume $l=0 \vee m=0$
then have $abs(l) \neq \infty\ abs(m) \neq \infty$ **using** *assms*(3) **by** *auto*
then obtain *lr mr* **where** $l = ereal\ lr\ m = ereal\ mr$ **by** *auto*
then show *?thesis* **using** *tendsto_mult_real_ereal assms* **by** *auto*
next
have *sgn_finite*: $\bigwedge a::ereal.\ abs(sgn\ a) \neq \infty$
by (*metis MInfty_neq_ereal*(2) *PInfty_neq_ereal*(2) *abs_eq_infinity_cases ereal_times*(1) *ereal_times*(3) *ereal_uminus_eq_reorder sgn_ereal.elims*)
then have *sgn_finite2*: $\bigwedge a\ b::ereal.\ abs(sgn\ a * sgn\ b) \neq \infty$
by (*metis abs_eq_infinity_cases abs_ereal.simps*(2) *abs_ereal.simps*(3) *ereal_mult_eq_MInfty ereal_mult_eq_PInfty*)
assume $\neg(l=0 \vee m=0)$
then have $l \neq 0\ m \neq 0$ **by** *auto*
then have $abs(l) > 0\ abs(m) > 0$
by (*metis abs_ereal_ge0 abs_ereal_less0 abs_ereal_pos ereal_uminus_uminus ereal_uminus_zero less_le not_less*)+
then have $sgn(l) * l > 0\ sgn(m) * m > 0$ **using** *ereal_sgn_abs* **by** *auto*
moreover have $((\lambda x.\ sgn(l) * f\ x) \longrightarrow (sgn(l) * l))\ F$

by (*rule tendsto_cmult_ereal*, *auto simp*: *sgn_finite assms(1)*)
moreover have $((\lambda x.\ sgn(m) * g\ x) \longrightarrow (sgn(m) * m))\ F$
by (*rule tendsto_cmult_ereal*, *auto simp*: *sgn_finite assms(2)*)
ultimately have $*$: $((\lambda x.\ (sgn(l) * f\ x) * (sgn(m) * g\ x)) \longrightarrow (sgn(l) * l) * (sgn(m) * m))\ F$
using *tendsto_mult_ereal_pos* **by** *force*
have $((\lambda x.\ (sgn(l) * sgn(m)) * ((sgn(l) * f\ x) * (sgn(m) * g\ x))) \longrightarrow (sgn(l) * sgn(m)) * ((sgn(l) * l) * (sgn(m) * m)))\ F$
by (*rule tendsto_cmult_ereal*, *auto simp*: *sgn_finite2* $*$)
moreover have $\bigwedge x.\ (sgn(l) * sgn(m)) * ((sgn(l) * f\ x) * (sgn(m) * g\ x)) = f\ x * g\ x$
by (*metis mult.left_neutral sgn_squared_ereal*[*OF* ‹$l \neq 0$›] *sgn_squared_ereal*[*OF* ‹$m \neq 0$›] *mult.assoc mult.commute*)
moreover have $(sgn(l) * sgn(m)) * ((sgn(l) * l) * (sgn(m) * m)) = l * m$
by (*metis mult.left_neutral sgn_squared_ereal*[*OF* ‹$l \neq 0$›] *sgn_squared_ereal*[*OF* ‹$m \neq 0$›] *mult.assoc mult.commute*)
ultimately show *?thesis* **by** *auto*
qed

lemma *tendsto_cmult_ereal_general* [*tendsto_intros*]:
fixes f::_ $\Rightarrow$ *ereal* **and** c::*ereal*
assumes $(f \longrightarrow l)\ F$ $\neg\ (l{=}0 \wedge abs(c) = \infty)$
shows $((\lambda x.\ c * f\ x) \longrightarrow c * l)\ F$
by (*cases* $c = 0$, *auto simp*: *assms tendsto_mult_ereal*)

Continuity of division

lemma *tendsto_inverse_ereal_PInf*:
fixes u::_ $\Rightarrow$ *ereal*
assumes $(u \longrightarrow \infty)\ F$
shows $((\lambda x.\ 1/\ u\ x) \longrightarrow 0)\ F$
proof −
{
fix e::*real* **assume** $e{>}0$
have $1/e < \infty$ **by** *auto*
then have *eventually* $(\lambda n.\ u\ n > 1/e)\ F$ **using** *assms(1)* **by** (*simp add*: *tendsto_PInfty*)
moreover
{
fix z::*ereal* **assume** $z{>}1/e$
then have $z{>}0$ **using** ‹$e{>}0$› **using** *less_le_trans not_le* **by** *fastforce*
then have $1/z \geq 0$ **by** *auto*
moreover have $1/z < e$
proof (*cases z*)
case (*real r*)
then show *?thesis*
using ‹$0 < e$› ‹$0 < z$› ‹*ereal* $(1\ /\ e) < z$› *divide_less_eq ereal_divide_less_pos* **by** *fastforce*
qed (*use* ‹$0 < e$› ‹$0 < z$› **in** *auto*)

```
    ultimately have 1/z ≥ 0 1/z < e by auto
   }
   ultimately have eventually (λn. 1/u n<e) F eventually (λn. 1/u n≥0) F by
(auto simp: eventually_mono)
  } note * = this
  show ?thesis
  proof (subst order_tendsto_iff, auto)
   fix a::ereal assume a<0
    then show eventually (λn. 1/u n > a) F using *(2) eventually_mono
less_le_trans linordered_field_no_ub by fastforce
  next
   fix a::ereal assume a>0
  then obtain e::real where e>0 a>e using ereal_dense2 ereal_less(2) by blast
   then have eventually (λn. 1/u n < e) F using *(1) by auto
   then show eventually (λn. 1/u n < a) F using ‹a>e› by (metis (mono_tags,
lifting) eventually_mono less_trans)
  qed
qed
```

The next lemma deserves to exist by itself, as it is so common and useful.

```
lemma tendsto_inverse_real [tendsto_intros]:
  fixes u::_ ⇒ real
  shows (u ⟶ l) F ⟹ l ≠ 0 ⟹ ((λx. 1/ u x) ⟶ 1/l) F
  using tendsto_inverse unfolding inverse_eq_divide .

lemma tendsto_inverse_ereal [tendsto_intros]:
  fixes u::_ ⇒ ereal
  assumes (u ⟶ l) F l ≠ 0
  shows ((λx. 1/ u x) ⟶ 1/l) F
proof (cases l)
  case (real r)
  then have r ≠ 0 using assms(2) by auto
  then have 1/l = ereal(1/r) using real by (simp add: one_ereal_def)
  define v where v = (λn. real_of_ereal(u n))
 have ureal: eventually (λn. u n = ereal(v n)) F unfolding v_def using real_lim_then_eventually_real
assms(1) real by auto
  then have ((λn. ereal(v n)) ⟶ ereal r) F using assms real v_def by auto
  then have *: ((λn. v n) ⟶ r) F by auto
  then have ((λn. 1/v n) ⟶ 1/r) F using ‹r ≠ 0› tendsto_inverse_real by
auto
  then have lim: ((λn. ereal(1/v n)) ⟶ 1/l) F using ‹1/l = ereal(1/r)› by
auto

  have r ∈ -{0} open (-{(0::real)}) using ‹r ≠ 0› by auto
  then have eventually (λn. v n ∈ -{0}) F using * using topological_tendstoD
by blast
  then have eventually (λn. v n ≠ 0) F by auto
  moreover
  {
```

```
    fix n assume H: v n ≠ 0 u n = ereal(v n)
    then have ereal(1/v n) = 1/ereal(v n) by (simp add: one_ereal_def)
    then have ereal(1/v n) = 1/u n using H(2) by simp
  }
  ultimately have eventually (λn. ereal(1/v n) = 1/u n) F using ureal eventu-
ally_elim2 by force
  with Lim_transform_eventually[OF lim this] show ?thesis by simp
next
  case (PInf)
  then have 1/l = 0 by auto
  then show ?thesis using tendsto_inverse_ereal_PInf assms PInf by auto
next
  case (MInf)
  then have 1/l = 0 by auto
  have 1/z = −1/ −z if z < 0 for z::ereal
    apply (cases z) using divide_ereal_def ‹ z < 0 › by auto
 moreover have eventually (λn. u n < 0) F by (metis (no_types) MInf assms(1)
tendsto_MInfty zero_ereal_def)
  ultimately have ∗: eventually (λn. −1/−u n = 1/u n) F by (simp add: even-
tually_mono)

  define v where v = (λn. − u n)
 have (v ⟶ ∞) F unfolding v_def using MInf assms(1) tendsto_uminus_ereal
by fastforce
  then have ((λn. 1/v n) ⟶ 0) F using tendsto_inverse_ereal_PInf by auto
  then have ((λn. −1/v n) ⟶ 0) F using tendsto_uminus_ereal by fastforce
  then show ?thesis unfolding v_def using Lim_transform_eventually[OF _ ∗]
‹ 1/l = 0 › by auto
qed

lemma tendsto_divide_ereal [tendsto_intros]:
  fixes f g::_ ⇒ ereal
  assumes (f ⟶ l) F (g ⟶ m) F m ≠ 0 ¬(abs(l) = ∞ ∧ abs(m) = ∞)
  shows ((λx. f x / g x) ⟶ l / m) F
proof −
  define h where h = (λx. 1/ g x)
   have ∗: (h ⟶ 1/m) F unfolding h_def using assms(2) assms(3) tend-
sto_inverse_ereal by auto
  have ((λx. f x ∗ h x) ⟶ l ∗ (1/m)) F
    apply (rule tendsto_mult_ereal[OF assms(1) ∗]) using assms(3) assms(4) by
(auto simp: divide_ereal_def)
  moreover have f x ∗ h x = f x / g x for x unfolding h_def by (simp add:
divide_ereal_def)
  moreover have l ∗ (1/m) = l/m by (simp add: divide_ereal_def)
   ultimately show ?thesis unfolding h_def using Lim_transform_eventually
by auto
qed
```

Further limits

The assumptions of $[\![|?x| \neq \infty; |?y| \neq \infty; (?f \longrightarrow ?x)\ ?F; (?g \longrightarrow ?y)\ ?F]\!] \Longrightarrow ((\lambda x.\ ?f\ x - ?g\ x) \longrightarrow ?x - ?y)\ ?F$ are too strong, we weaken them here.

lemma *tendsto_diff_ereal_general* [*tendsto_intros*]:
fixes *u v*::*'a* ⇒ *ereal*
assumes $(u \longrightarrow l)\ F\ (v \longrightarrow m)\ F\ \neg((l = \infty \wedge m = \infty) \vee (l = -\infty \wedge m = -\infty))$
shows $((\lambda n.\ u\ n - v\ n) \longrightarrow l - m)\ F$
proof −
have $\infty = l \longrightarrow ((\lambda a.\ u\ a + - v\ a) \longrightarrow l + - m)\ F$
by (*metis* (*no_types*) *assms ereal_uminus_uminus tendsto_add_ereal_general tendsto_uminus_ereal*)
then have $((\lambda n.\ u\ n + (-v\ n)) \longrightarrow l + (-m))\ F$
by (*simp add*: *assms ereal_uminus_eq_reorder tendsto_add_ereal_general*)
then show *?thesis*
by (*simp add*: *minus_ereal_def*)
qed

lemma *id_nat_ereal_tendsto_PInf* [*tendsto_intros*]:
$(\lambda\ n{::}nat.\ real\ n) \longrightarrow \infty$
by (*simp add*: *filterlim_real_sequentially tendsto_PInfty_eq_at_top*)

lemma *tendsto_at_top_pseudo_inverse* [*tendsto_intros*]:
fixes *u*::*nat* ⇒ *nat*
assumes *LIM n sequentially. u n :> at_top*
shows *LIM n sequentially. Inf* $\{N.\ u\ N \geq n\}$ *:> at_top*
proof −
{
fix *C*::*nat*
define *M* **where** $M = Max\ \{u\ n|\ n.\ n \leq C\}+1$
{
fix *n* **assume** $n \geq M$
have *eventually* $(\lambda N.\ u\ N \geq n)$ *sequentially* **using** *assms*
by (*simp add*: *filterlim_at_top*)
then have ∗: $\{N.\ u\ N \geq n\} \neq \{\}$ **by** *force*

have $N > C$ **if** $u\ N \geq n$ **for** *N*
proof (*rule ccontr*)
assume $\neg(N > C)$
then have $u\ N \leq Max\ \{u\ n|\ n.\ n \leq C\}$
using *Max_ge* **by** (*simp add*: *setcompr_eq_image not_less*)
then show *False* **using** ‹$u\ N \geq n$› ‹$n \geq M$› **unfolding** *M_def* **by** *auto*
qed
then have ∗∗: $\{N.\ u\ N \geq n\} \subseteq \{C..\}$ **by** *fastforce*
have *Inf* $\{N.\ u\ N \geq n\} \geq C$
by (*metis* ∗ ∗∗ *Inf_nat_def1 atLeast_iff subset_eq*)
}

```
    then have eventually (λn. Inf {N. u N ≥ n} ≥ C) sequentially
      using eventually_sequentially by auto
  }
  then show ?thesis using filterlim_at_top by auto
qed

lemma pseudo_inverse_finite_set:
  fixes u::nat ⇒ nat
  assumes LIM n sequentially. u n :> at_top
  shows finite {N. u N ≤ n}
proof −
  fix n
  have eventually (λN. u N ≥ n+1) sequentially using assms
    by (simp add: filterlim_at_top)
  then obtain N1 where N1: ⋀N. N ≥ N1 ⟹ u N ≥ n + 1
    using eventually_sequentially by auto
  have {N. u N ≤ n} ⊆ {..<N1}
    by (metis (no_types, lifting) N1 Suc_eq_plus1 lessThan_iff less_le_not_le
mem_Collect_eq nle_le not_less_eq_eq subset_eq)
  then show finite {N. u N ≤ n} by (simp add: finite_subset)
qed

lemma tendsto_at_top_pseudo_inverse2 [tendsto_intros]:
  fixes u::nat ⇒ nat
  assumes LIM n sequentially. u n :> at_top
  shows LIM n sequentially. Max {N. u N ≤ n} :> at_top
proof −
  {
    fix N0::nat
    have N0 ≤ Max {N. u N ≤ n} if n ≥ u N0 for n
      by (simp add: assms pseudo_inverse_finite_set that)
    then have eventually (λn. N0 ≤ Max {N. u N ≤ n}) sequentially
      using eventually_sequentially by blast
  }
  then show ?thesis using filterlim_at_top by auto
qed

lemma ereal_truncation_top [tendsto_intros]:
  fixes x::ereal
  shows (λn::nat. min x n) ⟶ x
proof (cases x)
  case (real r)
  then obtain K::nat where K>0 K > abs(r) using reals_Archimedean2 gr0I
by auto
  then have min x n = x if n ≥ K for n apply (subst real, subst real, auto)
using that eq_iff by fastforce
 then have eventually (λn. min x n = x) sequentially using eventually_at_top_linorder
by blast
  then show ?thesis by (simp add: tendsto_eventually)
```

next
case (*PInf*)
then have *min x n* = *n* **for** *n*::*nat* **by** (*auto simp*: *min_def*)
then show *?thesis* **using** *id_nat_ereal_tendsto_PInf PInf* **by** *auto*
next
case (*MInf*)
then have *min x n* = *x* **for** *n*::*nat* **by** (*auto simp*: *min_def*)
then show *?thesis* **by** *auto*
qed

lemma *ereal_truncation_real_top* [*tendsto_intros*]:
fixes *x*::*ereal*
assumes *x* ≠ − ∞
shows (λ*n*::*nat*. *real_of_ereal*(*min x n*)) ⟶ *x*
proof (*cases x*)
case (*real r*)
then obtain *K*::*nat* **where** *K*>*0 K* > *abs*(*r*) **using** *reals_Archimedean2 gr0I* **by** *auto*
then have *min x n* = *x* **if** *n* ≥ *K* **for** *n* **apply** (*subst real*, *subst real*, *auto*) **using** *that eq_iff* **by** *fastforce*
then have *real_of_ereal*(*min x n*) = *r* **if** *n* ≥ *K* **for** *n* **using** *real that* **by** *auto*
then have *eventually* (λ*n*. *real_of_ereal*(*min x n*) = *r*) *sequentially* **using** *eventually_at_top_linorder* **by** *blast*
then have (λ*n*. *real_of_ereal*(*min x n*)) ⟶ *r* **by** (*simp add*: *tendsto_eventually*)
then show *?thesis* **using** *real* **by** *auto*
next
case (*PInf*)
then have *real_of_ereal*(*min x n*) = *n* **for** *n*::*nat* **by** (*auto simp*: *min_def*)
then show *?thesis* **using** *id_nat_ereal_tendsto_PInf PInf* **by** *auto*
qed (*simp add*: *assms*)

lemma *ereal_truncation_bottom* [*tendsto_intros*]:
fixes *x*::*ereal*
shows (λ*n*::*nat*. *max x* (− *real n*)) ⟶ *x*
proof (*cases x*)
case (*real r*)
then obtain *K*::*nat* **where** *K*>*0 K* > *abs*(*r*) **using** *reals_Archimedean2 gr0I* **by** *auto*
then have *max x* (−*real n*) = *x* **if** *n* ≥ *K* **for** *n* **apply** (*subst real*, *subst real*, *auto*) **using** *that eq_iff* **by** *fastforce*
then have *eventually* (λ*n*. *max x* (−*real n*) = *x*) *sequentially* **using** *eventually_at_top_linorder* **by** *blast*
then show *?thesis* **by** (*simp add*: *tendsto_eventually*)
next
case (*MInf*)
then have *max x* (−*real n*) = (−*1*)∗ *ereal*(*real n*) **for** *n*::*nat* **by** (*auto simp*: *max_def*)
moreover have (λ*n*. (−*1*)∗ *ereal*(*real n*)) ⟶ −∞
using *tendsto_cmult_ereal*[*of* −*1*, *OF* _ *id_nat_ereal_tendsto_PInf*] **by**

(*simp add*: *one_ereal_def*)
ultimately show *?thesis* **using** *MInf* **by** *auto*
next
case (*PInf*)
then have *max x* (−*real n*) = *x* **for** *n*::*nat* **by** (*auto simp*: *max_def*)
then show *?thesis* **by** *auto*
qed

lemma *ereal_truncation_real_bottom* [*tendsto_intros*]:
fixes *x*::*ereal*
assumes *x* ≠ ∞
shows (λ*n*::*nat*. *real_of_ereal*(*max x* (− *real n*))) ⟶ *x*
proof (*cases x*)
case (*real r*)
then obtain *K*::*nat* **where** *K*>*0 K* > *abs*(*r*) **using** *reals_Archimedean2 gr0I*
by *auto*
then have *max x* (−*real n*) = *x* **if** *n* ≥ *K* **for** *n* **apply** (*subst real*, *subst real*, *auto*) **using** *that eq_iff* **by** *fastforce*
then have *real_of_ereal*(*max x* (−*real n*)) = *r* **if** *n* ≥ *K* **for** *n* **using** *real that*
by *auto*
then have *eventually* (λ*n*. *real_of_ereal*(*max x* (−*real n*)) = *r*) *sequentially*
using *eventually_at_top_linorder* **by** *blast*
then have (λ*n*. *real_of_ereal*(*max x* (−*real n*))) ⟶ *r* **by** (*simp add*: *tendsto_eventually*)
then show *?thesis* **using** *real* **by** *auto*
next
case (*MInf*)
then have *real_of_ereal*(*max x* (−*real n*)) = (−*1*)∗ *ereal*(*real n*) **for** *n*::*nat* **by**
(*auto simp*: *max_def*)
moreover have (λ*n*. (−*1*)∗ *ereal*(*real n*)) ⟶ −∞
using *tendsto_cmult_ereal*[*of* −*1*, *OF* _ *id_nat_ereal_tendsto_PInf*] **by**
(*simp add*: *one_ereal_def*)
ultimately show *?thesis* **using** *MInf* **by** *auto*
qed (*simp add*: *assms*)

the next one is copied from *tendsto_sum*.

lemma *tendsto_sum_ereal* [*tendsto_intros*]:
fixes *f* :: ′*a* ⇒ ′*b* ⇒ *ereal*
assumes ⋀*i*. *i* ∈ *S* ⟹ (*f i* ⟶ *a i*) *F*
⋀*i*. *abs*(*a i*) ≠ ∞
shows ((λ*x*. ∑ *i*∈*S*. *f i x*) ⟶ (∑ *i*∈*S*. *a i*)) *F*
proof (*cases finite S*)
assume *finite S* **then show** *?thesis* **using** *assms*
by (*induct*, *simp*, *simp add*: *tendsto_add_ereal_general2 assms*)
qed(*simp*)

lemma *continuous_ereal_abs*:
continuous_on (*UNIV*::*ereal set*) *abs*

proof −
have *continuous_on* ({..*0*} ∪ {(*0*::*ereal*)..}) *abs*
proof (*intro continuous_on_closed_Un continuous_intros*)
show *continuous_on* {..*0*::*ereal*} *abs*
by (*metis abs_ereal_ge0 abs_ereal_less0 continuous_on_eq antisym_conv1 atMost_iff continuous_uminus_ereal ereal_uminus_zero*)
show *continuous_on* {*0*::*ereal*..} *abs*
by (*metis abs_ereal_ge0 atLeast_iff continuous_on_eq continuous_on_id*)
qed
moreover have (*UNIV*::*ereal set*) = {..*0*} ∪ {(*0*::*ereal*)..} **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemmas *continuous_on_compose_ereal_abs*[*continuous_intros*] =
continuous_on_compose2[*OF continuous_ereal_abs _ subset_UNIV*]

lemma *tendsto_abs_ereal* [*tendsto_intros*]:
assumes (*u* ⟶ (*l*::*ereal*)) *F*
shows ((λ*n*. *abs*(*u n*)) ⟶ *abs l*) *F*
using *continuous_ereal_abs assms* **by** (*metis UNIV_I continuous_on tendsto_compose*)

lemma *ereal_minus_real_tendsto_MInf* [*tendsto_intros*]:
(λ*x*. *ereal* (− *real x*)) ⟶ − ∞
by (*subst uminus_ereal.simps*(*1*)[*symmetric*], *intro tendsto_intros*)

5.13.2 Extended-Nonnegative-Real.thy

lemma *tendsto_diff_ennreal_general* [*tendsto_intros*]:
fixes *u v*::*'a* ⇒ *ennreal*
assumes (*u* ⟶ *l*) *F* (*v* ⟶ *m*) *F* ¬(*l* = ∞ ∧ *m* = ∞)
shows ((λ*n*. *u n* − *v n*) ⟶ *l* − *m*) *F*
proof −
have ((λ*n*. *e2ennreal*(*enn2ereal*(*u n*) − *enn2ereal*(*v n*))) ⟶ *e2ennreal*(*enn2ereal l* − *enn2ereal m*)) *F*
by (*intro tendsto_intros*) (*use assms* **in** *auto*)
then show *?thesis* **by** *auto*
qed

lemma *tendsto_mult_ennreal* [*tendsto_intros*]:
fixes *l m*::*ennreal*
assumes (*u* ⟶ *l*) *F* (*v* ⟶ *m*) *F* ¬((*l* = *0* ∧ *m* = ∞) ∨ (*l* = ∞ ∧ *m* = *0*))
shows ((λ*n*. *u n* ∗ *v n*) ⟶ *l* ∗ *m*) *F*
proof −
have ((λ*n*. *e2ennreal*(*enn2ereal* (*u n*) ∗ *enn2ereal* (*v n*))) ⟶ *e2ennreal*(*enn2ereal l* ∗ *enn2ereal m*)) *F*
by (*intro tendsto_intros*) (*use assms enn2ereal_inject zero_ennreal.rep_eq* **in** *fastforce*)+
moreover have *e2ennreal*(*enn2ereal* (*u n*) ∗ *enn2ereal* (*v n*)) = *u n* ∗ *v n* **for** *n*

by (*subst times_ennreal.abs_eq*[*symmetric*], *auto simp*: *eq_onp_same_args*)
moreover have *e2ennreal*(*enn2ereal l* ∗ *enn2ereal m*) = *l* ∗ *m*
by (*subst times_ennreal.abs_eq*[*symmetric*], *auto simp*: *eq_onp_same_args*)
ultimately show *?thesis*
by *auto*
qed

5.13.3 monoset

definition (**in** *order*) *mono_set*:
mono_set S ⟷ (∀ *x y*. *x* ≤ *y* ⟶ *x* ∈ *S* ⟶ *y* ∈ *S*)

lemma (**in** *order*) *mono_greaterThan* [*intro*, *simp*]: *mono_set* {*B*<..} **unfolding** *mono_set* **by** *auto*
lemma (**in** *order*) *mono_atLeast* [*intro*, *simp*]: *mono_set* {*B*..} **unfolding** *mono_set* **by** *auto*
lemma (**in** *order*) *mono_UNIV* [*intro*, *simp*]: *mono_set UNIV* **unfolding** *mono_set* **by** *auto*
lemma (**in** *order*) *mono_empty* [*intro*, *simp*]: *mono_set* {} **unfolding** *mono_set* **by** *auto*

lemma (**in** *complete_linorder*) *mono_set_iff*:
fixes *S* :: *'a set*
defines *a* ≡ *Inf S*
shows *mono_set S* ⟷ *S* = {*a* <..} ∨ *S* = {*a*..} (**is** _ = *?c*)
proof
assume *mono_set S*
then have *mono*: ⋀*x y*. *x* ≤ *y* ⟹ *x* ∈ *S* ⟹ *y* ∈ *S*
by (*auto simp*: *mono_set*)
show *?c*
proof *cases*
assume *a* ∈ *S*
show *?c*
using *mono*[*OF* _ ‹*a* ∈ *S*›]
by (*auto intro*: *Inf_lower simp*: *a_def*)
next
assume *a* ∉ *S*
have *S* = {*a* <..}
proof *safe*
fix *x* **assume** *x* ∈ *S*
then have *a* ≤ *x*
unfolding *a_def* **by** (*rule Inf_lower*)
then show *a* < *x*
using ‹*x* ∈ *S*› ‹*a* ∉ *S*› **by** (*cases a* = *x*) *auto*
next
fix *x* **assume** *a* < *x*
then obtain *y* **where** *y* < *x y* ∈ *S*
unfolding *a_def Inf_less_iff* **..**
with *mono*[*of y x*] **show** *x* ∈ *S*

```
      by auto
    qed
    then show ?c ..
  qed
qed auto

lemma ereal_open_mono_set:
  fixes S :: ereal set
  shows open S ∧ mono_set S ⟷ S = UNIV ∨ S = {Inf S <..}
  by (metis Inf_UNIV atLeast_eq_UNIV_iff ereal_open_atLeast
    ereal_open_closed mono_set_iff open_ereal_greaterThan)

lemma ereal_closed_mono_set:
  fixes S :: ereal set
  shows closed S ∧ mono_set S ⟷ S = {} ∨ S = {Inf S ..}
  by (metis Inf_UNIV atLeast_eq_UNIV_iff closed_ereal_atLeast
    ereal_open_closed mono_empty mono_set_iff open_ereal_greaterThan)

lemma ereal_Liminf_Sup_monoset:
  fixes f :: 'a ⇒ ereal
  shows Liminf net f =
    Sup {l. ∀ S. open S ⟶ mono_set S ⟶ l ∈ S ⟶ eventually (λx. f x ∈ S)
net}
    (is _ = Sup ?A)
proof (safe intro!: Liminf_eqI complete_lattice_class.Sup_upper complete_lattice_class.Sup_least)
  fix P
  assume P: eventually P net
  fix S
  assume S: mono_set S Inf (f ‘ (Collect P)) ∈ S
  {
    fix x
    assume P x
    then have Inf (f ‘ (Collect P)) ≤ f x
      by (intro complete_lattice_class.INF_lower) simp
    with S have f x ∈ S
      by (simp add: mono_set)
  }
  with P show eventually (λx. f x ∈ S) net
    by (auto elim: eventually_mono)
next
  fix y l
  assume S: ∀ S. open S ⟶ mono_set S ⟶ l ∈ S ⟶ eventually  (λx. f x ∈ S)
net
  assume P: ∀ P. eventually P net ⟶ Inf (f ‘ (Collect P)) ≤ y
  show l ≤ y
  proof (rule dense_le)
    fix B
    assume B < l
    then have eventually (λx. f x ∈ {B <..}) net
```

```
    by (intro S[rule_format]) auto
   then have Inf (f ‘ {x. B < f x}) ≤ y
    using P by auto
   moreover have B ≤ Inf (f ‘ {x. B < f x})
    by (intro INF_greatest) auto
   ultimately show B ≤ y
    by simp
  qed
qed

lemma ereal_Limsup_Inf_monoset:
  fixes f :: 'a ⇒ ereal
  shows Limsup net f =
    Inf {l. ∀ S. open S ⟶ mono_set (uminus ‘ S) ⟶ l ∈ S ⟶ eventually (λx.
f x ∈ S) net}
    (is _ = Inf ?A)
proof (safe intro!: Limsup_eqI complete_lattice_class.Inf_lower complete_lattice_class.Inf_greatest)
  fix P
  assume P: eventually P net
  fix S
  assume S: mono_set (uminus‘S) Sup (f ‘ (Collect P)) ∈ S
  {
    fix x
    assume P x
    then have f x ≤ Sup (f ‘ (Collect P))
     by (intro complete_lattice_class.SUP_upper) simp
    with S(1)[unfolded mono_set, rule_format, of − Sup (f ‘ (Collect P)) − f x]
S(2)
    have f x ∈ S
     by (simp add: inj_image_mem_iff) }
  with P show eventually (λx. f x ∈ S) net
    by (auto elim: eventually_mono)
next
  fix y l
  assume S: ∀ S. open S ⟶ mono_set (uminus ‘ S) ⟶ l ∈ S ⟶ eventually
(λx. f x ∈ S) net
  assume P: ∀ P. eventually P net ⟶ y ≤ Sup (f ‘ (Collect P))
  show y ≤ l
  proof (rule dense_ge)
    fix B
    assume l < B
    then have eventually (λx. f x ∈ {..< B}) net
     by (intro S[rule_format]) auto
    then have y ≤ Sup (f ‘ {x. f x < B})
     using P by auto
    moreover have Sup (f ‘ {x. f x < B}) ≤ B
     by (intro SUP_least) auto
    ultimately show y ≤ B
     by simp
```

```
  qed
qed

lemma liminf_bounded_open:
  fixes x :: nat ⇒ ereal
  shows x0 ≤ liminf x ⟷ (∀ S. open S ⟶ mono_set S ⟶ x0 ∈ S ⟶ (∃ N.
∀ n≥N. x n ∈ S))
  (is _ ⟷ ?P x0)
proof
  assume ?P x0
  then show x0 ≤ liminf x
    unfolding ereal_Liminf_Sup_monoset eventually_sequentially
    by (intro complete_lattice_class.Sup_upper) auto
next
  assume x0 ≤ liminf x
  {
    fix S :: ereal set
    assume om: open S mono_set S x0 ∈ S
    then have ∃ N. ∀ n≥N. x n ∈ S
          by (metis ‹x0 ≤ liminf x› ereal_open_mono_set greaterThan_iff lim-
inf_bounded_iff om UNIV_I)
  }
  then show ?P x0
    by auto
qed

lemma limsup_finite_then_bounded:
  fixes u::nat ⇒ real
  assumes limsup u < ∞
  shows ∃ C. ∀ n. u n ≤ C
proof −
  obtain C where C: limsup u < C C < ∞ using assms ereal_dense2 by blast
  then have C = ereal(real_of_ereal C) using ereal_real by force
  have eventually (λn. u n < C) sequentially
    using SUP_lessD eventually_mono C(1)
    by (fastforce simp: INF_less_iff Limsup_def)
 then obtain N where N: ⋀n. n ≥ N ⟹ u n < C using eventually_sequentially
by auto
  define D where D = max (real_of_ereal C) (Max {u n |n. n ≤ N})
  have ⋀n. u n ≤ D
  proof −
    fix n show u n ≤ D
    proof (cases)
      assume ∗: n ≤ N
      have u n ≤ Max {u n |n. n ≤ N} by (rule Max_ge, auto simp: ∗)
      then show u n ≤ D unfolding D_def by linarith
    next
      assume ¬(n ≤ N)
      then have n ≥ N by simp
```

```
    then have u n < C using N by auto
      then have u n < real_of_ereal C using ‹C = ereal(real_of_ereal C)›
less_ereal.simps(1) by fastforce
    then show u n ≤ D unfolding D_def by linarith
   qed
  qed
  then show ?thesis by blast
qed

lemma liminf_finite_then_bounded_below:
  fixes u::nat ⇒ real
  assumes liminf u > −∞
  shows ∃ C. ∀ n. u n ≥ C
proof −
  obtain C where C: liminf u > C C > −∞ using assms using ereal_dense2
by blast
  then have C = ereal(real_of_ereal C) using ereal_real by force
  have eventually (λn. u n > C) sequentially
    using eventually_elim2 less_INF_D C(1)
    by (fastforce simp: less_SUP_iff Liminf_def)
 then obtain N where N: ⋀n. n ≥ N ⟹ u n > C using eventually_sequentially
by auto
  define D where D = min (real_of_ereal C) (Min {u n |n. n ≤ N})
  have ⋀n. u n ≥ D
  proof −
    fix n show u n ≥ D
    proof (cases)
      assume ∗: n ≤ N
      have u n ≥ Min {u n |n. n ≤ N} by (rule Min_le, auto simp: ∗)
      then show u n ≥ D unfolding D_def by linarith
    next
      assume ¬(n ≤ N)
      then have n ≥ N by simp
      then have u n > C using N by auto
      then have u n > real_of_ereal C
        using ‹C = ereal(real_of_ereal C)› less_ereal.simps(1) by fastforce
      then show u n ≥ D unfolding D_def by linarith
    qed
  qed
  then show ?thesis by blast
qed

lemma liminf_upper_bound:
  fixes u:: nat ⇒ ereal
  assumes liminf u < l
  shows ∃ N>k. u N < l
by (metis assms gt_ex less_le_trans liminf_bounded_iff not_less)

lemma limsup_shift:
```

limsup (λ*n*. *u* (*n*+*1*)) = *limsup u*
proof −
have (*SUP m*∈{*n*+*1*..}. *u m*) = (*SUP m*∈{*n*..}. *u* (*m* + *1*)) **for** *n*
by (*rule SUP_eq*) (*use Suc_le_D* **in** *auto*)
then have *a*: (*INF n. SUP m*∈{*n*..}. *u* (*m* + *1*)) = (*INF n.* (*SUP m*∈{*n*+*1*..}. *u m*)) **by** *auto*
have *b*: (*INF n.* (*SUP m*∈{*n*+*1*..}. *u m*)) = (*INF n*∈{*1*..}. (*SUP m*∈{*n*..}. *u m*))
by (*rule INF_eq*) (*use Suc_le_D* **in** *auto*)
have (*INF n*∈{*1*..}. *v n*) = (*INF n. v n*) **if** *decseq v* **for** *v*::*nat* ⇒ *'a*
by (*rule INF_eq*) (*use* ‹*decseq v*› *decseq_Suc_iff* **in** *auto*)
moreover have *decseq* (λ*n*. (*SUP m*∈{*n*..}. *u m*)) **by** (*simp add*: *SUP_subset_mono decseq_def*)
ultimately have *c*: (*INF n*∈{*1*..}. (*SUP m*∈{*n*..}. *u m*)) = (*INF n.* (*SUP m*∈{*n*..}. *u m*)) **by** *simp*
have (*INF n. Sup* (*u* ‘ {*n*..})) = (*INF n. SUP m*∈{*n*..}. *u* (*m* + *1*)) **using** *a b c* **by** *simp*
then show *?thesis* **by** (*auto cong*: *limsup_INF_SUP*)
qed

lemma *limsup_shift_k*:
limsup (λ*n*. *u* (*n*+*k*)) = *limsup u*
proof (*induction k*)
case (*Suc k*)
have *limsup* (λ*n*. *u* (*n*+*k*+*1*)) = *limsup* (λ*n*. *u* (*n*+*k*)) **using** *limsup_shift*[**where** *?u*=λ*n*. *u*(*n*+*k*)] **by** *simp*
then show *?case* **using** *Suc.IH* **by** *simp*
qed (*auto*)

lemma *liminf_shift*:
liminf (λ*n*. *u* (*n*+*1*)) = *liminf u*
proof −
have (*INF m*∈{*n*+*1*..}. *u m*) = (*INF m*∈{*n*..}. *u* (*m* + *1*)) **for** *n*
by (*rule INF_eq*) (*use Suc_le_D* **in** *auto*)
then have *a*: (*SUP n. INF m*∈{*n*..}. *u* (*m* + *1*)) = (*SUP n.* (*INF m*∈{*n*+*1*..}. *u m*)) **by** *auto*
have *b*: (*SUP n.* (*INF m*∈{*n*+*1*..}. *u m*)) = (*SUP n*∈{*1*..}. (*INF m*∈{*n*..}. *u m*))
by (*rule SUP_eq*) (*use Suc_le_D* **in** *auto*)
have (*SUP n*∈{*1*..}. *v n*) = (*SUP n. v n*) **if** *incseq v* **for** *v*::*nat* ⇒ *'a*
by (*rule SUP_eq*) (*use* ‹*incseq v*› *incseq_Suc_iff* **in** *auto*)
moreover have *incseq* (λ*n*. (*INF m*∈{*n*..}. *u m*)) **by** (*simp add*: *INF_superset_mono mono_def*)
ultimately have *c*: (*SUP n*∈{*1*..}. (*INF m*∈{*n*..}. *u m*)) = (*SUP n.* (*INF m*∈{*n*..}. *u m*)) **by** *simp*
have (*SUP n. Inf* (*u* ‘ {*n*..})) = (*SUP n. INF m*∈{*n*..}. *u* (*m* + *1*)) **using** *a b c* **by** *simp*
then show *?thesis* **by** (*auto cong*: *liminf_SUP_INF*)
qed

lemma *liminf_shift_k*:
  liminf (λ*n*. *u* (*n*+*k*)) = *liminf u*
proof (*induction k*)
  case (*Suc k*)
  have *liminf* (λ*n*. *u* (*n*+*k*+*1*)) = *liminf* (λ*n*. *u* (*n*+*k*))
    using *liminf_shift*[**where** *?u*=λ*n*. *u*(*n*+*k*)] **by** *simp*
  then show *?case* **using** *Suc.IH* **by** *simp*
qed (*auto*)

lemma *Limsup_obtain*:
  fixes *u*::_ ⇒ *'a* :: *complete_linorder*
  assumes *Limsup F u* > *c*
  shows ∃ *i*. *u i* > *c*
proof −
  have (*INF P*∈{*P*. *eventually P F*}. *SUP x*∈{*x*. *P x*}. *u x*) > *c* **using** *assms* **by** (*simp add*: *Limsup_def*)
  then show *?thesis* **by** (*metis eventually_True mem_Collect_eq less_INF_D less_SUP_iff*)
qed

The next lemma is extremely useful, as it often makes it possible to reduce statements about limsups to statements about limits.

lemma *limsup_subseq_lim*:
  fixes *u*::*nat* ⇒ *'a* :: {*complete_linorder*, *linorder_topology*}
  shows ∃ *r*::*nat*⇒*nat*. *strict_mono r* ∧ (*u o r*) ⟶ *limsup u*
proof (*cases*)
  assume ∀ *n*. ∃ *p*>*n*. ∀ *m*≥*p*. *u m* ≤ *u p*
  then have ∃ *r*. ∀ *n*. (∀ *m*≥*r n*. *u m* ≤ *u* (*r n*)) ∧ *r n* < *r* (*Suc n*)
    by (*intro dependent_nat_choice*) (*auto simp*: *conj_commute*)
  then obtain *r* :: *nat* ⇒ *nat* **where** *strict_mono r* **and** *mono*: ⋀*n m*. *r n* ≤ *m* ⟹ *u m* ≤ *u* (*r n*)
    by (*auto simp*: *strict_mono_Suc_iff*)
  define *umax* **where** *umax* = (λ*n*. (*SUP m*∈{*n*..}. *u m*))
  have *decseq umax* **unfolding** *umax_def* **by** (*simp add*: *SUP_subset_mono antimono_def*)
  then have *umax* ⟶ *limsup u* **unfolding** *umax_def* **by** (*metis LIMSEQ_INF limsup_INF_SUP*)
  then have ∗: (*umax o r*) ⟶ *limsup u* **by** (*simp add*: *LIMSEQ_subseq_LIMSEQ* ‹*strict_mono r*›)
  have ⋀*n*. *umax*(*r n*) = *u*(*r n*) **unfolding** *umax_def* **using** *mono*
    by (*metis SUP_le_iff antisym atLeast_def mem_Collect_eq order_refl*)
  then have *umax o r* = *u o r* **unfolding** *o_def* **by** *simp*
  then have (*u o r*) ⟶ *limsup u* **using** ∗ **by** *simp*
  then show *?thesis* **using** ‹*strict_mono r*› **by** *blast*
next
  assume ¬ (∀ *n*. ∃ *p*>*n*. (∀ *m*≥*p*. *u m* ≤ *u p*))
  then obtain *N* **where** *N*: ⋀*p*. *p* > *N* ⟹ ∃ *m*>*p*. *u p* < *u m* **by** (*force simp*: *not_le le_less*)

```
  have ∃r. ∀n. N < r n ∧ r n < r (Suc n) ∧ (∀i∈ {N<..r (Suc n)}. u i ≤ u (r
(Suc n)))
  proof (rule dependent_nat_choice)
    fix x assume N < x
    then have a: finite {N<..x} {N<..x} ≠ {} by simp_all
    have Max {u i |i. i ∈ {N<..x}} ∈ {u i |i. i ∈ {N<..x}} apply (rule Max_in)
using a by (auto)
     then obtain p where p ∈ {N<..x} and upmax: u p = Max{u i |i. i ∈
{N<..x}} by auto
    define U where U = {m. m > p ∧ u p < u m}
    have U ≠ {} unfolding U_def using N[of p] ‹p ∈ {N<..x}› by auto
    define y where y = Inf U
    then have y ∈ U using ‹U ≠ {}› by (simp add: Inf_nat_def1)
    have a: ⋀i. i ∈ {N<..x} ⟹ u i ≤ u p
    proof −
      fix i assume i ∈ {N<..x}
      then have u i ∈ {u i |i. i ∈ {N<..x}} by blast
      then show u i ≤ u p using upmax by simp
    qed
    moreover have u p < u y using ‹y ∈ U› U_def by auto
    ultimately have y ∉ {N<..x} using not_le by blast
    moreover have y > N using ‹y ∈ U› U_def ‹p ∈ {N<..x}› by auto
    ultimately have y > x by auto

    have ⋀i. i ∈ {N<..y} ⟹ u i ≤ u y
    proof −
      fix i assume i ∈ {N<..y} show u i ≤ u y
      proof (cases)
        assume i = y
        then show ?thesis by simp
      next
        assume ¬(i=y)
        then have i:i ∈ {N<..<y} using ‹i ∈ {N<..y}› by simp
        have u i ≤ u p
        proof (cases)
          assume i ≤ x
          then have i ∈ {N<..x} using i by simp
          then show ?thesis using a by simp
        next
          assume ¬(i ≤ x)
          then have i > x by simp
          then have ∗: i > p using ‹p ∈ {N<..x}› by simp
          have i < Inf U using i y_def by simp
          then have i ∉ U using Inf_nat_def not_less_Least by auto
          then show ?thesis using U_def ∗ by auto
        qed
        then show u i ≤ u y using ‹u p < u y› by auto
      qed
    qed
```

then have $N < y \wedge x < y \wedge (\forall i{\in}\{N{<}..y\}.\ u\ i \leq u\ y)$ **using** ‹$y > x$› ‹$y > N$› **by** *auto*
then show $\exists y{>}N.\ x < y \wedge (\forall i{\in}\{N{<}..y\}.\ u\ i \leq u\ y)$ **by** *auto*
qed (*auto*)
then obtain r **where** r: $\forall n.\ N < r\ n \wedge r\ n < r\ (Suc\ n) \wedge (\forall i{\in}\ \{N{<}..r\ (Suc\ n)\}.\ u\ i \leq u\ (r\ (Suc\ n)))$ **by** *auto*
have *strict_mono r* **using** *r* **by** (*auto simp*: *strict_mono_Suc_iff*)
have *incseq* $(u\ o\ r)$ **unfolding** *o_def* **using** *r* **by** (*simp add*: *incseq_SucI order.strict_implies_order*)
then have $(u\ o\ r) \longrightarrow (SUP\ n.\ (u\ o\ r)\ n)$ **using** *LIMSEQ_SUP* **by** *blast*
then have *limsup* $(u\ o\ r) = (SUP\ n.\ (u\ o\ r)\ n)$ **by** (*simp add*: *lim_imp_Limsup*)
moreover have *limsup* $(u\ o\ r) \leq$ *limsup u* **using** ‹*strict_mono r*› **by** (*simp add*: *limsup_subseq_mono*)
ultimately have $(SUP\ n.\ (u\ o\ r)\ n) \leq$ *limsup u* **by** *simp*

{
fix *i* **assume** *i*: $i \in \{N{<}..\}$
obtain *n* **where** $i < r\ (Suc\ n)$ **using** ‹*strict_mono r*› **using** *Suc_le_eq seq_suble* **by** *blast*
then have $i \in \{N{<}..r(Suc\ n)\}$ **using** *i* **by** *simp*
then have $u\ i \leq u\ (r(Suc\ n))$ **using** *r* **by** *simp*
then have $u\ i \leq (SUP\ n.\ (u\ o\ r)\ n)$ **unfolding** *o_def* **by** (*meson SUP_upper2 UNIV_I*)
}
then have $(SUP\ i{\in}\{N{<}..\}.\ u\ i) \leq (SUP\ n.\ (u\ o\ r)\ n)$ **using** *SUP_least* **by** *blast*
then have *limsup* $u \leq (SUP\ n.\ (u\ o\ r)\ n)$ **unfolding** *Limsup_def*
by (*metis* (*mono_tags*, *lifting*) *INF_lower2 atLeast_Suc_greaterThan atLeast_def eventually_ge_at_top mem_Collect_eq*)
then have *limsup* $u = (SUP\ n.\ (u\ o\ r)\ n)$ **using** ‹$(SUP\ n.\ (u\ o\ r)\ n) \leq$ *limsup u*› **by** *simp*
then have $(u\ o\ r) \longrightarrow$ *limsup u* **using** ‹$(u\ o\ r) \longrightarrow (SUP\ n.\ (u\ o\ r)\ n)$› **by** *simp*
then show *?thesis* **using** ‹*strict_mono r*› **by** *auto*
qed

lemma *liminf_subseq_lim*:
fixes *u*::*nat* ⇒ *'a* :: {*complete_linorder*, *linorder_topology*}
shows $\exists r{::}nat{\Rightarrow}nat.$ *strict_mono r* $\wedge$ $(u\ o\ r) \longrightarrow$ *liminf u*
proof (*cases*)
assume $\forall n.\ \exists p{>}n.\ \forall m{\geq}p.\ u\ m \geq u\ p$
then have $\exists r.\ \forall n.\ (\forall m{\geq}r\ n.\ u\ m \geq u\ (r\ n)) \wedge r\ n < r\ (Suc\ n)$
by (*intro dependent_nat_choice*) (*auto simp*: *conj_commute*)
then obtain *r* :: *nat* ⇒ *nat* **where** *strict_mono r* **and** *mono*: $\bigwedge n\ m.\ r\ n \leq m \Longrightarrow u\ m \geq u\ (r\ n)$
by (*auto simp*: *strict_mono_Suc_iff*)
define *umin* **where** *umin* = $(\lambda n.\ (INF\ m{\in}\{n..\}.\ u\ m))$
have *incseq umin* **unfolding** *umin_def* **by** (*simp add*: *INF_superset_mono incseq_def*)

```
 then have umin ⟶ liminf u unfolding umin_def by (metis LIMSEQ_SUP
liminf_SUP_INF)
 then have ∗: (umin o r) ⟶ liminf u by (simp add: LIMSEQ_subseq_LIMSEQ
‹strict_mono r›)
 have ⋀n. umin(r n) = u(r n) unfolding umin_def using mono
  by (metis le_INF_iff antisym atLeast_def mem_Collect_eq order_refl)
 then have umin o r = u o r unfolding o_def by simp
 then have (u o r) ⟶ liminf u using ∗ by simp
 then show ?thesis using ‹strict_mono r› by blast
next
 assume ¬ (∀ n. ∃ p>n. (∀ m≥p. u m ≥ u p))
 then obtain N where N: ⋀p. p > N ⟹ ∃ m>p. u p > u m by (force simp:
not_le le_less)
 have ∃ r. ∀ n. N < r n ∧ r n < r (Suc n) ∧ (∀ i∈ {N<..r (Suc n)}. u i ≥ u (r
(Suc n)))
 proof (rule dependent_nat_choice)
  fix x assume N < x
  then have a: finite {N<..x} {N<..x} ≠ {} by simp_all
  have Min {u i |i. i ∈ {N<..x}} ∈ {u i |i. i ∈ {N<..x}} apply (rule Min_in)
using a by (auto)
  then obtain p where p ∈ {N<..x} and upmin: u p = Min{u i |i. i ∈ {N<..x}}
by auto
  define U where U = {m. m > p ∧ u p > u m}
  have U ≠ {} unfolding U_def using N[of p] ‹p ∈ {N<..x}› by auto
  define y where y = Inf U
  then have y ∈ U using ‹U ≠ {}› by (simp add: Inf_nat_def1)
  have a: ⋀i. i ∈ {N<..x} ⟹ u i ≥ u p
  proof −
   fix i assume i ∈ {N<..x}
   then have u i ∈ {u i |i. i ∈ {N<..x}} by blast
   then show u i ≥ u p using upmin by simp
  qed
  moreover have u p > u y using ‹y ∈ U› U_def by auto
  ultimately have y ∉ {N<..x} using not_le by blast
  moreover have y > N using ‹y ∈ U› U_def ‹p ∈ {N<..x}› by auto
  ultimately have y > x by auto

  have ⋀i. i ∈ {N<..y} ⟹ u i ≥ u y
  proof −
   fix i assume i ∈ {N<..y} show u i ≥ u y
   proof (cases)
    assume i = y
    then show ?thesis by simp
   next
    assume ¬(i=y)
    then have i:i ∈ {N<..<y} using ‹i ∈ {N<..y}› by simp
    have u i ≥ u p
    proof (cases)
     assume i ≤ x
```

```
      then show ?thesis using a ‹i ∈ {N<..y}› by force
     next
      assume ¬(i ≤ x)
      then have i > x by simp
      then have ∗: i > p using ‹p ∈ {N<..x}› by simp
      have i < Inf U using i y_def by simp
      then have i ∉ U using Inf_nat_def not_less_Least by auto
      then show ?thesis using U_def ∗ by auto
     qed
     then show u i ≥ u y using ‹u p > u y› by auto
    qed
   qed
   then have N < y ∧ x < y ∧ (∀ i∈{N<..y}. u i ≥ u y) using ‹y > x› ‹y >
N› by auto
   then show ∃ y>N. x < y ∧ (∀ i∈{N<..y}. u i ≥ u y) by auto
 qed (auto)
 then obtain r :: nat ⇒ nat
   where r: ∀ n. N < r n ∧ r n < r (Suc n) ∧ (∀ i∈ {N<..r (Suc n)}. u i ≥ u
(r (Suc n))) by auto
 have strict_mono r using r by (auto simp: strict_mono_Suc_iff)
 have decseq (u o r) unfolding o_def using r by (simp add: decseq_SucI or-
der.strict_implies_order)
 then have (u o r) ⟶ (INF n. (u o r) n) using LIMSEQ_INF by blast
 then have liminf (u o r) = (INF n. (u o r) n) by (simp add: lim_imp_Liminf)
 moreover have liminf (u o r) ≥ liminf u using ‹strict_mono r› by (simp add:
liminf_subseq_mono)
 ultimately have (INF n. (u o r) n) ≥ liminf u by simp

 {
  fix i assume i: i ∈ {N<..}
   obtain n where i < r (Suc n) using ‹strict_mono r› using Suc_le_eq
seq_suble by blast
   then have i ∈ {N<..r(Suc n)} using i by simp
   then have u i ≥ u (r(Suc n)) using r by simp
   then have u i ≥ (INF n. (u o r) n) unfolding o_def by (meson INF_lower2
UNIV_I)
 }
 then have (INF i∈{N<..}. u i) ≥ (INF n. (u o r) n) using INF_greatest by
blast
 then have liminf u ≥ (INF n. (u o r) n) unfolding Liminf_def
  by (metis (mono_tags, lifting) SUP_upper2 atLeast_Suc_greaterThan atLeast_def
eventually_ge_at_top mem_Collect_eq)
 then have liminf u = (INF n. (u o r) n) using ‹(INF n. (u o r) n) ≥ liminf u›
by simp
 then have (u o r) ⟶ liminf u using ‹(u o r) ⟶ (INF n. (u o r) n)›
by simp
 then show ?thesis using ‹strict_mono r› by auto
qed
```

The following statement about limsups is reduced to a statement about

limits using subsequences thanks to *limsup_subseq_lim*. The statement for limits follows for instance from *tendsto_add_ereal_general*.

lemma *ereal_limsup_add_mono*:
fixes $u\ v$::*nat* ⇒ *ereal*
shows *limsup* $(\lambda n.\ u\ n + v\ n) \leq$ *limsup* $u +$ *limsup* v
proof (*cases*)
assume (*limsup* $u = \infty) \vee$ (*limsup* $v = \infty$)
then have *limsup* $u +$ *limsup* $v = \infty$ **by** *simp*
then show *?thesis* **by** *auto*
next
assume ¬((*limsup* $u = \infty) \vee$ (*limsup* $v = \infty$))
then have *limsup* $u < \infty$ *limsup* $v < \infty$ **by** *auto*

define w **where** $w = (\lambda n.\ u\ n + v\ n)$
obtain r **where** r: *strict_mono* r $(w\ o\ r) \longrightarrow$ *limsup* w **using** *limsup_subseq_lim* **by** *auto*
obtain s **where** s: *strict_mono* s $(u\ o\ r\ o\ s) \longrightarrow$ *limsup* $(u\ o\ r)$ **using** *limsup_subseq_lim* **by** *auto*
obtain t **where** t: *strict_mono* t $(v\ o\ r\ o\ s\ o\ t) \longrightarrow$ *limsup* $(v\ o\ r\ o\ s)$ **using** *limsup_subseq_lim* **by** *auto*

define a **where** $a = r\ o\ s\ o\ t$
have *strict_mono* a **using** $r\ s\ t$ **by** (*simp add*: *a_def strict_mono_o*)
have l:$(w\ o\ a) \longrightarrow$ *limsup* w
$(u\ o\ a) \longrightarrow$ *limsup* $(u\ o\ r)$
$(v\ o\ a) \longrightarrow$ *limsup* $(v\ o\ r\ o\ s)$
apply (*metis* (*no_types*, *lifting*) $r(2)$ $s(1)$ $t(1)$ *LIMSEQ_subseq_LIMSEQ a_def comp_assoc*)
apply (*metis* (*no_types*, *lifting*) $s(2)$ $t(1)$ *LIMSEQ_subseq_LIMSEQ a_def comp_assoc*)
apply (*metis* (*no_types*, *lifting*) $t(2)$ *a_def comp_assoc*)
done

have *limsup* $(u\ o\ r) \leq$ *limsup* u **by** (*simp add*: *limsup_subseq_mono* $r(1)$)
then have a: *limsup* $(u\ o\ r) \neq \infty$ **using** ‹*limsup* $u < \infty$› **by** *auto*
have *limsup* $(v\ o\ r\ o\ s) \leq$ *limsup* v
by (*simp add*: *comp_assoc limsup_subseq_mono* $r(1)$ $s(1)$ *strict_mono_o*)
then have b: *limsup* $(v\ o\ r\ o\ s) \neq \infty$ **using** ‹*limsup* $v < \infty$› **by** *auto*

have $(\lambda n.\ (u\ o\ a)\ n + (v\ o\ a)\ n) \longrightarrow$ *limsup* $(u\ o\ r) +$ *limsup* $(v\ o\ r\ o\ s)$
using l *tendsto_add_ereal_general a b* **by** *fastforce*
moreover have $(\lambda n.\ (u\ o\ a)\ n + (v\ o\ a)\ n) = (w\ o\ a)$ **unfolding** *w_def* **by** *auto*
ultimately have $(w\ o\ a) \longrightarrow$ *limsup* $(u\ o\ r) +$ *limsup* $(v\ o\ r\ o\ s)$ **by** *simp*
then have *limsup* $w =$ *limsup* $(u\ o\ r) +$ *limsup* $(v\ o\ r\ o\ s)$ **using** $l(1)$ *LIMSEQ_unique* **by** *blast*
then have *limsup* $w \leq$ *limsup* $u +$ *limsup* v
using ‹*limsup* $(u\ o\ r) \leq$ *limsup* u› ‹*limsup* $(v\ o\ r\ o\ s) \leq$ *limsup* v› *add_mono* **by** *simp*

then show *?thesis* **unfolding** *w_def* **by** *simp*
qed

There is an asymmetry between liminfs and limsups in *ereal*, as $\infty + (-\infty) = \infty$. This explains why there are more assumptions in the next lemma dealing with liminfs that in the previous one about limsups.

lemma *ereal_liminf_add_mono*:
fixes *u v*::*nat* ⇒ *ereal*
assumes ¬((*liminf u* = ∞ ∧ *liminf v* = −∞) ∨ (*liminf u* = −∞ ∧ *liminf v* = ∞))
shows *liminf* (λ*n*. *u n* + *v n*) ≥ *liminf u* + *liminf v*
proof (*cases*)
assume (*liminf u* = −∞) ∨ (*liminf v* = −∞)
then have ∗: *liminf u* + *liminf v* = −∞ **using** *assms* **by** *auto*
show *?thesis* **by** (*simp add*: ∗)
next
assume ¬((*liminf u* = −∞) ∨ (*liminf v* = −∞))
then have *liminf u* > −∞ *liminf v* > −∞ **by** *auto*

define *w* **where** *w* = (λ*n*. *u n* + *v n*)
obtain *r* **where** *r*: *strict_mono r* (*w o r*) ⟶ *liminf w* **using** *liminf_subseq_lim* **by** *auto*
obtain *s* **where** *s*: *strict_mono s* (*u o r o s*) ⟶ *liminf* (*u o r*) **using** *liminf_subseq_lim* **by** *auto*
obtain *t* **where** *t*: *strict_mono t* (*v o r o s o t*) ⟶ *liminf* (*v o r o s*) **using** *liminf_subseq_lim* **by** *auto*

define *a* **where** *a* = *r o s o t*
have *strict_mono a* **using** *r s t* **by** (*simp add*: *a_def strict_mono_o*)
have *l*:(*w o a*) ⟶ *liminf w*
(*u o a*) ⟶ *liminf* (*u o r*)
(*v o a*) ⟶ *liminf* (*v o r o s*)
apply (*metis* (*no_types*, *lifting*) *r*(*2*) *s*(*1*) *t*(*1*) *LIMSEQ_subseq_LIMSEQ a_def comp_assoc*)
apply (*metis* (*no_types*, *lifting*) *s*(*2*) *t*(*1*) *LIMSEQ_subseq_LIMSEQ a_def comp_assoc*)
apply (*metis* (*no_types*, *lifting*) *t*(*2*) *a_def comp_assoc*)
done

have *liminf* (*u o r*) ≥ *liminf u* **by** (*simp add*: *liminf_subseq_mono r*(*1*))
then have *a*: *liminf* (*u o r*) ≠ −∞ **using** ‹*liminf u* > −∞› **by** *auto*
have *liminf* (*v o r o s*) ≥ *liminf v*
by (*simp add*: *comp_assoc liminf_subseq_mono r*(*1*) *s*(*1*) *strict_mono_o*)
then have *b*: *liminf* (*v o r o s*) ≠ −∞ **using** ‹*liminf v* > −∞› **by** *auto*

have (λ*n*. (*u o a*) *n* + (*v o a*) *n*) ⟶ *liminf* (*u o r*) + *liminf* (*v o r o s*)
using *l tendsto_add_ereal_general a b* **by** *fastforce*
moreover have (λ*n*. (*u o a*) *n* + (*v o a*) *n*) = (*w o a*) **unfolding** *w_def* **by** *auto*

ultimately have $(w \circ a) \longrightarrow liminf\ (u \circ r) + liminf\ (v \circ r \circ s)$ **by** *simp*
then have $liminf\ w = liminf\ (u \circ r) + liminf\ (v \circ r \circ s)$ **using** *l(1) LIMSEQ_unique* **by** *blast*
then have $liminf\ w \geq liminf\ u + liminf\ v$
using ‹$liminf\ (u \circ r) \geq liminf\ u$› ‹$liminf\ (v \circ r \circ s) \geq liminf\ v$› *add_mono* **by** *simp*
then show *?thesis* **unfolding** *w_def* **by** *simp*
qed

lemma *ereal_limsup_lim_add*:
fixes $u\ v{::}nat \Rightarrow ereal$
assumes $u \longrightarrow a$ $abs(a) \neq \infty$
shows $limsup\ (\lambda n.\ u\ n + v\ n) = a + limsup\ v$
proof −
have $limsup\ u = a$ **using** *assms(1)* **using** *tendsto_iff_Liminf_eq_Limsup trivial_limit_at_top_linorder* **by** *blast*
have $(\lambda n.\ -u\ n) \longrightarrow -a$ **using** *assms(1)* **by** *auto*
then have $limsup\ (\lambda n.\ -u\ n) = -a$ **using** *tendsto_iff_Liminf_eq_Limsup trivial_limit_at_top_linorder* **by** *blast*

have $limsup\ (\lambda n.\ u\ n + v\ n) \leq limsup\ u + limsup\ v$
by (*rule ereal_limsup_add_mono*)
then have *up*: $limsup\ (\lambda n.\ u\ n + v\ n) \leq a + limsup\ v$ **using** ‹$limsup\ u = a$› **by** *simp*

have *a*: $limsup\ (\lambda n.\ (u\ n + v\ n) + (-u\ n)) \leq limsup\ (\lambda n.\ u\ n + v\ n) + limsup\ (\lambda n.\ -u\ n)$
by (*rule ereal_limsup_add_mono*)
have *eventually* $(\lambda n.\ u\ n = ereal(real\_of\_ereal(u\ n)))$ *sequentially* **using** *assms real_lim_then_eventually_real* **by** *auto*
moreover have $\bigwedge x.\ x = ereal(real\_of\_ereal(x)) \Longrightarrow x + (-x) = 0$
by (*metis plus_ereal.simps(1) right_minus uminus_ereal.simps(1) zero_ereal_def*)
ultimately have *eventually* $(\lambda n.\ u\ n + (-u\ n) = 0)$ *sequentially*
by (*metis* (*mono_tags*, *lifting*) *eventually_mono*)
moreover have $\bigwedge n.\ u\ n + (-u\ n) = 0 \Longrightarrow u\ n + v\ n + (-u\ n) = v\ n$
by (*metis add.commute add.left_commute add.left_neutral*)
ultimately have *eventually* $(\lambda n.\ u\ n + v\ n + (-u\ n) = v\ n)$ *sequentially*
using *eventually_mono* **by** *force*
then have $limsup\ v = limsup\ (\lambda n.\ u\ n + v\ n + (-u\ n))$ **using** *Limsup_eq* **by** *force*
then have $limsup\ v \leq limsup\ (\lambda n.\ u\ n + v\ n) - a$ **using** *a* ‹$limsup\ (\lambda n.\ -u\ n) = -a$› **by** (*simp add*: *minus_ereal_def*)
then have $limsup\ (\lambda n.\ u\ n + v\ n) \geq a + limsup\ v$ **using** *assms(2)* **by** (*metis add.commute ereal_le_minus*)
then show *?thesis* **using** *up* **by** *simp*
qed

lemma *ereal_limsup_lim_mult*:
fixes $u\ v{::}nat \Rightarrow ereal$

assumes $u \longrightarrow a$ $a{>}0$ $a \neq \infty$
shows $limsup\ (\lambda n.\ u\ n * v\ n) = a * limsup\ v$
proof −
define w **where** $w = (\lambda n.\ u\ n * v\ n)$
obtain r **where** r: $strict\_mono\ r\ (v\ o\ r) \longrightarrow limsup\ v$ **using** $limsup\_subseq\_lim$
by $auto$
have $(u\ o\ r) \longrightarrow a$ **using** $assms(1)\ LIMSEQ\_subseq\_LIMSEQ\ r$ **by** $auto$
with $tendsto\_mult\_ereal[OF\ this\ r(2)]$ **have** $(\lambda n.\ (u\ o\ r)\ n * (v\ o\ r)\ n) \longrightarrow$
$a * limsup\ v$ **using** $assms(2)\ assms(3)$ **by** $auto$
moreover have $\bigwedge n.\ (w\ o\ r)\ n = (u\ o\ r)\ n * (v\ o\ r)\ n$ **unfolding** w_def **by**
$auto$
ultimately have $(w\ o\ r) \longrightarrow a * limsup\ v$ **unfolding** w_def **by** $presburger$
then have $limsup\ (w\ o\ r) = a * limsup\ v$ **by** $(simp\ add{:}\ tendsto\_iff\_Liminf\_eq\_Limsup)$
then have I: $limsup\ w \geq a * limsup\ v$ **by** $(metis\ limsup\_subseq\_mono\ r(1))$

obtain s **where** s: $strict\_mono\ s\ (w\ o\ s) \longrightarrow limsup\ w$ **using** $limsup\_subseq\_lim$
by $auto$
have $*$: $(u\ o\ s) \longrightarrow a$ **using** $assms(1)\ LIMSEQ\_subseq\_LIMSEQ\ s$ **by** $auto$
have $eventually\ (\lambda n.\ (u\ o\ s)\ n > 0)\ sequentially$ **using** $assms(2) * order\_tendsto\_iff$
by $blast$
moreover have $eventually\ (\lambda n.\ (u\ o\ s)\ n < \infty)\ sequentially$ **using** $assms(3) *$
$order\_tendsto\_iff$ **by** $blast$
moreover have $(w\ o\ s)\ n\ /\ (u\ o\ s)\ n = (v\ o\ s)\ n$ **if** $(u\ o\ s)\ n > 0\ (u\ o\ s)\ n <$
∞ **for** n
unfolding w_def **using** $that$ **by** $(auto\ simp{:}\ ereal\_divide\_eq)$
ultimately have $eventually\ (\lambda n.\ (w\ o\ s)\ n\ /\ (u\ o\ s)\ n = (v\ o\ s)\ n)\ sequentially$
using $eventually\_elim2$ **by** $force$
moreover have $(\lambda n.\ (w\ o\ s)\ n\ /\ (u\ o\ s)\ n) \longrightarrow (limsup\ w)\ /\ a$
apply $(rule\ tendsto\_divide\_ereal[OF\ s(2)\ *])$ **using** $assms(2)\ assms(3)$ **by**
$auto$
ultimately have $(v\ o\ s) \longrightarrow (limsup\ w)\ /\ a$ **using** $Lim\_transform\_eventually$
by $fastforce$
then have $limsup\ (v\ o\ s) = (limsup\ w)\ /\ a$ **by** $(simp\ add{:}\ tendsto\_iff\_Liminf\_eq\_Limsup)$
then have $limsup\ v \geq (limsup\ w)\ /\ a$ **by** $(metis\ limsup\_subseq\_mono\ s(1))$
then have $a * limsup\ v \geq limsup\ w$ **using** $assms(2)\ assms(3)$ **by** $(simp\ add{:}$
$ereal\_divide\_le\_pos)$
then show $?thesis$ **using** I **unfolding** w_def **by** $auto$
qed

lemma $ereal\_liminf\_lim\_mult$:
fixes $u\ v{::}nat \Rightarrow ereal$
assumes $u \longrightarrow a$ $a{>}0$ $a \neq \infty$
shows $liminf\ (\lambda n.\ u\ n * v\ n) = a * liminf\ v$
proof −
define w **where** $w = (\lambda n.\ u\ n * v\ n)$
obtain r **where** r: $strict\_mono\ r\ (v\ o\ r) \longrightarrow liminf\ v$ **using** $liminf\_subseq\_lim$
by $auto$
have $(u\ o\ r) \longrightarrow a$ **using** $assms(1)\ LIMSEQ\_subseq\_LIMSEQ\ r$ **by** $auto$
with $tendsto\_mult\_ereal[OF\ this\ r(2)]$ **have** $(\lambda n.\ (u\ o\ r)\ n * (v\ o\ r)\ n) \longrightarrow$

$a * \mathit{liminf}\ v$ **using** *assms*(2) *assms*(3) **by** *auto*
moreover have $\bigwedge n.\ (w\ o\ r)\ n = (u\ o\ r)\ n * (v\ o\ r)\ n$ **unfolding** *w_def* **by** *auto*
ultimately have $(w\ o\ r) \longrightarrow a * \mathit{liminf}\ v$ **unfolding** *w_def* **by** *presburger*
then have $\mathit{liminf}\ (w\ o\ r) = a * \mathit{liminf}\ v$ **by** (*simp add*: *tendsto_iff_Liminf_eq_Limsup*)
then have *I*: $\mathit{liminf}\ w \leq a * \mathit{liminf}\ v$ **by** (*metis liminf_subseq_mono r*(1))

obtain *s* **where** *s*: *strict_mono s* $(w\ o\ s) \longrightarrow \mathit{liminf}\ w$ **using** *liminf_subseq_lim* **by** *auto*
have ∗: $(u\ o\ s) \longrightarrow a$ **using** *assms*(1) *LIMSEQ_subseq_LIMSEQ s* **by** *auto*
have *eventually* $(\lambda n.\ (u\ o\ s)\ n > 0)$ *sequentially* **using** *assms*(2) ∗ *order_tendsto_iff* **by** *blast*
moreover have *eventually* $(\lambda n.\ (u\ o\ s)\ n < \infty)$ *sequentially* **using** *assms*(3) ∗ *order_tendsto_iff* **by** *blast*
moreover have $(w\ o\ s)\ n\ /\ (u\ o\ s)\ n = (v\ o\ s)\ n$ **if** $(u\ o\ s)\ n > 0$ $(u\ o\ s)\ n < \infty$ **for** *n*
unfolding *w_def* **using** *that* **by** (*auto simp*: *ereal_divide_eq*)
ultimately have *eventually* $(\lambda n.\ (w\ o\ s)\ n\ /\ (u\ o\ s)\ n = (v\ o\ s)\ n)$ *sequentially* **using** *eventually_elim2* **by** *force*
moreover have $(\lambda n.\ (w\ o\ s)\ n\ /\ (u\ o\ s)\ n) \longrightarrow (\mathit{liminf}\ w)\ /\ a$
using ∗ *assms s tendsto_divide_ereal* **by** *fastforce*
ultimately have $(v\ o\ s) \longrightarrow (\mathit{liminf}\ w)\ /\ a$ **using** *Lim_transform_eventually* **by** *fastforce*
then have $\mathit{liminf}\ (v\ o\ s) = (\mathit{liminf}\ w)\ /\ a$ **by** (*simp add*: *tendsto_iff_Liminf_eq_Limsup*)
then have $\mathit{liminf}\ v \leq (\mathit{liminf}\ w)\ /\ a$ **by** (*metis liminf_subseq_mono s*(1))
then have $a * \mathit{liminf}\ v \leq \mathit{liminf}\ w$ **using** *assms*(2) *assms*(3) **by** (*simp add*: *ereal_le_divide_pos*)
then show *?thesis* **using** *I* **unfolding** *w_def* **by** *auto*
qed

lemma *ereal_liminf_lim_add*:
fixes *u v*::*nat* ⇒ *ereal*
assumes $u \longrightarrow a$ $\mathit{abs}(a) \neq \infty$
shows $\mathit{liminf}\ (\lambda n.\ u\ n + v\ n) = a + \mathit{liminf}\ v$
proof −
have $\mathit{liminf}\ u = a$ **using** *assms*(1) *tendsto_iff_Liminf_eq_Limsup trivial_limit_at_top_linorder* **by** *blast*
then have ∗: $\mathit{abs}(\mathit{liminf}\ u) \neq \infty$ **using** *assms*(2) **by** *auto*
have $(\lambda n.\ -u\ n) \longrightarrow -a$ **using** *assms*(1) **by** *auto*
then have $\mathit{liminf}\ (\lambda n.\ -u\ n) = -a$ **using** *tendsto_iff_Liminf_eq_Limsup trivial_limit_at_top_linorder* **by** *blast*
then have ∗∗: $\mathit{abs}(\mathit{liminf}\ (\lambda n.\ -u\ n)) \neq \infty$ **using** *assms*(2) **by** *auto*

have $\mathit{liminf}\ (\lambda n.\ u\ n + v\ n) \geq \mathit{liminf}\ u + \mathit{liminf}\ v$
using *abs_ereal.simps* **by** (*metis* (*full_types*) ∗ *ereal_liminf_add_mono*)
then have *up*: $\mathit{liminf}\ (\lambda n.\ u\ n + v\ n) \geq a + \mathit{liminf}\ v$ **using** ‹$\mathit{liminf}\ u = a$› **by** *simp*

have *a*: $\mathit{liminf}\ (\lambda n.\ (u\ n + v\ n) + (-u\ n)) \geq \mathit{liminf}\ (\lambda n.\ u\ n + v\ n) + \mathit{liminf}$

```
(λn. −u n)
    apply (rule ereal_liminf_add_mono) using ∗∗ by auto
  have eventually (λn. u n = ereal(real_of_ereal(u n))) sequentially using assms
    real_lim_then_eventually_real by auto
  moreover have ⋀x. x = ereal(real_of_ereal(x)) ⟹ x + (−x) = 0
   by (metis plus_ereal.simps(1) right_minus uminus_ereal.simps(1) zero_ereal_def)
  ultimately have eventually (λn. u n + (−u n) = 0) sequentially
    by (metis (mono_tags, lifting) eventually_mono)
  moreover have ⋀n. u n + (−u n) = 0 ⟹ u n + v n + (−u n) = v n
    by (metis add.commute add.left_commute add.left_neutral)
  ultimately have eventually (λn. u n + v n + (−u n) = v n) sequentially
    using eventually_mono by force
  then have liminf v = liminf (λn. u n + v n + (−u n)) using Liminf_eq by
force
  then have liminf v ≥ liminf (λn. u n + v n) −a using a ‹liminf (λn. −u n) =
−a› by (simp add: minus_ereal_def)
  then have liminf (λn. u n + v n) ≤ a + liminf v using assms(2) by (metis
add.commute ereal_minus_le)
  then show ?thesis using up by simp
qed

lemma ereal_liminf_limsup_add:
  fixes u v::nat ⇒ ereal
  shows liminf (λn. u n + v n) ≤ liminf u + limsup v
proof (cases)
  assume limsup v = ∞ ∨ liminf u = ∞
  then show ?thesis by auto
next
  assume ¬(limsup v = ∞ ∨ liminf u = ∞)
  then have limsup v < ∞ liminf u < ∞ by auto

  define w where w = (λn. u n + v n)
 obtain r where r: strict_mono r (u o r) ⟶ liminf u using liminf_subseq_lim
by auto
   obtain s where s: strict_mono s (w o r o s) ⟶ liminf (w o r) using
liminf_subseq_lim by auto
  obtain t where t: strict_mono t (v o r o s o t) ⟶ limsup (v o r o s) using
limsup_subseq_lim by auto

  define a where a = r o s o t
  have strict_mono a using r s t by (simp add: a_def strict_mono_o)
  have l:(u o a) ⟶ liminf u
        (w o a) ⟶ liminf (w o r)
        (v o a) ⟶ limsup (v o r o s)
 apply (metis (no_types, lifting) r(2) s(1) t(1) LIMSEQ_subseq_LIMSEQ a_def
comp_assoc)
   apply (metis (no_types, lifting) s(2) t(1) LIMSEQ_subseq_LIMSEQ a_def
comp_assoc)
  apply (metis (no_types, lifting) t(2) a_def comp_assoc)
```

done

have *liminf* (*w o r*) ≥ *liminf w* **by** (*simp add*: *liminf_subseq_mono r*(*1*))
have *limsup* (*v o r o s*) ≤ *limsup v*
by (*simp add*: *comp_assoc limsup_subseq_mono r*(*1*) *s*(*1*) *strict_mono_o*)
then have *b*: *limsup* (*v o r o s*) < ∞ **using** ‹*limsup v* < ∞› **by** *auto*

have (λ*n*. (*u o a*) *n* + (*v o a*) *n*) ⟶ *liminf u* + *limsup* (*v o r o s*)
apply (*rule tendsto_add_ereal_general*) **using** *b* ‹*liminf u* < ∞› *l*(*1*) *l*(*3*) **by** *force*+
moreover have (λ*n*. (*u o a*) *n* + (*v o a*) *n*) = (*w o a*) **unfolding** *w_def* **by** *auto*
ultimately have (*w o a*) ⟶ *liminf u* + *limsup* (*v o r o s*) **by** *simp*
then have *liminf* (*w o r*) = *liminf u* + *limsup* (*v o r o s*) **using** *l*(*2*) **using** *LIMSEQ_unique* **by** *blast*
then have *liminf w* ≤ *liminf u* + *limsup v*
using ‹*liminf* (*w o r*) ≥ *liminf w*› ‹*limsup* (*v o r o s*) ≤ *limsup v*›
by (*metis add_mono_thms_linordered_semiring*(*2*) *le_less_trans not_less*)
then show *?thesis* **unfolding** *w_def* **by** *simp*
qed

lemma *ereal_liminf_limsup_minus*:
fixes *u v*::*nat* ⇒ *ereal*
shows *liminf* (λ*n*. *u n* − *v n*) ≤ *limsup u* − *limsup v*
unfolding *minus_ereal_def*
apply (*subst add.commute*)
apply (*rule order_trans*[*OF ereal_liminf_limsup_add*])
using *ereal_Limsup_uminus*[*of sequentially* λ*n*. − *v n*]
apply (*simp add*: *add.commute*)
done

lemma *liminf_minus_ennreal*:
fixes *u v*::*nat* ⇒ *ennreal*
shows (⋀*n*. *v n* ≤ *u n*) ⟹ *liminf* (λ*n*. *u n* − *v n*) ≤ *limsup u* − *limsup v*
unfolding *liminf_SUP_INF limsup_INF_SUP*
including *ennreal.lifting*
proof (*transfer*, *clarsimp*)
fix *v u* :: *nat* ⇒ *ereal* **assume** ∗: ∀ *x*. *0* ≤ *v x* ∀ *x*. *0* ≤ *u x* ⋀*n*. *v n* ≤ *u n*
moreover have *0* ≤ *limsup u* − *limsup v*
using ∗ **by** (*intro ereal_diff_positive Limsup_mono always_eventually*) *simp*
moreover have *0* ≤ *Sup* (*u* ‘ {*x*..}) **for** *x*
using ∗ **by** (*intro SUP_upper2*[*of x*]) *auto*
moreover have *0* ≤ *Sup* (*v* ‘ {*x*..}) **for** *x*
using ∗ **by** (*intro SUP_upper2*[*of x*]) *auto*
ultimately show (*SUP n*. *INF n*∈{*n*..}. *max 0* (*u n* − *v n*))
≤ *max 0* ((*INF x*. *max 0* (*Sup* (*u* ‘ {*x*..}))) − (*INF x*. *max 0* (*Sup* (*v* ‘ {*x*..}))))
by (*auto simp*: ∗ *ereal_diff_positive max.absorb2 liminf_SUP_INF*[*symmetric*]

limsup_INF_SUP[*symmetric*] *ereal_liminf_limsup_minus*)
qed

5.13.4 Relate extended reals and the indicator function

lemma *ereal_indicator_le_0*: (*indicator S x*::*ereal*) ≤ *0* ⟷ *x* ∉ *S*
by (*auto split*: *split_indicator simp*: *one_ereal_def*)

lemma *ereal_indicator*: *ereal* (*indicator A x*) = *indicator A x*
by (*auto simp*: *indicator_def one_ereal_def*)

lemma *ereal_mult_indicator*: *ereal* (*x* ∗ *indicator A y*) = *ereal x* ∗ *indicator A y*
by (*simp split*: *split_indicator*)

lemma *ereal_indicator_mult*: *ereal* (*indicator A y* ∗ *x*) = *indicator A y* ∗ *ereal x*
by (*simp split*: *split_indicator*)

lemma *ereal_indicator_nonneg*[*simp*, *intro*]: *0* ≤ (*indicator A x* ::*ereal*)
unfolding *indicator_def* **by** *auto*

lemma *indicator_inter_arith_ereal*: *indicator A x* ∗ *indicator B x* = (*indicator* (*A* ∩ *B*) *x* :: *ereal*)
by (*simp split*: *split_indicator*)

end

5.14 Radius of Convergence and Summation Tests

theory *Summation_Tests*
imports
Complex_Main
HOL−Library.Discrete
HOL−Library.Extended_Real
HOL−Library.Liminf_Limsup
Extended_Real_Limits
begin

The definition of the radius of convergence of a power series, various summability tests, lemmas to compute the radius of convergence etc.

5.14.1 Convergence tests for infinite sums

Root test

lemma *limsup_root_powser*:
fixes *f* :: *nat* ⇒ *'a* :: {*banach*, *real_normed_div_algebra*}
shows *limsup* (λ*n*. *ereal* (*root n* (*norm* (*f n* ∗ *z* ^ *n*)))) =
limsup (λ*n*. *ereal* (*root n* (*norm* (*f n*)))) ∗ *ereal* (*norm z*)
proof −

```
  have A: (λn. ereal (root n (norm (f n * z ^ n)))) =
            (λn. ereal (root n (norm (f n))) * ereal (norm z)) (is ?g = ?h)
  proof
    fix n show ?g n = ?h n
     by (cases n = 0) (simp_all add: norm_mult real_root_mult real_root_pos2
norm_power)
  qed
  show ?thesis by (subst A, subst limsup_ereal_mult_right) simp_all
qed

lemma limsup_root_limit:
  assumes (λn. ereal (root n (norm (f n)))) ⟶ l (is ?g ⟶ _)
  shows   limsup (λn. ereal (root n (norm (f n)))) = l
proof −
  from assms have convergent ?g lim ?g = l
    unfolding convergent_def by (blast intro: limI)+
  with convergent_limsup_cl show ?thesis by force
qed

lemma limsup_root_limit':
  assumes (λn. root n (norm (f n))) ⟶ l
  shows   limsup (λn. ereal (root n (norm (f n)))) = ereal l
  by (intro limsup_root_limit tendsto_ereal assms)

theorem root_test_convergence':
  fixes f :: nat ⇒ 'a :: banach
  defines l ≡ limsup (λn. ereal (root n (norm (f n))))
  assumes l: l < 1
  shows   summable f
proof −
  have 0 = limsup (λn. 0) by (simp add: Limsup_const)
   also have ... ≤ l unfolding l_def by (intro Limsup_mono) (simp_all add:
real_root_ge_zero)
  finally have l ≥ 0 by simp
  with l obtain l' where l': l = ereal l' by (cases l) simp_all

  define c where c = (1 − l') / 2
  from l and ‹l ≥ 0› have c: l + c > l l' + c ≥ 0 l' + c < 1 unfolding c_def
    by (simp_all add: field_simps l')
  have ∀ C>l. eventually (λn. ereal (root n (norm (f n))) < C) sequentially
    by (subst Limsup_le_iff[symmetric]) (simp add: l_def)
 with c have eventually (λn. ereal (root n (norm (f n))) < l + ereal c) sequentially
by simp
  with eventually_gt_at_top[of 0::nat]
    have eventually (λn. norm (f n) ≤ (l' + c) ^ n) sequentially
  proof eventually_elim
    fix n :: nat assume n: n > 0
    assume ereal (root n (norm (f n))) < l + ereal c
    hence root n (norm (f n)) ≤ l' + c by (simp add: l')
```

```
    with c n have root n (norm (f n)) ^ n ≤ (l' + c) ^ n
      by (intro power_mono) (simp_all add: real_root_ge_zero)
    also from n have root n (norm (f n)) ^ n = norm (f n) by simp
    finally show norm (f n) ≤ (l' + c) ^ n by simp
  qed
  thus ?thesis
     by (rule summable_comparison_test_ev[OF _ summable_geometric]) (simp
add: c)
qed

theorem root_test_divergence:
  fixes f :: nat ⇒ 'a :: banach
  defines l ≡ limsup (λn. ereal (root n (norm (f n))))
  assumes l: l > 1
  shows   ¬summable f
proof
  assume summable f
  hence bounded: Bseq f by (simp add: summable_imp_Bseq)

  have 0 = limsup (λn. 0) by (simp add: Limsup_const)
   also have ... ≤ l unfolding l_def by (intro Limsup_mono) (simp_all add:
real_root_ge_zero)
  finally have l_nonneg: l ≥ 0 by simp

  define c where c = (if l = ∞ then 2 else 1 + (real_of_ereal l − 1) / 2)
  from l l_nonneg consider l = ∞ | ∃ l'. l = ereal l' by (cases l) simp_all
  hence c: c > 1 ∧ ereal c < l by cases (insert l, auto simp: c_def field_simps)

  have unbounded: ¬bdd_above {n. root n (norm (f n)) > c}
  proof
    assume bdd_above {n. root n (norm (f n)) > c}
    then obtain N where ∀ n. root n (norm (f n)) > c ⟶ n ≤ N unfolding
bdd_above_def by blast
    hence ∃ N. ∀ n≥N. root n (norm (f n)) ≤ c
      by (intro exI[of _ N + 1]) (force simp: not_less_eq_eq[symmetric])
    hence eventually (λn. root n (norm (f n)) ≤ c) sequentially
      by (auto simp: eventually_at_top_linorder)
    hence l ≤ c unfolding l_def by (intro Limsup_bounded) simp_all
    with c show False by auto
  qed

  from bounded obtain K where K: K > 0 ⋀n. norm (f n) ≤ K using BseqE
by blast
  define n where n = nat ⌈log c K⌉
 from unbounded have ∃ m>n. c < root m (norm (f m)) unfolding bdd_above_def
    by (auto simp: not_le)
  then obtain m where m: n < m c < root m (norm (f m)) by auto
  from c K have K = c powr log c K by (simp add: powr_def log_def)
  also from c have c powr log c K ≤ c powr real n unfolding n_def
```

by (*intro powr_mono*, *linarith*, *simp*)
finally have $K \leq c \mathbin{\hat{}} n$ **using** *c* **by** (*simp add*: *powr_realpow*)
also from *c m* **have** $c \mathbin{\hat{}} n < c \mathbin{\hat{}} m$ **by** *simp*
also from *c m* **have** $c \mathbin{\hat{}} m < root\ m\ (norm\ (f\ m)) \mathbin{\hat{}} m$ **by** (*intro power_strict_mono*) *simp_all*
also from *m* **have** $... = norm\ (f\ m)$ **by** *simp*
finally show *False* **using** $K(2)[of\ m]$ **by** *simp*
qed

Cauchy's condensation test

context
fixes $f :: nat \Rightarrow real$
begin

private lemma *condensation_inequality*:
assumes *mono*: $\bigwedge m\ n.\ 0 < m \Longrightarrow m \leq n \Longrightarrow f\ n \leq f\ m$
shows $(\sum k{=}1..{<}n.\ f\ k) \geq (\sum k{=}1..{<}n.\ f\ (2 * 2 \mathbin{\hat{}} Discrete.log\ k))$ (**is** *?thesis1*)
$(\sum k{=}1..{<}n.\ f\ k) \leq (\sum k{=}1..{<}n.\ f\ (2 \mathbin{\hat{}} Discrete.log\ k))$ (**is** *?thesis2*)
by (*intro sum_mono mono Discrete.log_exp2_ge Discrete.log_exp2_le*, *simp*, *simp*)+

private lemma *condensation_condense1*: $(\sum k{=}1..{<}2\hat{}n.\ f\ (2 \mathbin{\hat{}} Discrete.log\ k)) = (\sum k{<}n.\ 2\hat{}k * f\ (2 \mathbin{\hat{}} k))$
proof (*induction n*)
case (*Suc n*)
have $\{1..{<}2\hat{}Suc\ n\} = \{1..{<}2\hat{}n\} \cup \{2\hat{}n..{<}(2\hat{}Suc\ n :: nat)\}$ **by** *auto*
also have $(\sum k{\in}.... f\ (2 \mathbin{\hat{}} Discrete.log\ k)) =$
$(\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k)) + (\sum k = 2\hat{}n..{<}2\hat{}Suc\ n.\ f\ (2\hat{}Discrete.log\ k))$
by (*subst sum.union_disjoint*) (*insert Suc*, *auto*)
also have $Discrete.log\ k = n$ **if** $k \in \{2\hat{}n..{<}2\hat{}Suc\ n\}$ **for** *k* **using** *that* **by** (*intro Discrete.log_eqI*) *simp_all*
hence $(\sum k = 2\hat{}n..{<}2\hat{}Suc\ n.\ f\ (2\hat{}Discrete.log\ k)) = (\sum (\_::nat) = 2\hat{}n..{<}2\hat{}Suc\ n.\ f\ (2\hat{}n))$
by (*intro sum.cong*) *simp_all*
also have $\ldots = 2\hat{}n * f\ (2\hat{}n)$ **by** (*simp*)
finally show *?case* **by** *simp*
qed *simp*

private lemma *condensation_condense2*: $(\sum k{=}1..{<}2\hat{}n.\ f\ (2 * 2 \mathbin{\hat{}} Discrete.log\ k)) = (\sum k{<}n.\ 2\hat{}k * f\ (2 \mathbin{\hat{}} Suc\ k))$
proof (*induction n*)
case (*Suc n*)
have $\{1..{<}2\hat{}Suc\ n\} = \{1..{<}2\hat{}n\} \cup \{2\hat{}n..{<}(2\hat{}Suc\ n :: nat)\}$ **by** *auto*
also have $(\sum k{\in}.... f\ (2 * 2 \mathbin{\hat{}} Discrete.log\ k)) =$
$(\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}Suc\ k)) + (\sum k = 2\hat{}n..{<}2\hat{}Suc\ n.\ f\ (2 * 2\hat{}Discrete.log\ k))$
by (*subst sum.union_disjoint*) (*insert Suc*, *auto*)

```
  also have Discrete.log k = n if k ∈ {2^n..<2^Suc n} for k using that by (intro
Discrete.log_eqI) simp_all
  hence (∑ k = 2^n..<2^Suc n. f (2*2^Discrete.log k)) = (∑ (_::nat) = 2^n..<2^Suc
n. f (2^Suc n))
    by (intro sum.cong) simp_all
  also have ... = 2^n * f (2^Suc n) by (simp)
  finally show ?case by simp
qed simp

theorem condensation_test:
  assumes mono: ⋀m. 0 < m ⟹ f (Suc m) ≤ f m
  assumes nonneg: ⋀n. f n ≥ 0
  shows summable f ⟷ summable (λn. 2^n * f (2^n))
proof −
  define f' where f' n = (if n = 0 then 0 else f n) for n
  from mono have mono': decseq (λn. f (Suc n)) by (intro decseq_SucI) simp
  hence mono': f n ≤ f m if m ≤ n m > 0 for m n
    using that decseqD[OF mono', of m − 1 n − 1] by simp

  have (λn. f (Suc n)) = (λn. f' (Suc n)) by (intro ext) (simp add: f'_def)
  hence summable f ⟷ summable f'
    by (subst (1 2) summable_Suc_iff [symmetric]) (simp only:)
 also have ... ⟷ convergent (λn. ∑ k<n. f' k) unfolding summable_iff_convergent
..
  also have monoseq (λn. ∑ k<n. f' k) unfolding f'_def
    by (intro mono_SucI1) (auto intro!: mult_nonneg_nonneg nonneg)
  hence convergent (λn. ∑ k<n. f' k) ⟷ Bseq (λn. ∑ k<n. f' k)
    by (rule monoseq_imp_convergent_iff_Bseq)
  also have ... ⟷ Bseq (λn. ∑ k=1..<n. f' k) unfolding One_nat_def
   by (subst sum_shift_lb_Suc0_0_upt) (simp_all add: f'_def atLeast0LessThan)
  also have ... ⟷ Bseq (λn. ∑ k=1..<n. f k) unfolding f'_def by simp
  also have ... ⟷ Bseq (λn. ∑ k=1..<2^n. f k)
    by (rule nonneg_incseq_Bseq_subseq_iff[symmetric])
       (auto intro!: sum_nonneg incseq_SucI nonneg simp: strict_mono_def)
  also have ... ⟷ Bseq (λn. ∑ k<n. 2^k * f (2^k))
  proof (intro iffI)
    assume A: Bseq (λn. ∑ k=1..<2^n. f k)
   have eventually (λn. norm (∑ k<n. 2^k * f (2^Suc k)) ≤ norm (∑ k=1..<2^n.
f k)) sequentially
    proof (intro always_eventually allI)
      fix n :: nat
       have norm (∑ k<n. 2^k * f (2^Suc k)) = (∑ k<n. 2^k * f (2^Suc k))
unfolding real_norm_def
        by (intro abs_of_nonneg sum_nonneg ballI mult_nonneg_nonneg nonneg)
simp_all
      also have ... ≤ (∑ k=1..<2^n. f k)
     by (subst condensation_condense2 [symmetric]) (intro condensation_inequality
mono')
      also have ... = norm ... unfolding real_norm_def
```

by (*intro abs_of_nonneg*[*symmetric*] *sum_nonneg ballI mult_nonneg_nonneg nonneg*)
finally show *norm* ($\sum k{<}n.\ 2\ \hat{}\ k * f\ (2\ \hat{}\ Suc\ k)$) $\leq$ *norm* ($\sum k{=}1..{<}2\hat{}n.\ f\ k$) **.**
qed
from *this* **and** *A* **have** *Bseq* ($\lambda n.\ \sum k{<}n.\ 2\hat{}k * f\ (2\hat{}Suc\ k)$) **by** (*rule Bseq_eventually_mono*)
from *Bseq_mult*[*OF Bfun_const*[*of 2*] *this*] **have** *Bseq* ($\lambda n.\ \sum k{<}n.\ 2\hat{}Suc\ k * f\ (2\hat{}Suc\ k)$)
by (*simp add*: *sum_distrib_left sum_distrib_right mult_ac*)
hence *Bseq* ($\lambda n.\ (\sum k{=}Suc\ 0..{<}Suc\ n.\ 2\hat{}k * f\ (2\hat{}k)) + f\ 1$)
by (*intro Bseq_add*, *subst sum.shift_bounds_Suc_ivl*) (*simp add*: *atLeast0LessThan*)
hence *Bseq* ($\lambda n.\ (\sum k{=}0..{<}Suc\ n.\ 2\hat{}k * f\ (2\hat{}k))$)
by (*subst sum.atLeast_Suc_lessThan*) (*simp_all add*: *add_ac*)
thus *Bseq* ($\lambda n.\ (\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k))$)
by (*subst* (*asm*) *Bseq_Suc_iff*) (*simp add*: *atLeast0LessThan*)
next
assume *A*: *Bseq* ($\lambda n.\ (\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k))$)
have *eventually* ($\lambda n.$ *norm* ($\sum k{=}1..{<}2\hat{}n.\ f\ k$) $\leq$ *norm* ($\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k)$)) *sequentially*
proof (*intro always_eventually allI*)
fix *n* :: *nat*
have *norm* ($\sum k{=}1..{<}2\hat{}n.\ f\ k$) = ($\sum k{=}1..{<}2\hat{}n.\ f\ k$) **unfolding** *real_norm_def*
by (*intro abs_of_nonneg sum_nonneg ballI mult_nonneg_nonneg nonneg*)
also have ... $\leq$ ($\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k)$)
by (*subst condensation_condense1* [*symmetric*]) (*intro condensation_inequality mono'*)
also have ... = *norm* ... **unfolding** *real_norm_def*
by (*intro abs_of_nonneg* [*symmetric*] *sum_nonneg ballI mult_nonneg_nonneg nonneg*) *simp_all*
finally show *norm* ($\sum k{=}1..{<}2\hat{}n.\ f\ k$) $\leq$ *norm* ($\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k)$) **.**
qed
from *this* **and** *A* **show** *Bseq* ($\lambda n.\ \sum k{=}1..{<}2\hat{}n.\ f\ k$) **by** (*rule Bseq_eventually_mono*)
qed
also have *monoseq* ($\lambda n.\ (\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k))$)
by (*intro mono_SucI1*) (*auto intro*!: *mult_nonneg_nonneg nonneg*)
hence *Bseq* ($\lambda n.\ (\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k))$) $\longleftrightarrow$ *convergent* ($\lambda n.\ (\sum k{<}n.\ 2\hat{}k * f\ (2\hat{}k))$)
by (*rule monoseq_imp_convergent_iff_Bseq* [*symmetric*])
also have ... $\longleftrightarrow$ *summable* ($\lambda k.\ 2\hat{}k * f\ (2\hat{}k)$) **by** (*simp only*: *summable_iff_convergent*)
finally show *?thesis* **.**
qed

end

Summability of powers

lemma *abs_summable_complex_powr_iff*:
summable ($\lambda n.$ *norm* (*exp* (*of_real* (*ln* (*of_nat n*)) * *s*))) $\longleftrightarrow$ *Re s* < -1

proof (*cases Re s ≤ 0*)
let *?l* = *λn. complex_of_real* (*ln* (*of_nat n*))
case *False*
have *eventually* (*λn. norm* (*1* :: *real*) ≤ *norm* (*exp* (*?l n* ∗ *s*))) *sequentially*
apply (*rule eventually_mono* [*OF eventually_gt_at_top*[*of 0*::*nat*]])
using *False ge_one_powr_ge_zero* **by** *auto*
from *summable_comparison_test_ev*[*OF this*] *False* **show** *?thesis* **by** (*auto simp*: *summable_const_iff*)
next
let *?l* = *λn. complex_of_real* (*ln* (*of_nat n*))
case *True*
hence *summable* (*λn. norm* (*exp* (*?l n* ∗ *s*))) ⟷ *summable* (*λn. 2*^*n* ∗ *norm* (*exp* (*?l* (*2*^*n*) ∗ *s*)))
by (*intro condensation_test*) (*auto intro*!: *mult_right_mono_neg*)
also have (*λn. 2*^*n* ∗ *norm* (*exp* (*?l* (*2*^*n*) ∗ *s*))) = (*λn.* (*2 powr* (*Re s* + *1*)) ^ *n*)
proof
fix *n* :: *nat*
have *2*^*n* ∗ *norm* (*exp* (*?l* (*2*^*n*) ∗ *s*)) = *exp* (*real n* ∗ *ln 2*) ∗ *exp* (*real n* ∗ *ln 2* ∗ *Re s*)
using *True* **by** (*subst exp_of_nat_mult*) (*simp add*: *ln_realpow algebra_simps*)
also have ... = *exp* (*real n* ∗ (*ln 2* ∗ (*Re s* + *1*)))
by (*simp add*: *algebra_simps exp_add*)
also have ... = *exp* (*ln 2* ∗ (*Re s* + *1*)) ^ *n* **by** (*subst exp_of_nat_mult*) *simp*
also have *exp* (*ln 2* ∗ (*Re s* + *1*)) = *2 powr* (*Re s* + *1*) **by** (*simp add*: *powr_def*)
finally show *2*^*n* ∗ *norm* (*exp* (*?l* (*2*^*n*) ∗ *s*)) = (*2 powr* (*Re s* + *1*)) ^ *n* **.**
qed
also have *summable* ... ⟷ *2 powr* (*Re s* + *1*) < *2 powr 0*
by (*subst summable_geometric_iff*) *simp*
also have ... ⟷ *Re s* < −*1* **by** (*subst powr_less_cancel_iff*) (*simp*, *linarith*)
finally show *?thesis* **.**
qed

theorem *summable_complex_powr_iff*:
assumes *Re s* < −*1*
shows *summable* (*λn. exp* (*of_real* (*ln* (*of_nat n*)) ∗ *s*))
by (*rule summable_norm_cancel*, *subst abs_summable_complex_powr_iff*) *fact*

lemma *summable_real_powr_iff*: *summable* (*λn. of_nat n powr s* :: *real*) ⟷ *s* < −*1*
proof −
from *eventually_gt_at_top*[*of 0*::*nat*]
have *summable* (*λn. of_nat n powr s*) ⟷ *summable* (*λn. exp* (*ln* (*of_nat n*) ∗ *s*))
by (*intro summable_cong*) (*auto elim*!: *eventually_mono simp*: *powr_def*)
also have ... ⟷ *s* < −*1* **using** *abs_summable_complex_powr_iff*[*of of_real s*] **by** *simp*
finally show *?thesis* **.**

qed

lemma *inverse_power_summable*:
assumes *s*: $s \geq 2$
shows *summable* (λn. *inverse* (*of_nat* $n \mathbin{\hat{}} s$ :: $'a$ :: {*real_normed_div_algebra*,*banach*}))
proof (*rule summable_norm_cancel*, *subst summable_cong*)
from *eventually_gt_at_top*[*of* 0::*nat*]
show *eventually* (λn. *norm* (*inverse* (*of_nat* $n \mathbin{\hat{}} s$:: $'a$)) = *real_of_nat n powr* (−*real s*)) *at_top*
by *eventually_elim* (*simp add*: *norm_inverse norm_power powr_minus powr_realpow*)
qed (*insert s summable_real_powr_iff*[*of* $-s$], *simp_all*)

lemma *not_summable_harmonic*: ¬*summable* (λn. *inverse* (*of_nat n*) :: $'a$ :: *real_normed_field*)
proof
assume *summable* (λn. *inverse* (*of_nat n*) :: $'a$)
hence *convergent* (λn. *norm* (*of_real* ($\sum k{<}n$. *inverse* (*of_nat k*)) :: $'a$))
by (*simp add*: *summable_iff_convergent convergent_norm*)
hence *convergent* (λn. *abs* ($\sum k{<}n$. *inverse* (*of_nat k*)) :: *real*) **by** (*simp only*: *norm_of_real*)
also have (λn. *abs* ($\sum k{<}n$. *inverse* (*of_nat k*)) :: *real*) = (λn. $\sum k{<}n$. *inverse* (*of_nat k*))
by (*intro ext abs_of_nonneg sum_nonneg*) *auto*
also have *convergent* ... $\longleftrightarrow$ *summable* (λk. *inverse* (*of_nat k*) :: *real*)
by (*simp add*: *summable_iff_convergent*)
finally show *False* **using** *summable_real_powr_iff*[*of* −*1*] **by** (*simp add*: *powr_minus*)
qed

Kummer's test

theorem *kummers_test_convergence*:
fixes *f p* :: *nat* ⇒ *real*
assumes *pos_f*: *eventually* (λn. *f n* > *0*) *sequentially*
assumes *nonneg_p*: *eventually* (λn. *p n* ≥ *0*) *sequentially*
defines $l \equiv$ *liminf* (λn. *ereal* (*p n* ∗ *f n* / *f* (*Suc n*) − *p* (*Suc n*)))
assumes *l*: *l* > *0*
shows *summable f*
unfolding *summable_iff_convergent'*
proof −
define *r* **where** *r* = (*if l* = ∞ *then 1 else real_of_ereal l* / *2*)
from *l* **have** *r* > *0* ∧ *of_real r* < *l* **by** (*cases l*) (*simp_all add*: *r_def*)
hence *r*: *r* > *0 of_real r* < *l* **by** *simp_all*
hence *eventually* (λn. *p n* ∗ *f n* / *f* (*Suc n*) − *p* (*Suc n*) > *r*) *sequentially*
unfolding *l_def* **by** (*force dest*: *less_LiminfD*)
moreover from *pos_f* **have** *eventually* (λn. *f* (*Suc n*) > *0*) *sequentially*
by (*subst eventually_sequentially_Suc*)
ultimately have *eventually* (λn. *p n* ∗ *f n* − *p* (*Suc n*) ∗ *f* (*Suc n*) > *r* ∗ *f* (*Suc n*)) *sequentially*
by *eventually_elim* (*simp add*: *field_simps*)

```
  from eventually_conj[OF pos_f eventually_conj[OF nonneg_p this]]
    obtain m where m: ⋀n. n ≥ m ⟹ f n > 0 ⋀n. n ≥ m ⟹ p n ≥ 0
        ⋀n. n ≥ m ⟹ p n * f n − p (Suc n) * f (Suc n) > r * f (Suc n)
    unfolding eventually_at_top_linorder by blast

  let ?c = (norm (∑ k≤m. r * f k) + p m * f m) / r
  have Bseq (λn. (∑ k≤n + Suc m. f k))
  proof (rule BseqI')
    fix k :: nat
    define n where n = k + Suc m
    have n: n > m by (simp add: n_def)

    from r have r * norm (∑ k≤n. f k) = norm (∑ k≤n. r * f k)
      by (simp add: sum_distrib_left[symmetric] abs_mult)
    also from n have {..n} = {..m} ∪ {Suc m..n} by auto
    hence (∑ k≤n. r * f k) = (∑ k∈{..m} ∪ {Suc m..n}. r * f k) by (simp only:)
    also have ... = (∑ k≤m. r * f k) + (∑ k=Suc m..n. r * f k)
      by (subst sum.union_disjoint) auto
    also have norm ... ≤ norm (∑ k≤m. r * f k) + norm (∑ k=Suc m..n. r * f
k)
      by (rule norm_triangle_ineq)
    also from r less_imp_le[OF m(1)] have (∑ k=Suc m..n. r * f k) ≥ 0
      by (intro sum_nonneg) auto
    hence norm (∑ k=Suc m..n. r * f k) = (∑ k=Suc m..n. r * f k) by simp
    also have (∑ k=Suc m..n. r * f k) = (∑ k=m..<n. r * f (Suc k))
     by (subst sum.shift_bounds_Suc_ivl [symmetric])
         (simp only: atLeastLessThanSuc_atLeastAtMost)
    also from m have ... ≤ (∑ k=m..<n. p k * f k − p (Suc k) * f (Suc k))
      by (intro sum_mono[OF less_imp_le]) simp_all
    also have ... = −(∑ k=m..<n. p (Suc k) * f (Suc k) − p k * f k)
      by (simp add: sum_negf [symmetric] algebra_simps)
    also from n have ... = p m * f m − p n * f n
       by (cases n, simp, simp only: atLeastLessThanSuc_atLeastAtMost, subst
sum_Suc_diff) simp_all
    also from less_imp_le[OF m(1)] m(2) n have ... ≤ p m * f m by simp
    finally show norm (∑ k≤n. f k) ≤ (norm (∑ k≤m. r * f k) + p m * f m) /
r using r
      by (subst pos_le_divide_eq[OF r(1)]) (simp only: mult_ac)
  qed
  moreover have (∑ k≤n. f k) ≤ (∑ k≤n'. f k) if Suc m ≤ n n ≤ n' for n n'
    using less_imp_le[OF m(1)] that by (intro sum_mono2) auto
  ultimately show convergent (λn. ∑ k≤n. f k) by (rule Bseq_monoseq_convergent'_inc)
qed


theorem kummers_test_divergence:
  fixes f p :: nat ⇒ real
  assumes pos_f: eventually (λn. f n > 0) sequentially
  assumes pos_p: eventually (λn. p n > 0) sequentially
```

assumes *divergent_p*: ¬*summable* (λ*n*. *inverse* (*p n*))
defines *l* ≡ *limsup* (λ*n*. *ereal* (*p n* ∗ *f n* / *f* (*Suc n*) − *p* (*Suc n*)))
assumes *l*: *l* < *0*
shows ¬*summable f*
proof
assume *summable f*
from *eventually_conj*[*OF pos_f eventually_conj*[*OF pos_p Limsup_lessD*[*OF l*[*unfolded l_def*]]]]
obtain *N* **where** *N*: ⋀*n*. *n* ≥ *N* ⟹ *p n* > *0* ⋀*n*. *n* ≥ *N* ⟹ *f n* > *0*
⋀*n*. *n* ≥ *N* ⟹ *p n* ∗ *f n* / *f* (*Suc n*) − *p* (*Suc n*) < *0*
by (*auto simp*: *eventually_at_top_linorder*)
hence *A*: *p n* ∗ *f n* < *p* (*Suc n*) ∗ *f* (*Suc n*) **if** *n* ≥ *N* **for** *n* **using** *that N*[*of n*] *N*[*of Suc n*]
by (*simp add*: *field_simps*)
have *B*: *p n* ∗ *f n* ≥ *p N* ∗ *f N* **if** *n* ≥ *N* **for** *n* **using** *that* **and** *A*
by (*induction n rule*: *dec_induct*) (*auto intro*!: *less_imp_le elim*!: *order.trans*)
have *eventually* (λ*n*. *norm* (*p N* ∗ *f N* ∗ *inverse* (*p n*)) ≤ *f n*) *sequentially*
apply (*rule eventually_mono* [*OF eventually_ge_at_top*[*of N*]])
using *B N* **by** (*auto simp*: *field_simps abs_of_pos*)
from *this* **and** ‹*summable f*› **have** *summable* (λ*n*. *p N* ∗ *f N* ∗ *inverse* (*p n*))
by (*rule summable_comparison_test_ev*)
from *summable_mult*[*OF this*, *of inverse* (*p N* ∗ *f N*)] *N*[*OF le_refl*]
have *summable* (λ*n*. *inverse* (*p n*)) **by** (*simp add*: *field_split_simps*)
with *divergent_p* **show** *False* **by** *contradiction*
qed

Ratio test

theorem *ratio_test_convergence*:
fixes *f* :: *nat* ⇒ *real*
assumes *pos_f*: *eventually* (λ*n*. *f n* > *0*) *sequentially*
defines *l* ≡ *liminf* (λ*n*. *ereal* (*f n* / *f* (*Suc n*)))
assumes *l*: *l* > *1*
shows *summable f*
proof (*rule kummers_test_convergence*[*OF pos_f*])
note *l*
also have *l* = *liminf* (λ*n*. *ereal* (*f n* / *f* (*Suc n*) − *1*)) + *1*
by (*subst Liminf_add_ereal_right*[*symmetric*]) (*simp_all add*: *minus_ereal_def l_def one_ereal_def*)
finally show *liminf* (λ*n*. *ereal* (*1* ∗ *f n* / *f* (*Suc n*) − *1*)) > *0*
by (*cases liminf* (λ*n*. *ereal* (*1* ∗ *f n* / *f* (*Suc n*) − *1*))) *simp_all*
qed *simp*

theorem *ratio_test_divergence*:
fixes *f* :: *nat* ⇒ *real*
assumes *pos_f*: *eventually* (λ*n*. *f n* > *0*) *sequentially*
defines *l* ≡ *limsup* (λ*n*. *ereal* (*f n* / *f* (*Suc n*)))
assumes *l*: *l* < *1*
shows ¬*summable f*

proof (*rule kummers_test_divergence*[*OF pos_f*])
have *limsup* (λ*n*. *ereal* (*f n* / *f* (*Suc n*) − *1*)) + *1* = *l*
by (*subst Limsup_add_ereal_right*[*symmetric*]) (*simp_all add*: *minus_ereal_def l_def one_ereal_def*)
also note *l*
finally show *limsup* (λ*n*. *ereal* (*1* ∗ *f n* / *f* (*Suc n*) − *1*)) < *0*
by (*cases limsup* (λ*n*. *ereal* (*1* ∗ *f n* / *f* (*Suc n*) − *1*))) *simp_all*
qed (*simp_all add*: *summable_const_iff*)

Raabe's test

theorem *raabes_test_convergence*:
fixes *f* :: *nat* ⇒ *real*
assumes *pos*: *eventually* (λ*n*. *f n* > *0*) *sequentially*
defines *l* ≡ *liminf* (λ*n*. *ereal* (*of_nat n* ∗ (*f n* / *f* (*Suc n*) − *1*)))
assumes *l*: *l* > *1*
shows *summable f*
proof (*rule kummers_test_convergence*)
let *?l′* = *liminf* (λ*n*. *ereal* (*of_nat n* ∗ *f n* / *f* (*Suc n*) − *of_nat* (*Suc n*)))
have *1* < *l* **by** *fact*
also have *l* = *liminf* (λ*n*. *ereal* (*of_nat n* ∗ *f n* / *f* (*Suc n*) − *of_nat* (*Suc n*)) + *1*)
by (*simp add*: *l_def algebra_simps*)
also have ... = *?l′* + *1* **by** (*subst Liminf_add_ereal_right*) *simp_all*
finally show *?l′* > *0* **by** (*cases ?l′*) (*simp_all add*: *algebra_simps*)
qed (*simp_all add*: *pos*)

theorem *raabes_test_divergence*:
fixes *f* :: *nat* ⇒ *real*
assumes *pos*: *eventually* (λ*n*. *f n* > *0*) *sequentially*
defines *l* ≡ *limsup* (λ*n*. *ereal* (*of_nat n* ∗ (*f n* / *f* (*Suc n*) − *1*)))
assumes *l*: *l* < *1*
shows ¬*summable f*
proof (*rule kummers_test_divergence*)
let *?l′* = *limsup* (λ*n*. *ereal* (*of_nat n* ∗ *f n* / *f* (*Suc n*) − *of_nat* (*Suc n*)))
note *l*
also have *l* = *limsup* (λ*n*. *ereal* (*of_nat n* ∗ *f n* / *f* (*Suc n*) − *of_nat* (*Suc n*)) + *1*)
by (*simp add*: *l_def algebra_simps*)
also have ... = *?l′* + *1* **by** (*subst Limsup_add_ereal_right*) *simp_all*
finally show *?l′* < *0* **by** (*cases ?l′*) (*simp_all add*: *algebra_simps*)
qed (*insert pos eventually_gt_at_top*[*of 0*::*nat*] *not_summable_harmonic*, *simp_all*)

5.14.2 Radius of convergence

The radius of convergence of a power series. This value always exists, ranges from *0* to ∞, and the power series is guaranteed to converge for all inputs with a norm that is smaller than that radius and to diverge for all inputs with a norm that is greater.

definition *conv_radius* :: (*nat* ⇒ *'a* :: *banach*) ⇒ *ereal* **where**
conv_radius f = *inverse* (*limsup* (λ*n*. *ereal* (*root n* (*norm* (*f n*)))))

lemma *conv_radius_cong_weak* [*cong*]: (⋀*n*. *f n* = *g n*) ⟹ *conv_radius f* = *conv_radius g*
by (*drule ext*) *simp_all*

lemma *conv_radius_nonneg*: *conv_radius f* ≥ *0*
proof −
have *0* = *limsup* (λ*n*. *0*) **by** (*subst Limsup_const*) *simp_all*
also have ... ≤ *limsup* (λ*n*. *ereal* (*root n* (*norm* (*f n*))))
by (*intro Limsup_mono*) (*simp_all add*: *real_root_ge_zero*)
finally show *?thesis*
unfolding *conv_radius_def* **by** (*auto simp*: *ereal_inverse_nonneg_iff*)
qed

lemma *conv_radius_zero* [*simp*]: *conv_radius* (λ_. *0*) = ∞
by (*auto simp*: *conv_radius_def zero_ereal_def* [*symmetric*] *Limsup_const*)

lemma *conv_radius_altdef*:
conv_radius f = *liminf* (λ*n*. *inverse* (*ereal* (*root n* (*norm* (*f n*)))))
by (*subst Liminf_inverse_ereal*) (*simp_all add*: *real_root_ge_zero conv_radius_def*)

lemma *conv_radius_cong'*:
assumes *eventually* (λ*x*. *f x* = *g x*) *sequentially*
shows *conv_radius f* = *conv_radius g*
unfolding *conv_radius_altdef* **by** (*intro Liminf_eq eventually_mono* [*OF assms*])
auto

lemma *conv_radius_cong*:
assumes *eventually* (λ*x*. *norm* (*f x*) = *norm* (*g x*)) *sequentially*
shows *conv_radius f* = *conv_radius g*
unfolding *conv_radius_altdef* **by** (*intro Liminf_eq eventually_mono* [*OF assms*])
auto

theorem *abs_summable_in_conv_radius*:
fixes *f* :: *nat* ⇒ *'a* :: {*banach*, *real_normed_div_algebra*}
assumes *ereal* (*norm z*) < *conv_radius f*
shows *summable* (λ*n*. *norm* (*f n* ∗ *z* ^ *n*))
proof (*rule root_test_convergence'*)
define *l* **where** *l* = *limsup* (λ*n*. *ereal* (*root n* (*norm* (*f n*))))
have *0* = *limsup* (λ*n*. *0*) **by** (*simp add*: *Limsup_const*)
also have ... ≤ *l* **unfolding** *l_def* **by** (*intro Limsup_mono*) (*simp_all add*: *real_root_ge_zero*)
finally have *l_nonneg*: *l* ≥ *0* .

have *limsup* (λ*n*. *root n* (*norm* (*f n* ∗ *z*^*n*))) = *l* ∗ *ereal* (*norm z*) **unfolding** *l_def*
by (*rule limsup_root_powser*)

```
  also from l_nonneg consider l = 0 | l = ∞ | ∃ l'. l = ereal l' ∧ l' > 0
    by (cases l) (auto simp: less_le)
  hence l * ereal (norm z) < 1
  proof cases
    assume l = ∞
    hence conv_radius f = 0 unfolding conv_radius_def l_def by simp
    with assms show ?thesis by simp
  next
    assume ∃ l'. l = ereal l' ∧ l' > 0
    then obtain l' where l': l = ereal l' 0 < l' by auto
    hence l ≠ ∞ by auto
    have l * ereal (norm z) < l * conv_radius f
      by (intro ereal_mult_strict_left_mono) (simp_all add: l' assms)
    also have conv_radius f = inverse l by (simp add: conv_radius_def l_def)
    also from l' have l * inverse l = 1 by simp
    finally show ?thesis .
  qed simp_all
  finally show limsup (λn. ereal (root n (norm (norm (f n * z ^ n))))) < 1 by
simp
qed

lemma summable_in_conv_radius:
  fixes f :: nat ⇒ 'a :: {banach, real_normed_div_algebra}
  assumes ereal (norm z) < conv_radius f
  shows   summable (λn. f n * z ^ n)
  by (rule summable_norm_cancel, rule abs_summable_in_conv_radius) fact+

theorem not_summable_outside_conv_radius:
  fixes f :: nat ⇒ 'a :: {banach, real_normed_div_algebra}
  assumes ereal (norm z) > conv_radius f
  shows   ¬summable (λn. f n * z ^ n)
proof (rule root_test_divergence)
  define l where l = limsup (λn. ereal (root n (norm (f n))))
  have 0 = limsup (λn. 0) by (simp add: Limsup_const)
  also have ... ≤ l unfolding l_def by (intro Limsup_mono) (simp_all add:
real_root_ge_zero)
  finally have l_nonneg: l ≥ 0 .
  from assms have l_nz: l ≠ 0 unfolding conv_radius_def l_def by auto

  have limsup (λn. ereal (root n (norm (f n * z^n)))) = l * ereal (norm z)
    unfolding l_def by (rule limsup_root_powser)
  also have ... > 1
  proof (cases l)
    assume l = ∞
    with assms conv_radius_nonneg[of f] show ?thesis
      by (auto simp: zero_ereal_def[symmetric])
  next
    fix l' assume l': l = ereal l'
    from l_nonneg l_nz have 1 = l * inverse l by (auto simp: l' field_simps)
```

```
    also from l_nz have inverse l = conv_radius f
      unfolding l_def conv_radius_def by auto
    also from l' l_nz l_nonneg assms have l * ... < l * ereal (norm z)
      by (intro ereal_mult_strict_left_mono) (auto simp: l')
    finally show ?thesis .
  qed (insert l_nonneg, simp_all)
  finally show limsup (λn. ereal (root n (norm (f n * z^n)))) > 1 .
qed


lemma conv_radius_geI:
  assumes summable (λn. f n * z ^ n :: 'a :: {banach, real_normed_div_algebra})
  shows   conv_radius f ≥ norm z
  using not_summable_outside_conv_radius[of f z] assms by (force simp: not_le[symmetric])

lemma conv_radius_leI:
  assumes ¬summable (λn. norm (f n * z ^ n :: 'a :: {banach, real_normed_div_algebra}))
  shows   conv_radius f ≤ norm z
  using abs_summable_in_conv_radius[of z f] assms by (force simp: not_le[symmetric])

lemma conv_radius_leI':
  assumes ¬summable (λn. f n * z ^ n :: 'a :: {banach, real_normed_div_algebra})
  shows   conv_radius f ≤ norm z
  using summable_in_conv_radius[of z f] assms by (force simp: not_le[symmetric])

lemma conv_radius_geI_ex:
  fixes f :: nat ⇒ 'a :: {banach, real_normed_div_algebra}
  assumes ⋀r. 0 < r ⟹ ereal r < R ⟹ ∃z. norm z = r ∧ summable (λn. f n
* z^n)
  shows   conv_radius f ≥ R
proof (rule linorder_cases[of conv_radius f R])
  assume R: conv_radius f < R
  with conv_radius_nonneg[of f] obtain conv_radius'
    where [simp]: conv_radius f = ereal conv_radius'
    by (cases conv_radius f) simp_all
  define r where r = (if R = ∞ then conv_radius' + 1 else (real_of_ereal R +
conv_radius') / 2)
   from R conv_radius_nonneg[of f] have 0 < r ∧ ereal r < R ∧ ereal r >
conv_radius f
    unfolding r_def by (cases R) (auto simp: r_def field_simps)
  with assms(1)[of r] obtain z where norm z > conv_radius f summable (λn. f
n * z^n) by auto
  with not_summable_outside_conv_radius[of f z] show ?thesis by simp
qed simp_all

lemma conv_radius_geI_ex':
  fixes f :: nat ⇒ 'a :: {banach, real_normed_div_algebra}
  assumes ⋀r. 0 < r ⟹ ereal r < R ⟹ summable (λn. f n * of_real r^n)
  shows   conv_radius f ≥ R
```

proof (*rule conv_radius_geI_ex*)
fix *r* **assume** *0 < r ereal r < R*
with *assms*[*of r*] **show** ∃ *z*. *norm z = r* ∧ *summable* (λ*n*. *f n* ∗ *z* ^ *n*)
by (*intro exI*[*of _ of_real r :: 'a*]) *auto*
qed

lemma *conv_radius_leI_ex*:
fixes *f* :: *nat* ⇒ *'a* :: {*banach*, *real_normed_div_algebra*}
assumes *R* ≥ *0*
assumes ⋀*r*. *0 < r* ⟹ *ereal r > R* ⟹ ∃ *z*. *norm z = r* ∧ ¬*summable* (λ*n*. *norm* (*f n* ∗ *z*^*n*))
shows *conv_radius f* ≤ *R*
proof (*rule linorder_cases*[*of conv_radius f R*])
assume *R*: *conv_radius f > R*
from *R assms*(*1*) **obtain** *R'* **where** *R'*: *R = ereal R'* **by** (*cases R*) *simp_all*
define *r* **where**
r = (*if conv_radius f* = ∞ *then R'* + *1 else* (*R'* + *real_of_ereal* (*conv_radius f*)) / *2*)
from *R conv_radius_nonneg*[*of f*] **have** *r > R* ∧ *r < conv_radius f* **unfolding** *r_def*
by (*cases conv_radius f*) (*auto simp*: *r_def field_simps R'*)
with *assms*(*1*) *assms*(*2*)[*of r*] *R'*
obtain *z* **where** *norm z < conv_radius f* ¬*summable* (λ*n*. *norm* (*f n* ∗ *z*^*n*))
by *auto*
with *abs_summable_in_conv_radius*[*of z f*] **show** *?thesis* **by** *auto*
qed *simp_all*

lemma *conv_radius_leI_ex'*:
fixes *f* :: *nat* ⇒ *'a* :: {*banach*, *real_normed_div_algebra*}
assumes *R* ≥ *0*
assumes ⋀*r*. *0 < r* ⟹ *ereal r > R* ⟹ ¬*summable* (λ*n*. *f n* ∗ *of_real r*^*n*)
shows *conv_radius f* ≤ *R*
proof (*rule conv_radius_leI_ex*)
fix *r* **assume** *0 < r ereal r > R*
with *assms*(*2*)[*of r*] **show** ∃ *z*. *norm z = r* ∧ ¬*summable* (λ*n*. *norm* (*f n* ∗ *z* ^ *n*))
by (*intro exI*[*of _ of_real r :: 'a*]) (*auto dest*: *summable_norm_cancel*)
qed *fact*+

lemma *conv_radius_eqI*:
fixes *f* :: *nat* ⇒ *'a* :: {*banach*, *real_normed_div_algebra*}
assumes *R* ≥ *0*
assumes ⋀*r*. *0 < r* ⟹ *ereal r < R* ⟹ ∃ *z*. *norm z = r* ∧ *summable* (λ*n*. *f n* ∗ *z*^*n*)
assumes ⋀*r*. *0 < r* ⟹ *ereal r > R* ⟹ ∃ *z*. *norm z = r* ∧ ¬*summable* (λ*n*. *norm* (*f n* ∗ *z*^*n*))
shows *conv_radius f = R*
by (*intro antisym conv_radius_geI_ex conv_radius_leI_ex assms*)

lemma *conv_radius_eqI'*:
fixes $f :: nat \Rightarrow 'a :: \{banach, real\_normed\_div\_algebra\}$
assumes $R \geq 0$
assumes $\bigwedge r.\ 0 < r \Longrightarrow ereal\ r < R \Longrightarrow summable\ (\lambda n.\ f\ n * (of\_real\ r) \hat{} n)$
assumes $\bigwedge r.\ 0 < r \Longrightarrow ereal\ r > R \Longrightarrow \neg summable\ (\lambda n.\ norm\ (f\ n * (of\_real\ r) \hat{} n))$
shows $conv\_radius\ f = R$
proof (*intro conv_radius_eqI*[*OF assms*(*1*)])
fix r **assume** $0 < r\ ereal\ r < R$ **with** *assms*(*2*)[*OF this*]
show $\exists z.\ norm\ z = r \wedge summable\ (\lambda n.\ f\ n * z \hat{} n)$ **by** *force*
next
fix r **assume** $0 < r\ ereal\ r > R$ **with** *assms*(*3*)[*OF this*]
show $\exists z.\ norm\ z = r \wedge \neg summable\ (\lambda n.\ norm\ (f\ n * z \hat{} n))$ **by** *force*
qed

lemma *conv_radius_zeroI*:
fixes $f :: nat \Rightarrow 'a :: \{banach,real\_normed\_div\_algebra\}$
assumes $\bigwedge z.\ z \neq 0 \Longrightarrow \neg summable\ (\lambda n.\ f\ n * z \hat{} n)$
shows $conv\_radius\ f = 0$
proof (*rule ccontr*)
assume $conv\_radius\ f \neq 0$
with *conv_radius_nonneg*[*of f*] **have** *pos*: $conv\_radius\ f > 0$ **by** *simp*
define r **where** $r = (if\ conv\_radius\ f = \infty\ then\ 1\ else\ real\_of\_ereal\ (conv\_radius\ f)\ /\ 2)$
from *pos* **have** r: $ereal\ r > 0 \wedge ereal\ r < conv\_radius\ f$
by (*cases conv_radius f*) (*simp_all add*: *r_def*)
hence $summable\ (\lambda n.\ f\ n * of\_real\ r \hat{} n)$ **by** (*intro summable_in_conv_radius*) *simp*
moreover from r **and** *assms*[*of of_real r*] **have** $\neg summable\ (\lambda n.\ f\ n * of\_real\ r \hat{} n)$ **by** *simp*
ultimately show *False* **by** *contradiction*
qed

lemma *conv_radius_inftyI'*:
fixes $f :: nat \Rightarrow 'a :: \{banach,real\_normed\_div\_algebra\}$
assumes $\bigwedge r.\ r > c \Longrightarrow \exists z.\ norm\ z = r \wedge summable\ (\lambda n.\ f\ n * z \hat{} n)$
shows $conv\_radius\ f = \infty$
proof −
{
fix $r :: real$
have $max\ r\ (c + 1) > c$ **by** (*auto simp*: *max_def*)
from *assms*[*OF this*] **obtain** z **where** $norm\ z = max\ r\ (c + 1)\ summable\ (\lambda n.\ f\ n * z \hat{} n)$ **by** *blast*
from *conv_radius_geI*[*OF this*(*2*)] *this*(*1*) **have** $conv\_radius\ f \geq r$ **by** *simp*
}
from *this*[*of real_of_ereal* (*conv_radius f + 1*)] **show** $conv\_radius\ f = \infty$
by (*cases conv_radius f*) *simp_all*
qed

lemma *conv_radius_inftyI*:
fixes $f :: nat \Rightarrow 'a :: \{banach,real\_normed\_div\_algebra\}$
assumes $\bigwedge r.\ \exists z.\ norm\ z = r \wedge summable\ (\lambda n.\ f\ n * z \hat{} n)$
shows $conv\_radius\ f = \infty$
using *assms* **by** (*rule conv_radius_inftyI'*)

lemma *conv_radius_inftyI''*:
fixes $f :: nat \Rightarrow 'a :: \{banach,real\_normed\_div\_algebra\}$
assumes $\bigwedge z.\ summable\ (\lambda n.\ f\ n * z \hat{} n)$
shows $conv\_radius\ f = \infty$
proof (*rule conv_radius_inftyI'*)
fix $r :: real$ **assume** $r > 0$
with *assms* **show** $\exists z.\ norm\ z = r \wedge summable\ (\lambda n.\ f\ n * z \hat{} n)$
by (*intro exI*[*of* _ *of_real r*]) *simp*
qed

lemma *conv_radius_conv_Sup*:
fixes $f :: nat \Rightarrow 'a :: \{banach, real\_normed\_div\_algebra\}$
shows $conv\_radius\ f = Sup\ \{r.\ \forall z.\ ereal\ (norm\ z) < r \longrightarrow summable\ (\lambda n.\ f\ n * z \hat{} n)\}$
proof (*rule Sup_eqI* [*symmetric*], *goal_cases*)
case (*1 r*)
thus *?case*
by (*intro conv_radius_geI_ex'*) *auto*
next
case (*2 r*)
from *2*[*of 0*] **have** *r*: $r \geq 0$ **by** *auto*
show *?case*
proof (*intro conv_radius_leI_ex' r*)
fix R **assume** R: $R > 0$ $ereal\ R > r$
with r **obtain** r' **where** [*simp*]: $r = ereal\ r'$ **by** (*cases r*) *auto*
show $\neg summable\ (\lambda n.\ f\ n * of\_real\ R \hat{} n)$
proof
assume $*$: $summable\ (\lambda n.\ f\ n * of\_real\ R \hat{} n)$
define R' **where** $R' = (R + r')\ /\ 2$
from R **have** R': $R' > r'$ $R' < R$ **by** (*simp_all add*: R'_def)
hence $\forall z.\ norm\ z < R' \longrightarrow summable\ (\lambda n.\ f\ n * z \hat{} n)$
using *powser_inside*[*OF* $*$] **by** *auto*
from *2*[*of R'*] **and** *this* **have** $R' \leq r'$ **by** *auto*
with $\langle R' > r' \rangle$ **show** *False* **by** *simp*
qed
qed
qed

lemma *conv_radius_shift*:
fixes $f :: nat \Rightarrow 'a :: \{banach, real\_normed\_div\_algebra\}$
shows $conv\_radius\ (\lambda n.\ f\ (n + m)) = conv\_radius\ f$
unfolding *conv_radius_conv_Sup summable_powser_ignore_initial_segment* **..**

lemma *conv_radius_norm* [*simp*]: *conv_radius* (λ*x*. *norm* (*f x*)) = *conv_radius f*
by (*simp add*: *conv_radius_def*)

lemma *conv_radius_ratio_limit_ereal*:
fixes *f* :: *nat* ⇒ *'a* :: {*banach*,*real_normed_div_algebra*}
assumes *nz*: *eventually* (λ*n*. *f n* ≠ *0*) *sequentially*
assumes *lim*: (λ*n*. *ereal* (*norm* (*f n*) / *norm* (*f* (*Suc n*)))) ⟶ *c*
shows *conv_radius f* = *c*
proof (*rule conv_radius_eqI'*)
show *c* ≥ *0* **by** (*intro Lim_bounded2*[*OF lim*]) *simp_all*
next
fix *r* **assume** *r*: *0* < *r ereal r* < *c*
let *?l* = *liminf* (λ*n*. *ereal* (*norm* (*f n* ∗ *of_real r* ^ *n*) / *norm* (*f* (*Suc n*) ∗ *of_real r* ^ *Suc n*)))
have *?l* = *liminf* (λ*n*. *ereal* (*norm* (*f n*) / (*norm* (*f* (*Suc n*)))) ∗ *ereal* (*inverse r*))
using *r* **by** (*simp add*: *norm_mult norm_power field_split_simps*)
also from *r* **have** ... = *liminf* (λ*n*. *ereal* (*norm* (*f n*) / (*norm* (*f* (*Suc n*))))) ∗ *ereal* (*inverse r*)
by (*intro Liminf_ereal_mult_right*) *simp_all*
also have *liminf* (λ*n*. *ereal* (*norm* (*f n*) / (*norm* (*f* (*Suc n*))))) = *c*
by (*intro lim_imp_Liminf lim*) *simp*
finally have *l*: *?l* = *c* ∗ *ereal* (*inverse r*) **by** *simp*
from *r* **have** *l'*: *c* ∗ *ereal* (*inverse r*) > *1* **by** (*cases c*) (*simp_all add*: *field_simps*)
show *summable* (λ*n*. *f n* ∗ *of_real r*^*n*)
by (*rule summable_norm_cancel*, *rule ratio_test_convergence*)
(*insert r nz l l'*, *auto elim*!: *eventually_mono*)
next
fix *r* **assume** *r*: *0* < *r ereal r* > *c*
let *?l* = *limsup* (λ*n*. *ereal* (*norm* (*f n* ∗ *of_real r* ^ *n*) / *norm* (*f* (*Suc n*) ∗ *of_real r* ^ *Suc n*)))
have *?l* = *limsup* (λ*n*. *ereal* (*norm* (*f n*) / (*norm* (*f* (*Suc n*)))) ∗ *ereal* (*inverse r*))
using *r* **by** (*simp add*: *norm_mult norm_power field_split_simps*)
also from *r* **have** ... = *limsup* (λ*n*. *ereal* (*norm* (*f n*) / (*norm* (*f* (*Suc n*))))) ∗ *ereal* (*inverse r*)
by (*intro Limsup_ereal_mult_right*) *simp_all*
also have *limsup* (λ*n*. *ereal* (*norm* (*f n*) / (*norm* (*f* (*Suc n*))))) = *c*
by (*intro lim_imp_Limsup lim*) *simp*
finally have *l*: *?l* = *c* ∗ *ereal* (*inverse r*) **by** *simp*
from *r* **have** *l'*: *c* ∗ *ereal* (*inverse r*) < *1* **by** (*cases c*) (*simp_all add*: *field_simps*)
show ¬*summable* (λ*n*. *norm* (*f n* ∗ *of_real r*^*n*))
by (*rule ratio_test_divergence*) (*insert r nz l l'*, *auto elim*!: *eventually_mono*)
qed

lemma *conv_radius_ratio_limit_ereal_nonzero*:
fixes *f* :: *nat* ⇒ *'a* :: {*banach*,*real_normed_div_algebra*}
assumes *nz*: *c* ≠ *0*
assumes *lim*: (λ*n*. *ereal* (*norm* (*f n*) / *norm* (*f* (*Suc n*)))) ⟶ *c*

```
  shows   conv_radius f = c
proof (rule conv_radius_ratio_limit_ereal[OF _ lim], rule ccontr)
  assume ¬eventually (λn. f n ≠ 0) sequentially
  hence frequently (λn. f n = 0) sequentially by (simp add: frequently_def)
  hence frequently (λn. ereal (norm (f n) / norm (f (Suc n))) = 0) sequentially
    by (force elim!: frequently_elim1)
  hence c = 0 by (intro limit_frequently_eq[OF _ _ lim]) auto
  with nz show False by contradiction
qed

lemma conv_radius_ratio_limit:
  fixes f :: nat ⇒ 'a :: {banach,real_normed_div_algebra}
  assumes c' = ereal c
  assumes nz:  eventually (λn. f n ≠ 0) sequentially
  assumes lim: (λn. norm (f n) / norm (f (Suc n))) ⟶ c
  shows   conv_radius f = c'
  using assms by (intro conv_radius_ratio_limit_ereal) simp_all

lemma conv_radius_ratio_limit_nonzero:
  fixes f :: nat ⇒ 'a :: {banach,real_normed_div_algebra}
  assumes c' = ereal c
  assumes nz:  c ≠ 0
  assumes lim: (λn. norm (f n) / norm (f (Suc n))) ⟶ c
  shows   conv_radius f = c'
  using assms by (intro conv_radius_ratio_limit_ereal_nonzero) simp_all

lemma conv_radius_cmult_left:
  assumes c ≠ (0 :: 'a :: {banach, real_normed_div_algebra})
  shows   conv_radius (λn. c * f n) = conv_radius f
proof −
  have conv_radius (λn. c * f n) =
         inverse (limsup (λn. ereal (root n (norm (c * f n)))))
    unfolding conv_radius_def ..
  also have (λn. ereal (root n (norm (c * f n)))) =
              (λn. ereal (root n (norm c)) * ereal (root n (norm (f n))))
    by (rule ext) (auto simp: norm_mult real_root_mult)
  also have limsup ... = ereal 1 * limsup (λn. ereal (root n (norm (f n))))
   using assms by (intro ereal_limsup_lim_mult tendsto_ereal LIMSEQ_root_const)
auto
  also have inverse ... = conv_radius f by (simp add: conv_radius_def)
  finally show ?thesis .
qed

lemma conv_radius_cmult_right:
  assumes c ≠ (0 :: 'a :: {banach, real_normed_div_algebra})
  shows   conv_radius (λn. f n * c) = conv_radius f
proof −
  have conv_radius (λn. f n * c) = conv_radius (λn. c * f n)
    by (simp add: conv_radius_def norm_mult mult.commute)
```

with *conv_radius_cmult_left*[*OF assms*, *of f*] **show** *?thesis* **by** *simp*
qed

lemma *conv_radius_mult_power*:
assumes $c \neq (0 :: 'a :: \{real\_normed\_div\_algebra, banach\})$
shows *conv_radius* $(\lambda n.\ c \mathbin{\hat{}} n * f\ n) =$ *conv_radius f* / *ereal* (*norm c*)
proof −
have *limsup* $(\lambda n.$ *ereal* (*root n* (*norm* $(c \mathbin{\hat{}} n * f\ n)))) =$
limsup $(\lambda n.$ *ereal* (*norm c*) $*$ *ereal* (*root n* (*norm* (*f n*))))
by (*intro Limsup_eq eventually_mono* [*OF eventually_gt_at_top*[*of 0*::*nat*]])
(*auto simp*: *norm_mult norm_power real_root_mult real_root_power*)
also have $\ldots =$ *ereal* (*norm c*) $*$ *limsup* $(\lambda n.$ *ereal* (*root n* (*norm* (*f n*))))
using *assms* **by** (*subst Limsup_ereal_mult_left*[*symmetric*]) *simp_all*
finally have *A*: *limsup* $(\lambda n.$ *ereal* (*root n* (*norm* $(c \mathbin{\hat{}} n * f\ n)))) =$
ereal (*norm c*) $*$ *limsup* $(\lambda n.$ *ereal* (*root n* (*norm* (*f n*)))) **.**
show *?thesis* **using** *assms*
apply (*cases limsup* $(\lambda n.$ *ereal* (*root n* (*norm* (*f n*)))) $= 0$)
apply (*simp add*: *A conv_radius_def*)
apply (*unfold conv_radius_def A divide_ereal_def*, *simp add*: *mult.commute ereal_inverse_mult*)
done
qed

lemma *conv_radius_mult_power_right*:
assumes $c \neq (0 :: 'a :: \{real\_normed\_div\_algebra, banach\})$
shows *conv_radius* $(\lambda n.\ f\ n * c \mathbin{\hat{}} n) =$ *conv_radius f* / *ereal* (*norm c*)
using *conv_radius_mult_power*[*OF assms*, *of f*]
unfolding *conv_radius_def* **by** (*simp add*: *mult.commute norm_mult*)

lemma *conv_radius_divide_power*:
assumes $c \neq (0 :: 'a :: \{real\_normed\_div\_algebra, banach\})$
shows *conv_radius* $(\lambda n.\ f\ n$ / $c \mathbin{\hat{}} n) =$ *conv_radius f* $*$ *ereal* (*norm c*)
proof −
from *assms* **have** *inverse c* $\neq 0$ **by** *simp*
from *conv_radius_mult_power_right*[*OF this*, *of f*] **show** *?thesis*
by (*simp add*: *divide_inverse divide_ereal_def assms norm_inverse power_inverse*)
qed

lemma *conv_radius_add_ge*:
min (*conv_radius f*) (*conv_radius g*) ≤
conv_radius $(\lambda x.\ f\ x + g\ x :: 'a :: \{banach, real\_normed\_div\_algebra\})$
by (*rule conv_radius_geI_ex'*)
(*auto simp*: *algebra_simps intro*!: *summable_add summable_in_conv_radius*)

lemma *conv_radius_mult_ge*:
fixes *f g* :: *nat* ⇒ $('a :: \{banach, real\_normed\_div\_algebra\})$
shows *conv_radius* $(\lambda x. \sum i \leq x.\ f\ i * g\ (x - i)) \geq$ *min* (*conv_radius f*) (*conv_radius g*)

proof (*rule conv_radius_geI_ex'*)
fix *r* **assume** *r*: *r* > *0 ereal r* < *min* (*conv_radius f*) (*conv_radius g*)
from *r* **have** *summable* (λ*n*. (∑ *i*≤*n*. (*f i* ∗ *of_real r*^*i*) ∗ (*g* (*n* − *i*) ∗ *of_real* *r*^(*n* − *i*))))
by (*intro summable_Cauchy_product abs_summable_in_conv_radius*) *simp_all*
thus *summable* (λ*n*. (∑ *i*≤*n*. *f i* ∗ *g* (*n* − *i*)) ∗ *of_real r* ^ *n*)
by (*simp add*: *algebra_simps of_real_def power_add* [*symmetric*] *scaleR_sum_right*)
qed

lemma *le_conv_radius_iff*:
fixes *a* :: *nat* ⇒ *'a*::{*real_normed_div_algebra*,*banach*}
shows *r* ≤ *conv_radius a* ⟷ (∀ *x*. *norm* (*x*−ξ) < *r* ⟶ *summable* (λ*i*. *a i* ∗ (*x* − ξ) ^ *i*))
apply (*intro iffI allI impI summable_in_conv_radius conv_radius_geI_ex*)
apply (*meson less_ereal.simps*(*1*) *not_le order_trans*)
apply (*rule_tac x*=*of_real ra* **in** *exI*, *simp*)
apply (*metis abs_of_nonneg add_diff_cancel_left' less_eq_real_def norm_of_real*)
done

end

5.15 Uniform Limit and Uniform Convergence

theory *Uniform_Limit*
imports *Connected Summation_Tests Infinite_Sum*
begin

5.15.1 Definition

definition *uniformly_on* :: *'a set* ⇒ (*'a* ⇒ *'b*::*metric_space*) ⇒ (*'a* ⇒ *'b*) *filter*
where *uniformly_on S l* = (*INF e*∈{*0* <..}. *principal* {*f*. ∀ *x*∈*S*. *dist* (*f x*) (*l x*) < *e*})

abbreviation
uniform_limit S f l ≡ *filterlim f* (*uniformly_on S l*)

definition *uniformly_convergent_on* **where**
uniformly_convergent_on X f ⟷ (∃ *l*. *uniform_limit X f l sequentially*)

definition *uniformly_Cauchy_on* **where**
uniformly_Cauchy_on X f ⟷ (∀ *e*>*0*. ∃ *M*. ∀ *x*∈*X*. ∀ (*m*::*nat*)≥*M*. ∀ *n*≥*M*. *dist* (*f m x*) (*f n x*) < *e*)

proposition *uniform_limit_iff*:
uniform_limit S f l F ⟷ (∀ *e*>*0*. ∀ *F n in F*. ∀ *x*∈*S*. *dist* (*f n x*) (*l x*) < *e*)
unfolding *filterlim_iff uniformly_on_def*
by (*subst eventually_INF_base*)
(*fastforce*
simp: *eventually_principal uniformly_on_def*

intro: *bexI*[**where** *x*=*min a b* **for** *a b*]
elim: *eventually_mono*)+

lemma *uniform_limitD*:
uniform_limit S f l F $\Longrightarrow$ *e* > *0* $\Longrightarrow$ $\forall_F$ *n in F*. $\forall x{\in}S$. *dist* (*f n x*) (*l x*) < *e*
by (*simp add*: *uniform_limit_iff*)

lemma *uniform_limitI*:
($\bigwedge e$. *e* > *0* $\Longrightarrow$ $\forall_F$ *n in F*. $\forall x{\in}S$. *dist* (*f n x*) (*l x*) < *e*) $\Longrightarrow$ *uniform_limit S f l F*
by (*simp add*: *uniform_limit_iff*)

lemma *uniform_limit_sequentially_iff*:
uniform_limit S f l sequentially $\longleftrightarrow$ ($\forall e{>}0$. $\exists N$. $\forall n{\geq}N$. $\forall x \in S$. *dist* (*f n x*) (*l x*) < *e*)
unfolding *uniform_limit_iff eventually_sequentially* **..**

lemma *uniform_limit_at_iff*:
uniform_limit S f l (*at x*) $\longleftrightarrow$
($\forall e{>}0$. $\exists d{>}0$. $\forall z$. *0* < *dist z x* $\wedge$ *dist z x* < *d* $\longrightarrow$ ($\forall x{\in}S$. *dist* (*f z x*) (*l x*) < *e*))
unfolding *uniform_limit_iff eventually_at* **by** *simp*

lemma *uniform_limit_at_le_iff*:
uniform_limit S f l (*at x*) $\longleftrightarrow$
($\forall e{>}0$. $\exists d{>}0$. $\forall z$. *0* < *dist z x* $\wedge$ *dist z x* < *d* $\longrightarrow$ ($\forall x{\in}S$. *dist* (*f z x*) (*l x*) $\leq$ *e*))
unfolding *uniform_limit_iff eventually_at*
by (*fastforce dest*: *spec*[**where** *x* = *e* / *2* **for** *e*])

lemma *metric_uniform_limit_imp_uniform_limit*:
assumes *f*: *uniform_limit S f a F*
assumes *le*: *eventually* (λx. $\forall y{\in}S$. *dist* (*g x y*) (*b y*) $\leq$ *dist* (*f x y*) (*a y*)) *F*
shows *uniform_limit S g b F*
proof (*rule uniform_limitI*)
fix *e* :: *real* **assume** *0* < *e*
from *uniform_limitD*[*OF f this*] *le*
show $\forall_F$ *x in F*. $\forall y{\in}S$. *dist* (*g x y*) (*b y*) < *e*
by *eventually_elim force*
qed

5.15.2 Exchange limits

proposition *swap_uniform_limit*:
assumes *f*: $\forall_F$ *n in F*. (*f n* $\longrightarrow$ *g n*) (*at x within S*)
assumes *g*: (*g* $\longrightarrow$ *l*) *F*
assumes *uc*: *uniform_limit S f h F*
assumes $\neg$*trivial_limit F*
shows (*h* $\longrightarrow$ *l*) (*at x within S*)

proof (*rule tendstoI*)
fix *e* :: *real*
define *e′* **where** *e′* = *e*/*3*
assume *0* < *e*
then have *0* < *e′* **by** (*simp add*: *e′_def*)
from *uniform_limitD*[*OF uc* ‹*0* < *e′*›]
have ∀ *F n in F*. ∀ *x*∈*S*. *dist* (*h x*) (*f n x*) < *e′*
by (*simp add*: *dist_commute*)
moreover
from *f*
have ∀ *F n in F*. ∀ *F x in at x within S*. *dist* (*g n*) (*f n x*) < *e′*
by *eventually_elim* (*auto dest*!: *tendstoD*[*OF _* ‹*0* < *e′*›] *simp*: *dist_commute*)
moreover
from *tendstoD*[*OF g* ‹*0* < *e′*›] **have** ∀ *F x in F*. *dist l* (*g x*) < *e′*
by (*simp add*: *dist_commute*)
ultimately
have ∀ *F _ in F*. ∀ *F x in at x within S*. *dist* (*h x*) *l* < *e*
proof *eventually_elim*
case (*elim n*)
note *fh* = *elim*(*1*)
note *gl* = *elim*(*3*)
have ∀ *F x in at x within S*. *x* ∈ *S*
by (*auto simp*: *eventually_at_filter*)
with *elim*(*2*)
show *?case*
proof *eventually_elim*
case (*elim x*)
from *fh*[*rule_format*, *OF* ‹*x* ∈ *S*›] *elim*(*1*)
have *dist* (*h x*) (*g n*) < *e′* + *e′*
by (*rule dist_triangle_lt*[*OF add_strict_mono*])
from *dist_triangle_lt*[*OF add_strict_mono*, *OF this gl*]
show *?case* **by** (*simp add*: *e′_def*)
qed
qed
thus ∀ *F x in at x within S*. *dist* (*h x*) *l* < *e*
using *eventually_happens* **by** (*metis* ‹¬*trivial_limit F*›)
qed

5.15.3 Uniform limit theorem

lemma *tendsto_uniform_limitI*:
assumes *uniform_limit S f l F*
assumes *x* ∈ *S*
shows ((λ*y*. *f y x*) ⟶ *l x*) *F*
using *assms*
by (*auto intro*!: *tendstoI simp*: *eventually_mono dest*!: *uniform_limitD*)

theorem *uniform_limit_theorem*:
assumes *c*: ∀ *F n in F*. *continuous_on A* (*f n*)

assumes *ul*: *uniform_limit A f l F*
assumes ¬ *trivial_limit F*
shows *continuous_on A l*
unfolding *continuous_on_def*
proof *safe*
fix *x* **assume** *x* ∈ *A*
then have ∀$_F$ *n in F*. (*f n* ⟶ *f n x*) (*at x within A*) ((λ*n*. *f n x*) ⟶ *l x*) *F*
using *c ul*
by (*auto simp*: *continuous_on_def eventually_mono tendsto_uniform_limitI*)
then show (*l* ⟶ *l x*) (*at x within A*)
by (*rule swap_uniform_limit*) *fact+*
qed

lemma *uniformly_Cauchy_onI*:
assumes ⋀*e*. *e* > *0* ⟹ ∃ *M*. ∀ *x*∈*X*. ∀ *m*≥*M*. ∀ *n*≥*M*. *dist* (*f m x*) (*f n x*) < *e*
shows *uniformly_Cauchy_on X f*
using *assms* **unfolding** *uniformly_Cauchy_on_def* **by** *blast*

lemma *uniformly_Cauchy_onI'*:
assumes ⋀*e*. *e* > *0* ⟹ ∃ *M*. ∀ *x*∈*X*. ∀ *m*≥*M*. ∀ *n*>*m*. *dist* (*f m x*) (*f n x*) < *e*
shows *uniformly_Cauchy_on X f*
proof (*rule uniformly_Cauchy_onI*)
fix *e* :: *real* **assume** *e*: *e* > *0*
from *assms*[*OF this*] **obtain** *M*
where *M*: ⋀*x m n*. *x* ∈ *X* ⟹ *m* ≥ *M* ⟹ *n* > *m* ⟹ *dist* (*f m x*) (*f n x*) < *e* **by** *fast*
{
fix *x m n* **assume** *x*: *x* ∈ *X* **and** *m*: *m* ≥ *M* **and** *n*: *n* ≥ *M*
with *M*[*OF this*(*1*,*2*), *of n*] *M*[*OF this*(*1*,*3*), *of m*] *e* **have** *dist* (*f m x*) (*f n x*) < *e*
by (*cases m n rule*: *linorder_cases*) (*simp_all add*: *dist_commute*)
}
thus ∃ *M*. ∀ *x*∈*X*. ∀ *m*≥*M*. ∀ *n*≥*M*. *dist* (*f m x*) (*f n x*) < *e* **by** *fast*
qed

lemma *uniformly_Cauchy_imp_Cauchy*:
uniformly_Cauchy_on X f ⟹ *x* ∈ *X* ⟹ *Cauchy* (λ*n*. *f n x*)
unfolding *Cauchy_def uniformly_Cauchy_on_def* **by** *fast*

lemma *uniform_limit_cong*:
fixes *f g* :: *'a* ⇒ *'b* ⇒ (*'c* :: *metric_space*) **and** *h i* :: *'b* ⇒ *'c*
assumes *eventually* (λ*y*. ∀ *x*∈*X*. *f y x* = *g y x*) *F*
assumes ⋀*x*. *x* ∈ *X* ⟹ *h x* = *i x*
shows *uniform_limit X f h F* ⟷ *uniform_limit X g i F*
proof −
{
fix *f g* :: *'a* ⇒ *'b* ⇒ *'c* **and** *h i* :: *'b* ⇒ *'c*
assume *C*: *uniform_limit X f h F* **and** *A*: *eventually* (λ*y*. ∀ *x*∈*X*. *f y x* = *g y x*) *F*

```
    and B: ⋀x. x ∈ X ⟹ h x = i x
   {
    fix e ::real assume e > 0
    with C have eventually (λy. ∀x∈X. dist (f y x) (h x) < e) F
     unfolding uniform_limit_iff by blast
    with A have eventually (λy. ∀x∈X. dist (g y x) (i x) < e) F
     by eventually_elim (insert B, simp_all)
   }
   hence uniform_limit X g i F unfolding uniform_limit_iff by blast
  } note A = this
  show ?thesis by (rule iffI) (erule A; insert assms; simp add: eq_commute)+
qed

lemma uniform_limit_cong':
  fixes f g :: 'a ⇒ 'b ⇒ ('c :: metric_space) and h i :: 'b ⇒ 'c
  assumes ⋀y x. x ∈ X ⟹ f y x = g y x
  assumes ⋀x. x ∈ X ⟹ h x = i x
  shows   uniform_limit X f h F ⟷ uniform_limit X g i F
  using assms by (intro uniform_limit_cong always_eventually) blast+

lemma uniformly_convergent_cong:
  assumes eventually (λx. ∀y∈A. f x y = g x y) sequentially A = B
  shows uniformly_convergent_on A f ⟷ uniformly_convergent_on B g
  unfolding uniformly_convergent_on_def assms(2) [symmetric]
  by (intro iff_exI uniform_limit_cong eventually_mono [OF assms(1)]) auto

lemma uniformly_convergent_on_compose:
  assumes uniformly_convergent_on A f
  assumes filterlim g sequentially sequentially
  shows   uniformly_convergent_on A (λn. f (g n))
proof −
  from assms(1) obtain f' where uniform_limit A f f' sequentially
    by (auto simp: uniformly_convergent_on_def)
  hence uniform_limit A (λn. f (g n)) f' sequentially
    by (rule filterlim_compose) fact
  thus ?thesis
    by (auto simp: uniformly_convergent_on_def)
qed

lemma uniformly_convergent_uniform_limit_iff:
   uniformly_convergent_on X f ⟷ uniform_limit X f (λx. lim (λn. f n x))
sequentially
proof
  assume uniformly_convergent_on X f
  then obtain l where l: uniform_limit X f l sequentially
    unfolding uniformly_convergent_on_def by blast
  from l have uniform_limit X f (λx. lim (λn. f n x)) sequentially ⟷
                uniform_limit X f l sequentially
    by (intro uniform_limit_cong' limI tendsto_uniform_limitI[of f X l]) simp_all
```

also note *l*
finally show *uniform_limit X f* (λ*x*. *lim* (λ*n*. *f n x*)) *sequentially* **.**
qed (*auto simp*: *uniformly_convergent_on_def*)

lemma *uniformly_convergentI*: *uniform_limit X f l sequentially* ⟹ *uniformly_convergent_on X f*
unfolding *uniformly_convergent_on_def* **by** *blast*

lemma *uniformly_convergent_on_empty* [*iff*]: *uniformly_convergent_on* {} *f*
by (*simp add*: *uniformly_convergent_on_def uniform_limit_sequentially_iff*)

lemma *uniformly_convergent_on_const* [*simp*,*intro*]:
uniformly_convergent_on A (λ_. *c*)
by (*auto simp*: *uniformly_convergent_on_def uniform_limit_iff intro*!: *exI*[*of* _ *c*])

Cauchy-type criteria for uniform convergence.

lemma *Cauchy_uniformly_convergent*:
fixes *f* :: *nat* ⇒ *'a* ⇒ *'b* :: *complete_space*
assumes *uniformly_Cauchy_on X f*
shows *uniformly_convergent_on X f*
unfolding *uniformly_convergent_uniform_limit_iff uniform_limit_iff*
proof *safe*
let *?f* = λ*x*. *lim* (λ*n*. *f n x*)
fix *e* :: *real* **assume** *e*: *e* > *0*
hence *e*/*2* > *0* **by** *simp*
with *assms* **obtain** *N* **where** *N*: ⋀*x m n*. *x* ∈ *X* ⟹ *m* ≥ *N* ⟹ *n* ≥ *N* ⟹ *dist* (*f m x*) (*f n x*) < *e*/*2*
unfolding *uniformly_Cauchy_on_def* **by** *fast*
show *eventually* (λ*n*. ∀ *x*∈*X*. *dist* (*f n x*) (*?f x*) < *e*) *sequentially*
using *eventually_ge_at_top*[*of N*]
proof *eventually_elim*
fix *n* **assume** *n*: *n* ≥ *N*
show ∀ *x*∈*X*. *dist* (*f n x*) (*?f x*) < *e*
proof
fix *x* **assume** *x*: *x* ∈ *X*
with *assms* **have** (λ*n*. *f n x*) ⟶ *?f x*
by (*auto dest*!: *Cauchy_convergent uniformly_Cauchy_imp_Cauchy simp*: *convergent_LIMSEQ_iff*)
with ‹*e*/*2* > *0*› **have** *eventually* (λ*m*. *m* ≥ *N* ∧ *dist* (*f m x*) (*?f x*) < *e*/*2*) *sequentially*
by (*intro tendstoD eventually_conj eventually_ge_at_top*)
then obtain *m* **where** *m*: *m* ≥ *N dist* (*f m x*) (*?f x*) < *e*/*2*
unfolding *eventually_at_top_linorder* **by** *blast*
have *dist* (*f n x*) (*?f x*) ≤ *dist* (*f n x*) (*f m x*) + *dist* (*f m x*) (*?f x*)
by (*rule dist_triangle*)
also from *x n* **have** ... < *e*/*2* + *e*/*2* **by** (*intro add_strict_mono N m*)
finally show *dist* (*f n x*) (*?f x*) < *e* **by** *simp*
qed

qed
qed

lemma *uniformly_convergent_Cauchy*:
assumes *uniformly_convergent_on X f*
shows *uniformly_Cauchy_on X f*
proof (*rule uniformly_Cauchy_onI*)
fix *e*::*real* **assume** *e* > *0*
then have *0* < *e* / *2* **by** *simp*
with *assms*[*unfolded uniformly_convergent_on_def uniform_limit_sequentially_iff*]
obtain *l N* **where** *l*:*x* ∈ *X* ⟹ *n* ≥ *N* ⟹ *dist* (*f n x*) (*l x*) < *e* / *2* **for** *n x*
by *metis*
from *l l* **have** *x* ∈ *X* ⟹ *n* ≥ *N* ⟹ *m* ≥ *N* ⟹ *dist* (*f n x*) (*f m x*) < *e* **for** *n m x*
by (*rule dist_triangle_half_l*)
then show ∃ *M*. ∀ *x*∈*X*. ∀ *m*≥*M*. ∀ *n*≥*M*. *dist* (*f m x*) (*f n x*) < *e* **by** *blast*
qed

lemma *uniformly_convergent_eq_Cauchy*:
uniformly_convergent_on X f = *uniformly_Cauchy_on X f* **for** *f*::*nat* ⇒ *'b* ⇒ *'a*::*complete_space*
using *Cauchy_uniformly_convergent uniformly_convergent_Cauchy* **by** *blast*

lemma *uniformly_convergent_eq_cauchy*:
fixes *s*::*nat* ⇒ *'b* ⇒ *'a*::*complete_space*
shows
(∃ *l*. ∀ *e*>*0*. ∃ *N*. ∀ *n x*. *N* ≤ *n* ∧ *P x* ⟶ *dist*(*s n x*)(*l x*) < *e*) ⟷
(∀ *e*>*0*. ∃ *N*. ∀ *m n x*. *N* ≤ *m* ∧ *N* ≤ *n* ∧ *P x* ⟶ *dist* (*s m x*) (*s n x*) < *e*)
proof −
have ∗: (∀ *n*≥*N*. ∀ *x*. *Q x* ⟶ *R n x*) ⟷ (∀ *n x*. *N* ≤ *n* ∧ *Q x* ⟶ *R n x*)
(∀ *x*. *Q x* ⟶ (∀ *m*≥*N*. ∀ *n*≥*N*. *S n m x*)) ⟷ (∀ *m n x*. *N* ≤ *m* ∧ *N* ≤ *n* ∧ *Q x* ⟶ *S n m x*)
for *N*::*nat* **and** *Q*::*'b* ⇒ *bool* **and** *R S*
by *blast*+
show *?thesis*
using *uniformly_convergent_eq_Cauchy*[*of Collect P s*]
unfolding *uniformly_convergent_on_def uniformly_Cauchy_on_def uniform_limit_sequentially_iff*
by (*simp add*: ∗)
qed

lemma *uniformly_cauchy_imp_uniformly_convergent*:
fixes *s* :: *nat* ⇒ *'a* ⇒ *'b*::*complete_space*
assumes ∀ *e*>*0*.∃ *N*. ∀ *m* (*n*::*nat*) *x*. *N* ≤ *m* ∧ *N* ≤ *n* ∧ *P x* −−> *dist*(*s m x*)(*s n x*) < *e*
and ∀ *x*. *P x* −−> (∀ *e*>*0*. ∃ *N*. ∀ *n*. *N* ≤ *n* ⟶ *dist*(*s n x*)(*l x*) < *e*)
shows ∀ *e*>*0*. ∃ *N*. ∀ *n x*. *N* ≤ *n* ∧ *P x* ⟶ *dist*(*s n x*)(*l x*) < *e*
proof −
obtain *l'* **where** *l*:∀ *e*>*0*. ∃ *N*. ∀ *n x*. *N* ≤ *n* ∧ *P x* ⟶ *dist* (*s n x*) (*l' x*) < *e*
using *assms*(*1*) **unfolding** *uniformly_convergent_eq_cauchy*[*symmetric*] **by**

auto
moreover
{
fix *x*
assume *P x*
then have *l x = l' x*
using *tendsto_unique*[*OF trivial_limit_sequentially*, *of* λ*n. s n x l x l' x*]
using *l* **and** *assms*(*2*) **unfolding** *lim_sequentially* **by** *blast*
}
ultimately show *?thesis* **by** *auto*
qed

lemma *uniformly_convergent_on_sum_E*:
fixes ε::*real* **and** *f* :: *nat* ⇒ *'a* ⇒ *'b*::{*complete_space*,*real_normed_vector*}
assumes *uconv*: *uniformly_convergent_on K* (λ*n z.* ∑ *k*<*n. f k z*) **and** ε>*0*
obtains *N* **where** ⋀*m n z.* ⟦*N* ≤ *m*; *m* ≤ *n*; *z*∈*K*⟧ ⟹ *norm*(∑ *k*=*m*..<*n. f k z*) < ε
proof −
obtain *N* **where** *N*: ⋀*m n z.* ⟦*N* ≤ *m*; *N* ≤ *n*; *z*∈*K*⟧ ⟹ *dist* (∑ *k*<*m. f k z*) (∑ *k*<*n. f k z*) < ε
using *uconv* ‹ε>*0*› **unfolding** *uniformly_Cauchy_on_def uniformly_convergent_eq_Cauchy*
by *meson*
show *thesis*
proof
fix *m n z*
assume *N* ≤ *m m* ≤ *n z* ∈ *K*
moreover have (∑ *k* = *m*..<*n. f k z*) = (∑ *k*<*n. f k z*) − (∑ *k*<*m. f k z*)
by (*metis atLeast0LessThan le0 sum_diff_nat_ivl* ‹*m* ≤ *n*›)
ultimately show *norm*(∑ *k* = *m*..<*n. f k z*) < ε
using *N* **by** (*simp add*: *dist_norm*)
qed
qed

lemma *uniformly_convergent_on_sum_iff*:
fixes *f* :: *nat* ⇒ *'a* ⇒ *'b*::{*complete_space*,*real_normed_vector*}
shows *uniformly_convergent_on K* (λ*n z.* ∑ *k*<*n. f k z*)
⟷ (∀ε>*0*. ∃*N*. ∀*m n z. N*≤*m* ⟶ *m*≤*n* ⟶ *z*∈*K* ⟶ *norm* (∑ *k*=*m*..<*n. f k z*) < ε) (**is** *?lhs*=*?rhs*)
proof
assume *R*: *?rhs*
show *?lhs*
unfolding *uniformly_Cauchy_on_def uniformly_convergent_eq_Cauchy*
proof (*intro strip*)
fix ε::*real*
assume ε>*0*
then obtain *N* **where** ⋀*m n z.* ⟦*N* ≤ *m*; *m* ≤ *n*; *z*∈*K*⟧ ⟹ *norm*(∑ *k* = *m*..<*n. f k z*) < ε
using *R* **by** *blast*
then have ∀ *x*∈*K.* ∀ *m*≥*N.* ∀ *n*≥*m. norm* ((∑ *k*<*m. f k x*) − (∑ *k*<*n. f k x*))

$< \varepsilon$
by (*metis atLeast0LessThan le0 sum_diff_nat_ivl norm_minus_commute*)
then have $\forall x{\in}K.\ \forall m{\geq}N.\ \forall n{\geq}N.\ norm\ ((\sum k{<}m.\ f\ k\ x) - (\sum k{<}n.\ f\ k\ x))$
$< \varepsilon$
by (*metis linorder_le_cases norm_minus_commute*)
then show $\exists M.\ \forall x{\in}K.\ \forall m{\geq}M.\ \forall n{\geq}M.\ dist\ (\sum k{<}m.\ f\ k\ x)\ (\sum k{<}n.\ f\ k\ x)$
$< \varepsilon$
by (*metis dist_norm*)
qed
qed (*metis uniformly_convergent_on_sum_E*)

lemma *uniform_limit_suminf*:
fixes *f*:: *nat* ⇒ *'a*::{*metric_space*, *comm_monoid_add*} ⇒ *'a*
assumes *uniformly_convergent_on X* (λ*n x*. $\sum k{<}n.\ f\ k\ x$)
shows *uniform_limit X* (λ*n x*. $\sum k{<}n.\ f\ k\ x$) (λ*x*. $\sum k.\ f\ k\ x$) *sequentially*
proof −
obtain *S* **where** *S*: *uniform_limit X* (λ*n x*. $\sum k{<}n.\ f\ k\ x$) *S sequentially*
using *assms uniformly_convergent_on_def* **by** *blast*
then have $(\sum k.\ f\ k\ x) = S\ x$ **if** $x \in X$ **for** *x*
using *that sums_iff sums_def* **by** (*blast intro*: *tendsto_uniform_limitI* [*OF S*])
with *S* **show** *?thesis*
by (*simp cong*: *uniform_limit_cong'*)
qed

TODO: remove explicit formulations $(\exists l.\ \forall e{>}0.\ \exists N.\ \forall n\ x.\ N \leq n \wedge ?P\ x \longrightarrow dist\ (?s\ n\ x)\ (l\ x) < e) = (\forall e{>}0.\ \exists N.\ \forall m\ n\ x.\ N \leq m \wedge N \leq n \wedge ?P\ x \longrightarrow dist\ (?s\ m\ x)\ (?s\ n\ x) < e)$

$[\![\forall e{>}0.\ \exists N.\ \forall m\ n\ x.\ N \leq m \wedge N \leq n \wedge ?P\ x \longrightarrow dist\ (?s\ m\ x)\ (?s\ n\ x) < e;\ \forall x.\ ?P\ x \longrightarrow (\forall e{>}0.\ \exists N.\ \forall n{\geq}N.\ dist\ (?s\ n\ x)\ (?l\ x) < e)]\!] \Longrightarrow \forall e{>}0.\ \exists N.\ \forall n\ x.\ N \leq n \wedge ?P\ x \longrightarrow dist\ (?s\ n\ x)\ (?l\ x) < e$?!

lemma *uniformly_convergent_imp_convergent*:
uniformly_convergent_on X f ⟹ $x \in X$ ⟹ *convergent* (λ*n*. *f n x*)
unfolding *uniformly_convergent_on_def convergent_def*
by (*auto dest*: *tendsto_uniform_limitI*)

5.15.4 Comparison Test

lemma *uniformly_summable_comparison_test*:
fixes *f* :: *nat* ⇒ *'a* ⇒ *'b* :: *banach*
assumes *uniformly_convergent_on A* (λ*N x*. $\sum n{<}N.\ g\ n\ x$)
assumes $\bigwedge n\ x.\ x \in A \Longrightarrow norm\ (f\ n\ x) \leq g\ n\ x$
shows *uniformly_convergent_on A* (λ*N x*. $\sum n{<}N.\ f\ n\ x$)
proof −
have *uniformly_Cauchy_on A* (λ*N x*. $\sum n{<}N.\ f\ n\ x$)
proof (*rule uniformly_Cauchy_onI'*)
fix *e* :: *real* **assume** *e*: *e* > *0*
obtain *M* **where** *M*: $\bigwedge x\ m\ n.\ x \in A \Longrightarrow m \geq M \Longrightarrow n \geq M \Longrightarrow dist\ (\sum k{<}m.\ g\ k\ x)\ (\sum k{<}n.\ g\ k\ x) < e$

```
    using assms(1) e unfolding uniformly_convergent_eq_Cauchy uniformly_Cauchy_on_def
by metis
    show ∃M. ∀x∈A. ∀m≥M. ∀n>m. dist (∑k<m. f k x) (∑k<n. f k x) < e
    proof (rule exI[of _ M], safe)
      fix x m n assume xmn: x ∈ A m ≥ M m < n
      have nonneg: g k x ≥ 0 for k
        by (rule order.trans[OF _ assms(2)]) (use xmn in auto)
      have dist (∑k<m. f k x) (∑k<n. f k x) = norm (∑k∈{..<n}−{..<m}. f k
x)
      using xmn by (subst sum_diff) (auto simp: dist_norm norm_minus_commute)
      also have {..<n}−{..<m} = {m..<n}
        by auto
      also have norm (∑k∈{m..<n}. f k x) ≤ (∑k∈{m..<n}. norm (f k x))
        using norm_sum by blast
      also have ... ≤ (∑k∈{m..<n}. g k x)
        by (intro sum_mono assms xmn)
      also have ... = |∑k∈{m..<n}. g k x|
        by (subst abs_of_nonneg) (auto simp: nonneg intro!: sum_nonneg)
      also have {m..<n} = {..<n} − {..<m}
        by auto
      also have |∑k∈.... g k x| = dist (∑k<m. g k x) (∑k<n. g k x)
      using xmn by (subst sum_diff) (auto simp: abs_minus_commute dist_norm)
      also have ... < e
        by (rule M) (use xmn in auto)
      finally show dist (∑k<m. f k x) (∑k<n. f k x) < e .
    qed
  qed
  thus ?thesis
    unfolding uniformly_convergent_eq_Cauchy .
qed

lemma uniform_limit_compose_uniformly_continuous_on:
  fixes f :: 'a :: metric_space ⇒ 'b :: metric_space
  assumes lim: uniform_limit A g g' F
  assumes cont: uniformly_continuous_on B f
  assumes ev: eventually (λx. ∀y∈A. g x y ∈ B) F and closed B
  shows   uniform_limit A (λx y. f (g x y)) (λy. f (g' y)) F
proof (cases F = bot)
  case [simp]: False
  show ?thesis
    unfolding uniform_limit_iff
  proof safe
    fix e :: real assume e: e > 0

    have g'_in_B: g' y ∈ B if y ∈ A for y
    proof (rule Lim_in_closed_set)
      show eventually (λx. g x y ∈ B) F
        using ev by eventually_elim (use that in auto)
      show ((λx. g x y) ⟶ g' y) F
```

```
      using lim that by (metis tendsto_uniform_limitI)
    qed (use ‹closed B› in auto)

    obtain d where d: d > 0 ⋀x y. x ∈ B ⟹ y ∈ B ⟹ dist x y < d ⟹ dist
(f x) (f y) < e
      using e cont unfolding uniformly_continuous_on_def by metis
    from lim have eventually (λx. ∀ y∈A. dist (g x y) (g' y) < d) F
      unfolding uniform_limit_iff using ‹d > 0› by meson
    thus eventually (λx. ∀ y∈A. dist (f (g x y)) (f (g' y)) < e) F
      using assms(3)
    proof eventually_elim
      case (elim x)
      show ∀ y∈A. dist (f (g x y)) (f (g' y)) < e
      proof safe
        fix y assume y: y ∈ A
        show dist (f (g x y)) (f (g' y)) < e
        proof (rule d)
          show dist (g x y) (g' y) < d
            using elim y by blast
        qed (use y elim g'_in_B in auto)
      qed
    qed
  qed
qed (auto simp: filterlim_def)

lemma uniformly_convergent_on_compose_uniformly_continuous_on:
  fixes f :: 'a :: metric_space ⇒ 'b :: metric_space
  assumes lim: uniformly_convergent_on A g
  assumes cont: uniformly_continuous_on B f
  assumes ev: eventually (λx. ∀ y∈A. g x y ∈ B) sequentially and closed B
  shows   uniformly_convergent_on A (λx y. f (g x y))
proof −
  from lim obtain g' where g': uniform_limit A g g' sequentially
    by (auto simp: uniformly_convergent_on_def)
  thus ?thesis
   using uniform_limit_compose_uniformly_continuous_on[OF g' cont ev ‹closed
B›]
    by (auto simp: uniformly_convergent_on_def)
qed
```

5.15.5 Weierstrass M-Test

```
proposition Weierstrass_m_test_ev:
fixes f :: _ ⇒ _ ⇒ _ :: banach
assumes eventually (λn. ∀ x∈A. norm (f n x) ≤ M n) sequentially
assumes summable M
shows uniform_limit A (λn x. ∑ i<n. f i x) (λx. suminf (λi. f i x)) sequentially
proof (rule uniform_limitI)
  fix e :: real
```

```
  assume 0 < e
  from suminf_exist_split[OF ‹0 < e› ‹summable M›]
  have ∀F k in sequentially. norm (∑ i. M (i + k)) < e
    by (auto simp: eventually_sequentially)
  with eventually_all_ge_at_top[OF assms(1)]
    show ∀F n in sequentially. ∀x∈A. dist (∑ i<n. f i x) (∑ i. f i x) < e
  proof eventually_elim
    case (elim k)
    show ?case
    proof safe
      fix x assume x ∈ A
      have ∃N. ∀n≥N. norm (f n x) ≤ M n
        using assms(1) ‹x ∈ A› by (force simp: eventually_at_top_linorder)
      hence summable_norm_f: summable (λn. norm (f n x))
        by(rule summable_norm_comparison_test[OF _ ‹summable M›])
      have summable_f: summable (λn. f n x)
        using summable_norm_cancel[OF summable_norm_f] .
      have summable_norm_f_plus_k: summable (λi. norm (f (i + k) x))
        using summable_ignore_initial_segment[OF summable_norm_f]
        by auto
      have summable_M_plus_k: summable (λi. M (i + k))
        using summable_ignore_initial_segment[OF ‹summable M›]
        by auto

      have dist (∑ i<k. f i x) (∑ i. f i x) = norm ((∑ i. f i x) - (∑ i<k. f i x))
        using dist_norm dist_commute by (subst dist_commute)
      also have ... = norm (∑ i. f (i + k) x)
          using suminf_minus_initial_segment[OF summable_f, where k=k] by
simp
      also have ... ≤ (∑ i. norm (f (i + k) x))
        using summable_norm[OF summable_norm_f_plus_k] .
      also have ... ≤ (∑ i. M (i + k))
      by (rule suminf_le[OF _ summable_norm_f_plus_k summable_M_plus_k])
          (insert elim(1) ‹x ∈ A›, simp)
      finally show dist (∑ i<k. f i x) (∑ i. f i x) < e
        using elim by auto
    qed
  qed
qed
```

Alternative version, formulated as in HOL Light

```
corollary series_comparison_uniform:
  fixes f :: _ ⇒ nat ⇒ _ :: banach
  assumes g: summable g and le: ⋀n x. N ≤ n ∧ x ∈ A ⟹ norm(f x n) ≤ g n
    shows ∃l. ∀e. 0 < e ⟶ (∃N. ∀n x. N ≤ n ∧ x ∈ A ⟶ dist(sum (f x)
{..<n}) (l x) < e)
proof -
  have 1: ∀F n in sequentially. ∀x∈A. norm (f x n) ≤ g n
    using le eventually_sequentially by auto
```

show *?thesis*
apply (*rule_tac x*=($\lambda x.\ \sum i.\ f\ x\ i$) **in** *exI*)
apply (*metis* (*no_types*, *lifting*) *eventually_sequentially uniform_limitD* [*OF Weierstrass_m_test_ev* [*OF 1 g*]])
done
qed

corollary *Weierstrass_m_test*:
fixes $f :: \_ \Rightarrow \_ \Rightarrow \_ ::$ *banach*
assumes $\bigwedge n\ x.\ x \in A \Longrightarrow$ *norm* $(f\ n\ x) \leq M\ n$
assumes *summable M*
shows *uniform_limit A* ($\lambda n\ x.\ \sum i{<}n.\ f\ i\ x$) ($\lambda x.$ *suminf* ($\lambda i.\ f\ i\ x$)) *sequentially*
using *assms* **by** (*intro Weierstrass_m_test_ev always_eventually*) *auto*

corollary *Weierstrass_m_test'_ev*:
fixes $f :: \_ \Rightarrow \_ \Rightarrow \_ ::$ *banach*
assumes *eventually* ($\lambda n.\ \forall x{\in}A.$ *norm* $(f\ n\ x) \leq M\ n$) *sequentially summable M*
shows *uniformly_convergent_on A* ($\lambda n\ x.\ \sum i{<}n.\ f\ i\ x$)
unfolding *uniformly_convergent_on_def* **by** (*rule exI*, *rule Weierstrass_m_test_ev*[*OF assms*])

corollary *Weierstrass_m_test'*:
fixes $f :: \_ \Rightarrow \_ \Rightarrow \_ ::$ *banach*
assumes $\bigwedge n\ x.\ x \in A \Longrightarrow$ *norm* $(f\ n\ x) \leq M\ n$ *summable M*
shows *uniformly_convergent_on A* ($\lambda n\ x.\ \sum i{<}n.\ f\ i\ x$)
unfolding *uniformly_convergent_on_def* **by** (*rule exI*, *rule Weierstrass_m_test*[*OF assms*])

lemma *Weierstrass_m_test_general*:
fixes $f :: 'a \Rightarrow 'b \Rightarrow 'c ::$ *banach*
fixes $M :: 'a \Rightarrow$ *real*
assumes *norm_le*: $\bigwedge x\ y.\ x \in X \Longrightarrow y \in Y \Longrightarrow$ *norm* $(f\ x\ y) \leq M\ x$
assumes *summable*: *M summable_on X*
shows *uniform_limit Y* ($\lambda X\ y.\ \sum x{\in}X.\ f\ x\ y$) ($\lambda y.\ \sum_\infty x{\in}X.\ f\ x\ y$) (*finite_subsets_at_top X*)
proof (*rule uniform_limitI*)
fix $\varepsilon ::$ *real*
assume $\varepsilon > 0$
define *S* **where** $S = (\lambda y.\ \sum_\infty x{\in}X.\ f\ x\ y)$
have *S*: (($\lambda x.\ f\ x\ y$) *has_sum S y*) *X* **if** *y*: $y \in Y$ **for** *y*
unfolding *S_def*
proof (*rule has_sum_infsum*)
have ($\lambda x.$ *norm* $(f\ x\ y)$) *summable_on X*
by (*rule abs_summable_on_comparison_test'*[*OF summable norm_le*]) (*use y* **in** *auto*)
thus ($\lambda x.\ f\ x\ y$) *summable_on X*
by (*metis abs_summable_summable*)
qed

define T **where** $T = (\sum_\infty x{\in}X.\ M\ x)$
have T: $(M\ has\_sum\ T)\ X$
unfolding T_def **by** (*simp add*: *local.summable*)
have $M\_summable$: $M\ summable\_on\ X'$ **if** $X' \subseteq X$ **for** X'
using *local.summable summable_on_subset_banach that* **by** *blast*

have $f\_summable$: $(\lambda x.\ f\ x\ y)\ summable\_on\ X'$ **if** $X' \subseteq X\ y \in Y$ **for** $X'\ y$
using S *summable_on_def summable_on_subset_banach that* **by** *blast*
have *eventually* $(\lambda X'.\ dist\ (\sum x{\in}X'.\ M\ x)\ T < \varepsilon)$ (*finite_subsets_at_top* X)
using T ‹$\varepsilon > 0$› **unfolding** T_def *has_sum_def tendsto_iff* **by** *blast*
moreover have *eventually* $(\lambda X'.$ *finite* $X' \wedge X' \subseteq X)$ (*finite_subsets_at_top* X)
by (*simp add*: *eventually_finite_subsets_at_top_weakI*)
ultimately show $\forall_F\ X'$ *in finite_subsets_at_top* $X.\ \forall y{\in}Y.\ dist\ (\sum x{\in}X'.\ f\ x\ y)\ (\sum_\infty x{\in}X.\ f\ x\ y) < \varepsilon$
proof *eventually_elim*
case X': (*elim* X')
show $\forall y{\in}Y.\ dist\ (\sum x{\in}X'.\ f\ x\ y)\ (\sum_\infty x{\in}X.\ f\ x\ y) < \varepsilon$
proof
fix y **assume** y: $y \in Y$
have *1*: $((\lambda x.\ f\ x\ y)\ has\_sum\ (S\ y - (\sum x{\in}X'.\ f\ x\ y)))\ (X - X')$
using $X'\ y$ **by** (*intro has_sum_Diff S has_sum_finite*[*of* X'] $f\_summable$) *auto*
have *2*: $(M\ has\_sum\ (T - (\sum x{\in}X'.\ M\ x)))\ (X - X')$
using $X'\ y$ **by** (*intro has_sum_Diff T has_sum_finite*[*of* X'] $M\_summable$) *auto*
have $dist\ (\sum x{\in}X'.\ f\ x\ y)\ (\sum_\infty x{\in}X.\ f\ x\ y) = norm\ (S\ y - (\sum x{\in}X'.\ f\ x\ y))$
by (*simp add*: *dist_norm norm_minus_commute S_def*)
also have $norm\ (S\ y - (\sum x{\in}X'.\ f\ x\ y)) \leq (\sum_\infty x{\in}X{-}X'.\ M\ x)$
using *2* y **by** (*intro norm_infsum_le*[*OF 1 _ norm_le*]) (*auto simp*: *infsumI*)
also have $\ldots = T - (\sum x{\in}X'.\ M\ x)$
using *2* **by** (*auto simp*: *infsumI*)
also have $\ldots < \varepsilon$
using X' **by** (*simp add*: *dist_norm*)
finally show $dist\ (\sum x{\in}X'.\ f\ x\ y)\ (\sum_\infty x{\in}X.\ f\ x\ y) < \varepsilon$ **.**
qed
qed
qed

5.15.6 Structural introduction rules

lemma *uniform_limit_eq_rhs*: *uniform_limit* $X\ f\ l\ F \Longrightarrow l = m \Longrightarrow$ *uniform_limit* $X\ f\ m\ F$
by *simp*

named_theorems *uniform_limit_intros introduction rules for uniform_limit*
setup ‹
Global_Theory.add_thms_dynamic (***binding*** ‹*uniform_limit_eq_intros*›,

```
    fn context =>
    Named_Theorems.get (Context.proof_of context) named_theorems ‹uniform_limit_intros›
      |> map_filter (try (fn thm => @{thm uniform_limit_eq_rhs} OF [thm])))
›

lemma (in bounded_linear) uniform_limit[uniform_limit_intros]:
  assumes uniform_limit X g l F
  shows uniform_limit X (λa b. f (g a b)) (λa. f (l a)) F
proof (rule uniform_limitI)
  fix e::real
  from pos_bounded obtain K
    where K: ⋀x y. dist (f x) (f y) ≤ K * dist x y K > 0
    by (auto simp: ac_simps dist_norm diff[symmetric])
  assume 0 < e with ‹K > 0› have e / K > 0 by simp
  from uniform_limitD[OF assms this]
  show ∀⇩F n in F. ∀x∈X. dist (f (g n x)) (f (l x)) < e
    by eventually_elim (metis le_less_trans mult.commute pos_less_divide_eq K)
qed

lemma (in bounded_linear) uniformly_convergent_on:
  assumes uniformly_convergent_on A g
  shows   uniformly_convergent_on A (λx y. f (g x y))
proof −
  from assms obtain l where uniform_limit A g l sequentially
    unfolding uniformly_convergent_on_def by blast
  hence uniform_limit A (λx y. f (g x y)) (λx. f (l x)) sequentially
    by (rule uniform_limit)
  thus ?thesis unfolding uniformly_convergent_on_def by blast
qed

lemmas bounded_linear_uniform_limit_intros[uniform_limit_intros] =
  bounded_linear.uniform_limit[OF bounded_linear_Im]
  bounded_linear.uniform_limit[OF bounded_linear_Re]
  bounded_linear.uniform_limit[OF bounded_linear_cnj]
  bounded_linear.uniform_limit[OF bounded_linear_fst]
  bounded_linear.uniform_limit[OF bounded_linear_snd]
  bounded_linear.uniform_limit[OF bounded_linear_zero]
  bounded_linear.uniform_limit[OF bounded_linear_of_real]
  bounded_linear.uniform_limit[OF bounded_linear_inner_left]
  bounded_linear.uniform_limit[OF bounded_linear_inner_right]
  bounded_linear.uniform_limit[OF bounded_linear_divide]
  bounded_linear.uniform_limit[OF bounded_linear_scaleR_right]
  bounded_linear.uniform_limit[OF bounded_linear_mult_left]
  bounded_linear.uniform_limit[OF bounded_linear_mult_right]
  bounded_linear.uniform_limit[OF bounded_linear_scaleR_left]


lemmas uniform_limit_uminus[uniform_limit_intros] =
 bounded_linear.uniform_limit[OF bounded_linear_minus[OF bounded_linear_ident]]
```

lemma *uniform_limit_const*[*uniform_limit_intros*]: *uniform_limit S* (λ*x*. *c*) *c f*
by (*auto intro*!: *uniform_limitI*)

lemma *uniform_limit_add*[*uniform_limit_intros*]:
fixes *f g*::′*a* ⇒ ′*b* ⇒ ′*c*::*real_normed_vector*
assumes *uniform_limit X f l F*
assumes *uniform_limit X g m F*
shows *uniform_limit X* (λ*a b*. *f a b* + *g a b*) (λ*a*. *l a* + *m a*) *F*
proof (*rule uniform_limitI*)
fix *e*::*real*
assume *0* < *e*
hence *0* < *e* / *2* **by** *simp*
from
uniform_limitD[*OF assms*(*1*) *this*]
uniform_limitD[*OF assms*(*2*) *this*]
show ∀ *F n in F*. ∀ *x*∈*X*. *dist* (*f n x* + *g n x*) (*l x* + *m x*) < *e*
by *eventually_elim* (*simp add*: *dist_triangle_add_half*)
qed

lemma *uniform_limit_minus*[*uniform_limit_intros*]:
fixes *f g*::′*a* ⇒ ′*b* ⇒ ′*c*::*real_normed_vector*
assumes *uniform_limit X f l F*
assumes *uniform_limit X g m F*
shows *uniform_limit X* (λ*a b*. *f a b* − *g a b*) (λ*a*. *l a* − *m a*) *F*
unfolding *diff_conv_add_uminus*
by (*rule uniform_limit_intros assms*)+

lemma *uniform_limit_norm*[*uniform_limit_intros*]:
assumes *uniform_limit S g l f*
shows *uniform_limit S* (λ*x y*. *norm* (*g x y*)) (λ*x*. *norm* (*l x*)) *f*
using *assms*
apply (*rule metric_uniform_limit_imp_uniform_limit*)
apply (*rule eventuallyI*)
by (*metis dist_norm norm_triangle_ineq3 real_norm_def*)

lemma (**in** *bounded_bilinear*) *bounded_uniform_limit*[*uniform_limit_intros*]:
assumes *uniform_limit X f l F*
assumes *uniform_limit X g m F*
assumes *bounded* (*m* ‘ *X*)
assumes *bounded* (*l* ‘ *X*)
shows *uniform_limit X* (λ*a b*. *prod* (*f a b*) (*g a b*)) (λ*a*. *prod* (*l a*) (*m a*)) *F*
proof (*rule uniform_limitI*)
fix *e*::*real*
from *pos_bounded* **obtain** *K* **where** *K*:
0 < *K* ⋀*a b*. *norm* (*prod a b*) ≤ *norm a* ∗ *norm b* ∗ *K*
by *auto*
hence *sqrt* (*K*∗*4*) > *0* **by** *simp*

```
  from assms obtain Km Kl
  where Km: Km > 0 ⋀x. x ∈ X ⟹ norm (m x) ≤ Km
    and Kl: Kl > 0 ⋀x. x ∈ X ⟹ norm (l x) ≤ Kl
    by (auto simp: bounded_pos)
  hence K * Km * 4 > 0 K * Kl * 4 > 0
    using ‹K > 0›
    by simp_all
  assume 0 < e

  hence sqrt e > 0 by simp
  from uniform_limitD[OF assms(1) divide_pos_pos[OF this ‹sqrt (K*4) > 0›]]
    uniform_limitD[OF assms(2) divide_pos_pos[OF this ‹sqrt (K*4) > 0›]]
    uniform_limitD[OF assms(1) divide_pos_pos[OF ‹e > 0› ‹K * Km * 4 > 0›]]
    uniform_limitD[OF assms(2) divide_pos_pos[OF ‹e > 0› ‹K * Kl * 4 > 0›]]
  show ∀ F n in F. ∀x∈X. dist (prod (f n x) (g n x)) (prod (l x) (m x)) < e
  proof eventually_elim
    case (elim n)
    show ?case
    proof safe
      fix x assume x ∈ X
      have dist (prod (f n x) (g n x)) (prod (l x) (m x)) ≤
        norm (prod (f n x - l x) (g n x - m x)) +
        norm (prod (f n x - l x) (m x)) +
        norm (prod (l x) (g n x - m x))
      by (auto simp: dist_norm prod_diff_prod intro: order_trans norm_triangle_ineq
add_mono)
      also note K(2)[of f n x - l x g n x - m x]
      also from elim(1)[THEN bspec, OF ‹_ ∈ X›, unfolded dist_norm]
      have norm (f n x - l x) ≤ sqrt e / sqrt (K * 4)
        by simp
      also from elim(2)[THEN bspec, OF ‹_ ∈ X›, unfolded dist_norm]
      have norm (g n x - m x) ≤ sqrt e / sqrt (K * 4)
        by simp
      also have sqrt e / sqrt (K * 4) * (sqrt e / sqrt (K * 4)) * K = e / 4
        using ‹K > 0› ‹e > 0› by auto
      also note K(2)[of f n x - l x m x]
      also note K(2)[of l x g n x - m x]
      also from elim(3)[THEN bspec, OF ‹_ ∈ X›, unfolded dist_norm]
      have norm (f n x - l x) ≤ e / (K * Km * 4)
        by simp
      also from elim(4)[THEN bspec, OF ‹_ ∈ X›, unfolded dist_norm]
      have norm (g n x - m x) ≤ e / (K * Kl * 4)
        by simp
      also note Kl(2)[OF ‹_ ∈ X›]
      also note Km(2)[OF ‹_ ∈ X›]
      also have e / (K * Km * 4) * Km * K = e / 4
        using ‹K > 0› ‹Km > 0› by simp
      also have  Kl * (e / (K * Kl * 4)) * K = e / 4
        using ‹K > 0› ‹Kl > 0› by simp
```

```
    also have e / 4 + e / 4 + e / 4 < e using ‹e > 0› by simp
    finally show dist (prod (f n x) (g n x)) (prod (l x) (m x)) < e
      using ‹K > 0› ‹Kl > 0› ‹Km > 0› ‹e > 0›
      by (simp add: algebra_simps mult_right_mono divide_right_mono)
   qed
  qed
qed

lemmas bounded_bilinear_bounded_uniform_limit_intros[uniform_limit_intros]
=
 bounded_bilinear.bounded_uniform_limit[OF Inner_Product.bounded_bilinear_inner]
 bounded_bilinear.bounded_uniform_limit[OF Real_Vector_Spaces.bounded_bilinear_mult]
 bounded_bilinear.bounded_uniform_limit[OF Real_Vector_Spaces.bounded_bilinear_scaleR]

lemma uniform_lim_mult:
  fixes f :: 'a ⇒ 'b ⇒ 'c::real_normed_algebra
  assumes f: uniform_limit S f l F
      and g: uniform_limit S g m F
      and l: bounded (l ‘ S)
      and m: bounded (m ‘ S)
    shows uniform_limit S (λa b. f a b * g a b) (λa. l a * m a) F
  by (intro bounded_bilinear_bounded_uniform_limit_intros assms)

lemma uniform_lim_inverse:
  fixes f :: 'a ⇒ 'b ⇒ 'c::real_normed_field
  assumes f: uniform_limit S f l F
      and b: ⋀x. x ∈ S ⟹ b ≤ norm(l x)
      and b > 0
    shows uniform_limit S (λx y. inverse (f x y)) (inverse ∘ l) F
proof (rule uniform_limitI)
  fix e::real
  assume e > 0
  have lte: dist (inverse (f x y)) ((inverse ∘ l) y) < e
          if b/2 ≤ norm (f x y) norm (f x y - l y) < e * b^2 / 2 y ∈ S
          for x y
  proof -
    have [simp]: l y ≠ 0 f x y ≠ 0
      using ‹b > 0› that b [OF ‹y ∈ S›] by fastforce+
    have norm (l y - f x y) <  e * b^2 / 2
      by (metis norm_minus_commute that(2))
    also have ... ≤ e * (norm (f x y) * norm (l y))
      using ‹e > 0› that b [OF ‹y ∈ S›] apply (simp add: power2_eq_square)
      by (metis ‹b > 0› less_eq_real_def mult.left_commute mult_mono')
    finally show ?thesis
      by (auto simp: dist_norm field_split_simps norm_mult norm_divide)
  qed
  have ∀F n in F. ∀x∈S. dist (f n x) (l x) < b/2
    using uniform_limitD [OF f, of b/2] by (simp add: ‹b > 0›)
  then have ∀F x in F. ∀y∈S. b/2 ≤ norm (f x y)
```

```
    apply (rule eventually_mono)
    using b apply (simp only: dist_norm)
   by (metis (no_types, opaque_lifting) diff_zero dist_commute dist_norm norm_triangle_half_l
not_less)
  then have ∀⇩F x in F. ∀y∈S. b/2 ≤ norm (f x y) ∧ norm (f x y − l y) < e ∗
b^2 / 2
    apply (simp only: ball_conj_distrib dist_norm [symmetric])
    apply (rule eventually_conj, assumption)
      apply (rule uniform_limitD [OF f, of e ∗ b ^ 2 / 2])
    using ‹b > 0› ‹e > 0› by auto
  then show ∀⇩F x in F. ∀y∈S. dist (inverse (f x y)) ((inverse ∘ l) y) < e
    using lte by (force intro: eventually_mono)
qed

lemma uniform_lim_divide:
  fixes f :: 'a ⇒ 'b ⇒ 'c::real_normed_field
  assumes f: uniform_limit S f l F
      and g: uniform_limit S g m F
      and l: bounded (l ‘ S)
      and b: ⋀x. x ∈ S ⟹ b ≤ norm(m x)
      and b > 0
    shows uniform_limit S (λa b. f a b / g a b) (λa. l a / m a) F
proof −
  have m: bounded ((inverse ∘ m) ‘ S)
    using b ‹b > 0›
    apply (simp add: bounded_iff)
    by (metis le_imp_inverse_le norm_inverse)
  have uniform_limit S (λa b. f a b ∗ inverse (g a b))
        (λa. l a ∗ (inverse ∘ m) a) F
    by (rule uniform_lim_mult [OF f uniform_lim_inverse [OF g b ‹b > 0›] l m])
  then show ?thesis
    by (simp add: field_class.field_divide_inverse)
qed

lemma uniform_limit_null_comparison:
  assumes ∀⇩F x in F. ∀a∈S. norm (f x a) ≤ g x a
  assumes uniform_limit S g (λ_. 0) F
  shows uniform_limit S f (λ_. 0) F
  using assms(2)
proof (rule metric_uniform_limit_imp_uniform_limit)
  show ∀⇩F x in F. ∀y∈S. dist (f x y) 0 ≤ dist (g x y) 0
    using assms(1) by (rule eventually_mono) (force simp add: dist_norm)
qed

lemma uniform_limit_on_Un:
  uniform_limit I f g F ⟹ uniform_limit J f g F ⟹ uniform_limit (I ∪ J) f g
F
  by (auto intro!: uniform_limitI dest!: uniform_limitD elim: eventually_elim2)
```

lemma *uniform_limit_on_empty* [*iff*]:
uniform_limit {} *f g F*
by (*auto intro*!: *uniform_limitI*)

lemma *uniform_limit_on_UNION*:
assumes *finite S*
assumes $\bigwedge s.\ s \in S \Longrightarrow$ *uniform_limit* (*h s*) *f g F*
shows *uniform_limit* ($\bigcup$(*h* ‘ *S*)) *f g F*
using *assms*
by *induct* (*auto intro*: *uniform_limit_on_empty uniform_limit_on_Un*)

lemma *uniform_limit_on_Union*:
assumes *finite I*
assumes $\bigwedge J.\ J \in I \Longrightarrow$ *uniform_limit J f g F*
shows *uniform_limit* (*Union I*) *f g F*
by (*metis SUP_identity_eq assms uniform_limit_on_UNION*)

lemma *uniform_limit_on_subset*:
uniform_limit J f g F $\Longrightarrow$ $I \subseteq J \Longrightarrow$ *uniform_limit I f g F*
by (*auto intro*!: *uniform_limitI dest*!: *uniform_limitD intro*: *eventually_mono*)

lemma *uniform_limit_bounded*:
fixes *f*::*'i* ⇒ *'a*::*topological_space* ⇒ *'b*::*metric_space*
assumes *l*: *uniform_limit S f l F*
assumes *bnd*: $\forall_F$ *i in F. bounded* (*f i* ‘ *S*)
assumes $F \neq$ *bot*
shows *bounded* (*l* ‘ *S*)
proof −
from *l* **have** $\forall_F$ *n in F.* $\forall x{\in}S.$ *dist* (*l x*) (*f n x*) < *1*
by (*auto simp*: *uniform_limit_iff dist_commute dest*!: *spec*[**where** *x=1*])
with *bnd*
have $\forall_F$ *n in F.* $\exists M.$ $\forall x{\in}S.$ *dist undefined* (*l x*) ≤ *M* + *1*
by *eventually_elim*
(*auto intro*!: *order_trans*[*OF dist_triangle2 add_mono*] *intro*: *less_imp_le*
simp: *bounded_any_center*[**where** *a=undefined*])
then show *?thesis* **using** *assms*
by (*auto simp*: *bounded_any_center*[**where** *a=undefined*] *dest*!: *eventually_happens*)
qed

lemma *uniformly_convergent_add*:
uniformly_convergent_on A f $\Longrightarrow$ *uniformly_convergent_on A g*$\Longrightarrow$
uniformly_convergent_on A (λ*k x. f k x* + *g k x* :: *'a* :: {*real_normed_algebra*})
unfolding *uniformly_convergent_on_def* **by** (*blast dest*: *uniform_limit_add*)

lemma *uniformly_convergent_minus*:
uniformly_convergent_on A f $\Longrightarrow$ *uniformly_convergent_on A g*$\Longrightarrow$
uniformly_convergent_on A (λ*k x. f k x* − *g k x* :: *'a* :: {*real_normed_algebra*})
unfolding *uniformly_convergent_on_def* **by** (*blast dest*: *uniform_limit_minus*)

lemma *uniformly_convergent_mult*:
  uniformly_convergent_on A f $\Longrightarrow$
    uniformly_convergent_on A ($\lambda k\ x.\ c * f\ k\ x$ :: *'a* :: {*real_normed_algebra*})
  unfolding *uniformly_convergent_on_def*
  by (*blast dest*: *bounded_linear_uniform_limit_intros*(*13*))

5.15.7 Power series and uniform convergence

proposition *powser_uniformly_convergent*:
  fixes *a* :: *nat* $\Rightarrow$ *'a*::{*real_normed_div_algebra*,*banach*}
  assumes *r* < *conv_radius a*
  shows *uniformly_convergent_on* (*cball* ξ *r*) ($\lambda n\ x.\ \sum i<n.\ a\ i * (x - \xi)\ \hat{}\ i$)
proof (*cases 0* $\leq$ *r*)
  case *True*
  then have $*$: *summable* ($\lambda n.$ *norm* (*a n*) $*$ *r* $\hat{}$ *n*)
    using *abs_summable_in_conv_radius* [*of of_real r a*] *assms*
    by (*simp add*: *norm_mult norm_power*)
  show *?thesis*
    by (*simp add*: *Weierstrass_m_test'_ev* [*OF _ *] *norm_mult norm_power*
        mult_left_mono power_mono dist_norm norm_minus_commute)
next
  case *False* **then show** *?thesis* **by** (*simp add*: *not_le*)
qed

lemma *powser_uniform_limit*:
  fixes *a* :: *nat* $\Rightarrow$ *'a*::{*real_normed_div_algebra*,*banach*}
  assumes *r* < *conv_radius a*
  shows *uniform_limit* (*cball* ξ *r*) ($\lambda n\ x.\ \sum i<n.\ a\ i * (x - \xi)\ \hat{}\ i$) ($\lambda x.$ *suminf*
($\lambda i.\ a\ i * (x - \xi)\ \hat{}\ i$)) *sequentially*
using *powser_uniformly_convergent* [*OF assms*]
by (*simp add*: *Uniform_Limit.uniformly_convergent_uniform_limit_iff Series.suminf_eq_lim*)

lemma *powser_continuous_suminf*:
  fixes *a* :: *nat* $\Rightarrow$ *'a*::{*real_normed_div_algebra*,*banach*}
  assumes *r* < *conv_radius a*
  shows *continuous_on* (*cball* ξ *r*) ($\lambda x.$ *suminf* ($\lambda i.\ a\ i * (x - \xi)\ \hat{}\ i$))
apply (*rule uniform_limit_theorem* [*OF _ powser_uniform_limit*])
apply (*rule eventuallyI continuous_intros assms*)+
apply (*simp add*:)
done

lemma *powser_continuous_sums*:
  fixes *a* :: *nat* $\Rightarrow$ *'a*::{*real_normed_div_algebra*,*banach*}
  assumes *r*: *r* < *conv_radius a*
    and *sm*: $\bigwedge x.\ x \in$ *cball* ξ *r* $\Longrightarrow$ ($\lambda n.\ a\ n * (x - \xi)\ \hat{}\ n$) *sums* (*f x*)
  shows *continuous_on* (*cball* ξ *r*) *f*
apply (*rule continuous_on_cong* [*THEN iffD1*, *OF refl _ powser_continuous_suminf*
[*OF r*]])
using *sm sums_unique* **by** *fastforce*

lemmas *uniform_limit_subset_union* = *uniform_limit_on_subset*[*OF uniform_limit_on_Union*]

end

5.16 Bounded Linear Function

theory *Bounded_Linear_Function*
imports
Topology_Euclidean_Space
Operator_Norm
Uniform_Limit
Function_Topology

begin

lemma *onorm_componentwise*:
assumes *bounded_linear f*
shows *onorm f* $\leq$ ($\sum$ *i*$\in$*Basis*. *norm* (*f i*))
proof −
{
fix *i*::*'a*
assume *i* $\in$ *Basis*
hence *onorm* (λ*x*. (*x* $\cdot$ *i*) $*_R$ *f i*) $\leq$ *onorm* (λ*x*. (*x* $\cdot$ *i*)) $*$ *norm* (*f i*)
by (*auto intro*!: *onorm_scaleR_left_lemma bounded_linear_inner_left*)
also have ... $\leq$ *norm i* $*$ *norm* (*f i*)
by (*rule mult_right_mono*)
(*auto simp*: *ac_simps Cauchy_Schwarz_ineq2 intro*!: *onorm_le*)
finally have *onorm* (λ*x*. (*x* $\cdot$ *i*) $*_R$ *f i*) $\leq$ *norm* (*f i*) **using** ‹*i* $\in$ *Basis*›
by *simp*
} hence *onorm* (λ*x*. $\sum$ *i*$\in$*Basis*. (*x* $\cdot$ *i*) $*_R$ *f i*) $\leq$ ($\sum$ *i*$\in$*Basis*. *norm* (*f i*))
by (*auto intro*!: *order_trans*[*OF onorm_sum_le*] *bounded_linear_scaleR_const*
sum_mono bounded_linear_inner_left)
also have (λ*x*. $\sum$ *i*$\in$*Basis*. (*x* $\cdot$ *i*) $*_R$ *f i*) = (λ*x*. *f* ($\sum$ *i*$\in$*Basis*. (*x* $\cdot$ *i*) $*_R$ *i*))
by (*simp add*: *linear_sum bounded_linear.linear assms linear_simps*)
also have ... = *f*
by (*simp add*: *euclidean_representation*)
finally show *?thesis* **.**
qed

lemmas *onorm_componentwise_le* = *order_trans*[*OF onorm_componentwise*]

5.16.1 Intro rules for *bounded_linear*

named_theorems *bounded_linear_intros*

lemma *onorm_inner_left*:
assumes *bounded_linear r*
shows *onorm* (λ*x*. *r x* $\cdot$ *f*) $\leq$ *onorm r* $*$ *norm f*

proof (*rule onorm_bound*)
fix *x*
have *norm* (*r x* · *f*) ≤ *norm* (*r x*) ∗ *norm f*
by (*simp add*: *Cauchy_Schwarz_ineq2*)
also have ... ≤ *onorm r* ∗ *norm x* ∗ *norm f*
by (*intro mult_right_mono onorm assms norm_ge_zero*)
finally show *norm* (*r x* · *f*) ≤ *onorm r* ∗ *norm f* ∗ *norm x*
by (*simp add*: *ac_simps*)
qed (*intro mult_nonneg_nonneg norm_ge_zero onorm_pos_le assms*)

lemma *onorm_inner_right*:
assumes *bounded_linear r*
shows *onorm* (λ*x*. *f* · *r x*) ≤ *norm f* ∗ *onorm r*
apply (*subst inner_commute*)
apply (*rule onorm_inner_left*[*OF assms*, *THEN order_trans*])
apply *simp*
done

lemmas [*bounded_linear_intros*] =
bounded_linear_zero
bounded_linear_add
bounded_linear_const_mult
bounded_linear_mult_const
bounded_linear_scaleR_const
bounded_linear_const_scaleR
bounded_linear_ident
bounded_linear_sum
bounded_linear_Pair
bounded_linear_sub
bounded_linear_fst_comp
bounded_linear_snd_comp
bounded_linear_inner_left_comp
bounded_linear_inner_right_comp

5.16.2 declaration of derivative/continuous/tendsto introduction rules for bounded linear functions

attribute_setup *bounded_linear* =
‹*Scan.succeed* (*Thm.declaration_attribute* (*fn thm* =>
fold (*fn* (*r*, *s*) => *Named_Theorems.add_thm s* (*thm RS r*))
[
(@{*thm bounded_linear.has_derivative*}, ***named_theorems*** ‹*derivative_intros*›),
(@{*thm bounded_linear.tendsto*}, ***named_theorems*** ‹*tendsto_intros*›),
(@{*thm bounded_linear.continuous*}, ***named_theorems*** ‹*continuous_intros*›),
(@{*thm bounded_linear.continuous_on*}, ***named_theorems*** ‹*continuous_intros*›),
(@{*thm bounded_linear.uniformly_continuous_on*}, ***named_theorems*** ‹*continuous_intros*›),
(@{*thm bounded_linear_compose*}, ***named_theorems*** ‹*bounded_linear_intros*›)
]))›

attribute_setup *bounded_bilinear* =
‹*Scan.succeed* (*Thm.declaration_attribute* (*fn thm* =>
fold (*fn* (*r*, *s*) => *Named_Theorems.add_thm s* (*thm RS r*))
[
(@{*thm bounded_bilinear.FDERIV*}, ***named_theorems*** ‹*derivative_intros*›),
(@{*thm bounded_bilinear.tendsto*}, ***named_theorems*** ‹*tendsto_intros*›),
(@{*thm bounded_bilinear.continuous*}, ***named_theorems*** ‹*continuous_intros*›),
(@{*thm bounded_bilinear.continuous_on*}, ***named_theorems*** ‹*continuous_intros*›),
(@{*thm bounded_linear_compose*[*OF bounded_bilinear.bounded_linear_left*]},
named_theorems ‹*bounded_linear_intros*›),
(@{*thm bounded_linear_compose*[*OF bounded_bilinear.bounded_linear_right*]},
named_theorems ‹*bounded_linear_intros*›),
(@{*thm bounded_linear.uniformly_continuous_on*[*OF bounded_bilinear.bounded_linear_left*]},
named_theorems ‹*continuous_intros*›),
(@{*thm bounded_linear.uniformly_continuous_on*[*OF bounded_bilinear.bounded_linear_right*]},
named_theorems ‹*continuous_intros*›)
]))›

5.16.3 Type of bounded linear functions

typedef (**overloaded**) (*'a*, *'b*) *blinfun* ((_ ⇒$_L$ /_) [*22*, *21*] *21*) =
{*f*::*'a*::*real_normed_vector*⇒*'b*::*real_normed_vector*. *bounded_linear f*}
morphisms *blinfun_apply Blinfun*
by (*blast intro*: *bounded_linear_intros*)

declare [[*coercion*
blinfun_apply :: (*'a*::*real_normed_vector* ⇒$_L$ *'b*::*real_normed_vector*) ⇒ *'a* ⇒
'b]]

lemma *bounded_linear_blinfun_apply*[*bounded_linear_intros*]:
bounded_linear g ⟹ *bounded_linear* (λ*x*. *blinfun_apply f* (*g x*))
by (*metis blinfun_apply mem_Collect_eq bounded_linear_compose*)

setup_lifting *type_definition_blinfun*

lemma *blinfun_eqI*: (⋀*i*. *blinfun_apply x i* = *blinfun_apply y i*) ⟹ *x* = *y*
by *transfer auto*

lemma *bounded_linear_Blinfun_apply*: *bounded_linear f* ⟹ *blinfun_apply* (*Blinfun*
f) = *f*
by (*auto simp*: *Blinfun_inverse*)

5.16.4 Type class instantiations

instantiation *blinfun* :: (*real_normed_vector*, *real_normed_vector*) *real_normed_vector*
begin

lift_definition *norm_blinfun* :: *'a* ⇒$_L$ *'b* ⇒ *real* **is** *onorm* .

lift_definition *minus_blinfun* :: *'a* ⇒$_L$ *'b* ⇒ *'a* ⇒$_L$ *'b* ⇒ *'a* ⇒$_L$ *'b*

is $\lambda f\ g\ x.\ f\ x - g\ x$
by (*rule bounded_linear_sub*)

definition *dist_blinfun* :: $'a \Rightarrow_L\ 'b \Rightarrow\ 'a \Rightarrow_L\ 'b \Rightarrow$ *real*
where *dist_blinfun a b* = *norm* (*a* − *b*)

definition [*code del*]:
(*uniformity* :: (($'a \Rightarrow_L\ 'b$) × ($'a \Rightarrow_L\ 'b$)) *filter*) = (*INF e*∈{*0* <..}. *principal* {(*x*, *y*). *dist x y* < *e*})

definition *open_blinfun* :: ($'a \Rightarrow_L\ 'b$) *set* ⇒ *bool*
where [*code del*]: *open_blinfun S* = (∀ *x*∈*S*. ∀ $_F$ (*x'*, *y*) *in uniformity*. *x'* = *x* ⟶ *y* ∈ *S*)

lift_definition *uminus_blinfun* :: $'a \Rightarrow_L\ 'b \Rightarrow\ 'a \Rightarrow_L\ 'b$ **is** $\lambda f\ x.\ -\ f\ x$
by (*rule bounded_linear_minus*)

lift_definition *zero_blinfun* :: $'a \Rightarrow_L\ 'b$ **is** $\lambda x.\ 0$
by (*rule bounded_linear_zero*)

lift_definition *plus_blinfun* :: $'a \Rightarrow_L\ 'b \Rightarrow\ 'a \Rightarrow_L\ 'b \Rightarrow\ 'a \Rightarrow_L\ 'b$
is $\lambda f\ g\ x.\ f\ x + g\ x$
by (*metis bounded_linear_add*)

lift_definition *scaleR_blinfun*::*real* ⇒ $'a \Rightarrow_L\ 'b \Rightarrow\ 'a \Rightarrow_L\ 'b$ **is** $\lambda r\ f\ x.\ r *_R f\ x$
by (*metis bounded_linear_compose bounded_linear_scaleR_right*)

definition *sgn_blinfun* :: $'a \Rightarrow_L\ 'b \Rightarrow\ 'a \Rightarrow_L\ 'b$
where *sgn_blinfun x* = *scaleR* (*inverse* (*norm x*)) *x*

instance
apply *standard*
unfolding *dist_blinfun_def open_blinfun_def sgn_blinfun_def uniformity_blinfun_def*
apply (*rule refl* | (*transfer*, *force simp*: *onorm_triangle onorm_scaleR onorm_eq_0 algebra_simps*))+
done

end

declare *uniformity_Abort*[**where** $'a$=($'a$ :: *real_normed_vector*) $\Rightarrow_L$ ($'b$ :: *real_normed_vector*), *code*]

lemma *norm_blinfun_eqI*:
assumes *n* ≤ *norm* (*blinfun_apply f x*) / *norm x*
assumes ⋀*x*. *norm* (*blinfun_apply f x*) ≤ *n* ∗ *norm x*
assumes *0* ≤ *n*
shows *norm f* = *n*
by (*auto simp*: *norm_blinfun_def*
intro!: *antisym onorm_bound assms order_trans*[*OF* _ *le_onorm*]

bounded_linear_intros)

lemma *norm_blinfun*: *norm* (*blinfun_apply f x*) ≤ *norm f* ∗ *norm x*
by *transfer* (*rule onorm*)

lemma *norm_blinfun_bound*: *0* ≤ *b* ⟹ (⋀*x*. *norm* (*blinfun_apply f x*) ≤ *b* ∗ *norm x*) ⟹ *norm f* ≤ *b*
by *transfer* (*rule onorm_bound*)

lemma *bounded_bilinear_blinfun_apply*[*bounded_bilinear*]: *bounded_bilinear blinfun_apply*
proof
fix *f g*::′*a* ⇒$_L$ ′*b* **and** *a b*::′*a* **and** *r*::*real*
show (*f* + *g*) *a* = *f a* + *g a* (*r* ∗$_R$ *f*) *a* = *r* ∗$_R$ *f a*
by (*transfer*, *simp*)+
interpret *bounded_linear f* **for** *f*::′*a* ⇒$_L$ ′*b*
by (*auto intro*!: *bounded_linear_intros*)
show *f* (*a* + *b*) = *f a* + *f b f* (*r* ∗$_R$ *a*) = *r* ∗$_R$ *f a*
by (*simp_all add*: *add scaleR*)
show ∃ *K*. ∀ *a b*. *norm* (*blinfun_apply a b*) ≤ *norm a* ∗ *norm b* ∗ *K*
by (*auto intro*!: *exI*[**where** *x*=*1*] *norm_blinfun*)
qed

interpretation *blinfun*: *bounded_bilinear blinfun_apply*
by (*rule bounded_bilinear_blinfun_apply*)

lemmas *bounded_linear_apply_blinfun*[*intro*, *simp*] = *blinfun.bounded_linear_left*

declare *blinfun.zero_left* [*simp*] *blinfun.zero_right* [*simp*]

context *bounded_bilinear*
begin

named_theorems *bilinear_simps*

lemmas [*bilinear_simps*] =
add_left
add_right
diff_left
diff_right
minus_left
minus_right
scaleR_left
scaleR_right
zero_left
zero_right
sum_left
sum_right

end

```
instance blinfun :: (real_normed_vector, banach) banach
proof
  fix X::nat ⇒ 'a ⇒_L 'b
  assume Cauchy X
  {
    fix x::'a
    {
      fix x::'a
      assume norm x ≤ 1
      have Cauchy (λn. X n x)
      proof (rule CauchyI)
        fix e::real
        assume 0 < e
        from CauchyD[OF ‹Cauchy X› ‹0 < e›] obtain M
          where M: ⋀m n. m ≥ M ⟹ n ≥ M ⟹ norm (X m − X n) < e
          by auto
        show ∃ M. ∀ m≥M. ∀ n≥M. norm (X m x − X n x) < e
        proof (safe intro!: exI[where x=M])
          fix m n
          assume le: M ≤ m M ≤ n
          have norm (X m x − X n x) = norm ((X m − X n) x)
            by (simp add: blinfun.bilinear_simps)
          also have ... ≤ norm (X m − X n) * norm x
             by (rule norm_blinfun)
          also have ... ≤ norm (X m − X n) * 1
            using ‹norm x ≤ 1› norm_ge_zero by (rule mult_left_mono)
          also have ... = norm (X m − X n) by simp
          also have ... < e using le by fact
          finally show norm (X m x − X n x) < e .
        qed
      qed
      hence convergent (λn. X n x)
        by (metis Cauchy_convergent_iff)
    } note convergent_norm1 = this
    define y where y = x /_R norm x
    have y: norm y ≤ 1 and xy: x = norm x *_R y
      by (simp_all add: y_def inverse_eq_divide)
    have convergent (λn. norm x *_R X n y)
      by (intro bounded_bilinear.convergent[OF bounded_bilinear_scaleR] convergent_const
          convergent_norm1 y)
    also have (λn. norm x *_R X n y) = (λn. X n x)
      by (subst xy) (simp add: blinfun.bilinear_simps)
    finally have convergent (λn. X n x) .
  }
```

```
  then obtain v where v: ⋀x. (λn. X n x) ⟶ v x
    unfolding convergent_def
    by metis

  have Cauchy (λn. norm (X n))
  proof (rule CauchyI)
    fix e::real
    assume e > 0
    from CauchyD[OF ‹Cauchy X› ‹0 < e›] obtain M
      where M: ⋀m n. m ≥ M ⟹ n ≥ M ⟹ norm (X m − X n) < e
      by auto
    show ∃M. ∀m≥M. ∀n≥M. norm (norm (X m) − norm (X n)) < e
    proof (safe intro!: exI[where x=M])
      fix m n assume mn: m ≥ M n ≥ M
      have norm (norm (X m) − norm (X n)) ≤ norm (X m − X n)
        by (metis norm_triangle_ineq3 real_norm_def)
      also have ... < e using mn by fact
      finally show norm (norm (X m) − norm (X n)) < e .
    qed
  qed
  then obtain K where K: (λn. norm (X n)) ⟶ K
    unfolding Cauchy_convergent_iff convergent_def
    by metis

  have bounded_linear v
  proof
    fix x y and r::real
   from tendsto_add[OF v[of x] v [of y]] v[of x + y, unfolded blinfun.bilinear_simps]
        tendsto_scaleR[OF tendsto_const[of r] v[of x]] v[of r *R x, unfolded blin-
fun.bilinear_simps]
    show v (x + y) = v x + v y v (r *R x) = r *R v x
      by (metis (poly_guards_query) LIMSEQ_unique)+
    show ∃K. ∀x. norm (v x) ≤ norm x * K
    proof (safe intro!: exI[where x=K])
      fix x
      have norm (v x) ≤ K * norm x
      by (rule tendsto_le[OF _ tendsto_mult[OF K tendsto_const] tendsto_norm[OF
v]])
         (auto simp: norm_blinfun)
      thus norm (v x) ≤ norm x * K
        by (simp add: ac_simps)
    qed
  qed
  hence Bv: ⋀x. (λn. X n x) ⟶ Blinfun v x
    by (auto simp: bounded_linear_Blinfun_apply v)

  have X ⟶ Blinfun v
  proof (rule LIMSEQ_I)
    fix r::real assume r > 0
```

```
    define r' where r' = r / 2
    have 0 < r' r' < r using ‹r > 0› by (simp_all add: r'_def)
    from CauchyD[OF ‹Cauchy X› ‹r' > 0›]
    obtain M where M: ⋀m n. m ≥ M ⟹ n ≥ M ⟹ norm (X m − X n) < r'
      by metis
    show ∃no. ∀n≥no. norm (X n − Blinfun v) < r
    proof (safe intro!: exI[where x=M])
      fix n assume n: M ≤ n
      have norm (X n − Blinfun v) ≤ r'
      proof (rule norm_blinfun_bound)
        fix x
        have eventually (λm. m ≥ M) sequentially
          by (metis eventually_ge_at_top)
          hence ev_le: eventually (λm. norm (X n x − X m x) ≤ r' * norm x)
sequentially
        proof eventually_elim
          case (elim m)
          have norm (X n x − X m x) = norm ((X n − X m) x)
            by (simp add: blinfun.bilinear_simps)
          also have ... ≤ norm ((X n − X m)) * norm x
            by (rule norm_blinfun)
          also have ... ≤ r' * norm x
            using M[OF n elim] by (simp add: mult_right_mono)
          finally show ?case .
        qed
         have tendsto_v: (λm. norm (X n x − X m x)) ⟶ norm (X n x −
Blinfun v x)
          by (auto intro!: tendsto_intros Bv)
        show norm ((X n − Blinfun v) x) ≤ r' * norm x
       by (auto intro!: tendsto_upperbound tendsto_v ev_le simp: blinfun.bilinear_simps)
      qed (simp add: ‹0 < r'› less_imp_le)
      thus norm (X n − Blinfun v) < r
        by (metis ‹r' < r› le_less_trans)
    qed
  qed
  thus convergent X
    by (rule convergentI)
qed
```

5.16.5 On Euclidean Space

```
lemma Zfun_sum:
  assumes finite s
  assumes f: ⋀i. i ∈ s ⟹ Zfun (f i) F
  shows Zfun (λx. sum (λi. f i x) s) F
  using assms by induct (auto intro!: Zfun_zero Zfun_add)

lemma norm_blinfun_euclidean_le:
  fixes a::'a::euclidean_space ⇒_L 'b::real_normed_vector
```

shows *norm a* ≤ *sum* (λ*x*. *norm* (*a x*)) *Basis*
apply (*rule norm_blinfun_bound*)
apply (*simp add*: *sum_nonneg*)
apply (*subst euclidean_representation*[*symmetric*, **where** *'a*=*'a*])
apply (*simp only*: *blinfun.bilinear_simps sum_distrib_right*)
apply (*rule order.trans*[*OF norm_sum sum_mono*])
apply (*simp add*: *abs_mult mult_right_mono ac_simps Basis_le_norm*)
done

lemma *tendsto_componentwise1*:
fixes *a*::*'a*::*euclidean_space* ⇒$_L$ *'b*::*real_normed_vector*
and *b*::*'c* ⇒ *'a* ⇒$_L$ *'b*
assumes (⋀*j*. *j* ∈ *Basis* ⟹ ((λ*n*. *b n j*) ⟶ *a j*) *F*)
shows (*b* ⟶ *a*) *F*
proof −
have ⋀*j*. *j* ∈ *Basis* ⟹ *Zfun* (λ*x*. *norm* (*b x j* − *a j*)) *F*
using *assms* **unfolding** *tendsto_Zfun_iff Zfun_norm_iff* **.**
hence *Zfun* (λ*x*. ∑*j*∈*Basis*. *norm* (*b x j* − *a j*)) *F*
by (*auto intro*!: *Zfun_sum*)
thus *?thesis*
unfolding *tendsto_Zfun_iff*
by (*rule Zfun_le*)
(*auto intro*!: *order_trans*[*OF norm_blinfun_euclidean_le*] *simp*: *blinfun.bilinear_simps*)
qed

lift_definition
blinfun_of_matrix::(*'b*::*euclidean_space* ⇒ *'a*::*euclidean_space* ⇒ *real*) ⇒ *'a* ⇒$_L$ *'b*
is λ*a x*. ∑*i*∈*Basis*. ∑*j*∈*Basis*. ((*x* ∙ *j*) ∗ *a i j*) ∗$_R$ *i*
by (*intro bounded_linear_intros*)

lemma *blinfun_of_matrix_works*:
fixes *f*::*'a*::*euclidean_space* ⇒$_L$ *'b*::*euclidean_space*
shows *blinfun_of_matrix* (λ*i j*. (*f j*) ∙ *i*) = *f*
proof (*transfer*, *rule*, *rule euclidean_eqI*)
fix *f*::*'a* ⇒ *'b* **and** *x*::*'a* **and** *b*::*'b* **assume** *bounded_linear f* **and** *b*: *b* ∈ *Basis*
then interpret *bounded_linear f* **by** *simp*
have (∑*j*∈*Basis*. ∑*i*∈*Basis*. (*x* ∙ *i* ∗ (*f i* ∙ *j*)) ∗$_R$ *j*) ∙ *b*
= (∑*j*∈*Basis*. *if j* = *b then* (∑*i*∈*Basis*. (*x* ∙ *i* ∗ (*f i* ∙ *j*))) *else* 0)
using *b*
by (*simp add*: *inner_sum_left inner_Basis if_distrib cong*: *if_cong*) (*simp add*: *sum.swap*)
also have ... = (∑*i*∈*Basis*. (*x* ∙ *i* ∗ (*f i* ∙ *b*)))
using *b* **by** (*simp*)
also have ... = *f x* ∙ *b*
by (*metis* (*mono_tags*, *lifting*) *Linear_Algebra.linear_componentwise linear_axioms*)
finally show (∑*j*∈*Basis*. ∑*i*∈*Basis*. (*x* ∙ *i* ∗ (*f i* ∙ *j*)) ∗$_R$ *j*) ∙ *b* = *f x* ∙ *b* **.**
qed

lemma *blinfun_of_matrix_apply*:
blinfun_of_matrix a x = (∑ *i*∈*Basis*. ∑ *j*∈*Basis*. ((*x* ∙ *j*) ∗ *a i j*) ∗$_R$ *i*)
by *transfer simp*

lemma *blinfun_of_matrix_minus*: *blinfun_of_matrix x* − *blinfun_of_matrix y* = *blinfun_of_matrix* (*x* − *y*)
by *transfer* (*auto simp*: *algebra_simps sum_subtractf*)

lemma *norm_blinfun_of_matrix*:
norm (*blinfun_of_matrix a*) ≤ (∑ *i*∈*Basis*. ∑ *j*∈*Basis*. |*a i j*|)
apply (*rule norm_blinfun_bound*)
apply (*simp add*: *sum_nonneg*)
apply (*simp only*: *blinfun_of_matrix_apply sum_distrib_right*)
apply (*rule order_trans*[*OF norm_sum sum_mono*])
apply (*rule order_trans*[*OF norm_sum sum_mono*])
apply (*simp add*: *abs_mult mult_right_mono ac_simps Basis_le_norm*)
done

lemma *tendsto_blinfun_of_matrix*:
assumes ⋀*i j*. *i* ∈ *Basis* ⟹ *j* ∈ *Basis* ⟹ ((λ*n*. *b n i j*) ⟶ *a i j*) *F*
shows ((λ*n*. *blinfun_of_matrix* (*b n*)) ⟶ *blinfun_of_matrix a*) *F*
proof −
have ⋀*i j*. *i* ∈ *Basis* ⟹ *j* ∈ *Basis* ⟹ *Zfun* (λ*x*. *norm* (*b x i j* − *a i j*)) *F*
using *assms* **unfolding** *tendsto_Zfun_iff Zfun_norm_iff* .
hence *Zfun* (λ*x*. (∑ *i*∈*Basis*. ∑ *j*∈*Basis*. |*b x i j* − *a i j*|)) *F*
by (*auto intro*!: *Zfun_sum*)
thus *?thesis*
unfolding *tendsto_Zfun_iff blinfun_of_matrix_minus*
by (*rule Zfun_le*) (*auto intro*!: *order_trans*[*OF norm_blinfun_of_matrix*])
qed

lemma *tendsto_componentwise*:
fixes *a*::′*a*::*euclidean_space* ⇒$_L$ ′*b*::*euclidean_space*
and *b*::′*c* ⇒ ′*a* ⇒$_L$ ′*b*
shows (⋀*i j*. *i* ∈ *Basis* ⟹ *j* ∈ *Basis* ⟹ ((λ*n*. *b n j* ∙ *i*) ⟶ *a j* ∙ *i*) *F*) ⟹ (*b* ⟶ *a*) *F*
apply (*subst blinfun_of_matrix_works*[*of a*, *symmetric*])
apply (*subst blinfun_of_matrix_works*[*of b x* **for** *x*, *symmetric*, *abs_def*])
by (*rule tendsto_blinfun_of_matrix*)

lemma
continuous_blinfun_componentwiseI:
fixes *f*:: ′*b*::*t2_space* ⇒ ′*a*::*euclidean_space* ⇒$_L$ ′*c*::*euclidean_space*
assumes ⋀*i j*. *i* ∈ *Basis* ⟹ *j* ∈ *Basis* ⟹ *continuous F* (λ*x*. (*f x*) *j* ∙ *i*)
shows *continuous F f*
using *assms* **by** (*auto simp*: *continuous_def intro*!: *tendsto_componentwise*)

lemma
continuous_blinfun_componentwiseI1:

fixes f:: $'b$::*t2_space* $\Rightarrow$ $'a$::*euclidean_space* $\Rightarrow_L$ $'c$::*real_normed_vector*
assumes $\bigwedge i.\ i \in Basis \Longrightarrow$ *continuous* F $(\lambda x.\ f\ x\ i)$
shows *continuous* F f
using *assms* **by** (*auto simp*: *continuous_def intro*!: *tendsto_componentwise1*)

lemma
continuous_on_blinfun_componentwise:
fixes f:: $'d$::*t2_space* $\Rightarrow$ $'e$::*euclidean_space* $\Rightarrow_L$ $'f$::*real_normed_vector*
assumes $\bigwedge i.\ i \in Basis \Longrightarrow$ *continuous_on* s $(\lambda x.\ f\ x\ i)$
shows *continuous_on* s f
using *assms*
by (*auto intro*!: *continuous_at_imp_continuous_on intro*!: *tendsto_componentwise1*
simp: *continuous_on_eq_continuous_within continuous_def*)

lemma *bounded_linear_blinfun_matrix*: *bounded_linear* $(\lambda x.\ (x$::_ $\Rightarrow_L$ _$)\ j \cdot i)$
by (*auto intro*!: *bounded_linearI'* *bounded_linear_intros*)

lemma *continuous_blinfun_matrix*:
fixes f:: $'b$::*t2_space* $\Rightarrow$ $'a$::*real_normed_vector* $\Rightarrow_L$ $'c$::*real_inner*
assumes *continuous* F f
shows *continuous* F $(\lambda x.\ (f\ x)\ j \cdot i)$
by (*rule bounded_linear.continuous*[*OF bounded_linear_blinfun_matrix assms*])

lemma *continuous_on_blinfun_matrix*:
fixes f::$'a$::*t2_space* $\Rightarrow$ $'b$::*real_normed_vector* $\Rightarrow_L$ $'c$::*real_inner*
assumes *continuous_on* S f
shows *continuous_on* S $(\lambda x.\ (f\ x)\ j \cdot i)$
using *assms*
by (*auto simp*: *continuous_on_eq_continuous_within continuous_blinfun_matrix*)

lemma *continuous_on_blinfun_of_matrix*[*continuous_intros*]:
assumes $\bigwedge i\ j.\ i \in Basis \Longrightarrow j \in Basis \Longrightarrow$ *continuous_on* S $(\lambda s.\ g\ s\ i\ j)$
shows *continuous_on* S $(\lambda s.\ blinfun\_of\_matrix\ (g\ s))$
using *assms*
by (*auto simp*: *continuous_on intro*!: *tendsto_blinfun_of_matrix*)

lemma *mult_if_delta*:
$(if\ P\ then\ (1$::$'a$::*comm_semiring_1*$)\ else\ 0) * q = (if\ P\ then\ q\ else\ 0)$
by *auto*

lemma *compact_blinfun_lemma*:
fixes f :: *nat* $\Rightarrow$ $'a$::*euclidean_space* $\Rightarrow_L$ $'b$::*euclidean_space*
assumes *bounded* (*range* f)
shows $\forall\ d \subseteq Basis.\ \exists\ l$::$'a \Rightarrow_L\ 'b.\ \exists\ r$::*nat*$\Rightarrow$*nat*.
strict_mono $r \wedge (\forall\ e{>}0.$ *eventually* $(\lambda n.\ \forall\ i{\in}d.$ *dist* $(f\ (r\ n)\ i)\ (l\ i) < e)$ *sequentially*)
by (*rule compact_lemma_general*[**where** *unproj* $= \lambda e.$ *blinfun_of_matrix* $(\lambda i\ j.\ e\ j \cdot i)$])
(*auto intro*!: *euclidean_eqI*[**where** $'a{=}'b$] *bounded_linear_image assms*

simp: *blinfun_of_matrix_works blinfun_of_matrix_apply inner_Basis mult_if_delta sum.delta'*
scaleR_sum_left[*symmetric*])

lemma *blinfun_euclidean_eqI*: $(\bigwedge i.\ i \in Basis \Longrightarrow blinfun\_apply\ x\ i = blinfun\_apply\ y\ i) \Longrightarrow x = y$
apply (*auto intro*!: *blinfun_eqI*)
apply (*subst* (*2*) *euclidean_representation*[*symmetric*, **where** *'a='a*])
apply (*subst* (*1*) *euclidean_representation*[*symmetric*, **where** *'a='a*])
apply (*simp add*: *blinfun.bilinear_simps*)
done

lemma *Blinfun_eq_matrix*: *bounded_linear f* $\Longrightarrow$ *Blinfun f = blinfun_of_matrix* $(\lambda i\ j.\ f\ j \cdot i)$
by (*intro blinfun_euclidean_eqI*)
(*auto simp*: *blinfun_of_matrix_apply bounded_linear_Blinfun_apply inner_Basis if_distrib*
if_distribR sum.delta' euclidean_representation
cong: *if_cong*)

TODO: generalize (via *compact_cball*)?

instance *blinfun* :: (*euclidean_space*, *euclidean_space*) *heine_borel*
proof
fix *f* :: *nat* $\Rightarrow$ *'a* $\Rightarrow_L$ *'b*
assume *f*: *bounded* (*range f*)
then obtain *l*::*'a* $\Rightarrow_L$ *'b* **and** *r* **where** *r*: *strict_mono r*
and *l*: $\forall\, e{>}0.$ *eventually* $(\lambda n.\ \forall\, i{\in}Basis.\ dist\ (f\ (r\ n)\ i)\ (l\ i) < e)$ *sequentially*
using *compact_blinfun_lemma* [*OF f*] **by** *blast*
{
fix *e*::*real*
let *?d* = *real_of_nat DIM*(*'a*) $*$ *real_of_nat DIM*(*'b*)
assume *e* > *0*
hence *e* / *?d* > *0* **by** (*simp*)
with *l* **have** *eventually* $(\lambda n.\ \forall\, i{\in}Basis.\ dist\ (f\ (r\ n)\ i)\ (l\ i) < e\ /\ ?d)$ *sequentially*
by *simp*
moreover
{
fix *n*
assume *n*: $\forall\, i{\in}Basis.\ dist\ (f\ (r\ n)\ i)\ (l\ i) < e\ /\ ?d$
have *norm* $(f\ (r\ n) - l) = norm\ (blinfun\_of\_matrix\ (\lambda i\ j.\ (f\ (r\ n) - l)\ j \cdot i))$
unfolding *blinfun_of_matrix_works* **..**
also note *norm_blinfun_of_matrix*
also have $(\sum i{\in}Basis.\ \sum j{\in}Basis.\ |(f\ (r\ n) - l)\ j \cdot i|) <$
$(\sum i{\in}(Basis{::}'b\ set).\ e\ /\ real\_of\_nat\ DIM('b))$
proof (*rule sum_strict_mono*)
fix *i*::*'b* **assume** *i*: *i* $\in$ *Basis*
have $(\sum j{::}'a{\in}Basis.\ |(f\ (r\ n) - l)\ j \cdot i|) < (\sum j{::}'a{\in}Basis.\ e\ /\ ?d)$
proof (*rule sum_strict_mono*)

```
      fix j::'a assume j: j ∈ Basis
      have |(f (r n) − l) j • i| ≤ norm ((f (r n) − l) j)
        by (simp add: Basis_le_norm i)
      also have ... < e / ?d
        using n i j by (auto simp: dist_norm blinfun.bilinear_simps)
      finally show |(f (r n) − l) j • i| < e / ?d by simp
    qed simp_all
    also have ... ≤ e / real_of_nat DIM('b)
      by simp
    finally show (∑j∈Basis. |(f (r n) − l) j • i|) < e / real_of_nat DIM('b)
      by simp
  qed simp_all
  also have ... ≤ e by simp
  finally have dist (f (r n)) l < e
    by (auto simp: dist_norm)
 }
 ultimately have eventually (λn. dist (f (r n)) l < e) sequentially
  using eventually_elim2 by force
}
then have ∗: ((f ∘ r) ⟶ l) sequentially
 unfolding o_def tendsto_iff by simp
with r show ∃ l r. strict_mono r ∧ ((f ∘ r) ⟶ l) sequentially
 by auto
qed
```

5.16.6 concrete bounded linear functions

```
lemma transfer_bounded_bilinear_bounded_linearI:
 assumes g = (λi x. (blinfun_apply (f i) x))
 shows bounded_bilinear g = bounded_linear f
proof
 assume bounded_bilinear g
 then interpret bounded_bilinear f by (simp add: assms)
 show bounded_linear f
 proof (unfold_locales, safe intro!: blinfun_eqI)
  fix i
  show f (x + y) i = (f x + f y) i f (r *R x) i = (r *R f x) i for r x y
   by (auto intro!: blinfun_eqI simp: blinfun.bilinear_simps)
  from _ nonneg_bounded show ∃ K. ∀ x. norm (f x) ≤ norm x ∗ K
   by (rule ex_reg) (auto intro!: onorm_bound simp: norm_blinfun.rep_eq
ac_simps)
 qed
qed (auto simp: assms intro!: blinfun.comp)

lemma transfer_bounded_bilinear_bounded_linear[transfer_rule]:
 (rel_fun (rel_fun (=) (pcr_blinfun (=) (=))) (=)) bounded_bilinear bounded_linear
 by (auto simp: pcr_blinfun_def cr_blinfun_def rel_fun_def OO_def
  intro!: transfer_bounded_bilinear_bounded_linearI)
```

context *bounded_bilinear*
begin

lift_definition *prod_left*::*'b* ⇒ *'a* ⇒$_L$ *'c* **is** (λ*b a*. *prod a b*)
by (*rule bounded_linear_left*)
declare *prod_left.rep_eq*[*simp*]

lemma *bounded_linear_prod_left*[*bounded_linear*]: *bounded_linear prod_left*
by *transfer* (*rule flip*)

lift_definition *prod_right*::*'a* ⇒ *'b* ⇒$_L$ *'c* **is** (λ*a b*. *prod a b*)
by (*rule bounded_linear_right*)
declare *prod_right.rep_eq*[*simp*]

lemma *bounded_linear_prod_right*[*bounded_linear*]: *bounded_linear prod_right*
by *transfer* (*rule bounded_bilinear_axioms*)

end

lift_definition *id_blinfun*::*'a*::*real_normed_vector* ⇒$_L$ *'a* **is** λ*x*. *x*
by (*rule bounded_linear_ident*)

lemmas *blinfun_apply_id_blinfun*[*simp*] = *id_blinfun.rep_eq*

lemma *norm_blinfun_id*[*simp*]:
norm (*id_blinfun*::*'a*::{*real_normed_vector*, *perfect_space*} ⇒$_L$ *'a*) = *1*
by *transfer* (*auto simp*: *onorm_id*)

lemma *norm_blinfun_id_le*:
norm (*id_blinfun*::*'a*::*real_normed_vector* ⇒$_L$ *'a*) ≤ *1*
by *transfer* (*auto simp*: *onorm_id_le*)

lift_definition *fst_blinfun*::(*'a*::*real_normed_vector* × *'b*::*real_normed_vector*)
⇒$_L$ *'a* **is** *fst*
by (*rule bounded_linear_fst*)

lemma *blinfun_apply_fst_blinfun*[*simp*]: *blinfun_apply fst_blinfun* = *fst*
by *transfer* (*rule refl*)

lift_definition *snd_blinfun*::(*'a*::*real_normed_vector* × *'b*::*real_normed_vector*)
⇒$_L$ *'b* **is** *snd*
by (*rule bounded_linear_snd*)

lemma *blinfun_apply_snd_blinfun*[*simp*]: *blinfun_apply snd_blinfun* = *snd*
by *transfer* (*rule refl*)

lift_definition *blinfun_compose*::
'a::*real_normed_vector* ⇒$_L$ *'b*::*real_normed_vector* ⇒
'c::*real_normed_vector* ⇒$_L$ *'a* ⇒
'c ⇒$_L$ *'b* (**infixl** o_L *55*) **is** (*o*)
parametric *comp_transfer*
unfolding *o_def*
by (*rule bounded_linear_compose*)

lemma *blinfun_apply_blinfun_compose*[*simp*]: (*a* o_L *b*) *c* = *a* (*b c*)
by (*simp add*: *blinfun_compose.rep_eq*)

lemma *norm_blinfun_compose*:
norm (*f* o_L *g*) ≤ *norm f* ∗ *norm g*
by *transfer* (*rule onorm_compose*)

lemma *bounded_bilinear_blinfun_compose*[*bounded_bilinear*]: *bounded_bilinear* (o_L)
by *unfold_locales*
(*auto intro*!: *blinfun_eqI exI*[**where** *x*=*1*] *simp*: *blinfun.bilinear_simps norm_blinfun_compose*)

lemma *blinfun_compose_zero*[*simp*]:
blinfun_compose 0 = (λ_. *0*)
blinfun_compose x 0 = *0*
by (*auto simp*: *blinfun.bilinear_simps intro*!: *blinfun_eqI*)

lemma *blinfun_bij2*:
fixes *f*::*'a* ⇒$_L$ *'a*::*euclidean_space*
assumes *f* o_L *g* = *id_blinfun*
shows *bij* (*blinfun_apply g*)
proof (*rule bijI*)
show *inj g*
using *assms*
by (*metis blinfun_apply_id_blinfun blinfun_compose.rep_eq injI inj_on_imageI2*)
then show *surj g*
using *blinfun.bounded_linear_right bounded_linear_def linear_inj_imp_surj*
by *blast*
qed

lemma *blinfun_bij1*:
fixes *f*::*'a* ⇒$_L$ *'a*::*euclidean_space*
assumes *f* o_L *g* = *id_blinfun*
shows *bij* (*blinfun_apply f*)
proof (*rule bijI*)
show *surj* (*blinfun_apply f*)
by (*metis assms blinfun_apply_blinfun_compose blinfun_apply_id_blinfun surjI*)
then show *inj* (*blinfun_apply f*)
using *blinfun.bounded_linear_right bounded_linear_def linear_surj_imp_inj*
by *blast*
qed

lift_definition *blinfun_inner_right*::'*a*::*real_inner* ⇒ '*a* ⇒$_L$ *real* **is** (·)
by (*rule bounded_linear_inner_right*)
declare *blinfun_inner_right.rep_eq*[*simp*]

lemma *bounded_linear_blinfun_inner_right*[*bounded_linear*]: *bounded_linear blinfun_inner_right*
by *transfer* (*rule bounded_bilinear_inner*)

lift_definition *blinfun_inner_left*::'*a*::*real_inner* ⇒ '*a* ⇒$_L$ *real* **is** λ*x y*. *y* · *x*
by (*rule bounded_linear_inner_left*)
declare *blinfun_inner_left.rep_eq*[*simp*]

lemma *bounded_linear_blinfun_inner_left*[*bounded_linear*]: *bounded_linear blinfun_inner_left*
by *transfer* (*rule bounded_bilinear.flip*[*OF bounded_bilinear_inner*])

lift_definition *blinfun_scaleR_right*::*real* ⇒ '*a* ⇒$_L$ '*a*::*real_normed_vector* **is** (∗$_R$)
by (*rule bounded_linear_scaleR_right*)
declare *blinfun_scaleR_right.rep_eq*[*simp*]

lemma *bounded_linear_blinfun_scaleR_right*[*bounded_linear*]: *bounded_linear blinfun_scaleR_right*
by *transfer* (*rule bounded_bilinear_scaleR*)

lift_definition *blinfun_scaleR_left*::'*a*::*real_normed_vector* ⇒ *real* ⇒$_L$ '*a* **is** λ*x y*. *y* ∗$_R$ *x*
by (*rule bounded_linear_scaleR_left*)
lemmas [*simp*] = *blinfun_scaleR_left.rep_eq*

lemma *bounded_linear_blinfun_scaleR_left*[*bounded_linear*]: *bounded_linear blinfun_scaleR_left*
by *transfer* (*rule bounded_bilinear.flip*[*OF bounded_bilinear_scaleR*])

lift_definition *blinfun_mult_right*::'*a* ⇒ '*a* ⇒$_L$ '*a*::*real_normed_algebra* **is** (∗)
by (*rule bounded_linear_mult_right*)
declare *blinfun_mult_right.rep_eq*[*simp*]

lemma *bounded_linear_blinfun_mult_right*[*bounded_linear*]: *bounded_linear blinfun_mult_right*
by *transfer* (*rule bounded_bilinear_mult*)

lift_definition *blinfun_mult_left*::'*a*::*real_normed_algebra* ⇒ '*a* ⇒$_L$ '*a* **is** λ*x y*.

$y * x$
by (*rule bounded_linear_mult_left*)
lemmas [*simp*] = *blinfun_mult_left.rep_eq*

lemma *bounded_linear_blinfun_mult_left*[*bounded_linear*]: *bounded_linear blinfun_mult_left*
by *transfer* (*rule bounded_bilinear.flip*[*OF bounded_bilinear_mult*])

lemmas *bounded_linear_function_uniform_limit_intros*[*uniform_limit_intros*] =
bounded_linear.uniform_limit[*OF bounded_linear_apply_blinfun*]
bounded_linear.uniform_limit[*OF bounded_linear_blinfun_apply*]
bounded_linear.uniform_limit[*OF bounded_linear_blinfun_matrix*]

5.16.7 The strong operator topology on continuous linear operators

Let $'a$ and $'b$ be two normed real vector spaces. Then the space of linear continuous operators from $'a$ to $'b$ has a canonical norm, and therefore a canonical corresponding topology (the type classes instantiation are given in `Bounded_Linear_Function.thy`).

However, there is another topology on this space, the strong operator topology, where T_n tends to T iff, for all x in $'a$, then $T_n\ x$ tends to $T\ x$. This is precisely the product topology where the target space is endowed with the norm topology. It is especially useful when $'b$ is the set of real numbers, since then this topology is compact.

We can not implement it using type classes as there is already a topology, but at least we can define it as a topology.

Note that there is yet another (common and useful) topology on operator spaces, the weak operator topology, defined analogously using the product topology, but where the target space is given the weak-* topology, i.e., the pullback of the weak topology on the bidual of the space under the canonical embedding of a space into its bidual. We do not define it there, although it could also be defined analogously.

definition *strong_operator_topology*::('*a*::*real_normed_vector* ⇒$_L$ '*b*::*real_normed_vector*) *topology*
where *strong_operator_topology* = *pullback_topology UNIV blinfun_apply euclidean*

lemma *strong_operator_topology_topspace*:
topspace strong_operator_topology = *UNIV*
unfolding *strong_operator_topology_def topspace_pullback_topology topspace_euclidean*
by *auto*

lemma *strong_operator_topology_basis*:
fixes *f*::('*a*::*real_normed_vector* ⇒$_L$ '*b*::*real_normed_vector*) **and** *U*::'*i* ⇒ '*b set*
and *x*::'*i* ⇒ '*a*
assumes *finite I* ⋀*i*. *i* ∈ *I* ⟹ *open* (*U i*)

shows *openin strong_operator_topology* $\{f.\ \forall i{\in}I.\ blinfun\_apply\ f\ (x\ i) \in U\ i\}$
proof −
have *open* $\{g{::}('a{\Rightarrow}'b).\ \forall i{\in}I.\ g\ (x\ i) \in U\ i\}$
by (*rule product_topology_basis'*[*OF assms*])
moreover have $\{f.\ \forall i{\in}I.\ blinfun\_apply\ f\ (x\ i) \in U\ i\}$
$= blinfun\_apply{-}`\{g{::}('a{\Rightarrow}'b).\ \forall i{\in}I.\ g\ (x\ i) \in U\ i\} \cap UNIV$
by *auto*
ultimately show *?thesis*
unfolding *strong_operator_topology_def* **by** (*subst openin_pullback_topology*)
auto
qed

lemma *strong_operator_topology_continuous_evaluation*:
continuous_map strong_operator_topology euclidean $(\lambda f.\ blinfun\_apply\ f\ x)$
proof −
have *continuous_map strong_operator_topology euclidean* $((\lambda f.\ f\ x)\ o\ blinfun\_apply)$
unfolding *strong_operator_topology_def* **apply** (*rule continuous_map_pullback*)
using *continuous_on_product_coordinates* **by** *fastforce*
then show *?thesis* **unfolding** *comp_def* **by** *simp*
qed

lemma *continuous_on_strong_operator_topo_iff_coordinatewise*:
continuous_map T strong_operator_topology f
$\longleftrightarrow$ $(\forall x.$ *continuous_map T euclidean* $(\lambda y.\ blinfun\_apply\ (f\ y)\ x))$
proof (*auto*)
fix $x{::}'b$
assume *continuous_map T strong_operator_topology f*
with *continuous_map_compose*[*OF this strong_operator_topology_continuous_evaluation*]
have *continuous_map T euclidean* $((\lambda z.\ blinfun\_apply\ z\ x)\ o\ f)$
by *simp*
then show *continuous_map T euclidean* $(\lambda y.\ blinfun\_apply\ (f\ y)\ x)$
unfolding *comp_def* **by** *auto*
next
assume $*$: $\forall x.$ *continuous_map T euclidean* $(\lambda y.\ blinfun\_apply\ (f\ y)\ x)$
have $\bigwedge i.$ *continuous_map T euclidean* $(\lambda x.\ blinfun\_apply\ (f\ x)\ i)$
using $*$ **unfolding** *comp_def* **by** *auto*
then have *continuous_map T euclidean* $(blinfun\_apply\ o\ f)$
unfolding *o_def*
by (*metis* (*no_types*) *continuous_map_componentwise_UNIV euclidean_product_topology*)
show *continuous_map T strong_operator_topology f*
unfolding *strong_operator_topology_def*
apply (*rule continuous_map_pullback'*)
by (*auto simp add*: ‹*continuous_map T euclidean* $(blinfun\_apply\ o\ f)$›)
qed

lemma *strong_operator_topology_weaker_than_euclidean*:
continuous_map euclidean strong_operator_topology $(\lambda f.\ f)$
by (*subst continuous_on_strong_operator_topo_iff_coordinatewise*,

auto simp add: *linear_continuous_on*)

end

5.17 Derivative

theory *Derivative*
imports
Bounded_Linear_Function
Line_Segment
Convex_Euclidean_Space
begin

declare *bounded_linear_inner_left* [*intro*]

declare *has_derivative_bounded_linear*[*dest*]

5.17.1 Derivatives

lemma *has_derivative_add_const*:
(*f has_derivative f'*) *net* $\Longrightarrow$ ($\lambda x.\ f\ x + c$) *has_derivative f'*) *net*
by (*intro derivative_eq_intros*) *auto*

5.17.2 Derivative with composed bilinear function

More explicit epsilon-delta forms.

proposition *has_derivative_within'*:
(*f has_derivative f'*)(*at x within s*) $\longleftrightarrow$
bounded_linear f' $\wedge$
($\forall\, e{>}0.\ \exists\, d{>}0.\ \forall\, x'{\in}s.\ 0 < norm\ (x' - x) \wedge norm\ (x' - x) < d \longrightarrow$
norm (*f x'* $-$ *f x* $-$ *f'*(*x'* $-$ *x*)) / *norm* (*x'* $-$ *x*) $<$ *e*)
unfolding *has_derivative_within Lim_within dist_norm*
by (*simp add*: *diff_diff_eq*)

lemma *has_derivative_at'*:
(*f has_derivative f'*) (*at x*)
$\longleftrightarrow$ *bounded_linear f'* $\wedge$
($\forall\, e{>}0.\ \exists\, d{>}0.\ \forall\, x'.\ 0 < norm\ (x' - x) \wedge norm\ (x' - x) < d \longrightarrow$
norm (*f x'* $-$ *f x* $-$ *f'*(*x'* $-$ *x*)) / *norm* (*x'* $-$ *x*) $<$ *e*)
using *has_derivative_within'* [*of f f' x UNIV*] **by** *simp*

lemma *has_derivative_componentwise_within*:
(*f has_derivative f'*) (*at a within S*) $\longleftrightarrow$
($\forall\, i \in Basis.$ (($\lambda x.\ f\ x \cdot i$) *has_derivative* ($\lambda x.\ f'\ x \cdot i$)) (*at a within S*))
apply (*simp add*: *has_derivative_within*)
apply (*subst tendsto_componentwise_iff*)
apply (*simp add*: *ball_conj_distrib inner_diff_left inner_left_distrib flip*: *bounded_linear_componentwise_iff*)
done

lemma *has_derivative_at_withinI*:
(*f has_derivative f'*) (*at x*) ⟹ (*f has_derivative f'*) (*at x within s*)
unfolding *has_derivative_within' has_derivative_at'*
by *blast*

lemma *has_derivative_right*:
fixes *f* :: *real* ⇒ *real*
and *y* :: *real*
shows (*f has_derivative* ((∗) *y*)) (*at x within* ({*x* <..} ∩ *I*)) ⟷
((λ*t*. (*f x* − *f t*) / (*x* − *t*)) ⟶ *y*) (*at x within* ({*x* <..} ∩ *I*))
proof −
have ((λ*t*. (*f t* − (*f x* + *y* ∗ (*t* − *x*))) / |*t* − *x*|) ⟶ *0*) (*at x within* ({*x*<..}
∩ *I*)) ⟷
((λ*t*. (*f t* − *f x*) / (*t* − *x*) − *y*) ⟶ *0*) (*at x within* ({*x*<..} ∩ *I*))
by (*intro Lim_cong_within*) (*auto simp add*: *diff_divide_distrib add_divide_distrib*)
also have ... ⟷ ((λ*t*. (*f t* − *f x*) / (*t* − *x*)) ⟶ *y*) (*at x within* ({*x*<..} ∩
I))
by (*simp add*: *Lim_null*[*symmetric*])
also have ... ⟷ ((λ*t*. (*f x* − *f t*) / (*x* − *t*)) ⟶ *y*) (*at x within* ({*x*<..} ∩
I))
by (*intro Lim_cong_within*) (*simp_all add*: *field_simps*)
finally show *?thesis*
by (*simp add*: *bounded_linear_mult_right has_derivative_within*)
qed

Caratheodory characterization

lemma *DERIV_caratheodory_within*:
(*f has_field_derivative l*) (*at x within S*) ⟷
(∃ *g*. (∀ *z*. *f z* − *f x* = *g z* ∗ (*z* − *x*)) ∧ *continuous* (*at x within S*) *g* ∧ *g x* = *l*)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
show *?rhs*
proof (*intro exI conjI*)
let *?g* = (%*z*. *if z* = *x then l else* (*f z* − *f x*) / (*z*−*x*))
show ∀ *z*. *f z* − *f x* = *?g z* ∗ (*z*−*x*) **by** *simp*
show *continuous* (*at x within S*) *?g* **using** ‹*?lhs*›
by (*auto simp add*: *continuous_within has_field_derivative_iff cong*: *Lim_cong_within*)
show *?g x* = *l* **by** *simp*
qed
next
assume *?rhs*
then obtain *g* **where**
(∀ *z*. *f z* − *f x* = *g z* ∗ (*z*−*x*)) **and** *continuous* (*at x within S*) *g* **and** *g x* = *l*
by *blast*
thus *?lhs*
by (*auto simp add*: *continuous_within has_field_derivative_iff cong*: *Lim_cong_within*)

qed

5.17.3 Differentiability

definition
differentiable_on :: ('*a*::*real_normed_vector* ⇒ '*b*::*real_normed_vector*) ⇒ '*a set* ⇒ *bool*
(**infix** *differentiable'_on 50*)
where *f differentiable_on s* ⟷ (∀ *x*∈*s*. *f differentiable* (*at x within s*))

lemma *differentiableI*: (*f has_derivative f'*) *net* ⟹ *f differentiable net*
unfolding *differentiable_def*
by *auto*

lemma *differentiable_onD*: ⟦*f differentiable_on S*; *x* ∈ *S*⟧ ⟹ *f differentiable* (*at x within S*)
using *differentiable_on_def* **by** *blast*

lemma *differentiable_at_withinI*: *f differentiable* (*at x*) ⟹ *f differentiable* (*at x within s*)
unfolding *differentiable_def*
using *has_derivative_at_withinI*
by *blast*

lemma *differentiable_at_imp_differentiable_on*:
(⋀*x*. *x* ∈ *s* ⟹ *f differentiable at x*) ⟹ *f differentiable_on s*
by (*metis differentiable_at_withinI differentiable_on_def*)

corollary *differentiable_iff_scaleR*:
fixes *f* :: *real* ⇒ '*a*::*real_normed_vector*
shows *f differentiable F* ⟷ (∃ *d*. (*f has_derivative* (λ*x*. *x* ∗$_R$ *d*)) *F*)
by (*auto simp*: *differentiable_def dest*: *has_derivative_linear linear_imp_scaleR*)

lemma *differentiable_on_eq_differentiable_at*:
open s ⟹ *f differentiable_on s* ⟷ (∀ *x*∈*s*. *f differentiable at x*)
unfolding *differentiable_on_def*
by (*metis at_within_interior interior_open*)

lemma *differentiable_transform_within*:
assumes *f differentiable* (*at x within s*)
and *0* < *d*
and *x* ∈ *s*
and ⋀*x'*. ⟦*x'*∈*s*; *dist x' x* < *d*⟧ ⟹ *f x'* = *g x'*
shows *g differentiable* (*at x within s*)
using *assms has_derivative_transform_within* **unfolding** *differentiable_def*
by *blast*

lemma *differentiable_on_ident* [*simp*, *derivative_intros*]: (λ*x*. *x*) *differentiable_on S*

by (*simp add*: *differentiable_at_imp_differentiable_on*)

lemma *differentiable_on_id* [*simp*, *derivative_intros*]: *id differentiable_on S*
by (*simp add*: *id_def*)

lemma *differentiable_on_const* [*simp*, *derivative_intros*]: (λ*z*. *c*) *differentiable_on S*
by (*simp add*: *differentiable_on_def*)

lemma *differentiable_on_mult* [*simp*, *derivative_intros*]:
fixes *f* :: *'M*::*real_normed_vector* ⇒ *'a*::*real_normed_algebra*
shows ⟦*f differentiable_on S*; *g differentiable_on S*⟧ ⟹ (λ*z*. *f z* ∗ *g z*) *differentiable_on S*
unfolding *differentiable_on_def differentiable_def*
using *differentiable_def differentiable_mult* **by** *blast*

lemma *differentiable_on_compose*:
⟦*g differentiable_on S*; *f differentiable_on* (*g* ‘ *S*)⟧ ⟹ (λ*x*. *f* (*g x*)) *differentiable_on S*
by (*simp add*: *differentiable_in_compose differentiable_on_def*)

lemma *bounded_linear_imp_differentiable_on*: *bounded_linear f* ⟹ *f differentiable_on S*
by (*simp add*: *differentiable_on_def bounded_linear_imp_differentiable*)

lemma *linear_imp_differentiable_on*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*real_normed_vector*
shows *linear f* ⟹ *f differentiable_on S*
by (*simp add*: *differentiable_on_def linear_imp_differentiable*)

lemma *differentiable_on_minus* [*simp*, *derivative_intros*]:
f differentiable_on S ⟹ (λ*z*. −(*f z*)) *differentiable_on S*
by (*simp add*: *differentiable_on_def*)

lemma *differentiable_on_add* [*simp*, *derivative_intros*]:
⟦*f differentiable_on S*; *g differentiable_on S*⟧ ⟹ (λ*z*. *f z* + *g z*) *differentiable_on S*
by (*simp add*: *differentiable_on_def*)

lemma *differentiable_on_diff* [*simp*, *derivative_intros*]:
⟦*f differentiable_on S*; *g differentiable_on S*⟧ ⟹ (λ*z*. *f z* − *g z*) *differentiable_on S*
by (*simp add*: *differentiable_on_def*)

lemma *differentiable_on_inverse* [*simp*, *derivative_intros*]:
fixes *f* :: *'a* :: *real_normed_vector* ⇒ *'b* :: *real_normed_field*
shows *f differentiable_on S* ⟹ (⋀*x*. *x* ∈ *S* ⟹ *f x* ≠ *0*) ⟹ (λ*x*. *inverse* (*f x*)) *differentiable_on S*
by (*simp add*: *differentiable_on_def*)

lemma *differentiable_on_scaleR* [*derivative_intros*, *simp*]:
⟦*f differentiable_on S*; *g differentiable_on S*⟧ ⟹ (λ*x*. *f x* ∗$_R$ *g x*) *differentiable_on S*
unfolding *differentiable_on_def*
by (*blast intro*: *differentiable_scaleR*)

lemma *has_derivative_sqnorm_at* [*derivative_intros*, *simp*]:
((λ*x*. (*norm x*)^2) *has_derivative* (λ*x*. 2 ∗$_R$ (*a* · *x*))) (*at a*)
using *bounded_bilinear.FDERIV* [*of* (·) *id id a _ id id*]
by (*auto simp*: *inner_commute dot_square_norm bounded_bilinear_inner*)

lemma *differentiable_sqnorm_at* [*derivative_intros*, *simp*]:
fixes *a* :: ′*a* :: {*real_normed_vector*,*real_inner*}
shows (λ*x*. (*norm x*)^2) *differentiable* (*at a*)
by (*force simp add*: *differentiable_def intro*: *has_derivative_sqnorm_at*)

lemma *differentiable_on_sqnorm* [*derivative_intros*, *simp*]:
fixes *S* :: ′*a* :: {*real_normed_vector*,*real_inner*} *set*
shows (λ*x*. (*norm x*)^2) *differentiable_on S*
by (*simp add*: *differentiable_at_imp_differentiable_on*)

lemma *differentiable_norm_at* [*derivative_intros*, *simp*]:
fixes *a* :: ′*a* :: {*real_normed_vector*,*real_inner*}
shows *a* ≠ *0* ⟹ *norm differentiable* (*at a*)
using *differentiableI has_derivative_norm* **by** *blast*

lemma *differentiable_on_norm* [*derivative_intros*, *simp*]:
fixes *S* :: ′*a* :: {*real_normed_vector*,*real_inner*} *set*
shows *0* ∉ *S* ⟹ *norm differentiable_on S*
by (*metis differentiable_at_imp_differentiable_on differentiable_norm_at*)

5.17.4 Frechet derivative and Jacobian matrix

definition *frechet_derivative f net* = (*SOME f* ′. (*f has_derivative f* ′) *net*)

proposition *frechet_derivative_works*:
f differentiable net ⟷ (*f has_derivative* (*frechet_derivative f net*)) *net*
unfolding *frechet_derivative_def differentiable_def*
unfolding *some_eq_ex*[*of* λ *f* ′ . (*f has_derivative f* ′) *net*] **..**

lemma *linear_frechet_derivative*: *f differentiable net* ⟹ *linear* (*frechet_derivative f net*)
unfolding *frechet_derivative_works has_derivative_def*
by (*auto intro*: *bounded_linear.linear*)

lemma *frechet_derivative_const* [*simp*]: *frechet_derivative* (λ*x*. *c*) (*at a*) = (λ*x*. *0*)
using *differentiable_const frechet_derivative_works has_derivative_const has_derivative_unique*

by *blast*

lemma *frechet_derivative_id* [*simp*]: *frechet_derivative id* (*at a*) = *id*
using *differentiable_def frechet_derivative_works has_derivative_id has_derivative_unique*
by *blast*

lemma *frechet_derivative_ident* [*simp*]: *frechet_derivative* (λ*x*. *x*) (*at a*) = (λ*x*. *x*)
by (*metis eq_id_iff frechet_derivative_id*)

5.17.5 Differentiability implies continuity

proposition *differentiable_imp_continuous_within*:
f differentiable (*at x within s*) ⟹ *continuous* (*at x within s*) *f*
by (*auto simp*: *differentiable_def intro*: *has_derivative_continuous*)

lemma *differentiable_imp_continuous_on*:
f differentiable_on s ⟹ *continuous_on s f*
unfolding *differentiable_on_def continuous_on_eq_continuous_within*
using *differentiable_imp_continuous_within* **by** *blast*

lemma *differentiable_on_subset*:
f differentiable_on t ⟹ *s* ⊆ *t* ⟹ *f differentiable_on s*
unfolding *differentiable_on_def*
using *differentiable_within_subset*
by *blast*

lemma *differentiable_on_empty*: *f differentiable_on* {}
unfolding *differentiable_on_def*
by *auto*

lemma *has_derivative_continuous_on*:
(⋀*x*. *x* ∈ *s* ⟹ (*f has_derivative f' x*) (*at x within s*)) ⟹ *continuous_on s f*
by (*auto intro*!: *differentiable_imp_continuous_on differentiableI simp*: *differentiable_on_def*)

Results about neighborhoods filter.

lemma *eventually_nhds_metric_le*:
eventually P (*nhds a*) = (∃ *d*>0. ∀ *x*. *dist x a* ≤ *d* ⟶ *P x*)
unfolding *eventually_nhds_metric* **by** (*safe*, *rule_tac x*=*d* / *2* **in** *exI*, *auto*)

lemma *le_nhds*: *F* ≤ *nhds a* ⟷ (∀ *S*. *open S* ∧ *a* ∈ *S* ⟶ *eventually* (λ*x*. *x* ∈ *S*) *F*)
unfolding *le_filter_def eventually_nhds* **by** (*fast elim*: *eventually_mono*)

lemma *le_nhds_metric*: *F* ≤ *nhds a* ⟷ (∀ *e*>0. *eventually* (λ*x*. *dist x a* < *e*) *F*)
unfolding *le_filter_def eventually_nhds_metric* **by** (*fast elim*: *eventually_mono*)

lemma *le_nhds_metric_le*: $F \leq nhds\ a \longleftrightarrow (\forall e{>}0.\ eventually\ (\lambda x.\ dist\ x\ a \leq e)\ F)$
unfolding *le_filter_def eventually_nhds_metric_le* **by** (*fast elim*: *eventually_mono*)

Several results are easier using a "multiplied-out" variant. (I got this idea from Dieudonne's proof of the chain rule).

lemma *has_derivative_within_alt*:
$(f\ has\_derivative\ f')\ (at\ x\ within\ s) \longleftrightarrow bounded\_linear\ f' \wedge$
$(\forall e{>}0.\ \exists d{>}0.\ \forall y{\in}s.\ norm(y - x) < d \longrightarrow norm\ (f\ y - f\ x - f'\ (y - x)) \leq e * norm\ (y - x))$
unfolding *has_derivative_within filterlim_def le_nhds_metric_le eventually_filtermap eventually_at dist_norm diff_diff_eq*
by (*force simp add*: *linear_0 bounded_linear.linear pos_divide_le_eq*)

lemma *has_derivative_within_alt2*:
$(f\ has\_derivative\ f')\ (at\ x\ within\ s) \longleftrightarrow bounded\_linear\ f' \wedge$
$(\forall e{>}0.\ eventually\ (\lambda y.\ norm\ (f\ y - f\ x - f'\ (y - x)) \leq e * norm\ (y - x))\ (at\ x\ within\ s))$
unfolding *has_derivative_within filterlim_def le_nhds_metric_le eventually_filtermap eventually_at dist_norm diff_diff_eq*
by (*force simp add*: *linear_0 bounded_linear.linear pos_divide_le_eq*)

lemma *has_derivative_at_alt*:
$(f\ has\_derivative\ f')\ (at\ x) \longleftrightarrow$
$bounded\_linear\ f' \wedge$
$(\forall e{>}0.\ \exists d{>}0.\ \forall y.\ norm(y - x) < d \longrightarrow norm\ (f\ y - f\ x - f'(y - x)) \leq e * norm\ (y - x))$
using *has_derivative_within_alt*[**where** *s*=*UNIV*]
by *simp*

5.17.6 The chain rule

proposition *diff_chain_within*[*derivative_intros*]:
assumes $(f\ has\_derivative\ f')\ (at\ x\ within\ s)$
and $(g\ has\_derivative\ g')\ (at\ (f\ x)\ within\ (f\ `\ s))$
shows $((g \circ f)\ has\_derivative\ (g' \circ f'))(at\ x\ within\ s)$
using *has_derivative_in_compose*[*OF assms*]
by (*simp add*: *comp_def*)

lemma *diff_chain_at*[*derivative_intros*]:
$(f\ has\_derivative\ f')\ (at\ x) \Longrightarrow$
$(g\ has\_derivative\ g')\ (at\ (f\ x)) \Longrightarrow ((g \circ f)\ has\_derivative\ (g' \circ f'))\ (at\ x)$
by (*meson diff_chain_within has_derivative_at_withinI*)

lemma *has_vector_derivative_shift*: $(f\ has\_vector\_derivative\ D\ x)\ (at\ x)$
$\Longrightarrow ((+)\ d \circ f\ has\_vector\_derivative\ D\ x)\ (at\ x)$
using *diff_chain_at* [*OF _ shift_has_derivative_id*]
by (*simp add*: *has_derivative_iff_Ex has_vector_derivative_def*)

lemma *has_vector_derivative_within_open*:
 $a \in S \Longrightarrow$ *open* $S \Longrightarrow$
 (*f has_vector_derivative f'*) (*at a within S*) $\longleftrightarrow$ (*f has_vector_derivative f'*) (*at a*)
 by (*simp only*: *at_within_interior interior_open*)

lemma *field_vector_diff_chain_within*:
 assumes *Df*: (*f has_vector_derivative f'*) (*at x within S*)
 and *Dg*: (*g has_field_derivative g'*) (*at* (*f x*) *within f ' S*)
 shows ((*g* ∘ *f*) *has_vector_derivative* (*f'* ∗ *g'*)) (*at x within S*)
using *diff_chain_within*[*OF Df*[*unfolded has_vector_derivative_def*]
 Dg [*unfolded has_field_derivative_def*]]
 by (*auto simp*: *o_def mult.commute has_vector_derivative_def*)

lemma *vector_derivative_diff_chain_within*:
 assumes *Df*: (*f has_vector_derivative f'*) (*at x within S*)
 and *Dg*: (*g has_derivative g'*) (*at* (*f x*) *within f'S*)
 shows ((*g* ∘ *f*) *has_vector_derivative* (*g' f'*)) (*at x within S*)
using *diff_chain_within*[*OF Df*[*unfolded has_vector_derivative_def*] *Dg*]
 linear.scaleR[*OF has_derivative_linear*[*OF Dg*]]
 unfolding *has_vector_derivative_def o_def*
 by (*auto simp*: *o_def mult.commute has_vector_derivative_def*)

5.17.7 Composition rules stated just for differentiability

lemma *differentiable_chain_at*:
 f differentiable (*at x*) $\Longrightarrow$
 g differentiable (*at* (*f x*)) $\Longrightarrow$ (*g* ∘ *f*) *differentiable* (*at x*)
 unfolding *differentiable_def*
 by (*meson diff_chain_at*)

lemma *differentiable_chain_within*:
 f differentiable (*at x within S*) $\Longrightarrow$
 g differentiable (*at*(*f x*) *within* (*f ' S*)) $\Longrightarrow$ (*g* ∘ *f*) *differentiable* (*at x within S*)
 unfolding *differentiable_def*
 by (*meson diff_chain_within*)

5.17.8 Uniqueness of derivative

The general result is a bit messy because we need approachability of the limit point from any direction. But OK for nontrivial intervals etc.

proposition *frechet_derivative_unique_within*:
 fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*real_normed_vector*
 assumes *1*: (*f has_derivative f'*) (*at x within S*)
 and *2*: (*f has_derivative f''*) (*at x within S*)
 and *S*: $\bigwedge i\ e.\ [\![i \in Basis;\ e>0]\!] \Longrightarrow \exists d.\ 0 < |d| \wedge |d| < e \wedge (x + d *_R i) \in S$
 shows *f'* = *f''*
proof −
 note *as* = *assms*(*1*,*2*)[*unfolded has_derivative_def*]

then interpret f': *bounded_linear* f' **by** *auto*
from *as* **interpret** f'': *bounded_linear* f'' **by** *auto*
have x *islimpt* S **unfolding** *islimpt_approachable*
proof (*intro allI impI*)
fix e :: *real*
assume $e > 0$
obtain d **where** $0 < |d|$ **and** $|d| < e$ **and** $x + d *_R (SOME\ i.\ i \in Basis) \in S$
using *assms*(3) *SOME_Basis* ‹$e>0$› **by** *blast*
then show $\exists x' \in S.\ x' \neq x \wedge dist\ x'\ x < e$
by (*rule_tac* $x=x + d *_R (SOME\ i.\ i \in Basis)$ **in** *bexI*) (*auto simp*: *dist_norm SOME_Basis nonzero_Basis*) **qed**
then have $*$: *netlimit* (*at* x *within* S) $= x$
by (*simp add*: *Lim_ident_at trivial_limit_within*)
show *?thesis*
proof (*rule linear_eq_stdbasis*)
show *linear* f' *linear* f''
unfolding *linear_conv_bounded_linear* **using** *as* **by** *auto*
next
fix i :: $'a$
assume i: $i \in Basis$
define e **where** $e = norm\ (f'\ i - f''\ i)$
show $f'\ i = f''\ i$
proof (*rule ccontr*)
assume $f'\ i \neq f''\ i$
then have $e > 0$
unfolding e_def **by** *auto*
obtain d **where** d:
$0 < d$
$(\bigwedge y.\ y \in S \longrightarrow 0 < dist\ y\ x \wedge dist\ y\ x < d \longrightarrow$
$dist\ ((f\ y - f\ x - f'\ (y - x)) /_R norm\ (y - x) -$
$(f\ y - f\ x - f''\ (y - x)) /_R norm\ (y - x))\ (0 - 0) < e)$
using *tendsto_diff* [*OF as*(1,2)[*THEN conjunct2*]]
unfolding $*$ *Lim_within*
using ‹$e>0$› **by** *blast*
obtain c **where** c: $0 < |c|$ $|c| < d \wedge x + c *_R i \in S$
using *assms*(3) i *d*(1) **by** *blast*
have $*$: $norm\ (-((1\ /\ |c|) *_R f'\ (c *_R i)) + (1\ /\ |c|) *_R f''\ (c *_R i)) =$
$norm\ ((1\ /\ |c|) *_R (-(f'\ (c *_R i)) + f''\ (c *_R i)))$
unfolding *scaleR_right_distrib* **by** *auto*
also have $\ldots = norm\ ((1\ /\ |c|) *_R (c *_R (-(f'\ i) + f''\ i)))$
unfolding $f'.scaleR$ $f''.scaleR$
unfolding *scaleR_right_distrib scaleR_minus_right*
by *auto*
also have $\ldots = e$
unfolding e_def
using $c(1)$
using *norm_minus_cancel*[*of* $f'\ i - f''\ i$]
by *auto*
finally show *False*

```
      using c
      using d(2)[of x + c *R i]
      unfolding dist_norm
      unfolding f'.scaleR f''.scaleR f'.add f''.add f'.diff f''.diff
        scaleR_scaleR scaleR_right_diff_distrib scaleR_right_distrib
      using i
      by (auto simp: inverse_eq_divide)
    qed
  qed
qed

proposition frechet_derivative_unique_within_closed_interval:
  fixes f::'a::euclidean_space ⇒ 'b::real_normed_vector
  assumes ab: ⋀i. i∈Basis ⟹ a•i < b•i
    and x: x ∈ cbox a b
    and (f has_derivative f' ) (at x within cbox a b)
    and (f has_derivative f'') (at x within cbox a b)
  shows f' = f''
proof (rule frechet_derivative_unique_within)
  fix e :: real
  fix i :: 'a
  assume e > 0 and i: i ∈ Basis
  then show ∃ d. 0 < |d| ∧ |d| < e ∧ x + d *R i ∈ cbox a b
  proof (cases x•i = a•i)
    case True
    with ab[of i] ‹e>0› x i show ?thesis
      by (rule_tac x=(min (b•i − a•i) e) / 2 in exI)
        (auto simp add: mem_box field_simps inner_simps inner_Basis)
  next
    case False
    moreover have a • i < x • i
      using False i mem_box(2) x by force
    moreover {
      have a • i * 2 + min (x • i − a • i) e ≤ a•i *2 + x•i − a•i
        by auto
      also have ... = a•i + x•i
        by auto
      also have ... ≤ 2 * (x•i)
        using ‹a • i < x • i› by auto
      finally have a • i * 2 + min (x • i − a • i) e ≤ x • i * 2
        by auto
    }
    moreover have min (x • i − a • i) e ≥ 0
      by (simp add: ‹0 < e› ‹a • i < x • i› less_eq_real_def)
    then have x • i * 2 ≤ b • i * 2 + min (x • i − a • i) e
      using i mem_box(2) x by force
    ultimately show ?thesis
    using ab[of i] ‹e>0› x i
      by (rule_tac x=− (min (x•i − a•i) e) / 2 in exI)
```

(*auto simp add*: *mem_box field_simps inner_simps inner_Basis*)
qed
qed (*use assms* **in** *auto*)

lemma *frechet_derivative_unique_within_open_interval*:
fixes *f*::'*a*::*euclidean_space* ⇒ '*b*::*real_normed_vector*
assumes *x*: *x* ∈ *box a b*
and *f*: (*f has_derivative f'*) (*at x within box a b*) (*f has_derivative f''*) (*at x within box a b*)
shows *f'* = *f''*
by (*metis at_within_open assms has_derivative_unique open_box*)

lemma *frechet_derivative_at*:
(*f has_derivative f'*) (*at x*) ⟹ *f'* = *frechet_derivative f* (*at x*)
using *differentiable_def frechet_derivative_works has_derivative_unique* **by** *blast*

lemma *frechet_derivative_compose*:
frechet_derivative (*f o g*) (*at x*) = *frechet_derivative* (*f*) (*at* (*g x*)) *o frechet_derivative g* (*at x*)
if *g differentiable at x f differentiable at* (*g x*)
by (*metis diff_chain_at frechet_derivative_at frechet_derivative_works that*)

lemma *frechet_derivative_within_cbox*:
fixes *f* :: '*a*::*euclidean_space* ⇒ '*b*::*real_normed_vector*
assumes ⋀*i*. *i*∈*Basis* ⟹ *a*•*i* < *b*•*i*
and *x* ∈ *cbox a b*
and (*f has_derivative f'*) (*at x within cbox a b*)
shows *frechet_derivative f* (*at x within cbox a b*) = *f'*
using *assms*
by (*metis Derivative.differentiableI frechet_derivative_unique_within_closed_interval frechet_derivative_works*)

lemma *frechet_derivative_transform_within_open*:
frechet_derivative f (*at x*) = *frechet_derivative g* (*at x*)
if *f differentiable at x open X x* ∈ *X* ⋀*x*. *x* ∈ *X* ⟹ *f x* = *g x*
by (*meson frechet_derivative_at frechet_derivative_works has_derivative_transform_within_open that*)

5.17.9 Derivatives of local minima and maxima are zero

lemma *has_derivative_local_min*:
fixes *f* :: '*a*::*real_normed_vector* ⇒ *real*
assumes *deriv*: (*f has_derivative f'*) (*at x*)
assumes *min*: *eventually* (λ*y*. *f x* ≤ *f y*) (*at x*)
shows *f'* = (λ*h*. *0*)
proof
fix *h* :: '*a*
interpret *f'*: *bounded_linear f'*
using *deriv* **by** (*rule has_derivative_bounded_linear*)

```
  show f' h = 0
  proof (cases h = 0)
    case False
    from min obtain d where d1: 0 < d and d2: ∀ y∈ball x d. f x ≤ f y
      unfolding eventually_at by (force simp: dist_commute)
    have FDERIV (λr. x + r *R h) 0 :> (λr. r *R h)
      by (intro derivative_eq_intros) auto
    then have FDERIV (λr. f (x + r *R h)) 0 :> (λk. f' (k *R h))
      by (rule has_derivative_compose, simp add: deriv)
    then have DERIV (λr. f (x + r *R h)) 0 :> f' h
     unfolding has_field_derivative_def by (simp add: f'.scaleR mult_commute_abs)
    moreover have 0 < d / norm h using d1 and ‹h ≠ 0› by simp
   moreover have ∀ y. |0 − y| < d / norm h ⟶ f (x + 0 *R h) ≤ f (x + y *R
h)
      using ‹h ≠ 0› by (auto simp add: d2 dist_norm pos_less_divide_eq)
    ultimately show f' h = 0
      by (rule DERIV_local_min)
  qed simp
qed

lemma has_derivative_local_max:
  fixes f :: 'a::real_normed_vector ⇒ real
  assumes (f has_derivative f') (at x)
  assumes eventually (λy. f y ≤ f x) (at x)
  shows f' = (λh. 0)
  using has_derivative_local_min [of λx. − f x λh. − f' h x]
  using assms unfolding fun_eq_iff by simp

lemma differential_zero_maxmin:
  fixes f::'a::real_normed_vector ⇒ real
  assumes x ∈ S
    and open S
    and deriv: (f has_derivative f') (at x)
    and mono: (∀ y∈S. f y ≤ f x) ∨ (∀ y∈S. f x ≤ f y)
  shows f' = (λv. 0)
  using mono
proof
  assume ∀ y∈S. f y ≤ f x
  with ‹x ∈ S› and ‹open S› have eventually (λy. f y ≤ f x) (at x)
    unfolding eventually_at_topological by auto
  with deriv show ?thesis
    by (rule has_derivative_local_max)
next
  assume ∀ y∈S. f x ≤ f y
  with ‹x ∈ S› and ‹open S› have eventually (λy. f x ≤ f y) (at x)
    unfolding eventually_at_topological by auto
  with deriv show ?thesis
    by (rule has_derivative_local_min)
qed
```

lemma *differential_zero_maxmin_component*:
fixes f :: *'a::euclidean_space* ⇒ *'b::euclidean_space*
assumes k: $k \in Basis$
and *ball*: $0 < e$ $(\forall y \in ball\ x\ e.\ (f\ y)\cdot k \leq (f\ x)\cdot k) \vee (\forall y \in ball\ x\ e.\ (f\ x)\cdot k \leq (f\ y)\cdot k)$
and *diff*: *f differentiable* $(at\ x)$
shows $(\sum j \in Basis.\ (frechet\_derivative\ f\ (at\ x)\ j \cdot k) *_R j) = (0::'a)$ (**is** *?D k* = *0*)
proof −
let *?f'* = *frechet_derivative f* $(at\ x)$
have $x \in ball\ x\ e$ **using** ‹$0 < e$› **by** *simp*
moreover have *open* $(ball\ x\ e)$ **by** *simp*
moreover have $((\lambda x.\ f\ x \cdot k)\ has\_derivative\ (\lambda h.\ ?f'\ h \cdot k))\ (at\ x)$
using *bounded_linear_inner_left diff*[*unfolded frechet_derivative_works*]
by (*rule bounded_linear.has_derivative*)
ultimately have $(\lambda h.\ frechet\_derivative\ f\ (at\ x)\ h \cdot k) = (\lambda v.\ 0)$
using *ball*(*2*) **by** (*rule differential_zero_maxmin*)
then show *?thesis*
unfolding *fun_eq_iff* **by** *simp*
qed

5.17.10 One-dimensional mean value theorem

lemma *mvt_simple*:
fixes f :: *real* ⇒ *real*
assumes $a < b$
and *derf*: $\bigwedge x.\ [\![a \leq x; x \leq b]\!] \Longrightarrow (f\ has\_derivative\ f'\ x)\ (at\ x\ within\ \{a..b\})$
shows $\exists x \in \{a<..<b\}.\ f\ b - f\ a = f'\ x\ (b - a)$
proof (*rule mvt*)
have *f differentiable_on* $\{a..b\}$
using *derf* **unfolding** *differentiable_on_def differentiable_def* **by** *force*
then show *continuous_on* $\{a..b\}$ *f*
by (*rule differentiable_imp_continuous_on*)
show $(f\ has\_derivative\ f'\ x)\ (at\ x)$ **if** $a < x\ x < b$ **for** x
by (*metis at_within_Icc_at derf leI order.asym that*)
qed (*use assms* **in** *auto*)

lemma *mvt_very_simple*:
fixes f :: *real* ⇒ *real*
assumes $a \leq b$
and *derf*: $\bigwedge x.\ [\![a \leq x; x \leq b]\!] \Longrightarrow (f\ has\_derivative\ f'\ x)\ (at\ x\ within\ \{a..b\})$
shows $\exists x \in \{a..b\}.\ f\ b - f\ a = f'\ x\ (b - a)$
proof (*cases* $a = b$)
interpret *bounded_linear f' b*
using *assms* **by** *auto*
case *True*
then show *?thesis*
by *force*

next
case *False*
then show *?thesis*
using *mvt_simple*[*OF _ derf*]
by (*metis* ‹$a \leq b$› *atLeastAtMost_iff dual_order.order_iff_strict greaterThanLessThan_iff*)
qed

A nice generalization (see Havin's proof of 5.19 from Rudin's book).

lemma *mvt_general*:
fixes $f :: real \Rightarrow 'a::real\_inner$
assumes $a < b$
and *contf*: *continuous_on* $\{a..b\}$ f
and *derf*: $\bigwedge x.\ [\![a < x; x < b]\!] \Longrightarrow (f\ has\_derivative\ f'\ x)\ (at\ x)$
shows $\exists x \in \{a<..<b\}.\ norm\ (f\ b - f\ a) \leq norm\ (f'\ x\ (b - a))$
proof −
have $\exists x \in \{a<..<b\}.\ (f\ b - f\ a) \cdot f\ b - (f\ b - f\ a) \cdot f\ a = (f\ b - f\ a) \cdot f'\ x\ (b - a)$
apply (*rule mvt* [*OF* ‹$a < b$›, **where** $f = \lambda x.\ (f\ b - f\ a) \cdot f\ x$])
apply (*intro continuous_intros contf*)
using *derf* **apply** (*auto intro*: *has_derivative_inner_right*)
done
then obtain x **where** x: $x \in \{a<..<b\}$
$(f\ b - f\ a) \cdot f\ b - (f\ b - f\ a) \cdot f\ a = (f\ b - f\ a) \cdot f'\ x\ (b - a)$ **..**
show *?thesis*
proof (*cases* $f\ a = f\ b$)
case *False*
have $norm\ (f\ b - f\ a) * norm\ (f\ b - f\ a) = (norm\ (f\ b - f\ a))^2$
by (*simp add*: *power2_eq_square*)
also have $\ldots = (f\ b - f\ a) \cdot (f\ b - f\ a)$
unfolding *power2_norm_eq_inner* **..**
also have $\ldots = (f\ b - f\ a) \cdot f'\ x\ (b - a)$
using *x*(*2*) **by** (*simp only*: *inner_diff_right*)
also have $\ldots \leq norm\ (f\ b - f\ a) * norm\ (f'\ x\ (b - a))$
by (*rule norm_cauchy_schwarz*)
finally show *?thesis*
using *False x*(*1*)
by (*auto simp add*: *mult_left_cancel*)
next
case *True*
then show *?thesis*
using ‹$a < b$› **by** (*rule_tac x*=$(a + b)\ /2$ **in** *bexI*) *auto*
qed
qed

5.17.11 More general bound theorems

proposition *differentiable_bound_general*:
fixes $f :: real \Rightarrow 'a::real\_normed\_vector$
assumes $a < b$

```
    and f_cont: continuous_on {a..b} f
    and phi_cont: continuous_on {a..b} φ
    and f': ⋀x. a < x ⟹ x < b ⟹ (f has_vector_derivative f' x) (at x)
    and phi': ⋀x. a < x ⟹ x < b ⟹ (φ has_vector_derivative φ' x) (at x)
    and bnd: ⋀x. a < x ⟹ x < b ⟹ norm (f' x) ≤ φ' x
  shows norm (f b − f a) ≤ φ b − φ a
proof −
  {
    fix x assume x: a < x x < b
    have 0 ≤ norm (f' x) by simp
    also have ... ≤ φ' x using x by (auto intro!: bnd)
    finally have 0 ≤ φ' x .
  } note phi'_nonneg = this
  note f_tendsto = assms(2)[simplified continuous_on_def, rule_format]
  note phi_tendsto = assms(3)[simplified continuous_on_def, rule_format]
  {
    fix e::real assume e > 0
    define e2 where e2 = e / 2
    with ‹e > 0› have e2 > 0 by simp
    let ?le = λx1. norm (f x1 − f a) ≤ φ x1 − φ a + e * (x1 − a) + e
    define A where A = {x2. a ≤ x2 ∧ x2 ≤ b ∧ (∀x1∈{a ..< x2}. ?le x1)}
    have A_subset: A ⊆ {a..b} by (auto simp: A_def)
    {
      fix x2
      assume a: a ≤ x2 x2 ≤ b and le: ∀x1∈{a..<x2}. ?le x1
      have ?le x2 using ‹e > 0›
      proof cases
        assume x2 ≠ a with a have a < x2 by simp
        have at x2 within {a <..<x2} ≠ bot
          using ‹a < x2›
          by (auto simp: trivial_limit_within islimpt_in_closure)
        moreover
        have ((λx1. (φ x1 − φ a) + e * (x1 − a) + e) ⟶ (φ x2 − φ a) + e *
(x2 − a) + e) (at x2 within {a <..<x2})
            ((λx1. norm (f x1 − f a)) ⟶ norm (f x2 − f a)) (at x2 within {a
<..<x2})
          using a
          by (auto intro!: tendsto_eq_intros f_tendsto phi_tendsto
            intro: tendsto_within_subset[where S={a..b}])
        moreover
        have eventually (λx. x > a) (at x2 within {a <..<x2})
          by (auto simp: eventually_at_filter)
        hence eventually ?le (at x2 within {a <..<x2})
          unfolding eventually_at_filter
          by eventually_elim (insert le, auto)
        ultimately
        show ?thesis
          by (rule tendsto_le)
      qed simp
```

```
} note le_cont = this
have a ∈ A
  using assms by (auto simp: A_def)
hence [simp]: A ≠ {} by auto
have A_ivl: ⋀x1 x2. x2 ∈ A ⟹ x1 ∈ {a ..x2} ⟹ x1 ∈ A
  by (simp add: A_def)
have [simp]: bdd_above A by (auto simp: A_def)
define y where y = Sup A
have y ≤ b
  unfolding y_def
  by (simp add: cSup_le_iff) (simp add: A_def)
 have leI: ⋀x x1. a ≤ x1 ⟹ x ∈ A ⟹ x1 < x ⟹ ?le x1
   by (auto simp: A_def intro!: le_cont)
have y_all_le: ∀x1∈{a..<y}. ?le x1
  by (auto simp: y_def less_cSup_iff leI)
have a ≤ y
  by (metis ‹a ∈ A› ‹bdd_above A› cSup_upper y_def)
have y ∈ A
  using y_all_le ‹a ≤ y› ‹y ≤ b›
  by (auto simp: A_def)
hence A = {a .. y}
  using A_subset by (auto simp: subset_iff y_def cSup_upper intro: A_ivl)
from le_cont[OF ‹a ≤ y› ‹y ≤ b› y_all_le] have le_y: ?le y .
have y = b
proof (cases a = y)
  case True
  with ‹a < b› have y < b by simp
  with ‹a = y› f_cont phi_cont ‹e2 > 0›
  have 1: ∀F x in at y within {y..b}. dist (f x) (f y) < e2
   and 2: ∀F x in at y within {y..b}. dist (φ x) (φ y) < e2
    by (auto simp: continuous_on_def tendsto_iff)
  have 3: eventually (λx. y < x) (at y within {y..b})
    by (auto simp: eventually_at_filter)
  have 4: eventually (λx::real. x < b) (at y within {y..b})
    using _ ‹y < b›
    by (rule order_tendstoD) (auto intro!: tendsto_eq_intros)
  from 1 2 3 4
  have eventually_le: eventually (λx. ?le x) (at y within {y .. b})
  proof eventually_elim
    case (elim x1)
    have norm (f x1 - f a) = norm (f x1 - f y)
      by (simp add: ‹a = y›)
    also have norm (f x1 - f y) ≤ e2
      using elim ‹a = y› by (auto simp : dist_norm intro!:  less_imp_le)
    also have ... ≤ e2 + (φ x1 - φ a + e2 + e * (x1 - a))
      using ‹0 < e› elim
      by (intro add_increasing2[OF add_nonneg_nonneg order.refl])
        (auto simp: ‹a = y› dist_norm intro!: mult_nonneg_nonneg)
    also have ... = φ x1 - φ a + e * (x1 - a) + e
```

```
      by (simp add: e2_def)
    finally show ?le x1 .
  qed
  from this[unfolded eventually_at_topological] ‹?le y›
  obtain S where S: open S y ∈ S ⋀x. x∈S ⟹ x ∈ {y..b} ⟹ ?le x
    by metis
  from ‹open S› obtain d where d: ⋀x. dist x y < d ⟹ x ∈ S d > 0
    by (force simp: dist_commute open_dist ball_def dest!: bspec[OF _ ‹y ∈
S›])
  define d' where d' = min b (y + (d/2))
  have d' ∈ A
    unfolding A_def
  proof safe
    show a ≤ d' using ‹a = y› ‹0 < d› ‹y < b› by (simp add: d'_def)
    show d' ≤ b by (simp add: d'_def)
    fix x1
    assume x1 ∈ {a..<d'}
    hence x1 ∈ S x1 ∈ {y..b}
      by (auto simp: ‹a = y› d'_def dist_real_def intro!: d )
    thus ?le x1
      by (rule S)
  qed
  hence d' ≤ y
    unfolding y_def
    by (rule cSup_upper) simp
  then show y = b using ‹d > 0› ‹y < b›
    by (simp add: d'_def)
next
  case False
  with ‹a ≤ y› have a < y by simp
  show y = b
  proof (rule ccontr)
    assume y ≠ b
    hence y < b using ‹y ≤ b› by simp
    let ?F = at y within {y..<b}
    from f' phi'
    have (f has_vector_derivative f' y) ?F
      and (φ has_vector_derivative φ' y) ?F
      using ‹a < y› ‹y < b›
   by (auto simp add: at_within_open[of _ {a<..<b}] has_vector_derivative_def
       intro!: has_derivative_subset[where s={a<..<b} and t={y..<b}])
    hence ∀F x1 in ?F. norm (f x1 - f y - (x1 - y) *R f' y) ≤ e2 * |x1 - y|
        ∀F x1 in ?F. norm (φ x1 - φ y - (x1 - y) *R φ' y) ≤ e2 * |x1 - y|
      using ‹e2 > 0›
      by (auto simp: has_derivative_within_alt2 has_vector_derivative_def)
    moreover
    have ∀F x1 in ?F. y ≤ x1 ∀F x1 in ?F. x1 < b
      by (auto simp: eventually_at_filter)
    ultimately
```

```
    have ∀ F x1 in ?F. norm (f x1 − f y) ≤ (φ x1 − φ y) + e ∗ |x1 − y|
      (is ∀ F x1 in ?F. ?le′ x1)
    proof eventually_elim
      case (elim x1)
      from norm_triangle_ineq2[THEN order_trans, OF elim(1)]
      have norm (f x1 − f y) ≤ norm (f′ y) ∗ |x1 − y| + e2 ∗ |x1 − y|
        by (simp add: ac_simps)
      also have norm (f′ y) ≤ φ′ y using bnd ‹a < y› ‹y < b› by simp
      also have φ′ y ∗ |x1 − y| ≤ φ x1 − φ y + e2 ∗ |x1 − y|
        using elim by (simp add: ac_simps)
      finally
      have norm (f x1 − f y) ≤ φ x1 − φ y + e2 ∗ |x1 − y| + e2 ∗ |x1 − y|
        by (auto simp: mult_right_mono)
      thus ?case by (simp add: e2_def)
    qed
    moreover have ?le′ y by simp
    ultimately obtain S
    where S: open S y ∈ S ⋀x. x∈S ⟹ x ∈ {y..<b} ⟹ ?le′ x
      unfolding eventually_at_topological
      by metis
    from ‹open S› obtain d where d: ⋀x. dist x y < d ⟹ x ∈ S d > 0
      by (force simp: dist_commute open_dist ball_def dest!: bspec[OF _ ‹y ∈
S›])
    define d′ where d′ = min ((y + b)/2) (y + (d/2))
    have d′ ∈ A
      unfolding A_def
    proof safe
      show a ≤ d′ using ‹a < y› ‹0 < d› ‹y < b› by (simp add: d′_def)
      show d′ ≤ b using ‹y < b› by (simp add: d′_def min_def)
      fix x1
      assume x1: x1 ∈ {a..<d′}
      show ?le x1
      proof (cases x1 < y)
        case True
        then show ?thesis
          using ‹y ∈ A› local.leI x1 by auto
      next
        case False
        hence x1′: x1 ∈ S x1 ∈ {y..<b} using x1
          by (auto simp: d′_def dist_real_def intro!: d)
        have norm (f x1 − f a) ≤ norm (f x1 − f y) + norm (f y − f a)
          by (rule order_trans[OF _ norm_triangle_ineq]) simp
        also note S(3)[OF x1′]
        also note le_y
        finally show ?le x1
          using False by (auto simp: algebra_simps)
      qed
    qed
    hence d′ ≤ y
```

```
        unfolding y_def by (rule cSup_upper) simp
      thus False using ‹d > 0› ‹y < b›
        by (simp add: d'_def min_def split: if_split_asm)
    qed
  qed
  with le_y have norm (f b − f a) ≤ φ b − φ a + e * (b − a + 1)
    by (simp add: algebra_simps)
 } note * = this
 show ?thesis
 proof (rule field_le_epsilon)
  fix e::real assume e > 0
  then show norm (f b − f a) ≤ φ b − φ a + e
    using *[of e / (b − a + 1)] ‹a < b› by simp
 qed
qed

lemma differentiable_bound:
 fixes f :: 'a::real_normed_vector ⇒ 'b::real_normed_vector
 assumes convex S
  and derf: ⋀x. x∈S ⟹ (f has_derivative f' x) (at x within S)
  and B: ⋀x. x ∈ S ⟹ onorm (f' x) ≤ B
  and x: x ∈ S
  and y: y ∈ S
 shows norm (f x − f y) ≤ B * norm (x − y)
proof −
 let ?p = λu. x + u *R (y − x)
 let ?φ = λh. h * B * norm (x − y)
 have *: x + u *R (y − x) ∈ S if u ∈ {0..1} for u
 proof −
  have u *R y = u *R (y − x) + u *R x
    by (simp add: scale_right_diff_distrib)
  then show x + u *R (y − x) ∈ S
    using that ‹convex S› x y by (simp add: convex_alt)
      (metis pth_b(2) pth_c(1) scaleR_collapse)
 qed
 have ⋀z. z ∈ (λu. x + u *R (y − x)) ` {0..1} ⟹
       (f has_derivative f' z) (at z within (λu. x + u *R (y − x)) ` {0..1})
  by (auto intro: * has_derivative_subset [OF derf])
 then have continuous_on (?p ` {0..1}) f
  unfolding continuous_on_eq_continuous_within
  by (meson has_derivative_continuous)
 with * have 1: continuous_on {0 .. 1} (f ∘ ?p)
  by (intro continuous_intros)+
 {
  fix u::real assume u: u ∈{0 <..< 1}
  let ?u = ?p u
  interpret linear (f' ?u)
    using u by (auto intro!: has_derivative_linear derf *)
  have (f ∘ ?p has_derivative (f' ?u) ∘ (λu. 0 + u *R (y − x))) (at u within box
```

```
0 1)
    by (intro derivative_intros has_derivative_subset [OF derf]) (use u ∗ in auto)
   hence ((f ∘ ?p) has_vector_derivative f' ?u (y − x)) (at u)
     by (simp add: at_within_open[OF u open_greaterThanLessThan] scaleR
has_vector_derivative_def o_def)
  } note 2 = this
  have 3: continuous_on {0..1} ?φ
   by (rule continuous_intros)+
  have 4: (?φ has_vector_derivative B ∗ norm (x − y)) (at u) for u
   by (auto simp: has_vector_derivative_def intro!: derivative_eq_intros)
  {
   fix u::real assume u: u ∈{0 <..< 1}
   let ?u = ?p u
   interpret bounded_linear (f' ?u)
    using u by (auto intro!: has_derivative_bounded_linear derf ∗)
   have norm (f' ?u (y − x)) ≤ onorm (f' ?u) ∗ norm (y − x)
    by (rule onorm) (rule bounded_linear)
   also have onorm (f' ?u) ≤ B
    using u by (auto intro!: assms(3)[rule_format] ∗)
   finally have norm ((f' ?u) (y − x)) ≤ B ∗ norm (x − y)
    by (simp add: mult_right_mono norm_minus_commute)
  } note 5 = this
  have norm (f x − f y) = norm ((f ∘ (λu. x + u ∗R (y − x))) 1 − (f ∘ (λu. x
+ u ∗R (y − x))) 0)
   by (auto simp add: norm_minus_commute)
  also
  from differentiable_bound_general[OF zero_less_one 1, OF 3 2 4 5]
  have norm ((f ∘ ?p) 1 − (f ∘ ?p) 0) ≤ B ∗ norm (x − y)
   by simp
  finally show ?thesis .
qed

lemma field_differentiable_bound:
  fixes S :: 'a::real_normed_field set
  assumes cvs: convex S
     and df: ⋀z. z ∈ S ⟹ (f has_field_derivative f' z) (at z within S)
     and dn: ⋀z. z ∈ S ⟹ norm (f' z) ≤ B
     and x ∈ S  y ∈ S
   shows norm(f x − f y) ≤ B ∗ norm(x − y)
proof (rule differentiable_bound [OF cvs])
  show ⋀x. x ∈ S ⟹ (f has_derivative (∗) (f' x)) (at x within S)
   by (simp add: df has_field_derivative_imp_has_derivative)
  show ⋀x. x ∈ S ⟹ onorm ((∗) (f' x)) ≤ B
    by (metis (no_types, opaque_lifting) dn norm_mult onorm_le order.refl or-
der_trans)
qed (use assms in auto)

lemma
  differentiable_bound_segment:
```

fixes f::$'a$::*real_normed_vector* $\Rightarrow$ $'b$::*real_normed_vector*
assumes $\bigwedge t.\ t \in \{0..1\} \Longrightarrow x0 + t *_R a \in G$
assumes f': $\bigwedge x.\ x \in G \Longrightarrow (f\ has\_derivative\ f'\ x)\ (at\ x\ within\ G)$
assumes B: $\bigwedge x.\ x \in \{0..1\} \Longrightarrow onorm\ (f'\ (x0 + x *_R a)) \leq B$
shows $norm\ (f\ (x0 + a) - f\ x0) \leq norm\ a * B$
proof −
let $?G = (\lambda x.\ x0 + x *_R a)\ `\ \{0..1\}$
have $?G = (+)\ x0\ `\ (\lambda x.\ x *_R a)\ `\ \{0..1\}$ **by** *auto*
also have *convex* ...
by (*intro convex_translation convex_scaled convex_real_interval*)
finally have *convex* $?G$ **.**
moreover have $?G \subseteq G\ x0 \in ?G\ x0 + a \in ?G$ **using** *assms* **by** (*auto intro*: *image_eqI*[**where** x=*1*])
ultimately show *?thesis*
using *has_derivative_subset*[*OF* f' ‹$?G \subseteq G$›] B
differentiable_bound[*of* $(\lambda x.\ x0 + x *_R a)\ `\ \{0..1\}\ f\ f'\ B\ x0 + a\ x0$]
by (*force simp*: *ac_simps*)
qed

lemma *differentiable_bound_linearization*:
fixes f::$'a$::*real_normed_vector* $\Rightarrow$ $'b$::*real_normed_vector*
assumes S: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow a + t *_R (b - a) \in S$
assumes f'[*derivative_intros*]: $\bigwedge x.\ x \in S \Longrightarrow (f\ has\_derivative\ f'\ x)\ (at\ x\ within\ S)$
assumes B: $\bigwedge x.\ x \in S \Longrightarrow onorm\ (f'\ x - f'\ x0) \leq B$
assumes $x0 \in S$
shows $norm\ (f\ b - f\ a - f'\ x0\ (b - a)) \leq norm\ (b - a) * B$
proof −
define g **where** [*abs_def*]: $g\ x = f\ x - f'\ x0\ x$ **for** x
have g: $\bigwedge x.\ x \in S \Longrightarrow (g\ has\_derivative\ (\lambda i.\ f'\ x\ i - f'\ x0\ i))\ (at\ x\ within\ S)$
unfolding *g_def* **using** *assms*
by (*auto intro*!: *derivative_eq_intros*
bounded_linear.has_derivative[*OF has_derivative_bounded_linear*, *OF* f'])
from B **have** $\forall x{\in}\{0..1\}.\ onorm\ (\lambda i.\ f'\ (a + x *_R (b - a))\ i - f'\ x0\ i) \leq B$
using *assms* **by** (*auto simp*: *fun_diff_def*)
with *differentiable_bound_segment*[*OF S g*] ‹$x0 \in S$›
show *?thesis*
by (*simp add*: *g_def field_simps linear_diff*[*OF has_derivative_linear*[*OF* f']])
qed

lemma *vector_differentiable_bound_linearization*:
fixes f::*real* $\Rightarrow$ $'b$::*real_normed_vector*
assumes f': $\bigwedge x.\ x \in S \Longrightarrow (f\ has\_vector\_derivative\ f'\ x)\ (at\ x\ within\ S)$
assumes *closed_segment* $a\ b \subseteq S$
assumes B: $\bigwedge x.\ x \in S \Longrightarrow norm\ (f'\ x - f'\ x0) \leq B$
assumes $x0 \in S$
shows $norm\ (f\ b - f\ a - (b - a) *_R f'\ x0) \leq norm\ (b - a) * B$
using *assms*
by (*intro differentiable_bound_linearization*[*of a b S f* $\lambda x\ h.\ h *_R f'\ x\ x0\ B$])

(*force simp*: *closed_segment_real_eq has_vector_derivative_def*
scaleR_diff_right[*symmetric*] *mult.commute*[*of B*]
intro!: *onorm_le mult_left_mono*)+

In particular.

lemma *has_derivative_zero_constant*:
fixes *f* :: *'a*::*real_normed_vector* ⇒ *'b*::*real_normed_vector*
assumes *convex s*
and ⋀*x*. *x* ∈ *s* ⟹ (*f has_derivative* (λ*h*. *0*)) (*at x within s*)
shows ∃ *c*. ∀ *x*∈*s*. *f x* = *c*
proof −
{ fix *x y* **assume** *x* ∈ *s y* ∈ *s*
then have *norm* (*f x* − *f y*) ≤ *0* ∗ *norm* (*x* − *y*)
using *assms* **by** (*intro differentiable_bound*[*of s*]) (*auto simp*: *onorm_zero*)
then have *f x* = *f y*
by *simp* **}**
then show *?thesis*
by *metis*
qed

lemma *has_field_derivative_zero_constant*:
assumes *convex s* ⋀*x*. *x* ∈ *s* ⟹ (*f has_field_derivative 0*) (*at x within s*)
shows ∃ *c*. ∀ *x*∈*s*. *f* (*x*) = (*c* :: *'a* :: *real_normed_field*)
proof (*rule has_derivative_zero_constant*)
have *A*: (∗) *0* = (λ_. *0* :: *'a*) **by** (*intro ext*) *simp*
fix *x* **assume** *x* ∈ *s* **thus** (*f has_derivative* (λ*h*. *0*)) (*at x within s*)
using *assms*(*2*)[*of x*] **by** (*simp add*: *has_field_derivative_def A*)
qed *fact*

lemma
has_vector_derivative_zero_constant:
assumes *convex s*
assumes ⋀*x*. *x* ∈ *s* ⟹ (*f has_vector_derivative 0*) (*at x within s*)
obtains *c* **where** ⋀*x*. *x* ∈ *s* ⟹ *f x* = *c*
using *has_derivative_zero_constant*[*of s f*] *assms*
by (*auto simp*: *has_vector_derivative_def*)

lemma *has_derivative_zero_unique*:
fixes *f* :: *'a*::*real_normed_vector* ⇒ *'b*::*real_normed_vector*
assumes *convex s*
and ⋀*x*. *x* ∈ *s* ⟹ (*f has_derivative* (λ*h*. *0*)) (*at x within s*)
and *x* ∈ *s y* ∈ *s*
shows *f x* = *f y*
using *has_derivative_zero_constant*[*OF assms*(*1*,*2*)] *assms*(*3*−) **by** *force*

lemma *has_derivative_zero_unique_connected*:
fixes *f* :: *'a*::*real_normed_vector* ⇒ *'b*::*real_normed_vector*
assumes *open s connected s*
assumes *f*: ⋀*x*. *x* ∈ *s* ⟹ (*f has_derivative* (λ*x*. *0*)) (*at x*)

assumes $x \in s$ $y \in s$
shows $f\ x = f\ y$
proof (*rule connected_local_const*[**where** *f*=*f*, *OF* ‹*connected s*› ‹*x*∈*s*› ‹*y*∈*s*›])
show $\forall a{\in}s.$ *eventually* $(\lambda b.\ f\ a = f\ b)$ (*at a within s*)
proof
fix *a* **assume** $a \in s$
with ‹*open s*› **obtain** *e* **where** $0 < e$ *ball a e* $\subseteq s$
by (*rule openE*)
then have $\exists c.\ \forall x{\in}$*ball a e*. $f\ x = c$
by (*intro has_derivative_zero_constant*)
(*auto simp*: *at_within_open*[*OF* _ *open_ball*] *f*)
with ‹*0*<*e*› **have** $\forall x{\in}$*ball a e*. $f\ a = f\ x$
by *auto*
then show *eventually* $(\lambda b.\ f\ a = f\ b)$ (*at a within s*)
using ‹*0*<*e*› **unfolding** *eventually_at_topological*
by (*intro exI*[*of* _ *ball a e*]) *auto*
qed
qed

5.17.12 Differentiability of inverse function (most basic form)

lemma *has_derivative_inverse_basic*:
fixes *f* :: *'a*::*real_normed_vector* ⇒ *'b*::*real_normed_vector*
assumes *derf*: (*f has_derivative f'*) (*at* (*g y*))
and *ling'*: *bounded_linear g'*
and $g' \circ f' = id$
and *contg*: *continuous* (*at y*) *g*
and *open T*
and $y \in T$
and *fg*: $\bigwedge z.\ z \in T \Longrightarrow f\ (g\ z) = z$
shows (*g has_derivative g'*) (*at y*)
proof −
interpret *f'*: *bounded_linear f'*
using *assms* **unfolding** *has_derivative_def* **by** *auto*
interpret *g'*: *bounded_linear g'*
using *assms* **by** *auto*
obtain *C* **where** *C*: $0 < C$ $\bigwedge x.$ *norm* $(g'\ x) \leq$ *norm* $x * C$
using *bounded_linear.pos_bounded*[*OF assms*(*2*)] **by** *blast*
have *lem1*: $\forall e{>}0.\ \exists d{>}0.\ \forall z.$
norm $(z - y) < d \longrightarrow$ *norm* $(g\ z - g\ y - g'(z - y)) \leq e *$ *norm* $(g\ z - g\ y)$
proof (*intro allI impI*)
fix *e* :: *real*
assume $e > 0$
with *C*(*1*) **have** ∗: $e\ /\ C > 0$ **by** *auto*
obtain *d0* **where** $0 < d0$ **and** *d0*:
$\bigwedge u.$ *norm* $(u - g\ y) < d0 \Longrightarrow$ *norm* $(f\ u - f\ (g\ y) - f'\ (u - g\ y)) \leq e\ /$
$C *$ *norm* $(u - g\ y)$
using *derf* ∗ **unfolding** *has_derivative_at_alt* **by** *blast*
obtain *d1* **where** $0 < d1$ **and** *d1*: $\bigwedge x.$ $[\![0 < dist\ x\ y;\ dist\ x\ y < d1]\!] \Longrightarrow$ *dist*

```
(g x) (g y) < d0
      using contg ‹0 < d0› unfolding continuous_at Lim_at by blast
    obtain d2 where 0 < d2 and d2: ⋀u. dist u y < d2 ⟹ u ∈ T
      using ‹open T› ‹y ∈ T› unfolding open_dist by blast
    obtain d where d: 0 < d d < d1 d < d2
      using field_lbound_gt_zero[OF ‹0 < d1› ‹0 < d2›] by blast
    show ∃ d>0. ∀ z. norm (z − y) < d ⟶ norm (g z − g y − g' (z − y)) ≤ e *
norm (g z − g y)
    proof (intro exI allI impI conjI)
      fix z
      assume as: norm (z − y) < d
      then have z ∈ T
        using d2 d unfolding dist_norm by auto
      have norm (g z − g y − g' (z − y)) ≤ norm (g' (f (g z) − y − f' (g z − g
y)))
        unfolding g'.diff f'.diff
       unfolding assms(3)[unfolded o_def id_def, THEN fun_cong] fg[OF ‹z∈T›]
        by (simp add: norm_minus_commute)
      also have ... ≤ norm (f (g z) − y − f' (g z − g y)) * C
        by (rule C(2))
      also have ... ≤ (e / C) * norm (g z − g y) * C
      proof −
        have norm (g z − g y) < d0
          by (metis as cancel_comm_monoid_add_class.diff_cancel d(2) ‹0 < d0›
d1 diff_gt_0_iff_gt diff_strict_mono dist_norm dist_self zero_less_dist_iff)
        then show ?thesis
          by (metis C(1) ‹y ∈ T› d0 fg mult_le_cancel_iff1)
      qed
      also have ... ≤ e * norm (g z − g y)
        using C by (auto simp add: field_simps)
      finally show norm (g z − g y − g' (z − y)) ≤ e * norm (g z − g y)
        by simp
    qed (use d in auto)
  qed
  have *: (0::real) < 1 / 2
    by auto
  obtain d where 0 < d and d:
      ⋀z. norm (z − y) < d ⟹ norm (g z − g y − g' (z − y)) ≤ 1/2 * norm (g
z − g y)
    using lem1 * by blast
  define B where B = C * 2
  have B > 0
    unfolding B_def using C by auto
  have lem2: norm (g z − g y) ≤ B * norm (z − y) if z: norm(z − y) < d for z
  proof −
    have norm (g z − g y) ≤ norm(g' (z − y)) + norm ((g z − g y) − g'(z − y))
      by (rule norm_triangle_sub)
    also have ... ≤ norm (g' (z − y)) + 1 / 2 * norm (g z − g y)
      by (rule add_left_mono) (use d z in auto)
```

also have ... $\leq$ *norm* $(z - y) * C + 1 / 2 *$ *norm* $(g\ z - g\ y)$
by (*rule add_right_mono*) (*use C* **in** *auto*)
finally show *norm* $(g\ z - g\ y) \leq B *$ *norm* $(z - y)$
unfolding *B_def*
by (*auto simp add*: *field_simps*)
qed
show *?thesis*
unfolding *has_derivative_at_alt*
proof (*intro conjI assms allI impI*)
fix *e* :: *real*
assume $e > 0$
then have $*$: $e / B > 0$ **by** (*metis* ‹$B > 0$› *divide_pos_pos*)
obtain d' **where** $0 < d'$ **and** d':
$\bigwedge z.$ *norm* $(z - y) < d' \Longrightarrow$ *norm* $(g\ z - g\ y - g'\ (z - y)) \leq e / B *$ *norm* $(g\ z - g\ y)$
using *lem1* $*$ **by** *blast*
obtain *k* **where** *k*: $0 < k\ k < d\ k < d'$
using *field_lbound_gt_zero*[*OF* ‹$0 < d$› ‹$0 < d'$›] **by** *blast*
show $\exists\, d{>}0.\ \forall\, ya.$ *norm* $(ya - y) < d \longrightarrow$ *norm* $(g\ ya - g\ y - g'\ (ya - y)) \leq e *$ *norm* $(ya - y)$
proof (*intro exI allI impI conjI*)
fix *z*
assume *as*: *norm* $(z - y) < k$
then have *norm* $(g\ z - g\ y - g'\ (z - y)) \leq e / B *$ *norm*$(g\ z - g\ y)$
using $d'\ k$ **by** *auto*
also have ... $\leq e *$ *norm* $(z - y)$
unfolding *times_divide_eq_left pos_divide_le_eq*[*OF* ‹$B{>}0$›]
using *lem2*[*of z*] *k as* ‹$e > 0$›
by (*auto simp add*: *field_simps*)
finally show *norm* $(g\ z - g\ y - g'\ (z - y)) \leq e *$ *norm* $(z - y)$
by *simp*
qed (*use k* **in** *auto*)
qed
qed

Inverse function theorem for complex derivatives

lemma *has_field_derivative_inverse_basic*:
shows *DERIV f* $(g\ y)$:> $f' \Longrightarrow$
$f' \neq 0 \Longrightarrow$
continuous $(at\ y)\ g \Longrightarrow$
open $t \Longrightarrow$
$y \in t \Longrightarrow$
$(\bigwedge z.\ z \in t \Longrightarrow f\ (g\ z) = z)$
$\Longrightarrow$ *DERIV* $g\ y$:> *inverse* (f')
unfolding *has_field_derivative_def*
by (*rule has_derivative_inverse_basic*) (*auto simp*: *bounded_linear_mult_right*)

Simply rewrite that based on the domain point x.

lemma *has_derivative_inverse_basic_x*:

fixes $f :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$
assumes $(f\ has\_derivative\ f')\ (at\ x)$
and $bounded\_linear\ g'$
and $g' \circ f' = id$
and $continuous\ (at\ (f\ x))\ g$
and $g\ (f\ x) = x$
and $open\ T$
and $f\ x \in T$
and $\bigwedge y.\ y \in T \Longrightarrow f\ (g\ y) = y$
shows $(g\ has\_derivative\ g')\ (at\ (f\ x))$
by $(rule\ has\_derivative\_inverse\_basic)\ (use\ assms\ \textbf{in}\ auto)$

This is the version in Dieudonne', assuming continuity of f and g.

lemma *has_derivative_inverse_dieudonne*:
fixes $f :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$
assumes $open\ S$
and fS: $open\ (f\ `\ S)$
and A: $continuous\_on\ S\ f\ continuous\_on\ (f\ `\ S)\ g$
$\bigwedge x.\ x \in S \Longrightarrow g\ (f\ x) = x\ x \in S$
and B: $(f\ has\_derivative\ f')\ (at\ x)\ bounded\_linear\ g'\ g' \circ f' = id$
shows $(g\ has\_derivative\ g')\ (at\ (f\ x))$
using $A\ fS\ continuous\_on\_eq\_continuous\_at$
by $(intro\ has\_derivative\_inverse\_basic\_x[OF\ B\ \_\ \_\ fS])\ force+$

Here's the simplest way of not assuming much about g.

proposition *has_derivative_inverse*:
fixes $f :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$
assumes $compact\ S$
and $x \in S$
and fx: $f\ x \in interior\ (f\ `\ S)$
and $continuous\_on\ S\ f$
and gf: $\bigwedge y.\ y \in S \Longrightarrow g\ (f\ y) = y$
and B: $(f\ has\_derivative\ f')\ (at\ x)\ bounded\_linear\ g'\ g' \circ f' = id$
shows $(g\ has\_derivative\ g')\ (at\ (f\ x))$
proof −
have $*$: $\bigwedge y.\ y \in interior\ (f\ `\ S) \Longrightarrow f\ (g\ y) = y$
by $(metis\ gf\ image\_iff\ interior\_subset\ subsetCE)$
show *?thesis*
using $assms * continuous\_on\_interior\ continuous\_on\_inv\ fx$
by $(intro\ has\_derivative\_inverse\_basic\_x[OF\ B,\ \textbf{where}\ T = interior\ (f`S)])$
blast+
qed

Invertible derivative continuous at a point implies local injectivity. It's only for this we need continuity of the derivative, except of course if we want the fact that the inverse derivative is also continuous. So if we know for some other reason that the inverse function exists, it's OK.

proposition *has_derivative_locally_injective*:

```
  fixes f :: 'n::euclidean_space ⇒ 'm::euclidean_space
  assumes a ∈ S
      and open S
      and bling: bounded_linear g'
      and g' ∘ f' a = id
      and derf: ⋀x. x ∈ S ⟹ (f has_derivative f' x) (at x)
      and ⋀e. e > 0 ⟹ ∃ d>0. ∀ x. dist a x < d ⟶ onorm (λv. f' x v − f' a v)
< e
  obtains r where r > 0 ball a r ⊆ S inj_on f (ball a r)
proof −
  interpret bounded_linear g'
    using assms by auto
  note f'g' = assms(4)[unfolded id_def o_def,THEN cong]
  have g' (f' a (∑ Basis)) = (∑ Basis) (∑ Basis) ≠ (0::'n)
    using f'g' by auto
  then have ∗: 0 < onorm g'
    unfolding onorm_pos_lt[OF assms(3)]
    by fastforce
  define k where k = 1 / onorm g' / 2
  have ∗: k > 0
    unfolding k_def using ∗ by auto
  obtain d1 where d1:
      0 < d1
      ⋀x. dist a x < d1 ⟹ onorm (λv. f' x v − f' a v) < k
    using assms(6) ∗ by blast
  from ‹open S› obtain d2 where d2 > 0 ball a d2 ⊆ S
    using ‹a∈S› ..
  obtain d2 where d2: 0 < d2 ball a d2 ⊆ S
    using ‹0 < d2› ‹ball a d2 ⊆ S› by blast
  obtain d where d: 0 < d d < d1 d < d2
    using field_lbound_gt_zero[OF d1(1) d2(1)] by blast
  show ?thesis
  proof
    show 0 < d by (fact d)
    show ball a d ⊆ S
      using ‹d < d2› ‹ball a d2 ⊆ S› by auto
    show inj_on f (ball a d)
    unfolding inj_on_def
    proof (intro strip)
      fix x y
      assume as: x ∈ ball a d y ∈ ball a d f x = f y
      define ph where [abs_def]: ph w = w − g' (f w − f x) for w
      have ph':ph = g' ∘ (λw. f' a w − (f w − f x))
        unfolding ph_def o_def by (simp add: diff f'g')
      have norm (ph x − ph y) ≤ (1 / 2) ∗ norm (x − y)
      proof (rule differentiable_bound[OF convex_ball _ _ as(1−2)])
        fix u
        assume u: u ∈ ball a d
        then have u ∈ S
```

using *d d2* **by** *auto*
have ∗: $(\lambda v.\ v - g'\ (f'\ u\ v)) = g' \circ (\lambda w.\ f'\ a\ w - f'\ u\ w)$
unfolding *o_def* **and** *diff*
using *f'g'* **by** *auto*
have *blin*: *bounded_linear* $(f'\ a)$
using ‹$a \in S$› *derf* **by** *blast*
show $(ph\ has\_derivative\ (\lambda v.\ v - g'\ (f'\ u\ v)))\ (at\ u\ within\ ball\ a\ d)$
unfolding $ph' * comp\_def$
by (*rule* ‹$u \in S$› *derivative_eq_intros has_derivative_at_withinI* [*OF derf*] *bounded_linear.has_derivative* [*OF blin*] *bounded_linear.has_derivative* [*OF bling*] |*simp*)+
have ∗∗: *bounded_linear* $(\lambda x.\ f'\ u\ x - f'\ a\ x)$ *bounded_linear* $(\lambda x.\ f'\ a\ x - f'\ u\ x)$
using ‹$u \in S$› *blin bounded_linear_sub derf* **by** *auto*
then have *onorm* $(\lambda v.\ v - g'\ (f'\ u\ v)) \leq onorm\ g' * onorm\ (\lambda w.\ f'\ a\ w - f'\ u\ w)$
by (*simp add*: ∗ *bounded_linear_axioms onorm_compose*)
also have $\ldots \leq onorm\ g' * k$
apply (*rule mult_left_mono*)
using *d1*(*2*)[*of u*]
using *onorm_neg*[**where** $f{=}\lambda x.\ f'\ u\ x - f'\ a\ x$] *d u onorm_pos_le*[*OF bling*]
apply (*auto simp*: *algebra_simps*)
done
also have $\ldots \leq 1\ /\ 2$
unfolding *k_def* **by** *auto*
finally show $onorm\ (\lambda v.\ v - g'\ (f'\ u\ v)) \leq 1\ /\ 2$ **.**
qed
moreover have $norm\ (ph\ y - ph\ x) = norm\ (y - x)$
by (*simp add*: *as*(*3*) *ph_def*)
ultimately show $x = y$
unfolding *norm_minus_commute* **by** *auto*
qed
qed
qed

5.17.13 Uniformly convergent sequence of derivatives

lemma *has_derivative_sequence_lipschitz_lemma*:
fixes f :: $nat \Rightarrow$ 'a::*real_normed_vector* $\Rightarrow$ 'b::*real_normed_vector*
assumes *convex S*
and *derf*: $\bigwedge n\ x.\ x \in S \Longrightarrow ((f\ n)\ has\_derivative\ (f'\ n\ x))\ (at\ x\ within\ S)$
and *nle*: $\bigwedge n\ x\ h.\ [\![n \geq N;\ x \in S]\!] \Longrightarrow norm\ (f'\ n\ x\ h - g'\ x\ h) \leq e * norm\ h$
and $0 \leq e$
shows $\forall m \geq N.\ \forall n \geq N.\ \forall x \in S.\ \forall y \in S.\ norm\ ((f\ m\ x - f\ n\ x) - (f\ m\ y - f\ n\ y)) \leq 2 * e * norm\ (x - y)$
proof *clarify*
fix *m n x y*
assume *as*: $N \leq m\ N \leq n\ x \in S\ y \in S$

```
  show norm ((f m x − f n x) − (f m y − f n y)) ≤ 2 ∗ e ∗ norm (x − y)
  proof (rule differentiable_bound[where f'=λx h. f' m x h − f' n x h, OF ‹convex
S› _ _ as(3−4)])
    fix x
    assume x ∈ S
     show ((λa. f m a − f n a) has_derivative (λh. f' m x h − f' n x h)) (at x
within S)
      by (rule derivative_intros derf ‹x∈S›)+
    show onorm (λh. f' m x h − f' n x h) ≤ 2 ∗ e
    proof (rule onorm_bound)
      fix h
      have norm (f' m x h − f' n x h) ≤ norm (f' m x h − g' x h) + norm (f' n
x h − g' x h)
        using norm_triangle_ineq[of f' m x h − g' x h − f' n x h + g' x h]
        by (auto simp add: algebra_simps norm_minus_commute)
      also have ... ≤ e ∗ norm h + e ∗ norm h
        using nle[OF ‹N ≤ m› ‹x ∈ S›, of h] nle[OF ‹N ≤ n› ‹x ∈ S›, of h]
        by (auto simp add: field_simps)
      finally show norm (f' m x h − f' n x h) ≤ 2 ∗ e ∗ norm h
        by auto
    qed (simp add: ‹0 ≤ e›)
  qed
qed

lemma has_derivative_sequence_Lipschitz:
  fixes f :: nat ⇒ 'a::real_normed_vector ⇒ 'b::real_normed_vector
  assumes convex S
    and ⋀n x. x ∈ S ⟹ ((f n) has_derivative (f' n x)) (at x within S)
    and nle: ⋀e. e > 0 ⟹ ∀F n in sequentially. ∀x∈S. ∀h. norm (f' n x h − g'
x h) ≤ e ∗ norm h
    and e > 0
  shows ∃N. ∀m≥N. ∀n≥N. ∀x∈S. ∀y∈S.
    norm ((f m x − f n x) − (f m y − f n y)) ≤ e ∗ norm (x − y)
proof −
  have ∗: 2 ∗ (e/2) = e
    using ‹e > 0› by auto
  obtain N where ∀n≥N. ∀x∈S. ∀h. norm (f' n x h − g' x h) ≤ (e/2) ∗ norm
h
    using nle ‹e > 0›
    unfolding eventually_sequentially
    by (metis less_divide_eq_numeral1(1) mult_zero_left)
  then show ∃N. ∀m≥N. ∀n≥N. ∀x∈S. ∀y∈S. norm (f m x − f n x − (f m y
− f n y)) ≤ e ∗ norm (x − y)
    apply (rule_tac x=N in exI)
   apply (rule has_derivative_sequence_lipschitz_lemma[where e=e/2, unfolded
∗])
    using assms ‹e > 0›
    apply auto
    done
```

qed

proposition *has_derivative_sequence*:
fixes f :: *nat* ⇒ 'a::*real_normed_vector* ⇒ 'b::*banach*
assumes *convex S*
and *derf*: ⋀*n x*. *x* ∈ *S* ⟹ ((*f n*) *has_derivative* (*f' n x*)) (*at x within S*)
and *nle*: ⋀*e*. *e* > *0* ⟹ ∀$_F$ *n in sequentially*. ∀ *x*∈*S*. ∀ *h*. *norm* (*f' n x h* − *g' x h*) ≤ *e* ∗ *norm h*
and *x0* ∈ *S*
and *lim*: ((λ*n*. *f n x0*) ⟶ *l*) *sequentially*
shows ∃ *g*. ∀ *x*∈*S*. (λ*n*. *f n x*) ⟶ *g x* ∧ (*g has_derivative g'*(*x*)) (*at x within S*)
proof −
have *lem1*: ⋀*e*. *e* > *0* ⟹ ∃ *N*. ∀ *m*≥*N*. ∀ *n*≥*N*. ∀ *x*∈*S*. ∀ *y*∈*S*.
norm ((*f m x* − *f n x*) − (*f m y* − *f n y*)) ≤ *e* ∗ *norm* (*x* − *y*)
using *assms*(*1*,*2*,*3*) **by** (*rule has_derivative_sequence_Lipschitz*)
have ∃ *g*. ∀ *x*∈*S*. ((λ*n*. *f n x*) ⟶ *g x*) *sequentially*
proof (*intro ballI bchoice*)
fix *x*
assume *x* ∈ *S*
show ∃ *y*. (λ*n*. *f n x*) ⟶ *y*
unfolding *convergent_eq_Cauchy*
proof (*cases x* = *x0*)
case *True*
then show *Cauchy* (λ*n*. *f n x*)
using *LIMSEQ_imp_Cauchy*[*OF lim*] **by** *auto*
next
case *False*
show *Cauchy* (λ*n*. *f n x*)
unfolding *Cauchy_def*
proof (*intro allI impI*)
fix *e* :: *real*
assume *e* > *0*
hence ∗: *e* / *2* > *0 e* / *2* / *norm* (*x* − *x0*) > *0* **using** *False* **by** *auto*
obtain *M* **where** *M*: ∀ *m*≥*M*. ∀ *n*≥*M*. *dist* (*f m x0*) (*f n x0*) < *e* / *2*
using *LIMSEQ_imp_Cauchy*[*OF lim*] ∗ **unfolding** *Cauchy_def* **by** *blast*
obtain *N* **where** *N*:
∀ *m*≥*N*. ∀ *n*≥*N*.
∀ *u*∈*S*. ∀ *y*∈*S*. *norm* (*f m u* − *f n u* − (*f m y* − *f n y*)) ≤
e / *2* / *norm* (*x* − *x0*) ∗ *norm* (*u* − *y*)
using *lem1* ∗(*2*) **by** *blast*
show ∃ *M*. ∀ *m*≥*M*. ∀ *n*≥*M*. *dist* (*f m x*) (*f n x*) < *e*
proof (*intro exI allI impI*)
fix *m n*
assume *as*: *max M N* ≤*m max M N*≤*n*
have *dist* (*f m x*) (*f n x*) ≤ *norm* (*f m x0* − *f n x0*) + *norm* (*f m x* − *f n x* − (*f m x0* − *f n x0*))
unfolding *dist_norm*
by (*rule norm_triangle_sub*)

```
        also have ... ≤ norm (f m x0 − f n x0) + e / 2
          using N ‹x∈S› ‹x0∈S› as False by fastforce
        also have ... < e / 2 + e / 2
         by (rule add_strict_right_mono) (use as M in ‹auto simp: dist_norm›)
        finally show dist (f m x) (f n x) < e
          by auto
       qed
      qed
    qed
  qed
  then obtain g where g: ∀x∈S. (λn. f n x) ⟶ g x ..
  have lem2: ∃N. ∀n≥N. ∀x∈S. ∀y∈S. norm ((f n x − f n y) − (g x − g y)) ≤
e ∗ norm (x − y) if e > 0 for e
  proof −
    obtain N where
      N: ∀m≥N. ∀n≥N. ∀x∈S. ∀y∈S. norm (f m x − f n x − (f m y − f n y))
≤ e ∗ norm (x − y)
      using lem1 ‹e > 0› by blast
    show ∃N. ∀n≥N. ∀x∈S. ∀y∈S. norm (f n x − f n y − (g x − g y)) ≤ e ∗
norm (x − y)
    proof (intro exI ballI allI impI)
      fix n x y
      assume as: N ≤ n x ∈ S y ∈ S
      have ((λm. norm (f n x − f n y − (f m x − f m y))) ⟶ norm (f n x − f
n y − (g x − g y))) sequentially
        by (intro tendsto_intros g[rule_format] as)
      moreover have eventually (λm. norm (f n x − f n y − (f m x − f m y)) ≤
e ∗ norm (x − y)) sequentially
        unfolding eventually_sequentially
      proof (intro exI allI impI)
        fix m
        assume N ≤ m
        then show norm (f n x − f n y − (f m x − f m y)) ≤ e ∗ norm (x − y)
          using N as by (auto simp add: algebra_simps)
      qed
      ultimately show norm (f n x − f n y − (g x − g y)) ≤ e ∗ norm (x − y)
        by (simp add: tendsto_upperbound)
    qed
  qed
  have ∀x∈S. ((λn. f n x) ⟶ g x) sequentially ∧ (g has_derivative g′ x) (at x
within S)
    unfolding has_derivative_within_alt2
  proof (intro ballI conjI allI impI)
    fix x
    assume x ∈ S
    then show (λn. f n x) ⟶ g x
      by (simp add: g)
    have tog′: (λn. f′ n x u) ⟶ g′ x u for u
      unfolding filterlim_def le_nhds_metric_le eventually_filtermap dist_norm
```

```
    proof (intro allI impI)
      fix e :: real
      assume e > 0
      show eventually (λn. norm (f' n x u − g' x u) ≤ e) sequentially
      proof (cases u = 0)
        case True
        have eventually (λn. norm (f' n x u − g' x u) ≤ e * norm u) sequentially
          using nle ‹0 < e› ‹x ∈ S› by (fast elim: eventually_mono)
        then show ?thesis
          using ‹u = 0› ‹0 < e› by (auto elim: eventually_mono)
      next
        case False
        with ‹0 < e› have 0 < e / norm u by simp
        then have eventually (λn. norm (f' n x u − g' x u) ≤ e / norm u * norm
u) sequentially
          using nle ‹x ∈ S› by (fast elim: eventually_mono)
        then show ?thesis
          using ‹u ≠ 0› by simp
      qed
    qed
    show bounded_linear (g' x)
    proof
      fix x' y z :: 'a
      fix c :: real
    note lin = assms(2)[rule_format,OF ‹x∈S›,THEN has_derivative_bounded_linear]
      have (λn. f' n x (c *R x')) ⟶ c *R g' x x'
        unfolding lin[THEN bounded_linear.linear, THEN linear_cmul]
        by (intro tendsto_intros tog')
      then show g' x (c *R x') = c *R g' x x'
        using LIMSEQ_unique tog' by blast
      have (λn. f' n x (y + z)) ⟶ g' x y + g' x z
        unfolding lin[THEN bounded_linear.linear, THEN linear_add]
        by (simp add: tendsto_add tog')
      then show g' x (y + z) = g' x y + g' x z
        using LIMSEQ_unique tog' by blast
      obtain N where N: ∀ h. norm (f' N x h − g' x h) ≤ 1 * norm h
          using nle ‹x ∈ S› unfolding eventually_sequentially by (fast intro:
zero_less_one)
      have bounded_linear (f' N x)
        using derf ‹x ∈ S› by fast
      from bounded_linear.bounded [OF this]
      obtain K where K: ∀ h. norm (f' N x h) ≤ norm h * K ..
      {
        fix h
        have norm (g' x h) = norm (f' N x h − (f' N x h − g' x h))
          by simp
        also have ... ≤ norm (f' N x h) + norm (f' N x h − g' x h)
          by (rule norm_triangle_ineq4)
        also have ... ≤ norm h * K + 1 * norm h
```

```
      using N K by (fast intro: add_mono)
     finally have norm (g' x h) ≤ norm h * (K + 1)
      by (simp add: ring_distribs)
    }
    then show ∃K. ∀h. norm (g' x h) ≤ norm h * K by fast
   qed
   show eventually (λy. norm (g y - g x - g' x (y - x)) ≤ e * norm (y - x))
(at x within S)
    if e > 0 for e
   proof -
    have *: e / 3 > 0
     using that by auto
    obtain N1 where N1: ∀n≥N1. ∀x∈S. ∀h. norm (f' n x h - g' x h) ≤ e /
3 * norm h
     using nle * unfolding eventually_sequentially by blast
    obtain N2 where
      N2[rule_format]: ∀n≥N2. ∀x∈S. ∀y∈S. norm (f n x - f n y - (g x - g
y)) ≤ e / 3 * norm (x - y)
     using lem2 * by blast
    let ?N = max N1 N2
    have eventually (λy. norm (f ?N y - f ?N x - f' ?N x (y - x)) ≤ e / 3 *
norm (y - x)) (at x within S)
     using derf[unfolded has_derivative_within_alt2] and ‹x ∈ S› and * by
fast
    moreover have eventually (λy. y ∈ S) (at x within S)
     unfolding eventually_at by (fast intro: zero_less_one)
    ultimately show ∀F y in at x within S. norm (g y - g x - g' x (y - x)) ≤
e * norm (y - x)
    proof (rule eventually_elim2)
     fix y
     assume y ∈ S
    assume norm (f ?N y - f ?N x - f' ?N x (y - x)) ≤ e / 3 * norm (y - x)
     moreover have norm (g y - g x - (f ?N y - f ?N x)) ≤ e / 3 * norm (y
- x)
      using N2[OF _ ‹y ∈ S› ‹x ∈ S›]
      by (simp add: norm_minus_commute)
     ultimately have norm (g y - g x - f' ?N x (y - x)) ≤ 2 * e / 3 * norm
(y - x)
      using norm_triangle_le[of g y - g x - (f ?N y - f ?N x) f ?N y - f ?N
x - f' ?N x (y - x) 2 * e / 3 * norm (y - x)]
      by (auto simp add: algebra_simps)
     moreover
     have  norm (f' ?N x (y - x) - g' x (y - x)) ≤ e / 3 * norm (y - x)
      using N1 ‹x ∈ S› by auto
     ultimately show norm (g y - g x - g' x (y - x)) ≤ e * norm (y - x)
      using norm_triangle_le[of g y - g x - f' (max N1 N2) x (y - x) f' (max
N1 N2) x (y - x) - g' x (y - x)]
      by (auto simp add: algebra_simps)
    qed
```

```
    qed
  qed
  then show ?thesis by fast
qed
```

Can choose to line up antiderivatives if we want.

```
lemma has_antiderivative_sequence:
  fixes f :: nat ⇒ 'a::real_normed_vector ⇒ 'b::banach
  assumes convex S
    and der: ⋀n x. x ∈ S ⟹ ((f n) has_derivative (f' n x)) (at x within S)
    and no: ⋀e. e > 0 ⟹ ∀F n in sequentially.
      ∀x∈S. ∀h. norm (f' n x h − g' x h) ≤ e * norm h
  shows ∃g. ∀x∈S. (g has_derivative g' x) (at x within S)
proof (cases S = {})
  case False
  then obtain a where a ∈ S
    by auto
  have *: ⋀P Q. ∃g. ∀x∈S. P g x ∧ Q g x ⟹ ∃g. ∀x∈S. Q g x
    by auto
  show ?thesis
    apply (rule *)
    apply (rule has_derivative_sequence [OF ‹convex S› _ no, of λn x. f n x +
(f 0 a − f n a)])
      apply (metis assms(2) has_derivative_add_const)
    using ‹a ∈ S›
     apply auto
    done
qed auto

lemma has_antiderivative_limit:
  fixes g' :: 'a::real_normed_vector ⇒ 'a ⇒ 'b::banach
  assumes convex S
    and ⋀e. e>0 ⟹ ∃f f'. ∀x∈S.
        (f has_derivative (f' x)) (at x within S) ∧ (∀h. norm (f' x h − g' x h) ≤
e * norm h)
  shows ∃g. ∀x∈S. (g has_derivative g' x) (at x within S)
proof −
  have *: ∀n. ∃f f'. ∀x∈S.
    (f has_derivative (f' x)) (at x within S) ∧
    (∀h. norm(f' x h − g' x h) ≤ inverse (real (Suc n)) * norm h)
    by (simp add: assms(2))
  obtain f where
    *: ⋀x. ∃f'. ∀xa∈S. (f x has_derivative f' xa) (at xa within S) ∧
       (∀h. norm (f' xa h − g' xa h) ≤ inverse (real (Suc x)) * norm h)
    using * by metis
  obtain f' where
    f': ⋀x. ∀z∈S. (f x has_derivative f' x z) (at z within S) ∧
          (∀h. norm (f' x z h − g' z h) ≤ inverse (real (Suc x)) * norm h)
    using * by metis
```

show *?thesis*
proof (*rule has_antiderivative_sequence*[*OF* ‹*convex S*›, *of f f'*])
fix *e* :: *real*
assume *e* > *0*
obtain *N* **where** *N*: *inverse* (*real* (*Suc N*)) < *e*
using *reals_Archimedean*[*OF* ‹*e*>*0*›] **..**
show ∀$_F$ *n in sequentially*. ∀*x*∈*S*. ∀*h*. *norm* (*f' n x h* − *g' x h*) ≤ *e* ∗ *norm h*
unfolding *eventually_sequentially*
proof (*intro exI allI ballI impI*)
fix *n x h*
assume *n*: *N* ≤ *n* **and** *x*: *x* ∈ *S*
have ∗: *inverse* (*real* (*Suc n*)) ≤ *e*
using *n N*
by (*smt* (*verit*, *best*) *le_imp_inverse_le of_nat_0_less_iff of_nat_Suc of_nat_le_iff zero_less_Suc*)
show *norm* (*f' n x h* − *g' x h*) ≤ *e* ∗ *norm h*
by (*meson* ∗ *mult_right_mono norm_ge_zero order.trans x f'*)
qed
qed (*use f'* **in** *auto*)
qed

5.17.14 Differentiation of a series

proposition *has_derivative_series*:
fixes *f* :: *nat* ⇒ *'a*::*real_normed_vector* ⇒ *'b*::*banach*
assumes *convex S*
and ⋀*n x*. *x* ∈ *S* ⟹ ((*f n*) *has_derivative* (*f' n x*)) (*at x within S*)
and ⋀*e*. *e*>*0* ⟹ ∀$_F$ *n in sequentially*. ∀*x*∈*S*. ∀*h*. *norm* (*sum* (λ*i*. *f' i x h*) {..<*n*} − *g' x h*) ≤ *e* ∗ *norm h*
and *x* ∈ *S*
and (λ*n*. *f n x*) *sums l*
shows ∃*g*. ∀*x*∈*S*. (λ*n*. *f n x*) *sums* (*g x*) ∧ (*g has_derivative g' x*) (*at x within S*)
unfolding *sums_def*
apply (*rule has_derivative_sequence*[*OF assms*(*1*) _ *assms*(*3*)])
apply (*metis assms*(*2*) *has_derivative_sum*)
using *assms*(*4*−*5*)
unfolding *sums_def*
apply *auto*
done

lemma *has_field_derivative_series*:
fixes *f* :: *nat* ⇒ (*'a* :: {*real_normed_field*,*banach*}) ⇒ *'a*
assumes *convex S*
assumes ⋀*n x*. *x* ∈ *S* ⟹ (*f n has_field_derivative f' n x*) (*at x within S*)
assumes *uniform_limit S* (λ*n x*. ∑ *i*<*n*. *f' i x*) *g' sequentially*
assumes *x0* ∈ *S summable* (λ*n*. *f n x0*)
shows ∃*g*. ∀*x*∈*S*. (λ*n*. *f n x*) *sums g x* ∧ (*g has_field_derivative g' x*) (*at x within S*)

unfolding *has_field_derivative_def*
proof (*rule has_derivative_series*)
show $\forall_F\ n$ *in sequentially.*
$\forall x{\in}S.\ \forall h.\ norm\ ((\sum i{<}n.\ f'\ i\ x * h) - g'\ x * h) \leq e * norm\ h$ **if** $e > 0$
for e
unfolding *eventually_sequentially*
proof −
from *that assms*(3) **obtain** N **where** N: $\bigwedge n\ x.\ n \geq N \Longrightarrow x \in S \Longrightarrow norm\ ((\sum i{<}n.\ f'\ i\ x) - g'\ x) < e$
unfolding *uniform_limit_iff eventually_at_top_linorder dist_norm* **by** *blast*
{
fix $n :: nat$ **and** $x\ h :: 'a$ **assume** nx: $n \geq N\ x \in S$
have $norm\ ((\sum i{<}n.\ f'\ i\ x * h) - g'\ x * h) = norm\ ((\sum i{<}n.\ f'\ i\ x) - g'\ x) * norm\ h$
by (*simp add*: *norm_mult* [*symmetric*] *ring_distribs sum_distrib_right*)
also from N[*OF nx*] **have** $norm\ ((\sum i{<}n.\ f'\ i\ x) - g'\ x) \leq e$ **by** *simp*
hence $norm\ ((\sum i{<}n.\ f'\ i\ x) - g'\ x) * norm\ h \leq e * norm\ h$
by (*intro mult_right_mono*) *simp_all*
finally have $norm\ ((\sum i{<}n.\ f'\ i\ x * h) - g'\ x * h) \leq e * norm\ h$ **.**
}
thus $\exists N.\ \forall n{\geq}N.\ \forall x{\in}S.\ \forall h.\ norm\ ((\sum i{<}n.\ f'\ i\ x * h) - g'\ x * h) \leq e * norm\ h$ **by** *blast*
qed
qed (*use assms* **in** ‹*auto simp*: *has_field_derivative_def*›)

lemma *has_field_derivative_series'*:
fixes $f :: nat \Rightarrow ('a :: \{real\_normed\_field, banach\}) \Rightarrow 'a$
assumes *convex S*
assumes $\bigwedge n\ x.\ x \in S \Longrightarrow (f\ n\ has\_field\_derivative\ f'\ n\ x)\ (at\ x\ within\ S)$
assumes *uniformly_convergent_on S* $(\lambda n\ x.\ \sum i{<}n.\ f'\ i\ x)$
assumes $x0 \in S$ *summable* $(\lambda n.\ f\ n\ x0)$ $x \in interior\ S$
shows *summable* $(\lambda n.\ f\ n\ x)$ $((\lambda x.\ \sum n.\ f\ n\ x)\ has\_field\_derivative\ (\sum n.\ f'\ n\ x))\ (at\ x)$
proof −
from ‹$x \in interior\ S$› **have** $x \in S$ **using** *interior_subset* **by** *blast*
define g' **where** [*abs_def*]: $g'\ x = (\sum i.\ f'\ i\ x)$ **for** x
from *assms*(3) **have** *uniform_limit S* $(\lambda n\ x.\ \sum i{<}n.\ f'\ i\ x)\ g'$ *sequentially*
by (*simp add*: *uniformly_convergent_uniform_limit_iff suminf_eq_lim g'_def*)
from *has_field_derivative_series*[*OF assms*(1,2) *this assms*(4,5)] **obtain** g **where** g:
$\bigwedge x.\ x \in S \Longrightarrow (\lambda n.\ f\ n\ x)\ sums\ g\ x$
$\bigwedge x.\ x \in S \Longrightarrow (g\ has\_field\_derivative\ g'\ x)\ (at\ x\ within\ S)$ **by** *blast*
from *g*(1)[*OF* ‹$x \in S$›] **show** *summable* $(\lambda n.\ f\ n\ x)$ **by** (*simp add*: *sums_iff*)
from *g*(2)[*OF* ‹$x \in S$›] ‹$x \in interior\ S$› **have** $(g\ has\_field\_derivative\ g'\ x)\ (at\ x)$
by (*simp add*: *at_within_interior*[*of x S*])
also have $(g\ has\_field\_derivative\ g'\ x)\ (at\ x) \longleftrightarrow$
$((\lambda x.\ \sum n.\ f\ n\ x)\ has\_field\_derivative\ g'\ x)\ (at\ x)$
using *eventually_nhds_in_nhd*[*OF* ‹$x \in interior\ S$›] *interior_subset*[*of S*] *g*(1)

by (*intro DERIV_cong_ev*) (*auto elim*!: *eventually_mono simp*: *sums_iff*)
finally show ((λ*x*. ∑ *n*. *f n x*) *has_field_derivative g' x*) (*at x*) **.**
qed

lemma *differentiable_series*:
fixes *f* :: *nat* ⇒ ('*a* :: {*real_normed_field*,*banach*}) ⇒ '*a*
assumes *convex S open S*
assumes ⋀*n x*. *x* ∈ *S* ⟹ (*f n has_field_derivative f' n x*) (*at x*)
assumes *uniformly_convergent_on S* (λ*n x*. ∑ *i*<*n*. *f' i x*)
assumes *x0* ∈ *S summable* (λ*n*. *f n x0*) **and** *x*: *x* ∈ *S*
shows *summable* (λ*n*. *f n x*) **and** (λ*x*. ∑ *n*. *f n x*) *differentiable* (*at x*)
proof −
from *assms*(*4*) **obtain** *g'* **where** *A*: *uniform_limit S* (λ*n x*. ∑ *i*<*n*. *f' i x*) *g'*
sequentially
unfolding *uniformly_convergent_on_def* **by** *blast*
from *x* **and** ‹*open S*› **have** *S*: *at x within S* = *at x* **by** (*rule at_within_open*)
have ∃ *g*. ∀ *x*∈*S*. (λ*n*. *f n x*) *sums g x* ∧ (*g has_field_derivative g' x*) (*at x within*
S)
by (*intro has_field_derivative_series*[*of S f f' g' x0*] *assms A has_field_derivative_at_within*)
then obtain *g* **where** *g*: ⋀*x*. *x* ∈ *S* ⟹ (λ*n*. *f n x*) *sums g x*
⋀*x*. *x* ∈ *S* ⟹ (*g has_field_derivative g' x*) (*at x within S*) **by** *blast*
from *g*[*OF x*] **show** *summable* (λ*n*. *f n x*) **by** (*auto simp*: *summable_def*)
from *g*(*2*)[*OF x*] **have** *g'*: (*g has_derivative* (∗) (*g' x*)) (*at x*)
by (*simp add*: *has_field_derivative_def S*)
have ((λ*x*. ∑ *n*. *f n x*) *has_derivative* (∗) (*g' x*)) (*at x*)
by (*rule has_derivative_transform_within_open*[*OF g' ‹open S› x*])
(*insert g*, *auto simp*: *sums_iff*)
thus (λ*x*. ∑ *n*. *f n x*) *differentiable* (*at x*) **unfolding** *differentiable_def*
by (*auto simp*: *summable_def differentiable_def has_field_derivative_def*)
qed

lemma *differentiable_series'*:
fixes *f* :: *nat* ⇒ ('*a* :: {*real_normed_field*,*banach*}) ⇒ '*a*
assumes *convex S open S*
assumes ⋀*n x*. *x* ∈ *S* ⟹ (*f n has_field_derivative f' n x*) (*at x*)
assumes *uniformly_convergent_on S* (λ*n x*. ∑ *i*<*n*. *f' i x*)
assumes *x0* ∈ *S summable* (λ*n*. *f n x0*)
shows (λ*x*. ∑ *n*. *f n x*) *differentiable* (*at x0*)
using *differentiable_series*[*OF assms*, *of x0*] ‹*x0* ∈ *S*› **by** *blast*+

5.17.15 Derivative as a vector

Considering derivative *real* ⇒ '*b* as a vector.

definition *vector_derivative f net* = (*SOME f'*. (*f has_vector_derivative f'*) *net*)

lemma *vector_derivative_unique_within*:
assumes *not_bot*: *at x within S* ≠ *bot*
and *f'*: (*f has_vector_derivative f'*) (*at x within S*)
and *f''*: (*f has_vector_derivative f''*) (*at x within S*)

```
  shows f' = f''
proof −
  have (λx. x *R f') = (λx. x *R f'')
  proof (rule frechet_derivative_unique_within, simp_all)
    show ∃d. d ≠ 0 ∧ |d| < e ∧ x + d ∈ S if 0 < e  for e
    proof −
      from that
      obtain x' where x' ∈ S x' ≠ x |x' − x| < e
        using islimpt_approachable_real[of x S] not_bot
        by (auto simp add: trivial_limit_within)
      then show ?thesis
        using eq_iff_diff_eq_0 by fastforce
    qed
  qed (use f' f'' in ‹auto simp: has_vector_derivative_def›)
  then show ?thesis
    unfolding fun_eq_iff by (metis scaleR_one)
qed

lemma vector_derivative_unique_at:
  (f has_vector_derivative f') (at x) ⟹ (f has_vector_derivative f'') (at x) ⟹
f' = f''
  by (rule vector_derivative_unique_within) auto

lemma differentiableI_vector: (f has_vector_derivative y) F ⟹ f differentiable
F
  by (auto simp: differentiable_def has_vector_derivative_def)

proposition vector_derivative_works:
  f differentiable net ⟷ (f has_vector_derivative (vector_derivative f net)) net
    (is ?l = ?r)
proof
  assume ?l
  obtain f' where f': (f has_derivative f') net
    using ‹?l› unfolding differentiable_def ..
  then interpret bounded_linear f'
    by auto
  show ?r
    unfolding vector_derivative_def has_vector_derivative_def
    by (rule someI[of _ f' 1]) (simp add: scaleR[symmetric] f')
qed (auto simp: vector_derivative_def has_vector_derivative_def differentiable_def)

lemma vector_derivative_within:
  assumes not_bot: at x within S ≠ bot and y: (f has_vector_derivative y) (at x
within S)
  shows vector_derivative f (at x within S) = y
  using y
 by (intro vector_derivative_unique_within[OF not_bot vector_derivative_works[THEN
iffD1] y])
     (auto simp: differentiable_def has_vector_derivative_def)
```

lemma *deriv_of_real* [*simp*]:
at x within A ≠ *bot* ⟹ *vector_derivative of_real* (*at x within A*) = *1*
by (*auto intro*!: *vector_derivative_within derivative_eq_intros*)

lemma *frechet_derivative_eq_vector_derivative*:
assumes *f differentiable* (*at x*)
shows (*frechet_derivative f* (*at x*)) = (λ*r*. *r* ∗$_R$ *vector_derivative f* (*at x*))
using *assms*
by (*auto simp*: *differentiable_iff_scaleR vector_derivative_def has_vector_derivative_def*
intro: *someI frechet_derivative_at* [*symmetric*])

lemma *has_real_derivative*:
fixes *f* :: *real* ⇒ *real*
assumes (*f has_derivative f'*) *F*
obtains *c* **where** (*f has_real_derivative c*) *F*
proof −
obtain *c* **where** *f'* = (λ*x*. *x* ∗ *c*)
by (*metis assms has_derivative_bounded_linear real_bounded_linear*)
then show *?thesis*
by (*metis assms that has_field_derivative_def mult_commute_abs*)
qed

lemma *has_real_derivative_iff*:
fixes *f* :: *real* ⇒ *real*
shows (∃ *c*. (*f has_real_derivative c*) *F*) = (∃ *D*. (*f has_derivative D*) *F*)
by (*metis has_field_derivative_def has_real_derivative*)

lemma *has_vector_derivative_cong_ev*:
assumes ∗: *eventually* (λ*x*. *x* ∈ *S* ⟶ *f x* = *g x*) (*nhds x*) *f x* = *g x*
shows (*f has_vector_derivative f'*) (*at x within S*) = (*g has_vector_derivative*
f') (*at x within S*)
proof (*cases at x within S* = *bot*)
case *True*
then show *?thesis*
by (*simp add*: *has_derivative_def has_vector_derivative_def*)
next
case *False*
then show *?thesis*
unfolding *has_vector_derivative_def has_derivative_def*
using ∗
apply (*intro refl conj_cong filterlim_cong*)
apply (*auto simp*: *Lim_ident_at eventually_at_filter elim*: *eventually_mono*)
done
qed

lemma *vector_derivative_cong_eq*:
assumes *eventually* (λ*x*. *x* ∈ *A* ⟶ *f x* = *g x*) (*nhds x*) *x* = *y A* = *B x* ∈ *A*
shows *vector_derivative f* (*at x within A*) = *vector_derivative g* (*at y within*

```
B)
proof −
  have f x = g x
    using assms eventually_nhds_x_imp_x by blast
  hence (λD. (f has_vector_derivative D) (at x within A)) =
          (λD. (g has_vector_derivative D) (at x within A)) using assms
    by (intro ext has_vector_derivative_cong_ev refl assms) simp_all
  thus ?thesis by (simp add: vector_derivative_def assms)
qed

lemma islimpt_closure_open:
  fixes s :: 'a::perfect_space set
  assumes open s and t: t = closure s x ∈ t
  shows x islimpt t
proof cases
  assume x ∈ s
  { fix T assume x ∈ T open T
    then have open (s ∩ T)
      using ‹open s› by auto
    then have s ∩ T ≠ {x}
      using not_open_singleton[of x] by auto
    with ‹x ∈ T› ‹x ∈ s› have ∃ y∈t. y ∈ T ∧ y ≠ x
      using closure_subset[of s] by (auto simp: t) }
  then show ?thesis
    by (auto intro!: islimptI)
next
  assume x ∉ s with t show ?thesis
    unfolding t closure_def by (auto intro: islimpt_subset)
qed

lemma vector_derivative_unique_within_closed_interval:
  assumes ab: a < b x ∈ cbox a b
 assumes D: (f has_vector_derivative f') (at x within cbox a b) (f has_vector_derivative
f'') (at x within cbox a b)
  shows f' = f''
  using ab
  by (intro vector_derivative_unique_within[OF _ D])
       (auto simp: trivial_limit_within intro!: islimpt_closure_open[where s={a
<..< b}])

lemma vector_derivative_at:
  (f has_vector_derivative f') (at x) ⟹ vector_derivative f (at x) = f'
  by (intro vector_derivative_within at_neq_bot)

lemma has_vector_derivative_id_at [simp]: vector_derivative (λx. x) (at a) = 1
  by (simp add: vector_derivative_at)

lemma vector_derivative_minus_at [simp]:
  f differentiable at a
```

$\Longrightarrow$ *vector_derivative* $(\lambda x.\ -\ f\ x)\ (at\ a) = -$ *vector_derivative* $f\ (at\ a)$
by (*simp add*: *vector_derivative_at has_vector_derivative_minus vector_derivative_works* [*symmetric*])

lemma *vector_derivative_add_at* [*simp*]:
$[\![f$ *differentiable at* $a;\ g$ *differentiable at* $a]\!]$
$\Longrightarrow$ *vector_derivative* $(\lambda x.\ f\ x\ +\ g\ x)\ (at\ a) =$ *vector_derivative* $f\ (at\ a) +$ *vector_derivative* $g\ (at\ a)$
by (*simp add*: *vector_derivative_at has_vector_derivative_add vector_derivative_works* [*symmetric*])

lemma *vector_derivative_diff_at* [*simp*,*derivative_intros*]:
$[\![f$ *differentiable at* $a;\ g$ *differentiable at* $a]\!]$
$\Longrightarrow$ *vector_derivative* $(\lambda x.\ f\ x\ -\ g\ x)\ (at\ a) =$ *vector_derivative* $f\ (at\ a) -$ *vector_derivative* $g\ (at\ a)$
by (*simp add*: *vector_derivative_at has_vector_derivative_diff vector_derivative_works* [*symmetric*])

lemma *vector_derivative_mult_at* [*simp*]:
fixes $f\ g$ :: *real* $\Rightarrow$ *'a* :: *real_normed_algebra*
shows $[\![f$ *differentiable at* $a;\ g$ *differentiable at* $a]\!]$
$\Longrightarrow$ *vector_derivative* $(\lambda x.\ f\ x\ *\ g\ x)\ (at\ a) = f\ a\ *$ *vector_derivative* $g\ (at\ a)$ $+$ *vector_derivative* $f\ (at\ a)\ *\ g\ a$
by (*simp add*: *vector_derivative_at has_vector_derivative_mult vector_derivative_works* [*symmetric*])

lemma *vector_derivative_scaleR_at* [*simp*]:
$[\![f$ *differentiable at* $a;\ g$ *differentiable at* $a]\!]$
$\Longrightarrow$ *vector_derivative* $(\lambda x.\ f\ x\ *_R\ g\ x)\ (at\ a) = f\ a\ *_R$ *vector_derivative* $g\ (at\ a) +$ *vector_derivative* $f\ (at\ a)\ *_R\ g\ a$
apply (*intro vector_derivative_at has_vector_derivative_scaleR*)
apply (*auto simp*: *vector_derivative_works has_vector_derivative_def has_field_derivative_def mult_commute_abs*)
done

lemma *vector_derivative_within_cbox*:
assumes *ab*: $a < b\ x \in$ *cbox* $a\ b$
assumes f: $(f$ *has_vector_derivative* $f')\ (at\ x$ *within* *cbox* $a\ b)$
shows *vector_derivative* $f\ (at\ x$ *within* *cbox* $a\ b) = f'$
by (*metis assms box_real*(2) *f islimpt_Icc trivial_limit_within vector_derivative_within*)

lemma *vector_derivative_within_closed_interval*:
fixes f::*real* $\Rightarrow$ *'a*::*euclidean_space*
assumes $a < b$ **and** $x \in \{a..b\}$
assumes $(f$ *has_vector_derivative* $f')\ (at\ x$ *within* $\{a..b\})$
shows *vector_derivative* $f\ (at\ x$ *within* $\{a..b\}) = f'$
using *assms vector_derivative_within_cbox*
by *fastforce*

lemma *has_vector_derivative_within_subset*:
(*f has_vector_derivative f'*) (*at x within S*) ⟹ *T* ⊆ *S* ⟹ (*f has_vector_derivative f'*) (*at x within T*)
by (*auto simp*: *has_vector_derivative_def intro*: *has_derivative_subset*)

lemma *has_vector_derivative_at_within*:
(*f has_vector_derivative f'*) (*at x*) ⟹ (*f has_vector_derivative f'*) (*at x within S*)
unfolding *has_vector_derivative_def*
by (*rule has_derivative_at_withinI*)

lemma *has_vector_derivative_weaken*:
fixes *x D* **and** *f g S T*
assumes *f*: (*f has_vector_derivative D*) (*at x within T*)
and *x* ∈ *S S* ⊆ *T*
and ⋀*x*. *x* ∈ *S* ⟹ *f x* = *g x*
shows (*g has_vector_derivative D*) (*at x within S*)
proof −
have (*f has_vector_derivative D*) (*at x within S*) ⟷ (*g has_vector_derivative D*) (*at x within S*)
unfolding *has_vector_derivative_def has_derivative_iff_norm*
using *assms* **by** (*intro conj_cong Lim_cong_within refl*) *auto*
then show *?thesis*
using *has_vector_derivative_within_subset*[*OF f* ‹*S* ⊆ *T*›] **by** *simp*
qed

lemma *has_vector_derivative_transform_within*:
assumes (*f has_vector_derivative f'*) (*at x within S*)
and *0* < *d*
and *x* ∈ *S*
and ⋀*x'*. ⟦*x'*∈*S*; *dist x' x* < *d*⟧ ⟹ *f x'* = *g x'*
shows (*g has_vector_derivative f'*) (*at x within S*)
using *assms*
unfolding *has_vector_derivative_def*
by (*rule has_derivative_transform_within*)

lemma *has_vector_derivative_transform_within_open*:
assumes (*f has_vector_derivative f'*) (*at x*)
and *open S*
and *x* ∈ *S*
and ⋀*y*. *y*∈*S* ⟹ *f y* = *g y*
shows (*g has_vector_derivative f'*) (*at x*)
using *assms*
unfolding *has_vector_derivative_def*
by (*rule has_derivative_transform_within_open*)

lemma *has_vector_derivative_transform*:
assumes *x* ∈ *S* ⋀*x*. *x* ∈ *S* ⟹ *g x* = *f x*
assumes *f'*: (*f has_vector_derivative f'*) (*at x within S*)

shows (*g has_vector_derivative f'*) (*at x within S*)
using *assms*
unfolding *has_vector_derivative_def*
by (*rule has_derivative_transform*)

lemma *vector_diff_chain_at*:
assumes (*f has_vector_derivative f'*) (*at x*)
and (*g has_vector_derivative g'*) (*at* (*f x*))
shows ((*g* ∘ *f*) *has_vector_derivative* (*f' *$_R$ *g'*)) (*at x*)
using *assms has_vector_derivative_at_within has_vector_derivative_def vector_derivative_diff_chain_within* **by** *blast*

lemma *vector_diff_chain_within*:
assumes (*f has_vector_derivative f'*) (*at x within s*)
and (*g has_vector_derivative g'*) (*at* (*f x*) *within f ' s*)
shows ((*g* ∘ *f*) *has_vector_derivative* (*f' *$_R$ *g'*)) (*at x within s*)
using *assms has_vector_derivative_def vector_derivative_diff_chain_within* **by** *blast*

lemma *vector_derivative_const_at* [*simp*]: *vector_derivative* (λ*x*. *c*) (*at a*) = *0*
by (*simp add*: *vector_derivative_at*)

lemma *vector_derivative_at_within_ivl*:
(*f has_vector_derivative f'*) (*at x*) ⟹
a ≤ *x* ⟹ *x* ≤ *b* ⟹ *a*<*b* ⟹ *vector_derivative f* (*at x within* {*a*..*b*}) = *f'*
using *has_vector_derivative_at_within vector_derivative_within_cbox* **by** *fastforce*

lemma *vector_derivative_chain_at*:
assumes *f differentiable at x* (*g differentiable at* (*f x*))
shows *vector_derivative* (*g* ∘ *f*) (*at x*) =
vector_derivative f (*at x*) *$*$_R$ *vector_derivative g* (*at* (*f x*))
by (*metis vector_diff_chain_at vector_derivative_at vector_derivative_works assms*)

lemma *field_vector_diff_chain_at*:
assumes *Df*: (*f has_vector_derivative f'*) (*at x*)
and *Dg*: (*g has_field_derivative g'*) (*at* (*f x*))
shows ((*g* ∘ *f*) *has_vector_derivative* (*f' * *g'*)) (*at x*)
using *diff_chain_at*[*OF Df*[*unfolded has_vector_derivative_def*]
Dg [*unfolded has_field_derivative_def*]]
by (*auto simp*: *o_def mult.commute has_vector_derivative_def*)

lemma *vector_derivative_chain_within*:
assumes *at x within S* ≠ *bot f differentiable* (*at x within S*)
(*g has_derivative g'*) (*at* (*f x*) *within f ' S*)
shows *vector_derivative* (*g* ∘ *f*) (*at x within S*) =
g' (*vector_derivative f* (*at x within S*))
apply (*rule vector_derivative_within* [*OF* ‹*at x within S* ≠ *bot*›])
apply (*rule vector_derivative_diff_chain_within*)

using *assms*(2−3) *vector_derivative_works*
by *auto*

5.17.16 Field differentiability

definition *field_differentiable* :: [′*a* ⇒ ′*a*::*real_normed_field*, ′*a filter*] ⇒ *bool*
(**infixr** (*field′_differentiable*) 50)
where *f field_differentiable F* ≡ ∃*f′*. (*f has_field_derivative f′*) *F*

lemma *field_differentiable_imp_differentiable*:
f field_differentiable F ⟹ *f differentiable F*
unfolding *field_differentiable_def differentiable_def*
using *has_field_derivative_imp_has_derivative* **by** *auto*

lemma *field_differentiable_imp_continuous_at*:
f field_differentiable (*at x within S*) ⟹ *continuous* (*at x within S*) *f*
by (*metis DERIV_continuous field_differentiable_def*)

lemma *field_differentiable_within_subset*:
⟦*f field_differentiable* (*at x within S*); *T* ⊆ *S*⟧ ⟹ *f field_differentiable* (*at x within T*)
by (*metis DERIV_subset field_differentiable_def*)

lemma *field_differentiable_at_within*:
⟦*f field_differentiable* (*at x*)⟧
⟹ *f field_differentiable* (*at x within S*)
unfolding *field_differentiable_def*
by (*metis DERIV_subset top_greatest*)

lemma *field_differentiable_linear* [*simp*,*derivative_intros*]: ((∗) *c*) *field_differentiable F*
unfolding *field_differentiable_def has_field_derivative_def mult_commute_abs*
by (*force intro*: *has_derivative_mult_right*)

lemma *field_differentiable_const* [*simp*,*derivative_intros*]: (λ*z*. *c*) *field_differentiable F*
unfolding *field_differentiable_def has_field_derivative_def*
using *DERIV_const has_field_derivative_imp_has_derivative* **by** *blast*

lemma *field_differentiable_ident* [*simp*,*derivative_intros*]: (λ*z*. *z*) *field_differentiable F*
unfolding *field_differentiable_def has_field_derivative_def*
using *DERIV_ident has_field_derivative_def* **by** *blast*

lemma *field_differentiable_id* [*simp*,*derivative_intros*]: *id field_differentiable F*
unfolding *id_def* **by** (*rule field_differentiable_ident*)

lemma *field_differentiable_minus* [*derivative_intros*]:
f field_differentiable F ⟹ (λ*z*. − (*f z*)) *field_differentiable F*

unfolding *field_differentiable_def* **by** (*metis field_differentiable_minus*)

lemma *field_differentiable_diff_const* [*simp*,*derivative_intros*]:
(−)*c field_differentiable F*
unfolding *field_differentiable_def* **by** (*rule derivative_eq_intros exI* | *force*)+

lemma *field_differentiable_add* [*derivative_intros*]:
assumes *f field_differentiable F g field_differentiable F*
shows (λ*z*. *f z* + *g z*) *field_differentiable F*
using *assms* **unfolding** *field_differentiable_def*
by (*metis field_differentiable_add*)

lemma *field_differentiable_add_const* [*simp*,*derivative_intros*]:
(+) *c field_differentiable F*
by (*simp add*: *field_differentiable_add*)

lemma *field_differentiable_sum* [*derivative_intros*]:
(⋀*i*. *i* ∈ *I* ⟹ (*f i*) *field_differentiable F*) ⟹ (λ*z*. ∑ *i*∈*I*. *f i z*) *field_differentiable F*
by (*induct I rule*: *infinite_finite_induct*)
(*auto intro*: *field_differentiable_add field_differentiable_const*)

lemma *field_differentiable_diff* [*derivative_intros*]:
assumes *f field_differentiable F g field_differentiable F*
shows (λ*z*. *f z* − *g z*) *field_differentiable F*
using *assms* **unfolding** *field_differentiable_def*
by (*metis field_differentiable_diff*)

lemma *field_differentiable_inverse* [*derivative_intros*]:
assumes *f field_differentiable* (*at a within S*) *f a* ≠ *0*
shows (λ*z*. *inverse* (*f z*)) *field_differentiable* (*at a within S*)
using *assms* **unfolding** *field_differentiable_def*
by (*metis DERIV_inverse_fun*)

lemma *field_differentiable_mult* [*derivative_intros*]:
assumes *f field_differentiable* (*at a within S*)
g field_differentiable (*at a within S*)
shows (λ*z*. *f z* ∗ *g z*) *field_differentiable* (*at a within S*)
using *assms* **unfolding** *field_differentiable_def*
by (*metis DERIV_mult* [*of f _ a S g*])

lemma *field_differentiable_divide* [*derivative_intros*]:
assumes *f field_differentiable* (*at a within S*)
g field_differentiable (*at a within S*)
g a ≠ *0*
shows (λ*z*. *f z* / *g z*) *field_differentiable* (*at a within S*)
using *assms* **unfolding** *field_differentiable_def*
by (*metis DERIV_divide* [*of f _ a S g*])

lemma *field_differentiable_power* [*derivative_intros*]:
assumes *f field_differentiable* (*at a within S*)
shows (λ*z*. *f z* ^ *n*) *field_differentiable* (*at a within S*)
using *assms* **unfolding** *field_differentiable_def*
by (*metis DERIV_power*)

lemma *field_differentiable_cnj_cnj*:
assumes *f field_differentiable* (*at* (*cnj z*))
shows (*cnj* ∘ *f* ∘ *cnj*) *field_differentiable* (*at z*)
using *has_field_derivative_cnj_cnj assms*
by (*auto simp*: *field_differentiable_def*)

lemma *field_differentiable_transform_within*:
0 < *d* ⟹
x ∈ *S* ⟹
(⋀*x'*. *x'* ∈ *S* ⟹ *dist x' x* < *d* ⟹ *f x'* = *g x'*) ⟹
f field_differentiable (*at x within S*)
⟹ *g field_differentiable* (*at x within S*)
unfolding *field_differentiable_def has_field_derivative_def*
by (*blast intro*: *has_derivative_transform_within*)

lemma *field_differentiable_compose_within*:
assumes *f field_differentiable* (*at a within S*)
g field_differentiable (*at* (*f a*) *within f*‘*S*)
shows (*g o f*) *field_differentiable* (*at a within S*)
using *assms* **unfolding** *field_differentiable_def*
by (*metis DERIV_image_chain*)

lemma *field_differentiable_compose*:
f field_differentiable at z ⟹ *g field_differentiable at* (*f z*)
⟹ (*g o f*) *field_differentiable at z*
by (*metis field_differentiable_at_within field_differentiable_compose_within*)

lemma *field_differentiable_within_open*:
⟦*a* ∈ *S*; *open S*⟧ ⟹ *f field_differentiable at a within S* ⟷
f field_differentiable at a
unfolding *field_differentiable_def*
by (*metis at_within_open*)

lemma *exp_scaleR_has_vector_derivative_right*:
((λ*t*. *exp* (*t* ∗$_R$ *A*)) *has_vector_derivative exp* (*t* ∗$_R$ *A*) ∗ *A*) (*at t within T*)
unfolding *has_vector_derivative_def*
proof (*rule has_derivativeI*)
let *?F* = *at t within* (*T* ∩ {*t* − *1* <..< *t* + *1*})
have ∗: *at t within T* = *?F*
by (*rule at_within_nhd*[**where** *S*={*t* − *1* <..< *t* + *1*}]) *auto*
let *?e* = λ*i x*. (*inverse* (*1* + *real i*) ∗ *inverse* (*fact i*) ∗ (*x* − *t*) ^ *i*) ∗$_R$ (*A* ∗ *A* ^ *i*)
have ∀$_F$ *n in sequentially*.

```
    ∀x∈T ∩ {t − 1<..<t + 1}. norm (?e n x) ≤ norm (A ^(n + 1) /R fact (n
+ 1))
   apply (auto simp: algebra_split_simps intro!: eventuallyI)
   apply (rule mult_left_mono)
    apply (auto simp add: field_simps power_abs intro!: divide_right_mono
power_le_one)
   done
 then have uniform_limit (T ∩ {t − 1<..<t + 1}) (λn x. ∑i<n. ?e i x) (λx.
∑i. ?e i x) sequentially
    by (rule Weierstrass_m_test_ev) (intro summable_ignore_initial_segment
summable_norm_exp)
 moreover
 have ∀F x in sequentially. x > 0
   by (metis eventually_gt_at_top)
 then have
   ∀F n in sequentially. ((λx. ∑i<n. ?e i x) ⟶ A) ?F
   by eventually_elim
     (auto intro!: tendsto_eq_intros
       simp: power_0_left if_distrib if_distribR
       cong: if_cong)
 ultimately
 have [tendsto_intros]: ((λx. ∑i. ?e i x) ⟶ A) ?F
   by (auto intro!: swap_uniform_limit[where f=λn x. ∑i < n. ?e i x and F
= sequentially])
 have [tendsto_intros]: ((λx. if x = t then 0 else 1) ⟶ 1) ?F
   by (rule tendsto_eventually) (simp add: eventually_at_filter)
 have ((λy. ((y − t) / abs (y − t)) *R ((∑ n. ?e n y) − A)) ⟶ 0) (at t within
T)
   unfolding *
   by (rule tendsto_norm_zero_cancel) (auto intro!: tendsto_eq_intros)

 moreover have ∀F x in at t within T. x ≠ t
   by (simp add: eventually_at_filter)
 then have ∀F x in at t within T. ((x − t) / |x − t|) *R ((∑ n. ?e n x) − A) =
   (exp ((x − t) *R A) − 1 − (x − t) *R A) /R norm (x − t)
 proof eventually_elim
   case (elim x)
   have (exp ((x − t) *R A) − 1 − (x − t) *R A) /R norm (x − t) =
     ((∑ n. (x − t) *R ?e n x) − (x − t) *R A) /R norm (x − t)
     unfolding exp_first_term
     by (simp add: ac_simps)
   also
   have summable (λn. ?e n x)
   proof −
     from elim have ?e n x = (((x − t) *R A) ^(n + 1)) /R fact (n + 1) /R (x
− t) for n
       by simp
     then show ?thesis
       by (auto simp only:
```

intro!: *summable_scaleR_right summable_ignore_initial_segment summable_exp_generic*)
qed
then have $(\sum n.\ (x - t) *_R\ ?e\ n\ x) = (x - t) *_R\ (\sum n.\ ?e\ n\ x)$
by (*rule suminf_scaleR_right*[*symmetric*])
also have $(\ldots - (x - t) *_R A) /_R norm\ (x - t) = (x - t) *_R ((\sum n.\ ?e\ n\ x) - A) /_R norm\ (x - t)$
by (*simp add*: *algebra_simps*)
finally show *?case*
by *simp* (*simp add*: *field_simps*)
qed

ultimately have $((\lambda y.\ (exp\ ((y - t) *_R A) - 1 - (y - t) *_R A) /_R norm\ (y - t)) \longrightarrow 0)\ (at\ t\ within\ T)$
by (*rule Lim_transform_eventually*)
from *tendsto_mult_right_zero*[*OF this*, **where** *c*=$exp\ (t *_R A)$]
show $((\lambda y.\ (exp\ (y *_R A) - exp\ (t *_R A) - (y - t) *_R (exp\ (t *_R A) * A)) /_R norm\ (y - t)) \longrightarrow 0)$
$(at\ t\ within\ T)$
by (*rule Lim_transform_eventually*)
(*auto simp*: *field_split_simps exp_add_commuting*[*symmetric*])
qed (*rule bounded_linear_scaleR_left*)

lemma *exp_times_scaleR_commute*: $exp\ (t *_R A) * A = A * exp\ (t *_R A)$
using *exp_times_arg_commute*[*symmetric*, *of* $t *_R A$]
by (*auto simp*: *algebra_simps*)

lemma *exp_scaleR_has_vector_derivative_left*: $((\lambda t.\ exp\ (t *_R A))\ has\_vector\_derivative\ A * exp\ (t *_R A))\ (at\ t)$
using *exp_scaleR_has_vector_derivative_right*[*of A t*]
by (*simp add*: *exp_times_scaleR_commute*)

lemma *field_differentiable_series*:
fixes $f :: nat \Rightarrow 'a::\{real\_normed\_field,banach\} \Rightarrow 'a$
assumes *convex S open S*
assumes $\bigwedge n\ x.\ x \in S \Longrightarrow (f\ n\ has\_field\_derivative\ f'\ n\ x)\ (at\ x)$
assumes *uniformly_convergent_on S* $(\lambda n\ x.\ \sum i<n.\ f'\ i\ x)$
assumes *x0* $\in S$ *summable* $(\lambda n.\ f\ n\ x0)$ **and** *x*: $x \in S$
shows $(\lambda x.\ \sum n.\ f\ n\ x)\ field\_differentiable\ (at\ x)$
proof −
from *assms*(*4*) **obtain** g' **where** *A*: *uniform_limit S* $(\lambda n\ x.\ \sum i<n.\ f'\ i\ x)\ g'$ *sequentially*
unfolding *uniformly_convergent_on_def* **by** *blast*
from *x* **and** ‹*open S*› **have** *S*: *at x within S* = *at x* **by** (*rule at_within_open*)
have $\exists g.\ \forall x{\in}S.\ (\lambda n.\ f\ n\ x)\ sums\ g\ x \wedge (g\ has\_field\_derivative\ g'\ x)\ (at\ x\ within\ S)$
by (*intro has_field_derivative_series*[*of S f f' g' x0*] *assms A has_field_derivative_at_within*)
then obtain *g* **where** *g*: $\bigwedge x.\ x \in S \Longrightarrow (\lambda n.\ f\ n\ x)\ sums\ g\ x$
$\bigwedge x.\ x \in S \Longrightarrow (g\ has\_field\_derivative\ g'\ x)\ (at\ x\ within\ S)$ **by** *blast*
from *g*(*2*)[*OF x*] **have** *g'*: $(g\ has\_derivative\ (*)\ (g'\ x))\ (at\ x)$

by (*simp add*: *has_field_derivative_def S*)
have ((λ*x*. ∑ *n*. *f n x*) *has_derivative* (∗) (*g*′ *x*)) (*at x*)
by (*rule has_derivative_transform_within_open*[*OF g*′ ‹*open S*› *x*])
(*insert g*, *auto simp*: *sums_iff*)
thus (λ*x*. ∑ *n*. *f n x*) *field_differentiable* (*at x*) **unfolding** *differentiable_def*
by (*auto simp*: *summable_def field_differentiable_def has_field_derivative_def*)
qed

Caratheodory characterization

lemma *field_differentiable_caratheodory_at*:
f field_differentiable (*at z*) ⟷
(∃ *g*. (∀ *w*. *f*(*w*) − *f*(*z*) = *g*(*w*) ∗ (*w* − *z*)) ∧ *continuous* (*at z*) *g*)
using *CARAT_DERIV* [*of f*]
by (*simp add*: *field_differentiable_def has_field_derivative_def*)

lemma *field_differentiable_caratheodory_within*:
f field_differentiable (*at z within s*) ⟷
(∃ *g*. (∀ *w*. *f*(*w*) − *f*(*z*) = *g*(*w*) ∗ (*w* − *z*)) ∧ *continuous* (*at z within s*) *g*)
using *DERIV_caratheodory_within* [*of f*]
by (*simp add*: *field_differentiable_def has_field_derivative_def*)

5.17.17 Field derivative

definition *deriv* :: (′*a* ⇒ ′*a*::*real_normed_field*) ⇒ ′*a* ⇒ ′*a* **where**
deriv f x ≡ *SOME D*. *DERIV f x* :> *D*

lemma *DERIV_imp_deriv*: *DERIV f x* :> *f*′ ⟹ *deriv f x* = *f*′
unfolding *deriv_def* **by** (*metis some_equality DERIV_unique*)

lemma *DERIV_deriv_iff_has_field_derivative*:
DERIV f x :> *deriv f x* ⟷ (∃ *f*′. (*f has_field_derivative f*′) (*at x*))
by (*auto simp*: *has_field_derivative_def DERIV_imp_deriv*)

lemma *DERIV_deriv_iff_real_differentiable*:
fixes *x* :: *real*
shows *DERIV f x* :> *deriv f x* ⟷ *f differentiable at x*
unfolding *differentiable_def* **by** (*metis DERIV_imp_deriv has_real_derivative_iff*)

lemma *DERIV_deriv_iff_field_differentiable*:
DERIV f x :> *deriv f x* ⟷ *f field_differentiable at x*
unfolding *field_differentiable_def* **by** (*metis DERIV_imp_deriv*)

lemma *vector_derivative_of_real_left*:
assumes *f differentiable at x*
shows *vector_derivative* (λ*x*. *of_real* (*f x*)) (*at x*) = *of_real* (*deriv f x*)
by (*metis DERIV_deriv_iff_real_differentiable assms has_vector_derivative_of_real vector_derivative_at*)

lemma *vector_derivative_of_real_right*:

assumes *f field_differentiable at (of_real x)*
shows *vector_derivative (λx. f (of_real x)) (at x) = deriv f (of_real x)*
by (*metis DERIV_deriv_iff_field_differentiable assms has_vector_derivative_real_field vector_derivative_at*)

lemma *deriv_cong_ev*:
assumes *eventually (λx. f x = g x) (nhds x) x = y*
shows *deriv f x = deriv g y*
proof −
have *(λD. (f has_field_derivative D) (at x)) = (λD. (g has_field_derivative D) (at y))*
by (*intro ext DERIV_cong_ev refl assms*)
thus *?thesis* **by** (*simp add*: *deriv_def assms*)
qed

lemma *higher_deriv_cong_ev*:
assumes *eventually (λx. f x = g x) (nhds x) x = y*
shows *(deriv ^^ n) f x = (deriv ^^ n) g y*
proof −
from *assms(1)* **have** *eventually (λx. (deriv ^^ n) f x = (deriv ^^ n) g x) (nhds x)*
proof (*induction n arbitrary*: *f g*)
case (*Suc n*)
from *Suc.prems* **have** *eventually (λy. eventually (λz. f z = g z) (nhds y)) (nhds x)*
by (*simp add*: *eventually_eventually*)
hence *eventually (λx. deriv f x = deriv g x) (nhds x)*
by *eventually_elim* (*rule deriv_cong_ev*, *simp_all*)
thus *?case* **by** (*auto intro*!: *deriv_cong_ev Suc simp*: *funpow_Suc_right simp del*: *funpow.simps*)
qed *auto*
with ‹*x = y*› *eventually_nhds_x_imp_x* **show** *?thesis* **by** *blast*
qed

lemma *real_derivative_chain*:
fixes *x* :: *real*
shows *f differentiable at x ⟹ g differentiable at (f x)*
⟹ deriv (g o f) x = deriv g (f x) ∗ deriv f x
by (*metis DERIV_deriv_iff_real_differentiable DERIV_chain DERIV_imp_deriv*)
lemma *field_derivative_eq_vector_derivative*:
(deriv f x) = vector_derivative f (at x)
by (*simp add*: *mult.commute deriv_def vector_derivative_def has_vector_derivative_def has_field_derivative_def*)

proposition *field_differentiable_derivI*:
f field_differentiable (at x) ⟹ (f has_field_derivative deriv f x) (at x)
by (*simp add*: *field_differentiable_def DERIV_deriv_iff_has_field_derivative*)

lemma *vector_derivative_chain_at_general*:

assumes *f differentiable at x g field_differentiable at (f x)*
shows *vector_derivative (g ∘ f) (at x) = vector_derivative f (at x) ∗ deriv g (f x)*
using *assms field_differentiable_derivI field_vector_diff_chain_at vector_derivative_at vector_derivative_works* **by** *blast*

lemma *deriv_chain*:
f field_differentiable at x ⟹ g field_differentiable at (f x)
⟹ deriv (g o f) x = deriv g (f x) ∗ deriv f x
by (*metis DERIV_deriv_iff_field_differentiable DERIV_chain DERIV_imp_deriv*)

lemma *deriv_linear* [*simp*]: *deriv (λw. c ∗ w) = (λz. c)*
by (*metis DERIV_imp_deriv DERIV_cmult_Id*)

lemma *deriv_uminus* [*simp*]: *deriv (λw. −w) = (λz. −1)*
using *deriv_linear*[*of −1*] **by** (*simp del*: *deriv_linear*)

lemma *deriv_ident* [*simp*]: *deriv (λw. w) = (λz. 1)*
by (*metis DERIV_imp_deriv DERIV_ident*)

lemma *deriv_id* [*simp*]: *deriv id = (λz. 1)*
by (*simp add*: *id_def*)

lemma *deriv_const* [*simp*]: *deriv (λw. c) = (λz. 0)*
by (*metis DERIV_imp_deriv DERIV_const*)

lemma *deriv_add* [*simp*]:
⟦*f field_differentiable at z*; *g field_differentiable at z*⟧
⟹ deriv (λw. f w + g w) z = deriv f z + deriv g z
unfolding *DERIV_deriv_iff_field_differentiable*[*symmetric*]
by (*auto intro*!: *DERIV_imp_deriv derivative_intros*)

lemma *deriv_minus* [*simp*]:
f field_differentiable at z ⟹ deriv (λw. − f w) z = − deriv f z
by (*simp add*: *DERIV_deriv_iff_field_differentiable DERIV_imp_deriv Deriv.field_differentiable_minus*)

lemma *deriv_diff* [*simp*]:
⟦*f field_differentiable at z*; *g field_differentiable at z*⟧
⟹ deriv (λw. f w − g w) z = deriv f z − deriv g z
unfolding *DERIV_deriv_iff_field_differentiable*[*symmetric*]
by (*auto intro*!: *DERIV_imp_deriv derivative_intros*)

lemma *deriv_mult* [*simp*]:
⟦*f field_differentiable at z*; *g field_differentiable at z*⟧
⟹ deriv (λw. f w ∗ g w) z = f z ∗ deriv g z + deriv f z ∗ g z
unfolding *DERIV_deriv_iff_field_differentiable*[*symmetric*]
by (*auto intro*!: *DERIV_imp_deriv derivative_eq_intros*)

lemma *deriv_cmult*:
f field_differentiable at z $\Longrightarrow$ *deriv* $(\lambda w.\ c * f\ w)\ z = c * deriv\ f\ z$
by *simp*

lemma *deriv_cmult_right*:
f field_differentiable at z $\Longrightarrow$ *deriv* $(\lambda w.\ f\ w * c)\ z = deriv\ f\ z * c$
by *simp*

lemma *deriv_inverse* [*simp*]:
$[\![f\ field\_differentiable\ at\ z;\ f\ z \neq 0]\!]$
$\Longrightarrow$ *deriv* $(\lambda w.\ inverse\ (f\ w))\ z = -\ deriv\ f\ z\ /\ f\ z\ \hat{}\ 2$
unfolding *DERIV_deriv_iff_field_differentiable*[*symmetric*]
by (*safe intro*!: *DERIV_imp_deriv derivative_eq_intros*) (*auto simp*: *field_split_simps power2_eq_square*)

lemma *deriv_divide* [*simp*]:
$[\![f\ field\_differentiable\ at\ z;\ g\ field\_differentiable\ at\ z;\ g\ z \neq 0]\!]$
$\Longrightarrow$ *deriv* $(\lambda w.\ f\ w\ /\ g\ w)\ z = (deriv\ f\ z * g\ z - f\ z * deriv\ g\ z)\ /\ g\ z\ \hat{}\ 2$
by (*simp add*: *field_class.field_divide_inverse field_differentiable_inverse*)
(*simp add*: *field_split_simps power2_eq_square*)

lemma *deriv_cdivide_right*:
f field_differentiable at z $\Longrightarrow$ *deriv* $(\lambda w.\ f\ w\ /\ c)\ z = deriv\ f\ z\ /\ c$
by (*simp add*: *field_class.field_divide_inverse*)

lemma *deriv_pow*: $[\![f\ field\_differentiable\ at\ z]\!]$
$\Longrightarrow$ *deriv* $(\lambda w.\ f\ w\ \hat{}\ n)\ z = (if\ n{=}0\ then\ 0\ else\ n * deriv\ f\ z * f\ z\ \hat{}\ (n - Suc\ 0))$
unfolding *DERIV_deriv_iff_field_differentiable*[*symmetric*]
by (*auto intro*!: *DERIV_imp_deriv derivative_eq_intros*)

lemma *deriv_sum* [*simp*]:
$[\![\bigwedge i.\ f\ i\ field\_differentiable\ at\ z]\!]$
$\Longrightarrow$ *deriv* $(\lambda w.\ sum\ (\lambda i.\ f\ i\ w)\ S)\ z = sum\ (\lambda i.\ deriv\ (f\ i)\ z)\ S$
unfolding *DERIV_deriv_iff_field_differentiable*[*symmetric*]
by (*auto intro*!: *DERIV_imp_deriv derivative_intros*)

lemma *deriv_compose_linear*:
assumes *f field_differentiable at* $(c * z)$
shows *deriv* $(\lambda w.\ f\ (c * w))\ z = c * deriv\ f\ (c * z)$
proof −
have *deriv* $(\lambda a.\ f\ (c * a))\ z = deriv\ f\ (c * z) * c$
using *assms* **by** (*simp add*: *DERIV_chain2 DERIV_deriv_iff_field_differentiable DERIV_imp_deriv*)
then show *?thesis*
by *simp*
qed

lemma *nonzero_deriv_nonconstant*:
assumes *df*: *DERIV f ξ :> df* **and** *S*: *open S ξ ∈ S* **and** *df ≠ 0*
shows ¬ *f constant_on S*
unfolding *constant_on_def*
by (*metis ‹df ≠ 0›* *has_field_derivative_transform_within_open* [*OF df S*] *DERIV_const DERIV_unique*)

5.17.18 Relation between convexity and derivative

proposition *convex_on_imp_above_tangent*:
assumes *convex*: *convex_on A f* **and** *connected*: *connected A*
assumes *c*: *c ∈ interior A* **and** *x* : *x ∈ A*
assumes *deriv*: (*f has_field_derivative f'*) (*at c within A*)
shows *f x − f c ≥ f' ∗ (x − c)*
proof (*cases x c rule*: *linorder_cases*)
assume *xc*: *x > c*
let *?A' = interior A ∩ {c<..}*
from *c* **have** *c ∈ interior A ∩ closure {c<..}* **by** *auto*
also have *... ⊆ closure* (*interior A ∩ {c<..}*) **by** (*intro open_Int_closure_subset*) *auto*
finally have *at c within ?A' ≠ bot* **by** (*subst at_within_eq_bot_iff*) *auto*
moreover from *deriv* **have** ((*λy. (f y − f c) / (y − c)*) ⟶ *f'*) (*at c within ?A'*)
unfolding *has_field_derivative_iff* **using** *interior_subset*[*of A*] **by** (*blast intro*: *tendsto_mono at_le*)
moreover from *eventually_at_right_real*[*OF xc*]
have *eventually* (*λy. (f y − f c) / (y − c) ≤ (f x − f c) / (x − c)*) (*at_right c*)
proof *eventually_elim*
fix *y* **assume** *y*: *y ∈ {c<..<x}*
with *convex connected x c* **have** *f y ≤ (f x − f c) / (x − c) ∗ (y − c) + f c*
using *interior_subset*[*of A*]
by (*intro convex_onD_Icc' convex_on_subset*[*OF convex*] *connected_contains_Icc*) *auto*
hence *f y − f c ≤ (f x − f c) / (x − c) ∗ (y − c)* **by** *simp*
thus *(f y − f c) / (y − c) ≤ (f x − f c) / (x − c)* **using** *y xc* **by** (*simp add*: *field_split_simps*)
qed
hence *eventually* (*λy. (f y − f c) / (y − c) ≤ (f x − f c) / (x − c)*) (*at c within ?A'*)
by (*blast intro*: *filter_leD at_le*)
ultimately have *f' ≤ (f x − f c) / (x − c)* **by** (*simp add*: *tendsto_upperbound*)
thus *?thesis* **using** *xc* **by** (*simp add*: *field_simps*)
next
assume *xc*: *x < c*
let *?A' = interior A ∩ {..<c}*
from *c* **have** *c ∈ interior A ∩ closure {..<c}* **by** *auto*
also have *... ⊆ closure* (*interior A ∩ {..<c}*) **by** (*intro open_Int_closure_subset*) *auto*
finally have *at c within ?A' ≠ bot* **by** (*subst at_within_eq_bot_iff*) *auto*

moreover from *deriv* **have** $((\lambda y.\ (f\ y - f\ c)\ /\ (y - c)) \longrightarrow f')\ (at\ c\ within\ ?A')$
unfolding *has_field_derivative_iff* **using** *interior_subset*[*of A*] **by** (*blast intro*: *tendsto_mono at_le*)
moreover from *eventually_at_left_real*[*OF xc*]
have *eventually* $(\lambda y.\ (f\ y - f\ c)\ /\ (y - c) \geq (f\ x - f\ c)\ /\ (x - c))\ (at\_left\ c)$
proof *eventually_elim*
fix *y* **assume** *y*: $y \in \{x<..<c\}$
with *convex connected x c* **have** $f\ y \leq (f\ x - f\ c)\ /\ (c - x) * (c - y) + f\ c$
using *interior_subset*[*of A*]
by (*intro convex_onD_Icc'' convex_on_subset*[*OF convex*] *connected_contains_Icc*) *auto*
hence $f\ y - f\ c \leq (f\ x - f\ c) * ((c - y)\ /\ (c - x))$ **by** *simp*
also have $(c - y)\ /\ (c - x) = (y - c)\ /\ (x - c)$ **using** *y xc* **by** (*simp add*: *field_simps*)
finally show $(f\ y - f\ c)\ /\ (y - c) \geq (f\ x - f\ c)\ /\ (x - c)$ **using** *y xc*
by (*simp add*: *field_split_simps*)
qed
hence *eventually* $(\lambda y.\ (f\ y - f\ c)\ /\ (y - c) \geq (f\ x - f\ c)\ /\ (x - c))\ (at\ c\ within\ ?A')$
by (*blast intro*: *filter_leD at_le*)
ultimately have $f' \geq (f\ x - f\ c)\ /\ (x - c)$ **by** (*simp add*: *tendsto_lowerbound*)
thus *?thesis* **using** *xc* **by** (*simp add*: *field_simps*)
qed *simp_all*

5.17.19 Partial derivatives

lemma *eventually_at_Pair_within_TimesI1*:
fixes *x*::*'a*::*metric_space*
assumes $\forall_F\ x'$ *in at x within X*. *P x'*
assumes *P x*
shows $\forall_F\ (x', y')$ *in at* (x, y) *within* $X \times Y$. *P x'*
proof −
from *assms*[*unfolded eventually_at_topological*]
obtain *S* **where** *S*: *open S* $x \in S$ $\bigwedge x'.\ x' \in X \Longrightarrow x' \in S \Longrightarrow P\ x'$
by *metis*
show $\forall_F\ (x', y')$ *in at* (x, y) *within* $X \times Y$. *P x'*
unfolding *eventually_at_topological*
by (*auto intro*!: *exI*[**where** *x*=$S \times UNIV$] *S open_Times*)
qed

lemma *eventually_at_Pair_within_TimesI2*:
fixes *x*::*'a*::*metric_space*
assumes $\forall_F\ y'$ *in at y within Y*. *P y' P y*
shows $\forall_F\ (x', y')$ *in at* (x, y) *within* $X \times Y$. *P y'*
proof −
from *assms*[*unfolded eventually_at_topological*]
obtain *S* **where** *S*: *open S y* $\in S$ $\bigwedge y'.\ y' \in Y \Longrightarrow y' \in S \Longrightarrow P\ y'$
by *metis*

```
  show ∀F (x', y') in at (x, y) within X × Y. P y'
    unfolding eventually_at_topological
    by (auto intro!: exI[where x=UNIV × S] S open_Times)
qed

proposition has_derivative_partialsI:
 fixes f::'a::real_normed_vector ⇒ 'b::real_normed_vector ⇒ 'c::real_normed_vector
  assumes fx: ((λx. f x y) has_derivative fx) (at x within X)
 assumes fy: ⋀x y. x ∈ X ⟹ y ∈ Y ⟹ ((λy. f x y) has_derivative blinfun_apply
(fy x y)) (at y within Y)
 assumes fy_cont[unfolded continuous_within]: continuous (at (x, y) within X ×
Y) (λ(x, y). fy x y)
  assumes y ∈ Y convex Y
  shows ((λ(x, y). f x y) has_derivative (λ(tx, ty). fx tx + fy x y ty)) (at (x, y)
within X × Y)
proof (safe intro!: has_derivativeI tendstoI, goal_cases)
  case (2 e')
 interpret fx: bounded_linear fx using fx by (rule has_derivative_bounded_linear)
  define e where e = e' / 9
  have e > 0 using ‹e' > 0› by (simp add: e_def)

  from fy_cont[THEN tendstoD, OF ‹e > 0›]
  have ∀F (x', y') in at (x, y) within X × Y. dist (fy x' y') (fy x y) < e
    by (auto simp: split_beta')
  from this[unfolded eventually_at] obtain d' where
    d' > 0
    ⋀x' y'. x' ∈ X ⟹ y' ∈ Y ⟹ (x', y') ≠ (x, y) ⟹ dist (x', y') (x, y) < d'
⟹
      dist (fy x' y') (fy x y) < e
    by auto
  then
  have d': x' ∈ X ⟹ y' ∈ Y ⟹ dist (x', y') (x, y) < d' ⟹ dist (fy x' y') (fy
x y) < e
    for x' y'
    using ‹0 < e›
    by (cases (x', y') = (x, y)) auto
  define d where d = d' / sqrt 2
  have d > 0 using ‹0 < d'› by (simp add: d_def)
  have d: x' ∈ X ⟹ y' ∈ Y ⟹ dist x' x < d ⟹ dist y' y < d ⟹ dist (fy x'
y') (fy x y) < e
    for x' y'
    by (auto simp: dist_prod_def d_def intro!: d' real_sqrt_sum_squares_less)

  let ?S = ball y d ∩ Y
  have convex ?S
    by (auto intro!: convex_Int ‹convex Y›)
  {
    fix x'::'a and y'::'b
    assume x': x' ∈ X and y': y' ∈ Y
```

```
    assume dx': dist x' x < d and dy': dist y' y < d
    have norm (fy x' y' - fy x' y) ≤ dist (fy x' y') (fy x y) + dist (fy x' y) (fy x y)
      by norm
    also have dist (fy x' y') (fy x y) < e
      by (rule d; fact)
    also have dist (fy x' y) (fy x y) < e
      by (auto intro!: d simp: dist_prod_def x' ‹d > 0› ‹y ∈ Y› dx')
    finally
    have norm (fy x' y' - fy x' y) < e + e
      by arith
    then have onorm (blinfun_apply (fy x' y') - blinfun_apply (fy x' y)) < e + e
      by (auto simp: norm_blinfun.rep_eq blinfun.diff_left[abs_def] fun_diff_def)
  } note onorm = this

  have ev_mem: ∀F (x', y') in at (x, y) within X × Y. (x', y') ∈ X × Y
    using ‹y ∈ Y›
    by (auto simp: eventually_at intro!: zero_less_one)
  moreover
  have ev_dist: ∀F xy in at (x, y) within X × Y. dist xy (x, y) < d if d > 0 for
d
    using eventually_at_ball[OF that]
    by (rule eventually_elim2) (auto simp: dist_commute intro!: eventually_True)
  note ev_dist[OF ‹0 < d›]
  ultimately
  have ∀F (x', y') in at (x, y) within X × Y.
    norm (f x' y' - f x' y - (fy x' y) (y' - y)) ≤ norm (y' - y) * (e + e)
  proof (eventually_elim, safe)
    fix x' y'
    assume x' ∈ X and y': y' ∈ Y
    assume dist: dist (x', y') (x, y) < d
    then have dx: dist x' x < d and dy: dist y' y < d
      unfolding dist_prod_def fst_conv snd_conv atomize_conj
    by (metis le_less_trans real_sqrt_sum_squares_ge1 real_sqrt_sum_squares_ge2)
    {
      fix t::real
      assume t ∈ {0 .. 1}
      then have y + t *R (y' - y) ∈ closed_segment y y'
        by (auto simp: closed_segment_def algebra_simps intro!: exI[where x=t])
      also
      have ... ⊆ ball y d ∩ Y
        using ‹y ∈ Y› ‹0 < d› dy y'
        by (intro ‹convex ?S›[unfolded convex_contains_segment, rule_format, of
y y'])
           (auto simp: dist_commute)
      finally have y + t *R (y' - y) ∈ ?S .
    } note seg = this

    have ⋀x. x ∈ ball y d ∩ Y ⟹ onorm (blinfun_apply (fy x' x) - blinfun_apply
(fy x' y)) ≤ e + e
```

```
    by (safe intro!: onorm less_imp_le ‹x' ∈ X› dx) (auto simp: dist_commute
‹0 < d› ‹y ∈ Y›)
  with seg has_derivative_subset[OF assms(2)[OF ‹x' ∈ X›]]
  show norm (f x' y' − f x' y − (fy x' y) (y' − y)) ≤ norm (y' − y) * (e + e)
    by (rule differentiable_bound_linearization[where S=?S])
      (auto intro!: ‹0 < d› ‹y ∈ Y›)
 qed
 moreover
 let ?le = λx'. norm (f x' y − f x y − (fx) (x' − x)) ≤ norm (x' − x) * e
 from fx[unfolded has_derivative_within, THEN conjunct2, THEN tendstoD, OF
‹0 < e›]
 have ∀F x' in at x within X. ?le x'
  by eventually_elim (simp,
    simp add: dist_norm field_split_simps split: if_split_asm)
 then have ∀F (x', y') in at (x, y) within X × Y. ?le x'
  by (rule eventually_at_Pair_within_TimesI1)
    (simp add: blinfun.bilinear_simps)
 moreover have ∀F (x', y') in at (x, y) within X × Y. norm ((x', y') − (x, y))
≠ 0
  unfolding norm_eq_zero right_minus_eq
  by (auto simp: eventually_at intro!: zero_less_one)
 moreover
 from fy_cont[THEN tendstoD, OF ‹0 < e›]
 have ∀F x' in at x within X. norm (fy x' y − fy x y) < e
  unfolding eventually_at
  using ‹y ∈ Y›
  by (auto simp: dist_prod_def dist_norm)
 then have ∀F (x', y') in at (x, y) within X × Y. norm (fy x' y − fy x y) < e
  by (rule eventually_at_Pair_within_TimesI1)
    (simp add: blinfun.bilinear_simps ‹0 < e›)
 ultimately
 have ∀F (x', y') in at (x, y) within X × Y.
        norm ((f x' y' − f x y − (fx (x' − x) + fy x y (y' − y))) /R
          norm ((x', y') − (x, y)))
        < e'
 proof (eventually_elim, safe)
  fix x' y'
  have norm (f x' y' − f x y − (fx (x' − x) + fy x y (y' − y))) ≤
      norm (f x' y' − f x' y − fy x' y (y' − y)) +
      norm (fy x y (y' − y) − fy x' y (y' − y)) +
      norm (f x' y − f x y − fx (x' − x))
    by norm
  also
  assume nz: norm ((x', y') − (x, y)) ≠ 0
    and nfy: norm (fy x' y − fy x y) < e
  assume norm (f x' y' − f x' y − blinfun_apply (fy x' y) (y' − y)) ≤ norm (y'
− y) * (e + e)
  also assume norm (f x' y − f x y − (fx) (x' − x)) ≤ norm (x' − x) * e
  also
```

```
    have norm ((fy x y) (y' − y) − (fy x' y) (y' − y)) ≤ norm ((fy x y) − (fy x' y)) * norm (y' − y)
      by (auto simp: blinfun.bilinear_simps[symmetric] intro!: norm_blinfun)
    also have ... ≤ (e + e) * norm (y' − y)
      using ‹e > 0› nfy
      by (auto simp: norm_minus_commute intro!: mult_right_mono)
    also have norm (x' − x) * e ≤ norm (x' − x) * (e + e)
      using ‹0 < e› by simp
    also have norm (y' − y) * (e + e) + (e + e) * norm (y' − y) + norm (x' − x) * (e + e) ≤
        (norm (y' − y) + norm (x' − x)) * (4 * e)
      using ‹e > 0›
      by (simp add: algebra_simps)
    also have ... ≤ 2 * norm ((x', y') − (x, y)) * (4 * e)
      using ‹0 < e› real_sqrt_sum_squares_ge1[of norm (x' − x) norm (y' − y)]
        real_sqrt_sum_squares_ge2[of norm (y' − y) norm (x' − x)]
      by (auto intro!: mult_right_mono simp: norm_prod_def
        simp del: real_sqrt_sum_squares_ge1 real_sqrt_sum_squares_ge2)
    also have ... ≤ norm ((x', y') − (x, y)) * (8 * e)
      by simp
    also have ... < norm ((x', y') − (x, y)) * e'
      using ‹0 < e'› nz
      by (auto simp: e_def)
    finally show norm ((f x' y' − f x y − (fx (x' − x) + fy x y (y' − y))) /R norm ((x', y') − (x, y))) < e'
      by (simp add: dist_norm) (auto simp add: field_split_simps)
  qed
  then show ?case
    by eventually_elim (auto simp: dist_norm field_simps)
next
  from has_derivative_bounded_linear[OF fx]
  obtain fxb where fx = blinfun_apply fxb
    by (metis bounded_linear_Blinfun_apply)
  then show bounded_linear (λ(tx, ty). fx tx + blinfun_apply (fy x y) ty)
    by (auto intro!: bounded_linear_intros simp: split_beta')
qed
```

5.17.20 Differentiable case distinction

```
lemma has_derivative_within_If_eq:
  ((λx. if P x then f x else g x) has_derivative f') (at x within S) =
    (bounded_linear f' ∧
      ((λy.(if P y then (f y − ((if P x then f x else g x) + f' (y − x)))/R norm (y − x)
          else (g y − ((if P x then f x else g x) + f' (y − x)))/R norm (y − x)))
      ⟶ 0) (at x within S))
  (is _ = (_ ∧ (?if ⟶ 0) _))
proof −
  have (λy. (1 / norm (y − x)) *R
```

```
        ((if P y then f y else g y) −
         ((if P x then f x else g x) + f' (y − x)))) = ?if
    by (auto simp: inverse_eq_divide)
  thus ?thesis by (auto simp: has_derivative_within)
qed

lemma has_derivative_If_within_closures:
  assumes f': x ∈ S ∪ (closure S ∩ closure T) ⟹
    (f has_derivative f' x) (at x within S ∪ (closure S ∩ closure T))
  assumes g': x ∈ T ∪ (closure S ∩ closure T) ⟹
    (g has_derivative g' x) (at x within T ∪ (closure S ∩ closure T))
  assumes connect: x ∈ closure S ⟹ x ∈ closure T ⟹ f x = g x
  assumes connect': x ∈ closure S ⟹ x ∈ closure T ⟹ f' x = g' x
  assumes x_in: x ∈ S ∪ T
  shows ((λx. if x ∈ S then f x else g x) has_derivative
      (if x ∈ S then f' x else g' x)) (at x within (S ∪ T))
proof −
  from f' x_in interpret f': bounded_linear if x ∈ S then f' x else (λx. 0)
    by (auto simp add: has_derivative_within)
  from g' interpret g': bounded_linear if x ∈ T then g' x else (λx. 0)
    by (auto simp add: has_derivative_within)
  have bl: bounded_linear (if x ∈ S then f' x else g' x)
    using f'.scaleR f'.bounded f'.add g'.scaleR g'.bounded g'.add x_in
    by (unfold_locales; force)
  show ?thesis
    using f' g' closure_subset[of T] closure_subset[of S]
    unfolding has_derivative_within_If_eq
    by (intro conjI bl tendsto_If_within_closures x_in)
      (auto simp: has_derivative_within inverse_eq_divide connect connect' sub-
setD)
qed

lemma has_vector_derivative_If_within_closures:
  assumes x_in: x ∈ S ∪ T
  assumes u = S ∪ T
  assumes f': x ∈ S ∪ (closure S ∩ closure T) ⟹
    (f has_vector_derivative f' x) (at x within S ∪ (closure S ∩ closure T))
  assumes g': x ∈ T ∪ (closure S ∩ closure T) ⟹
    (g has_vector_derivative g' x) (at x within T ∪ (closure S ∩ closure T))
  assumes connect: x ∈ closure S ⟹ x ∈ closure T ⟹ f x = g x
  assumes connect': x ∈ closure S ⟹ x ∈ closure T ⟹ f' x = g' x
  shows ((λx. if x ∈ S then f x else g x) has_vector_derivative
    (if x ∈ S then f' x else g' x)) (at x within u)
  unfolding has_vector_derivative_def assms
  using x_in f' g'
  by (intro has_derivative_If_within_closures[where ?f' = λx a. a *R f' x and
?g' = λx a. a *R g' x,
      THEN has_derivative_eq_rhs]; force simp: assms has_vector_derivative_def)
```

5.17.21 The Inverse Function Theorem

lemma *linear_injective_contraction*:
assumes *linear f c < 1* **and** *le*: $\bigwedge x.\ norm\ (f\ x - x) \leq c * norm\ x$
shows *inj f*
unfolding *linear_injective_0*[*OF* ‹*linear f*›]
proof *safe*
fix *x*
assume *f x = 0*
with *le* [*of x*] **have** *norm x ≤ c * norm x*
by *simp*
then show *x = 0*
using ‹*c < 1*› **by** (*simp add*: *mult_le_cancel_right1*)
qed

From an online proof by J. Michael Boardman, Department of Mathematics, Johns Hopkins University

lemma *inverse_function_theorem_scaled*:
fixes *f*::*'a*::*euclidean_space* ⇒ *'a*
and *f'*::*'a* ⇒ (*'a* ⇒$_L$ *'a*)
assumes *open U*
and *derf*: $\bigwedge x.\ x \in U \Longrightarrow (f\ has\_derivative\ blinfun\_apply\ (f'\ x))\ (at\ x)$
and *contf*: *continuous_on U f'*
and *0 ∈ U* **and** [*simp*]: *f 0 = 0*
and *id*: *f' 0 = id_blinfun*
obtains *U' V g g'* **where** *open U' U' ⊆ U 0 ∈ U' open V 0 ∈ V homeomorphism U' V f g*
$\bigwedge y.\ y \in V \Longrightarrow (g\ has\_derivative\ (g'\ y))\ (at\ y)$
$\bigwedge y.\ y \in V \Longrightarrow g'\ y = inv\ (blinfun\_apply\ (f'(g\ y)))$
$\bigwedge y.\ y \in V \Longrightarrow bij\ (blinfun\_apply\ (f'(g\ y)))$
proof −
obtain *d1* **where** *cball 0 d1 ⊆ U d1 > 0*
using ‹*open U*› ‹*0 ∈ U*› *open_contains_cball* **by** *blast*
obtain *d2* **where** *d2*: $\bigwedge x.\ [\![x \in U;\ dist\ x\ 0 \leq d2]\!] \Longrightarrow dist\ (f'\ x)\ (f'\ 0) < 1/2$ *0 < d2*
using *continuous_onE* [*OF contf*, *of 0 1/2*] **by** (*metis* ‹*0 ∈ U*› *half_gt_zero_iff zero_less_one*)
obtain *δ* **where** *le*: $\bigwedge x.\ norm\ x \leq \delta \Longrightarrow dist\ (f'\ x)\ id\_blinfun \leq 1/2$ **and** *0 < δ*
and *subU*: *cball 0 δ ⊆ U*
proof
show *min d1 d2 > 0*
by (*simp add*: ‹*0 < d1*› ‹*0 < d2*›)
show *cball 0* (*min d1 d2*) *⊆ U*
using ‹*cball 0 d1 ⊆ U*› **by** *auto*
show *dist* (*f' x*) *id_blinfun ≤ 1/2* **if** *norm x ≤ min d1 d2* **for** *x*
using ‹*cball 0 d1 ⊆ U*› *d2 that id* **by** *fastforce*
qed
let *?D = cball 0 δ*
define *V*:: *'a set* **where** *V ≡ ball 0* (*δ/2*)

```
  have 4: norm (f (x + h) − f x − h) ≤ 1/2 ∗ norm h
    if x ∈ ?D x+h ∈ ?D for x h
  proof −
    let ?w = λx. f x − x
    have B: ⋀x. x ∈ ?D ⟹ onorm (blinfun_apply (f' x − id_blinfun)) ≤ 1/2
      by (metis dist_norm le mem_cball_0 norm_blinfun.rep_eq)
    have ⋀x. x ∈ ?D ⟹ (?w has_derivative (blinfun_apply (f' x − id_blinfun)))
(at x)
      by (rule derivative_eq_intros derf subsetD [OF subU] | force simp: blin-
fun.diff_left)+
    then have Dw: ⋀x. x ∈ ?D ⟹ (?w has_derivative (blinfun_apply (f' x −
id_blinfun))) (at x within ?D)
      using has_derivative_at_withinI by blast
    have norm (?w (x+h) − ?w x) ≤ (1/2) ∗ norm h
      using differentiable_bound [OF convex_cball Dw B] that by fastforce
    then show ?thesis
      by (auto simp: algebra_simps)
  qed
  have for_g: ∃!x. norm x < δ ∧ f x = y if y: norm y < δ/2 for y
  proof −
    let ?u = λx. x + (y − f x)
    have ∗: norm (?u x) < δ if x ∈ ?D for x
    proof −
      have fxx: norm (f x − x) ≤ δ/2
        using 4 [of 0 x] ‹0 < δ› ‹f 0 = 0› that by auto
      have norm (?u x) ≤ norm y + norm (f x − x)
      by (metis add.commute add_diff_eq norm_minus_commute norm_triangle_ineq)
      also have ... < δ/2 + δ/2
        using fxx y by auto
      finally show ?thesis
        by simp
    qed
    have ∃!x ∈ ?D. ?u x = x
    proof (rule banach_fix)
      show cball 0 δ ≠ {}
        using ‹0 < δ› by auto
      show (λx. x + (y − f x)) ‘ cball 0 δ ⊆ cball 0 δ
        using ∗ by force
      have dist (x + (y − f x)) (xh + (y − f xh)) ∗ 2 ≤ dist x xh
        if norm x ≤ δ and norm xh ≤ δ for x xh
        using that 4 [of x xh−x] by (auto simp: dist_norm norm_minus_commute
algebra_simps)
      then show ∀x∈cball 0 δ. ∀ya∈cball 0 δ. dist (x + (y − f x)) (ya + (y − f
ya)) ≤ (1/2) ∗ dist x ya
        by auto
    qed (auto simp: complete_eq_closed)
    then show ?thesis
      by (metis ∗ add_cancel_right_right eq_iff_diff_eq_0 le_less mem_cball_0)
  qed
```

define g **where** $g \equiv \lambda y.\ THE\ x.\ norm\ x < \delta \wedge f\ x = y$
have g: $norm\ (g\ y) < \delta \wedge f\ (g\ y) = y$ **if** $norm\ y < \delta/2$ **for** y
unfolding g_def **using** $that\ theI'\ [OF\ for\_g]$ **by** $meson$
then have $fg[simp]$: $f\ (g\ y) = y$ **if** $y \in V$ **for** y
using $that$ **by** $(auto\ simp\colon V\_def)$
have 5: $norm\ (g\ y' - g\ y) \leq 2 * norm\ (y' - y)$ **if** $y \in V\ y' \in V$ **for** $y\ y'$
proof −
have no: $norm\ (g\ y) \leq \delta\ norm\ (g\ y') \leq \delta$ **and** $[simp]$: $f\ (g\ y) = y$
using $that\ g$ **unfolding** V_def **by** $force+$
have $norm\ (g\ y' - g\ y) \leq norm\ (g\ y' - g\ y - (y' - y)) + norm\ (y' - y)$
by $(simp\ add\colon add.commute\ norm\_triangle\_sub)$
also have $\ldots \leq (1/2) * norm\ (g\ y' - g\ y) + norm\ (y' - y)$
using $4\ [of\ g\ y\ g\ y' - g\ y]\ that\ no$ **by** $(simp\ add\colon g\ norm\_minus\_commute\ V\_def)$
finally show $?thesis$
by $auto$
qed
have $contg$: $continuous\_on\ V\ g$
proof
fix y::$'a$ **and** e::$real$
assume $0 < e$ **and** y: $y \in V$
show $\exists\, d{>}0.\ \forall\, x'{\in}V.\ dist\ x'\ y < d \longrightarrow dist\ (g\ x')\ (g\ y) \leq e$
proof $(intro\ exI\ conjI\ ballI\ impI)$
show $0 < e/2$
by $(simp\ add\colon \langle 0 < e \rangle)$
qed $(use\ 5\ y$ **in** $\langle force\ simp\colon dist\_norm \rangle)$
qed
show $thesis$
proof
define U' **where** $U' \equiv (f\ -`\ V) \cap ball\ 0\ \delta$
have $contf$: $continuous\_on\ U\ f$
using $derf\ has\_derivative\_at\_withinI$ **by** $(fast\ intro\colon has\_derivative\_continuous\_on)$
then have $continuous\_on\ (ball\ 0\ \delta)\ f$
by $(meson\ ball\_subset\_cball\ continuous\_on\_subset\ subU)$
then show $open\ U'$
by $(simp\ add\colon U'\_def\ V\_def\ Int\_commute\ continuous\_open\_preimage)$
show $0 \in U'\ U' \subseteq U\ open\ V\ 0 \in V$
using $\langle 0 < \delta \rangle\ subU$ **by** $(auto\ simp\colon U'\_def\ V\_def)$
show $hom\colon homeomorphism\ U'\ V\ f\ g$
proof
show $continuous\_on\ U'\ f$
using $\langle U' \subseteq U \rangle\ contf\ continuous\_on\_subset$ **by** $blast$
show $continuous\_on\ V\ g$
using $contg$ **by** $blast$
show $f\ `\ U' \subseteq V$
using U'_def **by** $blast$
show $g\ `\ V \subseteq U'$
by $(simp\ add\colon U'\_def\ V\_def\ g\ image\_subset\_iff)$
show $g\ (f\ x) = x$ **if** $x \in U'$ **for** x

```
      by (metis that fg Int_iff U'_def V_def for_g g mem_ball_0 vimage_eq)
    show f (g y) = y if y ∈ V for y
      using that by (simp add: g V_def)
  qed
  show bij: bij (blinfun_apply (f'(g y))) if y ∈ V for y
  proof −
    have inj: inj (blinfun_apply (f' (g y)))
    proof (rule linear_injective_contraction)
      show linear (blinfun_apply (f' (g y)))
        using blinfun.bounded_linear_right bounded_linear_def by blast
    next
      fix x
      have norm (blinfun_apply (f' (g y)) x − x) = norm (blinfun_apply (f' (g
y) − id_blinfun) x)
        by (simp add: blinfun.diff_left)
      also have ... ≤ norm (f' (g y) − id_blinfun) ∗ norm x
        by (rule norm_blinfun)
      also have ... ≤ (1/2) ∗ norm x
      proof (rule mult_right_mono)
        show norm (f' (g y) − id_blinfun) ≤ 1/2
          using that g [of y] le by (auto simp: V_def dist_norm)
      qed auto
      finally show norm (blinfun_apply (f' (g y)) x − x) ≤ (1/2) ∗ norm x .
    qed auto
    moreover
    have surj (blinfun_apply (f' (g y)))
      using blinfun.bounded_linear_right bounded_linear_def
      by (blast intro!: linear_inj_imp_surj [OF _ inj])
    ultimately show ?thesis
      using bijI by blast
  qed
  define g' where g' ≡ λy. inv (blinfun_apply (f'(g y)))
  show (g has_derivative g' y) (at y) if y ∈ V for y
  proof −
    have gy: g y ∈ U
      using g subU that unfolding V_def by fastforce
    obtain e where e: ⋀h. f (g y + h) = y + blinfun_apply (f' (g y)) h + e h
      and e0: (λh. norm (e h) / norm h) −0→ 0
      using iffD1 [OF has_derivative_iff_Ex derf [OF gy]] ‹y ∈ V› by auto
    have [simp]: e 0 = 0
      using e [of 0] that by simp
    let ?INV = inv (blinfun_apply (f' (g y)))
    have inj: inj (blinfun_apply (f' (g y)))
      using bij bij_betw_def that by blast
    have (g has_derivative g' y) (at y within V)
      unfolding has_derivative_at_within_iff_Ex [OF ‹y ∈ V› ‹open V›]
    proof
      show blinv: bounded_linear (g' y)
        unfolding g'_def using derf gy inj inj_linear_imp_inv_bounded_linear
```

by *blast*
define *eg* **where** *eg* ≡ λ*k*. − *?INV* (*e* (*g* (*y*+*k*) − *g y*))
have *g* (*y*+*k*) = *g y* + *g*′ *y k* + *eg k* **if** *y* + *k* ∈ *V* **for** *k*
proof −
have *?INV k* = *?INV* (*blinfun_apply* (*f*′ (*g y*)) (*g* (*y*+*k*) − *g y*) + *e* (*g* (*y*+*k*) − *g y*))
using *e* [*of g*(*y*+*k*) − *g y*] *that* **by** *simp*
then have *g* (*y*+*k*) = *g y* + *?INV k* − *?INV* (*e* (*g* (*y*+*k*) − *g y*))
using *inj blinv* **by** (*simp add*: *linear_simps g*′*_def*)
then show *?thesis*
by (*auto simp*: *eg_def g*′*_def*)
qed
moreover have (λ*k*. *norm* (*eg k*) / *norm k*) −0→ *0*
proof (*rule Lim_null_comparison*)
let *?g* = λ*k*. *2* ∗ *onorm ?INV* ∗ *norm* (*e* (*g* (*y*+*k*) − *g y*)) / *norm* (*g* (*y*+*k*) − *g y*)
show ∀ *F k in at 0*. *norm* (*norm* (*eg k*) / *norm k*) ≤ *?g k*
unfolding *eventually_at_topological*
proof (*intro exI conjI ballI impI*)
show *open* ((+)(−*y*) ‘ *V*)
using ‹*open V*› *open_translation* **by** *blast*
show *0* ∈ (+)(−*y*) ‘ *V*
by (*simp add*: *that*)
show *norm* (*norm* (*eg k*) / *norm k*) ≤ *2* ∗ *onorm* (*inv* (*blinfun_apply* (*f*′ (*g y*)))) ∗ *norm* (*e* (*g* (*y*+*k*) − *g y*)) / *norm* (*g* (*y*+*k*) − *g y*)
if *k* ∈ (+)(−*y*) ‘ *V k* ≠ *0* **for** *k*
proof −
have *y*+*k* ∈ *V*
using *that* **by** *auto*
have *norm* (*norm* (*eg k*) / *norm k*) ≤ *onorm ?INV* ∗ *norm* (*e* (*g* (*y*+*k*) − *g y*)) / *norm k*
using *blinv g*′*_def onorm* **by** (*force simp*: *eg_def divide_simps*)
also have ... = (*norm* (*g* (*y*+*k*) − *g y*) / *norm k*) ∗ (*onorm ?INV* ∗ (*norm* (*e* (*g* (*y*+*k*) − *g y*)) / *norm* (*g* (*y*+*k*) − *g y*)))
by (*simp add*: *divide_simps*)
also have ... ≤ *2* ∗ (*onorm ?INV* ∗ (*norm* (*e* (*g* (*y*+*k*) − *g y*)) / *norm* (*g* (*y*+*k*) − *g y*)))
apply (*rule mult_right_mono*)
using *5* [*of y y*+*k*] ‹*y* ∈ *V*› ‹*y* + *k* ∈ *V*› *onorm_pos_le* [*OF blinv*]
apply (*auto simp*: *divide_simps zero_le_mult_iff zero_le_divide_iff g*′*_def*)
done
finally show *norm* (*norm* (*eg k*) / *norm k*) ≤ *2* ∗ *onorm ?INV* ∗ *norm* (*e* (*g* (*y*+*k*) − *g y*)) / *norm* (*g* (*y*+*k*) − *g y*)
by *simp*
qed
qed
have *1*: (λ*h*. *norm* (*e h*) / *norm h*) −0→ (*norm* (*e 0*) / *norm 0*)
using *e0* **by** *auto*

```
      have 2: (λk. g (y+k) − g y) −0→ 0
        using contg ‹open V› ‹y ∈ V› LIM_offset_zero_iff LIM_zero_iff
at_within_open continuous_on_def by fastforce
      from tendsto_compose [OF 1 2, simplified]
      have (λk. norm (e (g (y+k) − g y)) / norm (g (y+k) − g y)) −0→ 0 .
      from tendsto_mult_left [OF this] show ?g −0→ 0 by auto
    qed
    ultimately show ∃e. (∀k. y + k ∈ V ⟶ g (y+k) = g y + g' y k + e k)
∧ (λk. norm (e k) / norm k) −0→ 0
      by blast
  qed
  then show ?thesis
    by (metis ‹open V› at_within_open that)
 qed
 show g' y = inv (blinfun_apply (f' (g y)))
  if y ∈ V for y
  by (simp add: g'_def)
qed
qed
```

We need all this to justify the scaling and translations.

```
theorem inverse_function_theorem:
  fixes f::'a::euclidean_space ⇒ 'a
    and f'::'a ⇒ ('a ⇒_L 'a)
  assumes open U
    and derf: ⋀x. x ∈ U ⟹ (f has_derivative (blinfun_apply (f' x))) (at x)
    and contf:  continuous_on U f'
    and x0 ∈ U
    and invf: invf o_L f' x0 = id_blinfun
  obtains U' V g g' where open U' U' ⊆ U x0 ∈ U' open V f x0 ∈ V homeo-
morphism U' V f g
    ⋀y. y ∈ V ⟹ (g has_derivative (g' y)) (at y)
    ⋀y. y ∈ V ⟹ g' y = inv (blinfun_apply (f'(g y)))
    ⋀y. y ∈ V ⟹ bij (blinfun_apply (f'(g y)))
proof −
  have apply1 [simp]: ⋀i. blinfun_apply invf (blinfun_apply (f' x0) i) = i
    by (metis blinfun_apply_blinfun_compose blinfun_apply_id_blinfun invf)
  have apply2 [simp]: ⋀i. blinfun_apply (f' x0) (blinfun_apply invf i) = i
    by (metis apply1 bij_inv_eq_iff blinfun_bij1 invf)
  have [simp]: (range (blinfun_apply invf)) = UNIV
    using apply1 surjI by blast
  let ?f = invf ∘ (λx. (f ∘ (+)x0)x − f x0)
  let ?f' = λx. invf o_L (f' (x + x0))
  obtain U' V g g' where open U' and U': U' ⊆ (+)(−x0) ‘ U 0 ∈ U'
    and open V 0 ∈ V and hom: homeomorphism U' V ?f g
    and derg: ⋀y. y ∈ V ⟹ (g has_derivative (g' y)) (at y)
    and g': ⋀y. y ∈ V ⟹ g' y = inv (?f'(g y))
    and bij: ⋀y. y ∈ V ⟹ bij (?f'(g y))
  proof (rule inverse_function_theorem_scaled [of (+)(−x0) ‘ U ?f ?f'])
```

```
    show ope: open ((+) (− x0) ‘ U)
      using ‹open U› open_translation by blast
    show (?f has_derivative blinfun_apply (?f′ x)) (at x)
      if x ∈ (+) (− x0) ‘ U for x
      using that
      apply clarify
     apply (rule derf derivative_eq_intros | simp add: blinfun_compose.rep_eq)+
      done
    have YY: (λx. f′ (x + x0)) −u−x0→ f′ u
      if f′ −u→ f′ u u ∈ U for u
      using that LIM_offset [where k = x0] by (auto simp: algebra_simps)
    then have continuous_on ((+) (− x0) ‘ U) (λx. f′ (x + x0))
      using contf ‹open U› Lim_at_imp_Lim_at_within
      by (fastforce simp: continuous_on_def at_within_open_NO_MATCH ope)
    then show continuous_on ((+) (− x0) ‘ U) ?f′
      by (intro continuous_intros) simp
  qed (auto simp: invf ‹x0 ∈ U›)
  show thesis
  proof
    let ?U′ = (+)x0 ‘ U′
    let ?V = ((+)(f x0) ∘ f′ x0) ‘ V
    let ?g = (+)x0 ∘ g ∘ invf ∘ (+)(− f x0)
    let ?g′ = λy. inv (blinfun_apply (f′ (?g y)))
    show oU′: open ?U′
      by (simp add: ‹open U′› open_translation)
    show subU: ?U′ ⊆ U
      using ComplI ‹U′ ⊆ (+) (− x0) ‘ U› by auto
    show x0 ∈ ?U′
      by (simp add: ‹0 ∈ U′›)
    show open ?V
      using blinfun_bij2 [OF invf]
    by (metis ‹open V› bij_is_surj blinfun.bounded_linear_right bounded_linear_def
image_comp open_surjective_linear_image open_translation)
    show f x0 ∈ ?V
      using ‹0 ∈ V› image_iff by fastforce
    show homeomorphism ?U′ ?V f ?g
    proof
      show continuous_on ?U′ f
     by (meson subU continuous_on_eq_continuous_at derf has_derivative_continuous
oU′ subsetD)
      have ?f ‘ U′ ⊆ V
        using hom homeomorphism_image1 by blast
      then show f ‘ ?U′ ⊆ ?V
        unfolding image_subset_iff
       by (clarsimp simp: image_def) (metis apply2 add.commute diff_add_cancel)
      show ?g ‘ ?V ⊆ ?U′
        using hom invf by (auto simp: image_def homeomorphism_def)
      show ?g (f x) = x
        if x ∈ ?U′ for x
```

```
      using that hom homeomorphism_apply1 by fastforce
    have continuous_on V g
      using hom homeomorphism_def by blast
    then show continuous_on ?V ?g
      by (intro continuous_intros) (auto elim!: continuous_on_subset)
    have fg: ?f (g x) = x if x ∈ V for x
      using hom homeomorphism_apply2 that by blast
    show f (?g y) = y
      if y ∈ ?V for y
        using that fg by (simp add: image_iff) (metis apply2 add.commute
diff_add_cancel)
  qed
  show (?g has_derivative ?g' y) (at y) bij (blinfun_apply (f' (?g y)))
    if y ∈ ?V for y
  proof −
    have 1: bij (blinfun_apply invf)
      using blinfun_bij1 invf by blast
    then have 2: bij (blinfun_apply (f' (x0 + g x))) if x ∈ V for x
      by (metis add.commute bij bij_betw_comp_iff2 blinfun_compose.rep_eq
that top_greatest)
    then show bij (blinfun_apply (f' (?g y)))
      using that by auto
    have g' x ∘ blinfun_apply invf = inv (blinfun_apply (f' (x0 + g x)))
      if x ∈ V for x
      using that
      by (simp add: g' o_inv_distrib blinfun_compose.rep_eq 1 2 add.commute
bij_is_inj flip: o_assoc)
    then show (?g has_derivative ?g' y) (at y)
      using that invf
      by clarsimp (rule derg derivative_eq_intros | simp flip: id_def)+
  qed
 qed auto
qed
```

5.17.22 Piecewise differentiable functions

```
definition piecewise_differentiable_on
        (infixr piecewise'_differentiable'_on 50)
 where f piecewise_differentiable_on i ≡
        continuous_on i f ∧
        (∃ S. finite S ∧ (∀ x ∈ i − S. f differentiable (at x within i)))

lemma piecewise_differentiable_on_imp_continuous_on:
   f piecewise_differentiable_on S ⟹ continuous_on S f
by (simp add: piecewise_differentiable_on_def)

lemma piecewise_differentiable_on_subset:
   f piecewise_differentiable_on S ⟹ T ≤ S ⟹ f piecewise_differentiable_on
T
```

using *continuous_on_subset*
by (*smt* (*verit*) *Diff_iff differentiable_within_subset in_mono piecewise_differentiable_on_def*)

lemma *differentiable_on_imp_piecewise_differentiable*:
fixes *a*:: *'a*::{*linorder_topology*,*real_normed_vector*}
shows *f differentiable_on* {*a*..*b*} ⟹ *f piecewise_differentiable_on* {*a*..*b*}
using *differentiable_imp_continuous_on differentiable_onD piecewise_differentiable_on_def*
by *fastforce*

lemma *differentiable_imp_piecewise_differentiable*:
(⋀*x*. *x* ∈ *S* ⟹ *f differentiable* (*at x within S*))
⟹ *f piecewise_differentiable_on S*
by (*auto simp*: *piecewise_differentiable_on_def differentiable_imp_continuous_on differentiable_on_def*
intro: *differentiable_within_subset*)

lemma *piecewise_differentiable_const* [*iff*]: (λ*x*. *z*) *piecewise_differentiable_on S*
by (*simp add*: *differentiable_imp_piecewise_differentiable*)

lemma *piecewise_differentiable_compose*:
⟦*f piecewise_differentiable_on S*; *g piecewise_differentiable_on* (*f* ‘ *S*);
⋀*x*. *finite* (*S* ∩ *f*−‘{*x*})⟧
⟹ (*g* ∘ *f*) *piecewise_differentiable_on S*
apply (*simp add*: *piecewise_differentiable_on_def*, *safe*)
apply (*blast intro*: *continuous_on_compose2*)
apply (*rename_tac A B*)
apply (*rule_tac x*=*A* ∪ (⋃*x*∈*B*. *S* ∩ *f*−‘{*x*}) **in** *exI*)
apply (*blast intro*!: *differentiable_chain_within*)
done

lemma *piecewise_differentiable_affine*:
fixes *m*::*real*
assumes *f piecewise_differentiable_on* ((λ*x*. *m* ∗*R* *x* + *c*) ‘ *S*)
shows (*f* ∘ (λ*x*. *m* ∗*R* *x* + *c*)) *piecewise_differentiable_on S*
proof (*cases m* = *0*)
case *True*
then show *?thesis*
unfolding *o_def*
by (*force intro*: *differentiable_imp_piecewise_differentiable differentiable_const*)
next
case *False*
show *?thesis*
apply (*rule piecewise_differentiable_compose* [*OF differentiable_imp_piecewise_differentiable*])
apply (*rule assms derivative_intros* | *simp add*: *False vimage_def real_vector_affinity_eq*)+
done
qed

lemma *piecewise_differentiable_cases*:
fixes *c*::*real*

```
  assumes f piecewise_differentiable_on {a..c}
          g piecewise_differentiable_on {c..b}
          a ≤ c c ≤ b f c = g c
  shows (λx. if x ≤ c then f x else g x) piecewise_differentiable_on {a..b}
proof −
  obtain S T where st: finite S finite T
                and fd: ⋀x. x ∈ {a..c} − S ⟹ f differentiable at x within {a..c}
                and gd: ⋀x. x ∈ {c..b} − T ⟹ g differentiable at x within {c..b}
    using assms
    by (auto simp: piecewise_differentiable_on_def)
  have finabc: finite ({a,b,c} ∪ (S ∪ T))
    by (metis ‹finite S› ‹finite T› finite_Un finite_insert finite.emptyI)
  have continuous_on {a..c} f continuous_on {c..b} g
    using assms piecewise_differentiable_on_def by auto
  then have continuous_on {a..b} (λx. if x ≤ c then f x else g x)
    using continuous_on_cases [OF closed_real_atLeastAtMost [of a c],
                               OF closed_real_atLeastAtMost [of c b],
                               of f g λx. x≤c]  assms
    by (force simp: ivl_disj_un_two_touch)
  moreover
  { fix x
    assume x: x ∈ {a..b} − ({a,b,c} ∪ (S ∪ T))
   have (λx. if x ≤ c then f x else g x) differentiable at x within {a..b} (is ?diff_fg)
    proof (cases x c rule: le_cases)
      case le show ?diff_fg
      proof (rule differentiable_transform_within [where d = dist x c])
        have f differentiable at x
          using x le fd [of x] at_within_interior [of x {a..c}] by simp
        then show f differentiable at x within {a..b}
          by (simp add: differentiable_at_withinI)
      qed (use x le st dist_real_def in auto)
    next
      case ge show ?diff_fg
      proof (rule differentiable_transform_within [where d = dist x c])
        have g differentiable at x
          using x ge gd [of x] at_within_interior [of x {c..b}] by simp
        then show g differentiable at x within {a..b}
          by (simp add: differentiable_at_withinI)
      qed (use x ge st dist_real_def in auto)
    qed
  }
  then have ∃ S. finite S ∧
                (∀ x∈{a..b} − S. (λx. if x ≤ c then f x else g x) differentiable at x
within {a..b})
    by (meson finabc)
  ultimately show ?thesis
    by (simp add: piecewise_differentiable_on_def)
qed
```

lemma *piecewise_differentiable_neg*:
  f piecewise_differentiable_on S $\Longrightarrow$ $(\lambda x.\ -(f\ x))$ *piecewise_differentiable_on S*
 by (*auto simp*: *piecewise_differentiable_on_def continuous_on_minus*)

lemma *piecewise_differentiable_add*:
 assumes *f piecewise_differentiable_on i*
      g piecewise_differentiable_on i
   shows $(\lambda x.\ f\ x + g\ x)$ *piecewise_differentiable_on i*
proof −
 obtain *S T* **where** *st*: *finite S finite T*
                $\forall\, x{\in}i - S.$ *f differentiable at x within i*
                $\forall\, x{\in}i - T.$ *g differentiable at x within i*
   using *assms* **by** (*auto simp*: *piecewise_differentiable_on_def*)
 then have *finite* $(S \cup T) \wedge (\forall\, x{\in}i - (S \cup T).$ $(\lambda x.\ f\ x + g\ x)$ *differentiable at x within i*)
   by *auto*
 moreover have *continuous_on i f continuous_on i g*
   using *assms piecewise_differentiable_on_def* **by** *auto*
 ultimately show *?thesis*
   by (*auto simp*: *piecewise_differentiable_on_def continuous_on_add*)
qed

lemma *piecewise_differentiable_diff*:
  $[\![$*f piecewise_differentiable_on S*; *g piecewise_differentiable_on S*$]\!]$
   $\Longrightarrow$ $(\lambda x.\ f\ x - g\ x)$ *piecewise_differentiable_on S*
 unfolding *diff_conv_add_uminus*
 by (*metis piecewise_differentiable_add piecewise_differentiable_neg*)

5.17.23 The concept of continuously differentiable

John Harrison writes as follows:

"The usual assumption in complex analysis texts is that a path γ should be piecewise continuously differentiable, which ensures that the path integral exists at least for any continuous f, since all piecewise continuous functions are integrable. However, our notion of validity is weaker, just piecewise differentiability... [namely] continuity plus differentiability except on a finite set... [Our] underlying theory of integration is the Kurzweil-Henstock theory. In contrast to the Riemann or Lebesgue theory (but in common with a simple notion based on antiderivatives), this can integrate all derivatives."

"Formalizing basic complex analysis." From Insight to Proof: Festschrift in Honour of Andrzej Trybulec. Studies in Logic, Grammar and Rhetoric 10.23 (2007): 151-165.

And indeed he does not assume that his derivatives are continuous, but the penalty is unreasonably difficult proofs concerning winding numbers. We need a self-contained and straightforward theorem asserting that all derivatives can be integrated before we can adopt Harrison's choice.

definition *C1_differentiable_on* :: (*real* ⇒ *'a*::*real_normed_vector*) ⇒ *real set* ⇒ *bool*
(**infix** *C1'_differentiable'_on 50*)
where
f C1_differentiable_on S ⟷
(∃ *D*. (∀ *x* ∈ *S*. (*f has_vector_derivative* (*D x*)) (*at x*)) ∧ *continuous_on S D*)

lemma *C1_differentiable_on_eq*:
f C1_differentiable_on S ⟷
(∀ *x* ∈ *S*. *f differentiable at x*) ∧ *continuous_on S* (λ*x*. *vector_derivative f* (*at x*))
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
unfolding *C1_differentiable_on_def*
by (*metis* (*no_types*, *lifting*) *continuous_on_eq differentiableI_vector vector_derivative_at*)
next
assume *?rhs*
then show *?lhs*
using *C1_differentiable_on_def vector_derivative_works* **by** *fastforce*
qed

lemma *C1_differentiable_on_subset*:
f C1_differentiable_on T ⟹ *S* ⊆ *T* ⟹ *f C1_differentiable_on S*
unfolding *C1_differentiable_on_def continuous_on_eq_continuous_within*
by (*blast intro*: *continuous_within_subset*)

lemma *C1_differentiable_compose*:
assumes *fg*: *f C1_differentiable_on S g C1_differentiable_on* (*f* ‘ *S*) **and** *fin*: ⋀*x*. *finite* (*S* ∩ *f* −‘{*x*})
shows (*g* ∘ *f*) *C1_differentiable_on S*
proof −
have ⋀*x*. *x* ∈ *S* ⟹ *g* ∘ *f differentiable at x*
by (*meson C1_differentiable_on_eq assms differentiable_chain_at imageI*)
moreover have *continuous_on S* (λ*x*. *vector_derivative* (*g* ∘ *f*) (*at x*))
proof (*rule continuous_on_eq* [*of* _ λ*x*. *vector_derivative f* (*at x*) ∗*R vector_derivative g* (*at* (*f x*))])
show *continuous_on S* (λ*x*. *vector_derivative f* (*at x*) ∗*R vector_derivative g* (*at* (*f x*)))
using *fg*
apply (*clarsimp simp add*: *C1_differentiable_on_eq*)
apply (*rule Limits.continuous_on_scaleR*, *assumption*)
by (*metis* (*mono_tags*, *lifting*) *continuous_at_imp_continuous_on continuous_on_compose continuous_on_cong differentiable_imp_continuous_within o_def*)
show ⋀*x*. *x* ∈ *S* ⟹ *vector_derivative f* (*at x*) ∗*R vector_derivative g* (*at* (*f x*)) = *vector_derivative* (*g* ∘ *f*) (*at x*)

by (*metis* (*mono_tags*, *opaque_lifting*) *C1_differentiable_on_eq fg imageI vector_derivative_chain_at*)
qed
ultimately show *?thesis*
by (*simp add*: *C1_differentiable_on_eq*)
qed

lemma *C1_diff_imp_diff*: *f C1_differentiable_on S* ⟹ *f differentiable_on S*
by (*simp add*: *C1_differentiable_on_eq differentiable_at_imp_differentiable_on*)

lemma *C1_differentiable_on_ident* [*simp*, *derivative_intros*]: (λ*x*. *x*) *C1_differentiable_on S*
by (*auto simp*: *C1_differentiable_on_eq*)

lemma *C1_differentiable_on_const* [*simp*, *derivative_intros*]: (λ*z*. *a*) *C1_differentiable_on S*
by (*auto simp*: *C1_differentiable_on_eq*)

lemma *C1_differentiable_on_add* [*simp*, *derivative_intros*]:
f C1_differentiable_on S ⟹ *g C1_differentiable_on S* ⟹ (λ*x*. *f x* + *g x*) *C1_differentiable_on S*
unfolding *C1_differentiable_on_eq* **by** (*auto intro*: *continuous_intros*)

lemma *C1_differentiable_on_minus* [*simp*, *derivative_intros*]:
f C1_differentiable_on S ⟹ (λ*x*. − *f x*) *C1_differentiable_on S*
unfolding *C1_differentiable_on_eq* **by** (*auto intro*: *continuous_intros*)

lemma *C1_differentiable_on_diff* [*simp*, *derivative_intros*]:
f C1_differentiable_on S ⟹ *g C1_differentiable_on S* ⟹ (λ*x*. *f x* − *g x*) *C1_differentiable_on S*
unfolding *C1_differentiable_on_eq* **by** (*auto intro*: *continuous_intros*)

lemma *C1_differentiable_on_mult* [*simp*, *derivative_intros*]:
fixes *f g* :: *real* ⇒ *'a* :: *real_normed_algebra*
shows *f C1_differentiable_on S* ⟹ *g C1_differentiable_on S* ⟹ (λ*x*. *f x* ∗ *g x*) *C1_differentiable_on S*
unfolding *C1_differentiable_on_eq*
by (*auto simp*: *continuous_on_add continuous_on_mult continuous_at_imp_continuous_on differentiable_imp_continuous_within*)

lemma *C1_differentiable_on_scaleR* [*simp*, *derivative_intros*]:
f C1_differentiable_on S ⟹ *g C1_differentiable_on S* ⟹ (λ*x*. *f x* ∗*R* *g x*) *C1_differentiable_on S*
unfolding *C1_differentiable_on_eq*
by (*rule continuous_intros* | *simp add*: *continuous_at_imp_continuous_on differentiable_imp_continuous_within*)+

lemma *C1_differentiable_on_of_real* [*derivative_intros*]: *of_real C1_differentiable_on S*

unfolding *C1_differentiable_on_def*
using *vector_derivative_works* **by** *fastforce*

lemma *C1_differentiable_on_translation*:
f C1_differentiable_on U − S ⟹ (+) *d* ∘ *f C1_differentiable_on U − S*
by (*metis C1_differentiable_on_def has_vector_derivative_shift*)

lemma *C1_differentiable_on_translation_eq*:
fixes *d* :: *'a*::*real_normed_vector*
shows (+) *d* ∘ *f C1_differentiable_on i − S* ⟷ *f C1_differentiable_on i − S*
by (*force simp*: *o_def intro*: *C1_differentiable_on_translation dest*: *C1_differentiable_on_translation* [*of concl*: −*d*])

definition *piecewise_C1_differentiable_on*
(**infixr** *piecewise'_C1'_differentiable'_on 50*)
where *f piecewise_C1_differentiable_on i* ≡
continuous_on i f ∧
(∃ *S*. *finite S* ∧ (*f C1_differentiable_on* (*i − S*)))

lemma *C1_differentiable_imp_piecewise*:
f C1_differentiable_on S ⟹ *f piecewise_C1_differentiable_on S*
by (*auto simp*: *piecewise_C1_differentiable_on_def C1_differentiable_on_eq continuous_at_imp_continuous_on differentiable_imp_continuous_within*)

lemma *piecewise_C1_imp_differentiable*:
f piecewise_C1_differentiable_on i ⟹ *f piecewise_differentiable_on i*
by (*auto simp*: *piecewise_C1_differentiable_on_def piecewise_differentiable_on_def C1_differentiable_on_def differentiable_def has_vector_derivative_def intro*: *has_derivative_at_withinI*)

lemma *piecewise_C1_differentiable_on_translation_eq*:
((+) *d* ∘ *f piecewise_C1_differentiable_on i*) ⟷ (*f piecewise_C1_differentiable_on i*)
unfolding *piecewise_C1_differentiable_on_def continuous_on_translation_eq*
by (*metis C1_differentiable_on_translation_eq*)

lemma *piecewise_C1_differentiable_compose* [*derivative_intros*]:
assumes *fg*: *f piecewise_C1_differentiable_on S g piecewise_C1_differentiable_on* (*f* ‘ *S*) **and** *fin*: ⋀*x*. *finite* (*S* ∩ *f*−‘{*x*})
shows (*g* ∘ *f*) *piecewise_C1_differentiable_on S*
proof −
have *continuous_on S* (λ*x*. *g* (*f x*))
by (*metis continuous_on_compose2 fg order_refl piecewise_C1_differentiable_on_def*)
moreover have ∃ *T*. *finite T* ∧ *g* ∘ *f C1_differentiable_on S − T*
proof −
obtain *F* **where** *finite F* **and** *F*: *f C1_differentiable_on S − F* **and** *f*: *f piecewise_C1_differentiable_on S*
using *fg* **by** (*auto simp*: *piecewise_C1_differentiable_on_def*)

```
    obtain G where finite G and G: g C1_differentiable_on f ‘ S − G and g: g
piecewise_C1_differentiable_on f ‘ S
      using fg by (auto simp: piecewise_C1_differentiable_on_def)
    show ?thesis
    proof (intro exI conjI)
      show finite (F ∪ (⋃x∈G. S ∩ f−‘{x}))
        using fin by (auto simp only: Int_Union ‹finite F› ‹finite G› finite_UN
finite_imageI)
      show g ∘ f C1_differentiable_on S − (F ∪ (⋃x∈G. S ∩ f −‘ {x}))
        apply (rule C1_differentiable_compose)
          apply (blast intro: C1_differentiable_on_subset [OF F])
          apply (blast intro: C1_differentiable_on_subset [OF G])
        by (simp add:  C1_differentiable_on_subset G Diff_Int_distrib2 fin)
    qed
  qed
  ultimately show ?thesis
    by (simp add: piecewise_C1_differentiable_on_def)
qed

lemma piecewise_C1_differentiable_on_subset:
   f piecewise_C1_differentiable_on S ⟹ T ≤ S ⟹ f piecewise_C1_differentiable_on
T
  by (auto simp: piecewise_C1_differentiable_on_def elim!: continuous_on_subset
C1_differentiable_on_subset)

lemma C1_differentiable_imp_continuous_on:
  f C1_differentiable_on S ⟹ continuous_on S f
  unfolding C1_differentiable_on_eq continuous_on_eq_continuous_within
  using differentiable_at_withinI differentiable_imp_continuous_within by blast

lemma C1_differentiable_on_empty [iff,derivative_intros]: f C1_differentiable_on
{}
  unfolding C1_differentiable_on_def
  by auto

lemma piecewise_C1_differentiable_affine:
  fixes m::real
  assumes f piecewise_C1_differentiable_on ((λx. m * x + c) ‘ S)
  shows (f ∘ (λx. m *R x + c)) piecewise_C1_differentiable_on S
proof (cases m = 0)
  case True
  then show ?thesis
    unfolding o_def by (auto simp: piecewise_C1_differentiable_on_def)
next
  case False
  have *: ⋀x. finite (S ∩ {y. m * y + c = x})
    using False not_finite_existsD by fastforce
  show ?thesis
   apply (rule piecewise_C1_differentiable_compose [OF C1_differentiable_imp_piecewise])
```

```
    apply (rule * assms derivative_intros | simp add: False vimage_def)+
    done
qed

lemma piecewise_C1_differentiable_cases [derivative_intros]:
  fixes c::real
  assumes f piecewise_C1_differentiable_on {a..c}
          g piecewise_C1_differentiable_on {c..b}
          a ≤ c c ≤ b f c = g c
  shows (λx. if x ≤ c then f x else g x) piecewise_C1_differentiable_on {a..b}
proof -
  obtain S T where st: f C1_differentiable_on ({a..c} - S)
                      g C1_differentiable_on ({c..b} - T)
                      finite S finite T
    using assms
    by (force simp: piecewise_C1_differentiable_on_def)
  then have f_diff: f differentiable_on {a..<c} - S
       and g_diff: g differentiable_on {c<..b} - T
    by (simp_all add: C1_differentiable_on_eq differentiable_at_withinI differentiable_on_def)
  have continuous_on {a..c} f continuous_on {c..b} g
    using assms piecewise_C1_differentiable_on_def by auto
  then have cab: continuous_on {a..b} (λx. if x ≤ c then f x else g x)
    using continuous_on_cases [OF closed_real_atLeastAtMost [of a c],
                               OF closed_real_atLeastAtMost [of c b],
                               of f g λx. x≤c]  assms
    by (force simp: ivl_disj_un_two_touch)
  { fix x
    assume x: x ∈ {a..b} - insert c (S ∪ T)
    have (λx. if x ≤ c then f x else g x) differentiable at x (is ?diff_fg)
    proof (cases x c rule: le_cases)
      case le show ?diff_fg
       apply (rule differentiable_transform_within [where f=f and d = dist x c])
        using x dist_real_def le st by (auto simp: C1_differentiable_on_eq)
    next
      case ge show ?diff_fg
        apply (rule differentiable_transform_within [where f=g and d = dist x c])
     using dist_nz x dist_real_def ge st x by (auto simp: C1_differentiable_on_eq)
    qed
  }
  then have (∀x ∈ {a..b} - insert c (S ∪ T). (λx. if x ≤ c then f x else g x) differentiable at x)
    by auto
  moreover
  { assume fcon: continuous_on ({a<..<c} - S) (λx. vector_derivative f (at x))
       and gcon: continuous_on ({c<..<b} - T) (λx. vector_derivative g (at x))
    have open ({a<..<c} - S)  open ({c<..<b} - T)
      using st by (simp_all add: open_Diff finite_imp_closed)
```

moreover have *continuous_on* ({*a*<..<*c*} − *S*) (λ*x*. *vector_derivative* (λ*x*. *if* *x* ≤ *c* *then* *f* *x* *else* *g* *x*) (*at* *x*))
proof −
have ((λ*x*. *if* *x* ≤ *c* *then* *f* *x* *else* *g* *x*) *has_vector_derivative* *vector_derivative* *f* (*at* *x*)) (*at* *x*)
if *a* < *x* *x* < *c* *x* ∉ *S* **for** *x*
proof −
have *f*: *f differentiable at x*
by (*meson C1_differentiable_on_eq Diff_iff atLeastAtMost_iff less_eq_real_def* *st*(*1*) *that*)
show *?thesis*
using *that*
apply (*rule_tac f*=*f* **and** *d*=*dist x c* **in** *has_vector_derivative_transform_within*)
apply (*auto simp*: *dist_norm vector_derivative_works* [*symmetric*] *f*)
done
qed
then show *?thesis*
by (*metis* (*no_types*, *lifting*) *continuous_on_eq* [*OF fcon*] *DiffE greaterThanLessThan_iff* *vector_derivative_at*)
qed
moreover have *continuous_on* ({*c*<..<*b*} − *T*) (λ*x*. *vector_derivative* (λ*x*. *if* *x* ≤ *c* *then* *f* *x* *else* *g* *x*) (*at* *x*))
proof −
have ((λ*x*. *if* *x* ≤ *c* *then* *f* *x* *else* *g* *x*) *has_vector_derivative* *vector_derivative* *g* (*at* *x*)) (*at* *x*)
if *c* < *x* *x* < *b* *x* ∉ *T* **for** *x*
proof −
have *g*: *g differentiable at x*
by (*metis C1_differentiable_on_eq DiffD1 DiffI atLeastAtMost_diff_ends* *greaterThanLessThan_iff st*(*2*) *that*)
show *?thesis*
using *that*
apply (*rule_tac f*=*g* **and** *d*=*dist x c* **in** *has_vector_derivative_transform_within*)
apply (*auto simp*: *dist_norm vector_derivative_works* [*symmetric*] *g*)
done
qed
then show *?thesis*
by (*metis* (*no_types*, *lifting*) *continuous_on_eq* [*OF gcon*] *DiffE greaterThanLessThan_iff* *vector_derivative_at*)
qed
ultimately have *continuous_on* ({*a*<..<*b*} − *insert c* (*S* ∪ *T*)) (λ*x*. *vector_derivative* (λ*x*. *if* *x* ≤ *c* *then* *f* *x* *else* *g* *x*) (*at* *x*))
by (*rule continuous_on_subset* [*OF continuous_on_open_Un*], *auto*)
} **note** ∗ = *this*
have *continuous_on* ({*a*<..<*b*} − *insert c* (*S* ∪ *T*)) (λ*x*. *vector_derivative* (λ*x*. *if* *x* ≤ *c* *then* *f* *x* *else* *g* *x*) (*at* *x*))
using *st*
by (*auto simp*: *C1_differentiable_on_eq elim*!: *continuous_on_subset intro*: ∗)
ultimately have ∃ *S*. *finite S* ∧ ((λ*x*. *if* *x* ≤ *c* *then* *f* *x* *else* *g* *x*) *C1_differentiable_on*

$\{a..b\} - S$)
apply (*rule_tac* x=$\{a,b,c\} \cup S \cup T$ **in** *exI*)
using *st* **by** (*auto simp*: *C1_differentiable_on_eq elim*!: *continuous_on_subset*)
with *cab* **show** *?thesis*
by (*simp add*: *piecewise_C1_differentiable_on_def*)
qed

lemma *piecewise_C1_differentiable_const* [*derivative_intros*]:
$(\lambda x.\ c)$ *piecewise_C1_differentiable_on S*
by (*simp add*: *C1_differentiable_imp_piecewise*)

lemma *piecewise_C1_differentiable_scaleR* [*derivative_intros*]:
⟦*f piecewise_C1_differentiable_on S*⟧
$\Longrightarrow$ $(\lambda x.\ c *_R f\ x)$ *piecewise_C1_differentiable_on S*
by (*force simp add*: *piecewise_C1_differentiable_on_def continuous_on_scaleR*)

lemma *piecewise_C1_differentiable_neg* [*derivative_intros*]:
f piecewise_C1_differentiable_on S $\Longrightarrow$ $(\lambda x.\ -(f\ x))$ *piecewise_C1_differentiable_on S*
unfolding *piecewise_C1_differentiable_on_def*
by (*auto intro*!: *continuous_on_minus C1_differentiable_on_minus*)

lemma *piecewise_C1_differentiable_add* [*derivative_intros*]:
assumes *f piecewise_C1_differentiable_on i*
g piecewise_C1_differentiable_on i
shows $(\lambda x.\ f\ x + g\ x)$ *piecewise_C1_differentiable_on i*
proof −
obtain *S t* **where** *st*: *finite S finite t*
f C1_differentiable_on $(i-S)$
g C1_differentiable_on $(i-t)$
using *assms* **by** (*auto simp*: *piecewise_C1_differentiable_on_def*)
then have *finite* $(S \cup t) \wedge (\lambda x.\ f\ x + g\ x)$ *C1_differentiable_on i* $-$ $(S \cup t)$
by (*auto intro*: *C1_differentiable_on_add elim*!: *C1_differentiable_on_subset*)
moreover have *continuous_on i f continuous_on i g*
using *assms piecewise_C1_differentiable_on_def* **by** *auto*
ultimately show *?thesis*
by (*auto simp*: *piecewise_C1_differentiable_on_def continuous_on_add*)
qed

lemma *piecewise_C1_differentiable_diff* [*derivative_intros*]:
⟦*f piecewise_C1_differentiable_on S*; *g piecewise_C1_differentiable_on S*⟧
$\Longrightarrow$ $(\lambda x.\ f\ x - g\ x)$ *piecewise_C1_differentiable_on S*
unfolding *diff_conv_add_uminus*
by (*metis piecewise_C1_differentiable_add piecewise_C1_differentiable_neg*)

lemma *piecewise_C1_differentiable_cmult_right* [*derivative_intros*]:
fixes *c*::*complex*
shows *f piecewise_C1_differentiable_on S*
$\Longrightarrow$ $(\lambda x.\ f\ x * c)$ *piecewise_C1_differentiable_on S*

by (*force simp*: *piecewise_C1_differentiable_on_def continuous_on_mult_right*)

lemma *piecewise_C1_differentiable_cmult_left* [*derivative_intros*]:
fixes *c*::*complex*
shows *f piecewise_C1_differentiable_on S*
⟹ (λ*x*. *c* ∗ *f x*) *piecewise_C1_differentiable_on S*
using *piecewise_C1_differentiable_cmult_right* [*of f S c*] **by** (*simp add*: *mult.commute*)

lemma *piecewise_C1_differentiable_on_of_real* [*derivative_intros*]:
of_real piecewise_C1_differentiable_on S
by (*simp add*: *C1_differentiable_imp_piecewise C1_differentiable_on_of_real*)

end

5.18 Finite Cartesian Products of Euclidean Spaces

theory *Cartesian_Euclidean_Space*
imports *Derivative*
begin

lemma *subspace_special_hyperplane*: *subspace* {*x*. *x* $ *k* = *0*}
by (*simp add*: *subspace_def*)

lemma *sum_mult_product*:
sum h {..<*A* ∗ *B* :: *nat*} = (∑ *i*∈{..<*A*}. ∑ *j*∈{..<*B*}. *h* (*j* + *i* ∗ *B*))
unfolding *sum.nat_group*[*of h B A*, *unfolded atLeast0LessThan*, *symmetric*]
proof (*rule sum.cong*, *simp*, *rule sum.reindex_cong*)
fix *i*
show *inj_on* (λ*j*. *j* + *i* ∗ *B*) {..<*B*} **by** (*auto intro*!: *inj_onI*)
show {*i* ∗ *B*..<*i* ∗ *B* + *B*} = (λ*j*. *j* + *i* ∗ *B*) ‘ {..<*B*}
proof *safe*
fix *j* **assume** *j* ∈ {*i* ∗ *B*..<*i* ∗ *B* + *B*}
then show *j* ∈ (λ*j*. *j* + *i* ∗ *B*) ‘ {..<*B*}
by (*auto intro*!: *image_eqI*[*of _ _ j* − *i* ∗ *B*])
qed *simp*
qed *simp*

lemma *interval_cbox_cart*: {*a*::*real*^'*n*..*b*} = *cbox a b*
by (*auto simp add*: *less_eq_vec_def mem_box Basis_vec_def inner_axis*)

lemma *differentiable_vec*:
fixes *S* :: '*a*::*euclidean_space set*
shows *vec differentiable_on S*
by (*simp add*: *linear_linear bounded_linear_imp_differentiable_on*)

lemma *continuous_vec* [*continuous_intros*]:
fixes *x* :: '*a*::*euclidean_space*
shows *isCont vec x*
apply (*clarsimp simp add*: *continuous_def LIM_def dist_vec_def L2_set_def*)

apply (*rule_tac x=r / sqrt (real CARD('b)) in exI*)
by (*simp add: mult.commute pos_less_divide_eq real_sqrt_mult*)

lemma *box_vec_eq_empty* [*simp*]:
shows *cbox (vec a) (vec b) = {} ⟷ cbox a b = {}*
box (vec a) (vec b) = {} ⟷ box a b = {}
by (*auto simp: Basis_vec_def mem_box box_eq_empty inner_axis*)

5.18.1 Closures and interiors of halfspaces

lemma *interior_halfspace_component_le* [*simp*]:
interior {x. x$k ≤ a} = {x :: (real^'n). x$k < a} (**is** *?LE*)
and *interior_halfspace_component_ge* [*simp*]:
interior {x. x$k ≥ a} = {x :: (real^'n). x$k > a} (**is** *?GE*)
proof −
have *axis k (1::real) ≠ 0*
by (*simp add: axis_def vec_eq_iff*)
moreover have *axis k (1::real) · x = x$k* **for** *x*
by (*simp add: cart_eq_inner_axis inner_commute*)
ultimately show *?LE ?GE*
using *interior_halfspace_le* [*of axis k (1::real) a*]
interior_halfspace_ge [*of axis k (1::real) a*] **by** *auto*
qed

lemma *closure_halfspace_component_lt* [*simp*]:
closure {x. x$k < a} = {x :: (real^'n). x$k ≤ a} (**is** *?LE*)
and *closure_halfspace_component_gt* [*simp*]:
closure {x. x$k > a} = {x :: (real^'n). x$k ≥ a} (**is** *?GE*)
proof −
have *axis k (1::real) ≠ 0*
by (*simp add: axis_def vec_eq_iff*)
moreover have *axis k (1::real) · x = x$k* **for** *x*
by (*simp add: cart_eq_inner_axis inner_commute*)
ultimately show *?LE ?GE*
using *closure_halfspace_lt* [*of axis k (1::real) a*]
closure_halfspace_gt [*of axis k (1::real) a*] **by** *auto*
qed

lemma *interior_standard_hyperplane*:
interior {x :: (real^'n). x$k = a} = {}
proof −
have *axis k (1::real) ≠ 0*
by (*simp add: axis_def vec_eq_iff*)
moreover have *axis k (1::real) · x = x$k* **for** *x*
by (*simp add: cart_eq_inner_axis inner_commute*)
ultimately show *?thesis*
using *interior_hyperplane* [*of axis k (1::real) a*]
by *force*
qed

lemma *matrix_vector_mul_bounded_linear*[*intro*, *simp*]: *bounded_linear* ((∗*v*) *A*)
for *A* :: *'a*::{*euclidean_space*,*real_algebra_1*}^*'n*^*m*
using *matrix_vector_mul_linear*[*of A*]
by (*simp add*: *linear_conv_bounded_linear linear_matrix_vector_mul_eq*)

lemma
fixes *A* :: *'a*::{*euclidean_space*,*real_algebra_1*}^*'n*^*m*
shows *matrix_vector_mult_linear_continuous_at* [*continuous_intros*]: *isCont* ((∗*v*) *A*) *z*
and *matrix_vector_mult_linear_continuous_on* [*continuous_intros*]: *continuous_on S* ((∗*v*) *A*)
by (*simp_all add*: *linear_continuous_at linear_continuous_on*)

5.18.2 Bounds on components etc. relative to operator norm

lemma *norm_column_le_onorm*:
fixes *A* :: *real*^*'n*^*m*
shows *norm*(*column i A*) ≤ *onorm*((∗*v*) *A*)
proof −
have *norm* (*χ j*. *A* $ *j* $ *i*) ≤ *norm* (*A* ∗*v axis i 1*)
by (*simp add*: *matrix_mult_dot cart_eq_inner_axis*)
also have . . . ≤ *onorm* ((∗*v*) *A*)
using *onorm* [*OF matrix_vector_mul_bounded_linear*, *of A axis i 1*] **by** *auto*
finally have *norm* (*χ j*. *A* $ *j* $ *i*) ≤ *onorm* ((∗*v*) *A*) **.**
then show *?thesis*
unfolding *column_def* **.**
qed

lemma *matrix_component_le_onorm*:
fixes *A* :: *real*^*'n*^*m*
shows |*A* $ *i* $ *j*| ≤ *onorm*((∗*v*) *A*)
proof −
have |*A* $ *i* $ *j*| ≤ *norm* (*χ n*. (*A* $ *n* $ *j*))
by (*metis* (*full_types*, *lifting*) *component_le_norm_cart vec_lambda_beta*)
also have . . . ≤ *onorm* ((∗*v*) *A*)
by (*metis* (*no_types*) *column_def norm_column_le_onorm*)
finally show *?thesis* **.**
qed

lemma *component_le_onorm*:
fixes *f* :: *real*^*'m* ⇒ *real*^*'n*
shows *linear f* ⟹ |*matrix f* $ *i* $ *j*| ≤ *onorm f*
by (*metis matrix_component_le_onorm matrix_vector_mul*(*2*))

lemma *onorm_le_matrix_component_sum*:
fixes *A* :: *real*^*'n*^*m*
shows *onorm*((∗*v*) *A*) ≤ (∑ *i*∈*UNIV*. ∑ *j*∈*UNIV*. |*A* $ *i* $ *j*|)
proof (*rule onorm_le*)

fix x
have $norm\ (A\ *v\ x) \leq (\sum i{\in}UNIV.\ |(A\ *v\ x)\ \$\ i|)$
by (*rule norm_le_l1_cart*)
also have $\ldots \leq (\sum i{\in}UNIV.\ \sum j{\in}UNIV.\ |A\ \$\ i\ \$\ j| * norm\ x)$
proof (*rule sum_mono*)
fix i
have $|(A\ *v\ x)\ \$\ i| \leq |\sum j{\in}UNIV.\ A\ \$\ i\ \$\ j * x\ \$\ j|$
by (*simp add*: *matrix_vector_mult_def*)
also have $\ldots \leq (\sum j{\in}UNIV.\ |A\ \$\ i\ \$\ j * x\ \$\ j|)$
by (*rule sum_abs*)
also have $\ldots \leq (\sum j{\in}UNIV.\ |A\ \$\ i\ \$\ j| * norm\ x)$
by (*rule sum_mono*) (*simp add*: *abs_mult component_le_norm_cart mult_left_mono*)
finally show $|(A\ *v\ x)\ \$\ i| \leq (\sum j{\in}UNIV.\ |A\ \$\ i\ \$\ j| * norm\ x)$ **.**
qed
finally show $norm\ (A\ *v\ x) \leq (\sum i{\in}UNIV.\ \sum j{\in}UNIV.\ |A\ \$\ i\ \$\ j|) * norm\ x$
by (*simp add*: *sum_distrib_right*)
qed

lemma *onorm_le_matrix_component*:
fixes $A :: real^{\sim}{}'n^{\sim}{}'m$
assumes $\bigwedge i\ j.\ abs(A\$i\$j) \leq B$
shows $onorm((*v)\ A) \leq real\ (CARD('m)) * real\ (CARD('n)) * B$
proof (*rule onorm_le*)
fix $x :: real^{\sim}{}'n::\_$
have $norm\ (A\ *v\ x) \leq (\sum i{\in}UNIV.\ |(A\ *v\ x)\ \$\ i|)$
by (*rule norm_le_l1_cart*)
also have $\ldots \leq (\sum i::'m\in UNIV.\ real\ (CARD('n)) * B * norm\ x)$
proof (*rule sum_mono*)
fix i
have $|(A\ *v\ x)\ \$\ i| \leq norm(A\ \$\ i) * norm\ x$
by (*simp add*: *matrix_mult_dot Cauchy_Schwarz_ineq2*)
also have $\ldots \leq (\sum j{\in}UNIV.\ |A\ \$\ i\ \$\ j|) * norm\ x$
by (*simp add*: *mult_right_mono norm_le_l1_cart*)
also have $\ldots \leq real\ (CARD('n)) * B * norm\ x$
by (*simp add*: *assms sum_bounded_above mult_right_mono*)
finally show $|(A\ *v\ x)\ \$\ i| \leq real\ (CARD('n)) * B * norm\ x$ **.**
qed
also have $\ldots \leq CARD('m) * real\ (CARD('n)) * B * norm\ x$
by *simp*
finally show $norm\ (A\ *v\ x) \leq CARD('m) * real\ (CARD('n)) * B * norm\ x$ **.**
qed

lemma *vector_sub_project_orthogonal_cart*: $(b::real^{\sim}{}'n) \cdot (x - ((b \cdot x)\ /\ (b \cdot b)) *s\ b) = 0$
unfolding *inner_simps scalar_mult_eq_scaleR* **by** *auto*

lemma *infnorm_cart*:$infnorm\ (x::real^{\sim}{}'n) = Sup\ \{|x\$i|\ |i.\ i{\in}UNIV\}$
by (*simp add*: *infnorm_def inner_axis Basis_vec_def*) (*metis* (*lifting*) *inner_axis real_inner_1_right*)

lemma *component_le_infnorm_cart*: $|x\$i| \leq$ *infnorm* $(x::real\hat{}'n)$
using *Basis_le_infnorm*[*of axis i 1 x*]
by (*simp add*: *Basis_vec_def axis_eq_axis inner_axis*)

lemma *continuous_component*[*continuous_intros*]: *continuous F f* $\Longrightarrow$ *continuous F* $(\lambda x.\ f\ x\ \$\ i)$
unfolding *continuous_def* **by** (*rule tendsto_vec_nth*)

lemma *continuous_on_component*[*continuous_intros*]: *continuous_on s f* $\Longrightarrow$ *continuous_on s* $(\lambda x.\ f\ x\ \$\ i)$
unfolding *continuous_on_def* **by** (*fast intro*: *tendsto_vec_nth*)

lemma *continuous_on_vec_lambda*[*continuous_intros*]:
$(\bigwedge i.$ *continuous_on S* $(f\ i)) \Longrightarrow$ *continuous_on S* $(\lambda x.\ \chi\ i.\ f\ i\ x)$
unfolding *continuous_on_def* **by** (*auto intro*: *tendsto_vec_lambda*)

lemma *closed_positive_orthant*: *closed* $\{x::real\hat{}'n.\ \forall i.\ 0 \leq x\$i\}$
by (*simp add*: *Collect_all_eq closed_INT closed_Collect_le continuous_on_component*)

lemma *bounded_component_cart*: *bounded s* $\Longrightarrow$ *bounded* $((\lambda x.\ x\ \$\ i)\ `\ s)$
unfolding *bounded_def*
apply *clarify*
apply (*rule_tac x*=*x* $\$$ *i* **in** *exI*)
apply (*rule_tac x*=*e* **in** *exI*)
apply *clarify*
apply (*rule order_trans* [*OF dist_vec_nth_le*], *simp*)
done

lemma *compact_lemma_cart*:
fixes *f* :: *nat* $\Rightarrow$ *'a*::*heine_borel* $\hat{}$ *'n*
assumes *f*: *bounded* (*range f*)
shows $\exists\, l\ r.$ *strict_mono r* $\wedge$
$(\forall\, e{>}0.$ *eventually* $(\lambda n.\ \forall i{\in}d.$ *dist* $(f\ (r\ n)\ \$\ i)\ (l\ \$\ i) < e)$ *sequentially*)
(**is** *?th d*)
proof −
have $\forall\, d' \subseteq d.$ *?th d'*
by (*rule compact_lemma_general*[**where** *unproj*=*vec_lambda*])
(*auto intro*!: *f bounded_component_cart*)
then show *?th d* **by** *simp*
qed

instance *vec* :: (*heine_borel*, *finite*) *heine_borel*
proof
fix *f* :: *nat* $\Rightarrow$ *'a* $\hat{}$ *'b*
assume *f*: *bounded* (*range f*)
then obtain *l r* **where** *r*: *strict_mono r*
and *l*: $\forall\, e{>}0.$ *eventually* $(\lambda n.\ \forall i{\in}UNIV.$ *dist* $(f\ (r\ n)\ \$\ i)\ (l\ \$\ i) < e)$ *sequentially*

```
    using compact_lemma_cart [OF f] by blast
  let ?d = UNIV::'b set
  { fix e::real assume e>0
    hence 0 < e / (real_of_nat (card ?d))
      using zero_less_card_finite divide_pos_pos[of e, of real_of_nat (card ?d)]
by auto
    with l have eventually (λn. ∀ i. dist (f (r n) $ i) (l $ i) < e / (real_of_nat
(card ?d))) sequentially
      by simp
    moreover
    { fix n
      assume n: ∀ i. dist (f (r n) $ i) (l $ i) < e / (real_of_nat (card ?d))
      have dist (f (r n)) l ≤ (∑ i∈?d. dist (f (r n) $ i) (l $ i))
        unfolding dist_vec_def using zero_le_dist by (rule L2_set_le_sum)
      also have ... < (∑ i∈?d. e / (real_of_nat (card ?d)))
        by (rule sum_strict_mono) (simp_all add: n)
      finally have dist (f (r n)) l < e by simp
    }
    ultimately have eventually (λn. dist (f (r n)) l < e) sequentially
      by (rule eventually_mono)
  }
  hence ((f ∘ r) ⟶ l) sequentially unfolding o_def tendsto_iff by simp
  with r show ∃ l r. strict_mono r ∧ ((f ∘ r) ⟶ l) sequentially by auto
qed

lemma interval_cart:
  fixes a :: real^'n
  shows box a b = {x::real^'n. ∀ i. a$i < x$i ∧ x$i < b$i}
    and cbox a b = {x::real^'n. ∀ i. a$i ≤ x$i ∧ x$i ≤ b$i}
   by (auto simp add: set_eq_iff less_vec_def less_eq_vec_def mem_box Ba-
sis_vec_def inner_axis)

lemma mem_box_cart:
  fixes a :: real^'n
  shows x ∈ box a b ⟷ (∀ i. a$i < x$i ∧ x$i < b$i)
    and x ∈ cbox a b ⟷ (∀ i. a$i ≤ x$i ∧ x$i ≤ b$i)
 using interval_cart[of a b] by (auto simp add: set_eq_iff less_vec_def less_eq_vec_def)

lemma interval_eq_empty_cart:
  fixes a :: real^'n
  shows (box a b = {} ⟷ (∃ i. b$i ≤ a$i)) (is ?th1)
    and (cbox a b = {} ⟷ (∃ i. b$i < a$i)) (is ?th2)
proof −
  { fix i x assume as:b$i ≤ a$i and x:x∈box a b
    hence a $ i < x $ i ∧ x $ i < b $ i unfolding mem_box_cart by auto
    hence a$i < b$i by auto
    hence False using as by auto }
  moreover
  { assume as:∀ i. ¬ (b$i ≤ a$i)
```

```
    let ?x = (1/2) *R (a + b)
    { fix i
      have a$i < b$i using as[THEN spec[where x=i]] by auto
      hence a$i < ((1/2) *R (a+b)) $ i ((1/2) *R (a+b)) $ i < b$i
        unfolding vector_smult_component and vector_add_component
        by auto }
    hence box a b ≠ {} using mem_box_cart(1)[of ?x a b] by auto }
  ultimately show ?th1 by blast

  { fix i x assume as:b$i < a$i and x:x∈cbox a b
    hence a $ i ≤ x $ i ∧ x $ i ≤ b $ i unfolding mem_box_cart by auto
    hence a$i ≤ b$i by auto
    hence False using as by auto }
  moreover
  { assume as:∀ i. ¬ (b$i < a$i)
    let ?x = (1/2) *R (a + b)
    { fix i
      have a$i ≤ b$i using as[THEN spec[where x=i]] by auto
      hence a$i ≤ ((1/2) *R (a+b)) $ i ((1/2) *R (a+b)) $ i ≤ b$i
        unfolding vector_smult_component and vector_add_component
        by auto }
    hence cbox a b ≠ {} using mem_box_cart(2)[of ?x a b] by auto  }
  ultimately show ?th2 by blast
qed

lemma interval_ne_empty_cart:
  fixes a :: real^'n
  shows cbox a b ≠ {} ⟷ (∀ i. a$i ≤ b$i)
    and box a b ≠ {} ⟷ (∀ i. a$i < b$i)
  unfolding interval_eq_empty_cart[of a b] by (auto simp add: not_less not_le)


lemma subset_interval_imp_cart:
  fixes a :: real^'n
  shows (∀ i. a$i ≤ c$i ∧ d$i ≤ b$i) ⟹ cbox c d ⊆ cbox a b
    and (∀ i. a$i < c$i ∧ d$i < b$i) ⟹ cbox c d ⊆ box a b
    and (∀ i. a$i ≤ c$i ∧ d$i ≤ b$i) ⟹ box c d ⊆ cbox a b
    and (∀ i. a$i ≤ c$i ∧ d$i ≤ b$i) ⟹ box c d ⊆ box a b
  unfolding subset_eq[unfolded Ball_def] unfolding mem_box_cart
  by (auto intro: order_trans less_le_trans le_less_trans less_imp_le)

lemma interval_sing:
  fixes a :: 'a::linorder^'n
  shows {a .. a} = {a} ∧ {a<..<a} = {}
  apply (auto simp add: set_eq_iff less_vec_def less_eq_vec_def vec_eq_iff)
  done

lemma subset_interval_cart:
  fixes a :: real^'n
```

shows *cbox c d* ⊆ *cbox a b* ⟷ (∀ *i*. *c*$*i* ≤ *d*$*i*) −−> (∀ *i*. *a*$*i* ≤ *c*$*i* ∧ *d*$*i* ≤ *b*$*i*) (**is** *?th1*)
and *cbox c d* ⊆ *box a b* ⟷ (∀ *i*. *c*$*i* ≤ *d*$*i*) −−> (∀ *i*. *a*$*i* < *c*$*i* ∧ *d*$*i* < *b*$*i*) (**is** *?th2*)
and *box c d* ⊆ *cbox a b* ⟷ (∀ *i*. *c*$*i* < *d*$*i*) −−> (∀ *i*. *a*$*i* ≤ *c*$*i* ∧ *d*$*i* ≤ *b*$*i*) (**is** *?th3*)
and *box c d* ⊆ *box a b* ⟷ (∀ *i*. *c*$*i* < *d*$*i*) −−> (∀ *i*. *a*$*i* ≤ *c*$*i* ∧ *d*$*i* ≤ *b*$*i*) (**is** *?th4*)
using *subset_box*[*of c d a b*] **by** (*simp_all add*: *Basis_vec_def inner_axis*)

lemma *disjoint_interval_cart*:
fixes *a*::*real*^'*n*
shows *cbox a b* ∩ *cbox c d* = {} ⟷ (∃ *i*. (*b*$*i* < *a*$*i* ∨ *d*$*i* < *c*$*i* ∨ *b*$*i* < *c*$*i* ∨ *d*$*i* < *a*$*i*)) (**is** *?th1*)
and *cbox a b* ∩ *box c d* = {} ⟷ (∃ *i*. (*b*$*i* < *a*$*i* ∨ *d*$*i* ≤ *c*$*i* ∨ *b*$*i* ≤ *c*$*i* ∨ *d*$*i* ≤ *a*$*i*)) (**is** *?th2*)
and *box a b* ∩ *cbox c d* = {} ⟷ (∃ *i*. (*b*$*i* ≤ *a*$*i* ∨ *d*$*i* < *c*$*i* ∨ *b*$*i* ≤ *c*$*i* ∨ *d*$*i* ≤ *a*$*i*)) (**is** *?th3*)
and *box a b* ∩ *box c d* = {} ⟷ (∃ *i*. (*b*$*i* ≤ *a*$*i* ∨ *d*$*i* ≤ *c*$*i* ∨ *b*$*i* ≤ *c*$*i* ∨ *d*$*i* ≤ *a*$*i*)) (**is** *?th4*)
using *disjoint_interval*[*of a b c d*] **by** (*simp_all add*: *Basis_vec_def inner_axis*)

lemma *Int_interval_cart*:
fixes *a* :: *real*^'*n*
shows *cbox a b* ∩ *cbox c d* = {(χ *i*. *max* (*a*$*i*) (*c*$*i*)) .. (χ *i*. *min* (*b*$*i*) (*d*$*i*))}
unfolding *Int_interval*
by (*auto simp*: *mem_box less_eq_vec_def*)
(*auto simp*: *Basis_vec_def inner_axis*)

lemma *closed_interval_left_cart*:
fixes *b* :: *real*^'*n*
shows *closed* {*x*::*real*^'*n*. ∀ *i*. *x*$*i* ≤ *b*$*i*}
by (*simp add*: *Collect_all_eq closed_INT closed_Collect_le continuous_on_component*)

lemma *closed_interval_right_cart*:
fixes *a*::*real*^'*n*
shows *closed* {*x*::*real*^'*n*. ∀ *i*. *a*$*i* ≤ *x*$*i*}
by (*simp add*: *Collect_all_eq closed_INT closed_Collect_le continuous_on_component*)

lemma *is_interval_cart*:
is_interval (*s*::(*real*^'*n*) *set*) ⟷
(∀ *a*∈*s*. ∀ *b*∈*s*. ∀ *x*. (∀ *i*. ((*a*$*i* ≤ *x*$*i* ∧ *x*$*i* ≤ *b*$*i*) ∨ (*b*$*i* ≤ *x*$*i* ∧ *x*$*i* ≤ *a*$*i*))) ⟶ *x* ∈ *s*)
by (*simp add*: *is_interval_def Ball_def Basis_vec_def inner_axis imp_ex*)

lemma *closed_halfspace_component_le_cart*: *closed* {*x*::*real*^'*n*. *x*$*i* ≤ *a*}
by (*simp add*: *closed_Collect_le continuous_on_component*)

lemma *closed_halfspace_component_ge_cart*: *closed* {*x*::*real*^'*n*. *x*$*i* ≥ *a*}

by (*simp add*: *closed_Collect_le continuous_on_component*)

lemma *open_halfspace_component_lt_cart*: *open* {*x*::*real*^'*n*. *x*$*i* < *a*}
by (*simp add*: *open_Collect_less continuous_on_component*)

lemma *open_halfspace_component_gt_cart*: *open* {*x*::*real*^'*n*. *x*$*i* > *a*}
by (*simp add*: *open_Collect_less continuous_on_component*)

lemma *Lim_component_le_cart*:
fixes *f* :: 'a ⇒ *real*^'*n*
assumes (*f* ⟶ *l*) *net* ¬ (*trivial_limit net*) *eventually* (λ*x*. *f x* $*i* ≤ *b*) *net*
shows *l*$*i* ≤ *b*
by (*rule tendsto_le*[*OF assms*(*2*) *tendsto_const tendsto_vec_nth*, *OF assms*(*1*, *3*)])

lemma *Lim_component_ge_cart*:
fixes *f* :: 'a ⇒ *real*^'*n*
assumes (*f* ⟶ *l*) *net* ¬ (*trivial_limit net*) *eventually* (λ*x*. *b* ≤ (*f x*)$*i*) *net*
shows *b* ≤ *l*$*i*
by (*rule tendsto_le*[*OF assms*(*2*) *tendsto_vec_nth tendsto_const*, *OF assms*(*1*, *3*)])

lemma *Lim_component_eq_cart*:
fixes *f* :: 'a ⇒ *real*^'*n*
assumes *net*: (*f* ⟶ *l*) *net* ¬ *trivial_limit net* **and** *ev*:*eventually* (λ*x*. *f*(*x*)$*i* = *b*) *net*
shows *l*$*i* = *b*
using *ev*[*unfolded order_eq_iff eventually_conj_iff*] **and**
Lim_component_ge_cart[*OF net*, *of b i*] **and**
Lim_component_le_cart[*OF net*, *of i b*] **by** *auto*

lemma *connected_ivt_component_cart*:
fixes *x* :: *real*^'*n*
shows *connected s* ⟹ *x* ∈ *s* ⟹ *y* ∈ *s* ⟹ *x*$*k* ≤ *a* ⟹ *a* ≤ *y*$*k* ⟹ (∃ *z*∈*s*. *z*$*k* = *a*)
using *connected_ivt_hyperplane*[*of s x y axis k 1 a*]
by (*auto simp add*: *inner_axis inner_commute*)

lemma *subspace_substandard_cart*: *vec.subspace* {*x*. (∀ *i*. *P i* ⟶ *x*$*i* = *0*)}
unfolding *vec.subspace_def* **by** *auto*

lemma *closed_substandard_cart*:
closed {*x*::'*a*::*real_normed_vector* ^ '*n*. ∀ *i*. *P i* ⟶ *x*$*i* = *0*}
proof −
{ fix *i*::'*n*
have *closed* {*x*::'*a* ^ '*n*. *P i* ⟶ *x*$*i* = *0*}
by (*cases P i*) (*simp_all add*: *closed_Collect_eq continuous_on_component*)
}
thus *?thesis*

unfolding *Collect_all_eq* **by** (*simp add*: *closed_INT*)
qed

5.18.3 Convex Euclidean Space

lemma *Cart_1*:(*1*::*real*^'*n*) = $\sum$ *Basis*
using *const_vector_cart*[*of 1*] **by** (*simp add*: *one_vec_def*)

declare *vector_add_ldistrib*[*simp*] *vector_ssub_ldistrib*[*simp*] *vector_smult_assoc*[*simp*] *vector_smult_rneg*[*simp*]
declare *vector_sadd_rdistrib*[*simp*] *vector_sub_rdistrib*[*simp*]

lemmas *vector_component_simps* = *vector_minus_component vector_smult_component vector_add_component less_eq_vec_def vec_lambda_beta vector_uminus_component*

lemma *convex_box_cart*:
assumes $\bigwedge i$. *convex* {*x*. *P i x*}
shows *convex* {*x*. $\forall i$. *P i* (*x*$*i*)}
using *assms* **unfolding** *convex_def* **by** *auto*

5.18.4 Arbitrarily good rational approximations

lemma *rational_approximation*:
assumes *e* > *0*
obtains *r*::*real* **where** *r* ∈ ℚ |*r* − *x*| < *e*
using *Rats_dense_in_real* [*of x* − *e*/*2 x* + *e*/*2*] *assms* **by** *auto*

lemma *Rats_closure_real*: *closure* ℚ = (*UNIV*::*real set*)
proof −
have $\bigwedge$*x*::*real*. *x* ∈ *closure* ℚ
by (*metis closure_approachable dist_real_def rational_approximation*)
then show *?thesis* **by** *auto*
qed

proposition *matrix_rational_approximation*:
fixes *A* :: *real*^'*n*^'*m*
assumes *e* > *0*
obtains *B* **where** $\bigwedge i\ j$. *B*$*i*$*j* ∈ ℚ *onorm*(λ*x*. (*A* − *B*) *v x*) < *e*
proof −
have $\forall i\ j$. $\exists q \in \mathbb{Q}$. |*q* − *A* $ *i* $ *j*| < *e* / (*2* * *CARD*('*m*) * *CARD*('*n*))
using *assms* **by** (*force intro*: *rational_approximation* [*of e* / (*2* * *CARD*('*m*) * *CARD*('*n*))])
then obtain *B* **where** *B*: $\bigwedge i\ j$. *B*$*i*$*j* ∈ ℚ **and** *Bclo*: $\bigwedge i\ j$. |*B*$*i*$*j* − *A* $ *i* $ *j*| < *e* / (*2* * *CARD*('*m*) * *CARD*('*n*))
by (*auto simp*: *lambda_skolem Bex_def*)
show *?thesis*
proof
have *onorm* ((*v) (*A* − *B*)) ≤ *real CARD*('*m*) * *real CARD*('*n*) *
(*e* / (*2* * *real CARD*('*m*) * *real CARD*('*n*)))
apply (*rule onorm_le_matrix_component*)

using *Bclo* **by** (*simp add*: *abs_minus_commute less_imp_le*)
also have ... < *e*
using ‹*0* < *e*› **by** (*simp add*: *field_split_simps*)
finally show *onorm* ((∗*v*) (*A* − *B*)) < *e* **.**
qed (*use B* **in** *auto*)
qed

5.18.5 Derivative

definition *jacobian f net* = *matrix*(*frechet_derivative f net*)

proposition *jacobian_works*:
(*f*::(*real^'a*) ⇒ (*real^'b*)) *differentiable net* ⟷
(*f has_derivative* (λ*h*. (*jacobian f net*) ∗*v h*)) *net* (**is** *?lhs* = *?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*simp add*: *frechet_derivative_works has_derivative_linear jacobian_def*)
next
assume *?rhs* **then show** *?lhs*
by (*rule differentiableI*)
qed

Component of the differential must be zero if it exists at a local maximum or minimum for that corresponding component

proposition *differential_zero_maxmin_cart*:
fixes *f*::*real^'a* ⇒ *real^'b*
assumes *0* < *e* ((∀ *y* ∈ *ball x e*. (*f y*)$*k* ≤ (*f x*)$*k*) ∨ (∀ *y*∈*ball x e*. (*f x*)$*k* ≤ (*f y*)$*k*))
f differentiable (*at x*)
shows *jacobian f* (*at x*) $ *k* = *0*
using *differential_zero_maxmin_component*[*of axis k 1 e x f*] *assms*
vector_cart[*of* λ*j*. *frechet_derivative f* (*at x*) *j* $ *k*]
by (*simp add*: *Basis_vec_def axis_eq_axis inner_axis jacobian_def matrix_def*)

5.18.6 Routine results connecting the types (*real*, *1*) *vec* and *real*

lemma *vec_cbox_1_eq* [*simp*]:
shows *vec* ‘ *cbox u v* = *cbox* (*vec u*) (*vec v* ::*real^1*)
by (*force simp*: *Basis_vec_def cart_eq_inner_axis* [*symmetric*] *mem_box*)

lemma *vec_nth_cbox_1_eq* [*simp*]:
fixes *u v* :: *'a*::*euclidean_space^1*
shows (λ*x*. *x* $ *1*) ‘ *cbox u v* = *cbox* (*u*$*1*) (*v*$*1*)
by (*auto simp*: *Basis_vec_def cart_eq_inner_axis* [*symmetric*] *mem_box image_iff Bex_def inner_axis*) (*metis vec_component*)

lemma *vec_nth_1_iff_cbox* [*simp*]:
fixes *a b* :: *'a*::*euclidean_space*

shows $(\lambda x::'a\hat{}1.\ x\ \$\ 1)\ `\ S = cbox\ a\ b \longleftrightarrow S = cbox\ (vec\ a)\ (vec\ b)$
(**is** *?lhs = ?rhs*)
proof
assume *L*: *?lhs* **show** *?rhs*
proof (*intro equalityI subsetI*)
fix *x*
assume $x \in S$
then have $x\ \$\ 1 \in (\lambda v.\ v\ \$\ (1::1))\ `\ cbox\ (vec\ a)\ (vec\ b)$
using *L* **by** *auto*
then show $x \in cbox\ (vec\ a)\ (vec\ b)$
by (*metis* (*no_types*, *lifting*) *imageE vector_one_nth*)
next
fix $x :: 'a\hat{}1$
assume $x \in cbox\ (vec\ a)\ (vec\ b)$
then show $x \in S$
by (*metis* (*no_types*, *lifting*) *L imageE imageI vec_component vec_nth_cbox_1_eq vector_one_nth*)
qed
qed *simp*

lemma *vec_nth_real_1_iff_cbox* [*simp*]:
fixes *a b* :: *real*
shows $(\lambda x::real\hat{}1.\ x\ \$\ 1)\ `\ S = \{a..b\} \longleftrightarrow S = cbox\ (vec\ a)\ (vec\ b)$
using *vec_nth_1_iff_cbox*[*of S a b*]
by *simp*

lemma *interval_split_cart*:
$\{a..b::real\hat{}'n\} \cap \{x.\ x\$k \le c\} = \{a\ ..\ (\chi\ i.\ if\ i = k\ then\ min\ (b\$k)\ c\ else\ b\$i)\}$
$cbox\ a\ b \cap \{x.\ x\$k \ge c\} = \{(\chi\ i.\ if\ i = k\ then\ max\ (a\$k)\ c\ else\ a\$i)\ ..\ b\}$
unfolding *Int_iff mem_box_cart mem_Collect_eq interval_cbox_cart set_eq_iff*
unfolding *vec_lambda_beta*
by *auto*

lemmas *cartesian_euclidean_space_uniform_limit_intros*[*uniform_limit_intros*]
=
bounded_linear.uniform_limit[*OF blinfun.bounded_linear_right*]
bounded_linear.uniform_limit[*OF bounded_linear_vec_nth*]

end

5.19 Bernstein-Weierstrass and Stone-Weierstrass

By L C Paulson (2015)

theory *Weierstrass_Theorems*
imports *Uniform_Limit Path_Connected Derivative*
begin

5.19.1 Bernstein polynomials

definition *Bernstein* :: [*nat*,*nat*,*real*] ⇒ *real* **where**
Bernstein n k x ≡ *of_nat* (*n choose k*) ∗ *x*^*k* ∗ (*1* − *x*)^(*n* − *k*)

lemma *Bernstein_nonneg*: ⟦*0* ≤ *x*; *x* ≤ *1*⟧ ⟹ *0* ≤ *Bernstein n k x*
by (*simp add*: *Bernstein_def*)

lemma *Bernstein_pos*: ⟦*0* < *x*; *x* < *1*; *k* ≤ *n*⟧ ⟹ *0* < *Bernstein n k x*
by (*simp add*: *Bernstein_def*)

lemma *sum_Bernstein* [*simp*]: (∑ *k*≤*n*. *Bernstein n k x*) = *1*
using *binomial_ring* [*of x 1*−*x n*]
by (*simp add*: *Bernstein_def*)

lemma *binomial_deriv1*:
(∑ *k*≤*n*. (*of_nat k* ∗ *of_nat* (*n choose k*)) ∗ *a*^(*k*−*1*) ∗ *b*^(*n*−*k*)) = *real_of_nat n* ∗ (*a*+*b*)^(*n*−*1*)
apply (*rule DERIV_unique* [**where** *f* = λ*a*. (*a*+*b*)^*n* **and** *x*=*a*])
apply (*subst binomial_ring*)
apply (*rule derivative_eq_intros sum.cong* | *simp add*: *atMost_atLeast0*)+
done

lemma *binomial_deriv2*:
(∑ *k*≤*n*. (*of_nat k* ∗ *of_nat* (*k*−*1*) ∗ *of_nat* (*n choose k*)) ∗ *a*^(*k*−*2*) ∗ *b*^(*n*−*k*)) =
of_nat n ∗ *of_nat* (*n*−*1*) ∗ (*a*+*b*::*real*)^(*n*−*2*)
apply (*rule DERIV_unique* [**where** *f* = λ*a*. *of_nat n* ∗ (*a*+*b*::*real*)^(*n*−*1*) **and** *x*=*a*])
apply (*subst binomial_deriv1* [*symmetric*])
apply (*rule derivative_eq_intros sum.cong* | *simp add*: *Num.numeral_2_eq_2*)+
done

lemma *sum_k_Bernstein* [*simp*]: (∑ *k*≤*n*. *real k* ∗ *Bernstein n k x*) = *of_nat n* ∗ *x*
apply (*subst binomial_deriv1* [*of n x 1*−*x*, *simplified*, *symmetric*])
apply (*simp add*: *sum_distrib_right*)
apply (*auto simp*: *Bernstein_def algebra_simps power_eq_if intro*!: *sum.cong*)
done

lemma *sum_kk_Bernstein* [*simp*]: (∑ *k*≤*n*. *real k* ∗ (*real k* − *1*) ∗ *Bernstein n k x*) = *real n* ∗ (*real n* − *1*) ∗ *x*^*2*
proof −
have (∑ *k*≤*n*. *real k* ∗ (*real k* − *1*) ∗ *Bernstein n k x*) =
(∑ *k*≤*n*. *real k* ∗ *real* (*k* − *Suc 0*) ∗ *real* (*n choose k*) ∗ *x*^(*k* − *2*) ∗ (*1* − *x*)^(*n* − *k*) ∗ *x*^*2*)
proof (*rule sum.cong* [*OF refl*], *simp*)
fix *k*
assume *k* ≤ *n*
then consider *k* = *0* | *k* = *1* | *k'* **where** *k* = *Suc* (*Suc k'*)

by (*metis One_nat_def not0_implies_Suc*)
then show $k = 0 \vee$
$(real\ k - 1) * Bernstein\ n\ k\ x =$
$real\ (k - Suc\ 0) *$
$(real\ (n\ choose\ k) * (x \hat{} (k - 2) * ((1 - x) \hat{} (n - k) * x^2)))$
by *cases* (*auto simp add*: *Bernstein_def power2_eq_square algebra_simps*)
qed
also have ... $= real\_of\_nat\ n * real\_of\_nat\ (n - Suc\ 0) * x^2$
by (*subst binomial_deriv2* [*of n x 1−x*, *simplified*, *symmetric*]) (*simp add*: *sum_distrib_right*)
also have ... $= n * (n - 1) * x^2$
by *auto*
finally show *?thesis*
by *auto*
qed

5.19.2 Explicit Bernstein version of the 1D Weierstrass approximation theorem

theorem *Bernstein_Weierstrass*:
fixes $f :: real \Rightarrow real$
assumes *contf*: *continuous_on* $\{0..1\}$ f **and** e: $0 < e$
shows $\exists N.\ \forall n\ x.\ N \leq n \wedge x \in \{0..1\}$
$\longrightarrow |f\ x - (\sum k{\leq}n.\ f(k/n) * Bernstein\ n\ k\ x)| < e$
proof −
have *bounded* $(f\ `\ \{0..1\})$
using *compact_continuous_image compact_imp_bounded contf* **by** *blast*
then obtain M **where** M: $\bigwedge x.\ 0 \leq x \Longrightarrow x \leq 1 \Longrightarrow |f\ x| \leq M$
by (*force simp add*: *bounded_iff*)
then have $0 \leq M$ **by** *force*
have *ucontf*: *uniformly_continuous_on* $\{0..1\}$ f
using *compact_uniformly_continuous contf* **by** *blast*
then obtain d **where** d: $d{>}0$ $\bigwedge x\ x'.\ [\![x \in \{0..1\};\ x' \in \{0..1\};\ |x' - x| < d]\!] \Longrightarrow |f\ x' - f\ x| < e/2$
apply (*rule uniformly_continuous_onE* [**where** $e = e/2$])
using e **by** (*auto simp*: *dist_norm*)
{ fix n::*nat* **and** x::*real*
assume n: $Suc\ (nat\lceil 4{*}M/(e{*}d^2)\rceil) \leq n$ **and** x: $0 \leq x\ x \leq 1$
have $0 < n$ **using** n **by** *simp*
have *ed0*: $-(e * d^2) < 0$
using e ‹$0{<}d$› **by** *simp*
also have ... $\leq M * 4$
using ‹$0{\leq}M$› **by** *simp*
finally have [*simp*]: $real\_of\_int\ (nat\ \lceil 4 * M\ /\ (e * d^2)\rceil) = real\_of\_int\ \lceil 4 * M\ /\ (e * d^2)\rceil$
using ‹$0{\leq}M$› e ‹$0{<}d$›
by (*simp add*: *field_simps*)
have $4{*}M/(e{*}d^2) + 1 \leq real\ (Suc\ (nat\lceil 4{*}M/(e{*}d^2)\rceil))$
by (*simp add*: *real_nat_ceiling_ge*)

```
  also have ... ≤ real n
    using n by (simp add: field_simps)
  finally have nbig: 4*M/(e*d^2) + 1 ≤ real n .
  have sum_bern: (∑k≤n. (x - k/n)^2 * Bernstein n k x) = x * (1 - x) / n
  proof -
    have *: ⋀a b x::real. (a - b)^2 * x = a * (a - 1) * x + (1 - 2 * b) * a * x
+ b * b * x
      by (simp add: algebra_simps power2_eq_square)
    have (∑k≤n. (k - n * x)^2 * Bernstein n k x) = n * x * (1 - x)
      apply (simp add: * sum.distrib)
      apply (simp flip: sum_distrib_left add: mult.assoc)
      apply (simp add: algebra_simps power2_eq_square)
      done
    then have (∑k≤n. (k - n * x)^2 * Bernstein n k x)/n^2 = x * (1 - x) / n
      by (simp add: power2_eq_square)
    then show ?thesis
    using n by (simp add: sum_divide_distrib field_split_simps power2_commute)
  qed
  { fix k
    assume k: k ≤ n
    then have kn: 0 ≤ k / n k / n ≤ 1
      by (auto simp: field_split_simps)
    consider (lessd) |x - k / n| < d | (ged) d ≤ |x - k / n|
      by linarith
    then have |(f x - f (k/n))| ≤ e/2 + 2 * M / d^2 * (x - k/n)^2
    proof cases
      case lessd
      then have |(f x - f (k/n))| < e/2
        using d x kn by (simp add: abs_minus_commute)
      also have ... ≤ (e/2 + 2 * M / d^2 * (x - k/n)^2)
        using ‹M≥0› d by simp
      finally show ?thesis by simp
    next
      case ged
      then have dle: d^2 ≤ (x - k/n)^2
        by (metis d(1) less_eq_real_def power2_abs power_mono)
      have §: 1 ≤ (x - real k / real n)^2 / d^2
        using dle ‹d>0› by auto
      have |(f x - f (k/n))| ≤ |f x| + |f (k/n)|
        by (rule abs_triangle_ineq4)
      also have ... ≤ M+M
        by (meson M add_mono_thms_linordered_semiring(1) kn x)
      also have ... ≤ 2 * M * ((x - k/n)^2 / d^2)
        using § ‹M≥0› mult_left_mono by fastforce
      also have ... ≤ e/2 + 2 * M / d^2 * (x - k/n)^2
        using e  by simp
      finally show ?thesis .
      qed
  } note * = this
```

have $|f\ x - (\sum k{\le}n.\ f(k\ /\ n) * \mathit{Bernstein}\ n\ k\ x)| \le |\sum k{\le}n.\ (f\ x - f(k\ /\ n)) * \mathit{Bernstein}\ n\ k\ x|$
by (*simp add*: *sum_subtractf sum_distrib_left* [*symmetric*] *algebra_simps*)
also have $\ldots \le (\sum k{\le}n.\ |(f\ x - f(k\ /\ n)) * \mathit{Bernstein}\ n\ k\ x|)$
by (*rule sum_abs*)
also have $\ldots \le (\sum k{\le}n.\ (e/2 + (2 * M\ /\ d^2) * (x - k\ /\ n)^2) * \mathit{Bernstein}\ n\ k\ x)$
using $*$
by (*force simp add*: *abs_mult Bernstein_nonneg x mult_right_mono intro*: *sum_mono*)
also have $\ldots \le e/2 + (2 * M)\ /\ (d^2 * n)$
unfolding *sum.distrib Rings.semiring_class.distrib_right sum_distrib_left* [*symmetric*] *mult.assoc sum_bern*
using ‹$d{>}0$› x **by** (*simp add*: *divide_simps* ‹$M{\ge}0$› *mult_le_one mult_left_le*)
also have $\ldots < e$
using ‹$d{>}0$› *nbig e* ‹$n{>}0$›
apply (*simp add*: *field_split_simps*)
using *ed0* **by** *linarith*
finally have $|f\ x - (\sum k{\le}n.\ f\ (\mathit{real}\ k\ /\ \mathit{real}\ n) * \mathit{Bernstein}\ n\ k\ x)| < e$ **.**
}
then show *?thesis*
by *auto*
qed

5.19.3 General Stone-Weierstrass theorem

Source: Bruno Brosowski and Frank Deutsch. An Elementary Proof of the Stone-Weierstrass Theorem. Proceedings of the American Mathematical Society Volume 81, Number 1, January 1981. DOI: 10.2307/2043993 https://www.jstor.org/stable/2043993

locale *function_ring_on* =
fixes $R :: ('a{::}t2\_space \Rightarrow \mathit{real})\ \mathit{set}$ **and** $S :: 'a\ \mathit{set}$
assumes *compact*: *compact S*
assumes *continuous*: $f \in R \Longrightarrow \mathit{continuous\_on}\ S\ f$
assumes *add*: $f \in R \Longrightarrow g \in R \Longrightarrow (\lambda x.\ f\ x + g\ x) \in R$
assumes *mult*: $f \in R \Longrightarrow g \in R \Longrightarrow (\lambda x.\ f\ x * g\ x) \in R$
assumes *const*: $(\lambda\_.\ c) \in R$
assumes *separable*: $x \in S \Longrightarrow y \in S \Longrightarrow x \neq y \Longrightarrow \exists f{\in}R.\ f\ x \neq f\ y$

begin
lemma *minus*: $f \in R \Longrightarrow (\lambda x.\ -f\ x) \in R$
by (*frule mult* [*OF const* [*of* -1]]) *simp*

lemma *diff*: $f \in R \Longrightarrow g \in R \Longrightarrow (\lambda x.\ f\ x - g\ x) \in R$
unfolding *diff_conv_add_uminus* **by** (*metis add minus*)

lemma *power*: $f \in R \Longrightarrow (\lambda x.\ f\ x\hat{}n) \in R$
by (*induct n*) (*auto simp*: *const mult*)

lemma *sum*: $[\![\mathit{finite}\ I;\ \bigwedge i.\ i \in I \Longrightarrow f\ i \in R]\!] \Longrightarrow (\lambda x.\ \sum i \in I.\ f\ i\ x) \in R$
by (*induct I rule*: *finite_induct*; *simp add*: *const add*)

lemma *prod*: $[\![\mathit{finite}\ I;\ \bigwedge i.\ i \in I \Longrightarrow f\ i \in R]\!] \Longrightarrow (\lambda x.\ \prod i \in I.\ f\ i\ x) \in R$
by (*induct I rule*: *finite_induct*; *simp add*: *const mult*)

definition *normf* :: ('*a*::*t2_space* ⇒ *real*) ⇒ *real*
where *normf f* ≡ *SUP x*∈*S*. |*f x*|

lemma *normf_upper*:
assumes *continuous_on S f x* ∈ *S* **shows** |*f x*| ≤ *normf f*
proof −
have *bdd_above* ((λ*x*. |*f x*|) ‘ *S*)
by (*simp add*: *assms*(*1*) *bounded_imp_bdd_above compact compact_continuous_image compact_imp_bounded continuous_on_rabs*)
then show *?thesis*
using *assms cSUP_upper normf_def* **by** *fastforce*
qed

lemma *normf_least*: *S* ≠ {} ⟹ (⋀*x*. *x* ∈ *S* ⟹ |*f x*| ≤ *M*) ⟹ *normf f* ≤ *M*
by (*simp add*: *normf_def cSUP_least*)

end

lemma (**in** *function_ring_on*) *one*:
assumes *U*: *open U* **and** *t0*: *t0* ∈ *S t0* ∈ *U* **and** *t1*: *t1* ∈ *S*−*U*
shows ∃ *V*. *open V* ∧ *t0* ∈ *V* ∧ *S* ∩ *V* ⊆ *U* ∧
(∀ *e*>*0*. ∃*f* ∈ *R*. *f* ‘ *S* ⊆ {*0*..*1*} ∧ (∀ *t* ∈ *S* ∩ *V*. *f t* < *e*) ∧ (∀ *t* ∈ *S* − *U*. *f t* > *1* − *e*))
proof −
have ∃ *pt* ∈ *R*. *pt t0* = *0* ∧ *pt t* > *0* ∧ *pt* ‘ *S* ⊆ {*0*..*1*} **if** *t*: *t* ∈ *S* − *U* **for** *t*
proof −
have *t* ≠ *t0* **using** *t t0* **by** *auto*
then obtain *g* **where** *g*: *g* ∈ *R g t* ≠ *g t0*
using *separable t0* **by** (*metis Diff_subset subset_eq t*)
define *h* **where** [*abs_def*]: *h x* = *g x* − *g t0* **for** *x*
have *h* ∈ *R*
unfolding *h_def* **by** (*fast intro*: *g const diff*)
then have *hsq*: (λ*w*. (*h w*)^2) ∈ *R*
by (*simp add*: *power2_eq_square mult*)
have *h t* ≠ *h t0*
by (*simp add*: *h_def g*)
then have *h t* ≠ *0*
by (*simp add*: *h_def*)
then have *ht2*: *0* < (*h t*)^2
by *simp*
also have ... ≤ *normf* (λ*w*. (*h w*)^2)
using *t normf_upper* [**where** *x*=*t*] *continuous* [*OF hsq*] **by** *force*

```
    finally have nfp: 0 < normf (λw. (h w)²) .
    define p where [abs_def]: p x = (1 / normf (λw. (h w)²)) * (h x)^2 for x
    have p ∈ R
      unfolding p_def by (fast intro: hsq const mult)
    moreover have p t0 = 0
      by (simp add: p_def h_def)
    moreover have p t > 0
      using nfp ht2 by (simp add: p_def)
    moreover have ⋀x. x ∈ S ⟹ p x ∈ {0..1}
      using nfp normf_upper [OF continuous [OF hsq] ] by (auto simp: p_def)
    ultimately show ∃ pt ∈ R. pt t0 = 0 ∧ pt t > 0 ∧ pt ‘ S ⊆ {0..1}
      by auto
  qed
  then obtain pf where pf: ⋀t. t ∈ S−U ⟹ pf t ∈ R ∧ pf t t0 = 0 ∧ pf t t >
0
                  and pf01: ⋀t. t ∈ S−U ⟹ pf t ‘ S ⊆ {0..1}
    by metis
  have com_sU: compact (S−U)
   using compact closed_Int_compact U by (simp add: Diff_eq compact_Int_closed
open_closed)
  have ⋀t. t ∈ S−U ⟹ ∃ A. open A ∧ A ∩ S = {x∈S. 0 < pf t x}
    apply (rule open_Collect_positive)
    by (metis pf continuous)
  then obtain Uf where Uf: ⋀t. t ∈ S−U ⟹ open (Uf t) ∧ (Uf t) ∩ S = {x∈S.
0 < pf t x}
    by metis
  then have open_Uf: ⋀t. t ∈ S−U ⟹ open (Uf t)
    by blast
  have tUft: ⋀t. t ∈ S−U ⟹ t ∈ Uf t
    using pf Uf by blast
  then have *: S−U ⊆ (⋃ x ∈ S−U. Uf x)
    by blast
  obtain subU where subU: subU ⊆ S − U finite subU S − U ⊆ (⋃ x ∈ subU.
Uf x)
    by (blast intro: that compactE_image [OF com_sU open_Uf *])
  then have [simp]: subU ≠ {}
    using t1 by auto
  then have cardp: card subU > 0 using subU
    by (simp add: card_gt_0_iff)
  define p where [abs_def]: p x = (1 / card subU) * (∑ t ∈ subU. pf t x) for x
  have pR: p ∈ R
    unfolding p_def using subU pf by (fast intro: pf const mult sum)
  have pt0 [simp]: p t0 = 0
    using subU pf by (auto simp: p_def intro: sum.neutral)
  have pt_pos: p t > 0 if t: t ∈ S−U for t
  proof −
    obtain i where i: i ∈ subU t ∈ Uf i using subU t by blast
    show ?thesis
      using subU i t
```

```
    apply (clarsimp simp: p_def field_split_simps)
    apply (rule sum_pos2 [OF ‹finite subU›])
    using Uf t pf01 apply auto
    apply (force elim!: subsetCE)
    done
 qed
 have p01: p x ∈ {0..1} if t: x ∈ S for x
 proof −
  have 0 ≤ p x
    using subU cardp t pf01
    by (fastforce simp add: p_def field_split_simps intro: sum_nonneg)
  moreover have p x ≤ 1
    using subU cardp t
    apply (simp add: p_def field_split_simps)
    apply (rule sum_bounded_above [where 'a=real and K=1, simplified])
    using pf01 by force
  ultimately show ?thesis
    by auto
 qed
 have compact (p ‘ (S−U))
   by (meson Diff_subset com_sU compact_continuous_image continuous continuous_on_subset pR)
 then have open (− (p ‘ (S−U)))
  by (simp add: compact_imp_closed open_Compl)
 moreover have 0 ∈ − (p ‘ (S−U))
  by (metis (no_types) ComplI image_iff not_less_iff_gr_or_eq pt_pos)
 ultimately obtain delta0 where delta0: delta0 > 0 ball 0 delta0 ⊆ − (p ‘ (S−U))
  by (auto simp: elim!: openE)
 then have pt_delta: ⋀x. x ∈ S−U ⟹ p x ≥ delta0
  by (force simp: ball_def dist_norm dest: p01)
 define δ where δ = delta0/2
 have delta0 ≤ 1 using delta0 p01 [of t1] t1
   by (force simp: ball_def dist_norm dest: p01)
 with delta0 have δ01: 0 < δ δ < 1
  by (auto simp: δ_def)
 have pt_δ: ⋀x. x ∈ S−U ⟹ p x ≥ δ
  using pt_delta delta0 by (force simp: δ_def)
 have ∃ A. open A ∧ A ∩ S = {x∈S. p x < δ/2}
 by (rule open_Collect_less_Int [OF continuous [OF pR] continuous_on_const])
 then obtain V where V: open V V ∩ S = {x∈S. p x < δ/2}
  by blast
 define k where k = nat⌊1/δ⌋ + 1
 have k>0  by (simp add: k_def)
 have k−1 ≤ 1/δ
  using δ01 by (simp add: k_def)
 with δ01 have k ≤ (1+δ)/δ
  by (auto simp: algebra_simps add_divide_distrib)
 also have ... < 2/δ
```

```
  using δ01 by (auto simp: field_split_simps)
 finally have k2δ: k < 2/δ .
 have 1/δ < k
  using δ01 unfolding k_def by linarith
 with δ01 k2δ have kδ: 1 < k*δ k*δ < 2
  by (auto simp: field_split_simps)
 define q where [abs_def]: q n t = (1 - p t^n)^(k^n) for n t
 have qR: q n ∈ R for n
  by (simp add: q_def const diff power pR)
 have q01: ⋀n t. t ∈ S ⟹ q n t ∈ {0..1}
  using p01 by (simp add: q_def power_le_one algebra_simps)
 have qt0 [simp]: ⋀n. n>0 ⟹ q n t0 = 1
  using t0 pf by (simp add: q_def power_0_left)
 { fix t and n::nat
  assume t: t ∈ S ∩ V
  with ‹k>0› V have k * p t < k * δ / 2
    by force
  then have 1 - (k * δ / 2)^n ≤ 1 - (k * p t)^n
    using ‹k>0› p01 t by (simp add: power_mono)
  also have ... ≤ q n t
    using Bernoulli_inequality [of - ((p t)^n) k^n]
    apply (simp add: q_def)
    by (metis IntE atLeastAtMost_iff p01 power_le_one power_mult_distrib t)
  finally have 1 - (k * δ / 2)^n ≤ q n t .
 } note limitV = this
 { fix t and n::nat
  assume t: t ∈ S - U
  with ‹k>0› U have k * δ ≤ k * p t
    by (simp add: pt_δ)
  with kδ have kpt: 1 < k * p t
    by (blast intro: less_le_trans)
  have ptn_pos: 0 < p t^n
    using pt_pos [OF t] by simp
  have ptn_le: p t^n ≤ 1
    by (meson DiffE atLeastAtMost_iff p01 power_le_one t)
  have q n t = (1/(k^n * (p t)^n)) * (1 - p t^n)^(k^n) * k^n * (p t)^n
    using pt_pos [OF t] ‹k>0› by (simp add: q_def)
  also have ... ≤ (1/(k * (p t))^n) * (1 - p t^n)^(k^n) * (1 + k^n * (p t)^n)
    using pt_pos [OF t] ‹k>0›
    by (simp add: divide_simps mult_left_mono ptn_le)
  also have ... ≤ (1/(k * (p t))^n) * (1 - p t^n)^(k^n) * (1 + (p t)^n)^(k^n)
  proof (rule mult_left_mono [OF Bernoulli_inequality])
    show 0 ≤ 1 / (real k * p t)^n * (1 - p t^n)^k^n
      using ptn_pos ptn_le by (auto simp: power_mult_distrib)
  qed (use ptn_pos in auto)
  also have ... = (1/(k * (p t))^n) * (1 - p t^(2*n))^(k^n)
    using pt_pos [OF t] ‹k>0›
  by (simp add: algebra_simps power_mult power2_eq_square flip: power_mult_distrib)
  also have ... ≤ (1/(k * (p t))^n) * 1
```

```
    using pt_pos ‹k>0› p01 power_le_one t
    by (intro mult_left_mono [OF power_le_one]) auto
  also have ... ≤ (1 / (k*δ))^n
    using ‹k>0› δ01  power_mono pt_δ t
    by (fastforce simp: field_simps)
  finally have q n t ≤ (1 / (real k * δ))^n  .
 } note limitNonU = this
 define NN
  where NN e = 1 + nat ⌈max (ln e / ln (real k * δ / 2)) (- ln e / ln (real k
* δ))⌉ for e
 have NN: of_nat (NN e) > ln e / ln (real k * δ / 2)  of_nat (NN e) > - ln e
/ ln (real k * δ)
          if 0<e for e
    unfolding NN_def  by linarith+
  have NN1: (k * δ / 2)^NN e < e if e>0 for e
  proof -
    have ln ((real k * δ / 2)^NN e) = real (NN e) * ln (real k * δ / 2)
      by (simp add: ‹δ>0› ‹0 < k› ln_realpow)
    also have ... < ln e
      using NN kδ that by (force simp add: field_simps)
    finally show ?thesis
      by (simp add: ‹δ>0› ‹0 < k› that)
  qed
 have NN0: (1/(k*δ))^(NN e) < e if e>0 for e
 proof -
  have 0 < ln (real k) + ln δ
    using δ01(1) ‹0 < k› kδ(1) ln_gt_zero ln_mult by fastforce
  then have real (NN e) * ln (1 / (real k * δ)) < ln e
    using kδ(1) NN(2) [of e] that by (simp add: ln_div divide_simps)
  then have exp (real (NN e) * ln (1 / (real k * δ))) < e
    by (metis exp_less_mono exp_ln that)
  then show ?thesis
    by (simp add: δ01(1) ‹0 < k› exp_of_nat_mult)
 qed
 { fix t and e::real
  assume e>0
  have t ∈ S ∩ V ⟹ 1 - q (NN e) t < e t ∈ S - U ⟹ q (NN e) t < e
  proof -
    assume t: t ∈ S ∩ V
    show 1 - q (NN e) t < e
      by (metis add.commute diff_le_eq not_le limitV [OF t] less_le_trans [OF
NN1 [OF ‹e>0›]])
  next
    assume t: t ∈ S - U
    show q (NN e) t < e
     using  limitNonU [OF t] less_le_trans [OF NN0 [OF ‹e>0›]] not_le by
blast
  qed
 } then have ⋀e. e > 0 ⟹ ∃f∈R. f ‘ S ⊆ {0..1} ∧ (∀t ∈ S ∩ V. f t < e) ∧
```

```
(∀ t ∈ S − U. 1 − e < f t)
    using q01
    by (rule_tac x=λx. 1 − q (NN e) x in bexI) (auto simp: algebra_simps intro:
diff const qR)
  moreover have t0V: t0 ∈ V  S ∩ V ⊆ U
    using pt_δ t0 U V δ01  by fastforce+
  ultimately show ?thesis using V t0V
    by blast
qed
```

Non-trivial case, with A and B both non-empty

```
lemma (in function_ring_on) two_special:
  assumes A: closed A  A ⊆ S a ∈ A
      and B: closed B B ⊆ S b ∈ B
      and disj: A ∩ B = {}
      and e: 0 < e e < 1
    shows ∃f ∈ R. f ‘ S ⊆ {0..1} ∧ (∀ x ∈ A. f x < e) ∧ (∀ x ∈ B. f x > 1 − e)
proof −
  { fix w
    assume w ∈ A
    then have open ( − B) b ∈ S w ∉ B w ∈ S
      using assms by auto
    then have ∃ V. open V ∧ w ∈ V ∧ S ∩ V ⊆ −B ∧
            (∀ e>0. ∃f ∈ R. f ‘ S ⊆ {0..1} ∧ (∀ x ∈ S ∩ V. f x < e) ∧ (∀ x ∈ S
∩ B. f x > 1 − e))
      using one [of −B w b] assms ‹w ∈ A› by simp
  }
  then obtain Vf where Vf:
        ⋀w. w ∈ A ⟹ open (Vf w) ∧ w ∈ Vf w ∧ S ∩ Vf w ⊆ −B ∧
                    (∀ e>0. ∃f ∈ R. f ‘ S ⊆ {0..1} ∧ (∀ x ∈ S ∩ Vf w. f x < e) ∧
(∀ x ∈ S ∩ B. f x > 1 − e))
    by metis
  then have open_Vf: ⋀w. w ∈ A ⟹ open (Vf w)
    by blast
  have tVft: ⋀w. w ∈ A ⟹ w ∈ Vf w
    using Vf by blast
  then have sum_max_0: A ⊆ (⋃ x ∈ A. Vf x)
    by blast
  have com_A: compact A using A
    by (metis compact compact_Int_closed inf.absorb_iff2)
  obtain subA where subA: subA ⊆ A finite subA A ⊆ (⋃ x ∈ subA. Vf x)
    by (blast intro: that compactE_image [OF com_A open_Vf sum_max_0])
  then have [simp]: subA ≠ {}
    using ‹a ∈ A› by auto
  then have cardp: card subA > 0 using subA
    by (simp add: card_gt_0_iff)
  have ⋀w. w ∈ A ⟹ ∃f ∈ R. f ‘ S ⊆ {0..1} ∧ (∀ x ∈ S ∩ Vf w. f x < e / card
subA) ∧ (∀ x ∈ S ∩ B. f x > 1 − e / card subA)
    using Vf e cardp by simp
```

then obtain *ff* **where** *ff*:
$\bigwedge w.\ w \in A \Longrightarrow ff\ w \in R \wedge ff\ w\ `\ S \subseteq \{0..1\} \wedge$
$(\forall x \in S \cap Vf\ w.\ ff\ w\ x < e\ /\ card\ subA) \wedge (\forall x \in S \cap B.\ ff\ w\ x > 1 - e\ /\ card\ subA)$
by *metis*
define *pff* **where** [*abs_def*]: $pff\ x = (\prod w \in subA.\ ff\ w\ x)$ **for** *x*
have *pffR*: $pff \in R$
unfolding *pff_def* **using** *subA ff* **by** (*auto simp*: *intro*: *prod*)
moreover
have *pff01*: $pff\ x \in \{0..1\}$ **if** *t*: $x \in S$ **for** *x*
proof −
have $0 \leq pff\ x$
using *subA cardp t ff*
by (*fastforce simp*: *pff_def field_split_simps sum_nonneg intro*: *prod_nonneg*)
moreover have $pff\ x \leq 1$
using *subA cardp t ff*
by (*fastforce simp add*: *pff_def field_split_simps sum_nonneg intro*: *prod_mono* [**where** $g = \lambda x.\ 1$, *simplified*])
ultimately show *?thesis*
by *auto*
qed
moreover
{ fix *v x*
assume *v*: $v \in subA$ **and** *x*: $x \in Vf\ v\ x \in S$
from *subA v* **have** $pff\ x = ff\ v\ x * (\prod w \in subA - \{v\}.\ ff\ w\ x)$
unfolding *pff_def* **by** (*metis prod.remove*)
also have $... \leq ff\ v\ x * 1$
proof −
have $\bigwedge i.\ i \in subA - \{v\} \Longrightarrow 0 \leq ff\ i\ x \wedge ff\ i\ x \leq 1$
by (*metis Diff_subset atLeastAtMost_iff ff image_subset_iff subA*(*1*) *subsetD x*(*2*))
moreover have $0 \leq ff\ v\ x$
using *ff subA*(*1*) *v x*(*2*) **by** *fastforce*
ultimately show *?thesis*
by (*metis mult_left_mono prod_mono* [**where** $g = \lambda x.\ 1$, *simplified*])
qed
also have $... < e\ /\ card\ subA$
using *ff subA*(*1*) *v x* **by** *auto*
also have $... \leq e$
using *cardp e* **by** (*simp add*: *field_split_simps*)
finally have $pff\ x < e$ **.**
}
then have $\bigwedge x.\ x \in A \Longrightarrow pff\ x < e$
using *A Vf subA* **by** (*metis UN_E contra_subsetD*)
moreover
{ fix *x*
assume *x*: $x \in B$
then have $x \in S$
using *B* **by** *auto*

```
    have 1 − e ≤ (1 − e / card subA) ^ card subA
      using Bernoulli_inequality [of −e / card subA card subA] e cardp
      by (auto simp: field_simps)
    also have ... = (∏ w ∈ subA. 1 − e / card subA)
      by (simp add: subA(2))
    also have ... < pff x
    proof −
      have ⋀i. i ∈ subA ⟹ e / real (card subA) ≤ 1 ∧ 1 − e / real (card subA)
< ff i x
        using e ‹B ⊆ S› ff subA(1) x by (force simp: field_split_simps)
      then show ?thesis
        using prod_mono_strict [where f = λx. 1 − e / card subA] subA(2) by
(force simp add: pff_def)
    qed
    finally have 1 − e < pff x .
  }
  ultimately show ?thesis by blast
qed

lemma (in function_ring_on) two:
  assumes A: closed A A ⊆ S
      and B: closed B B ⊆ S
      and disj: A ∩ B = {}
      and e: 0 < e e < 1
    shows ∃f ∈ R. f ‘ S ⊆ {0..1} ∧ (∀x ∈ A. f x < e) ∧ (∀x ∈ B. f x > 1 − e)
proof (cases A ≠ {} ∧ B ≠ {})
  case True then show ?thesis
    using assms
    by (force simp flip: ex_in_conv intro!: two_special)
next
  case False
  then consider A={} | B={} by force
  then show ?thesis
  proof cases
    case 1
    with e show ?thesis
      by (rule_tac x=λx. 1 in bexI) (auto simp: const)
  next
    case 2
    with e show ?thesis
      by (rule_tac x=λx. 0 in bexI) (auto simp: const)
  qed
qed
```

The special case where f is non-negative and $e < (1::'a) \,/\, (3::'a)$

```
lemma (in function_ring_on) Stone_Weierstrass_special:
  assumes f: continuous_on S f and fpos: ⋀x. x ∈ S ⟹ f x ≥ 0
      and e: 0 < e e < 1/3
  shows ∃g ∈ R. ∀x∈S. |f x − g x| < 2∗e
```

```
proof −
  define n where n = 1 + nat ⌈normf f / e⌉
  define A where A j = {x ∈ S. f x ≤ (j − 1/3)∗e} for j :: nat
  define B where B j = {x ∈ S. f x ≥ (j + 1/3)∗e} for j :: nat
  have ngt: (n−1) ∗ e ≥ normf f
    using e pos_divide_le_eq real_nat_ceiling_ge[of normf f / e]
    by (fastforce simp add: divide_simps n_def)
  moreover have n≥1
    by (simp_all add: n_def)
  ultimately have ge_fx: (n−1) ∗ e ≥ f x if x ∈ S for x
    using f normf_upper that by fastforce
  have closed S
    by (simp add: compact compact_imp_closed)
  { fix j
    have closed (A j) A j ⊆ S
    using ‹closed S› continuous_on_closed_Collect_le [OF f continuous_on_const]
      by (simp_all add: A_def Collect_restrict)
    moreover have closed (B j) B j ⊆ S
     using ‹closed S› continuous_on_closed_Collect_le [OF continuous_on_const
f]
      by (simp_all add: B_def Collect_restrict)
    moreover have (A j) ∩ (B j) = {}
      using e by (auto simp: A_def B_def field_simps)
    ultimately have ∃f ∈ R. f ‘ S ⊆ {0..1} ∧ (∀x ∈ A j. f x < e/n) ∧ (∀x ∈ B
j. f x > 1 − e/n)
      using e ‹1 ≤ n› by (auto intro: two)
  }
  then obtain xf where xfR: ⋀j. xf j ∈ R and xf01: ⋀j. xf j ‘ S ⊆ {0..1}
                   and xfA: ⋀x j. x ∈ A j ⟹ xf j x < e/n
                   and xfB: ⋀x j. x ∈ B j ⟹ xf j x > 1 − e/n
    by metis
  define g where [abs_def]: g x = e ∗ (∑ i≤n. xf i x) for x
  have gR: g ∈ R
    unfolding g_def by (fast intro: mult const sum xfR)
  have gge0: ⋀x. x ∈ S ⟹ g x ≥ 0
   using e xf01 by (simp add: g_def zero_le_mult_iff image_subset_iff sum_nonneg)
  have A0: A 0 = {}
    using fpos e by (fastforce simp: A_def)
  have An: A n = S
     using e ngt ‹n≥1› f normf_upper by (fastforce simp: A_def field_simps
of_nat_diff)
  have Asub: A j ⊆ A i if i≥j for i j
    using e that by (force simp: A_def intro: order_trans)
  { fix t
    assume t: t ∈ S
    define j where j = (LEAST j. t ∈ A j)
    have jn: j ≤ n
      using t An by (simp add: Least_le j_def)
    have Aj: t ∈ A j
```

```
  using t An by (fastforce simp add: j_def intro: LeastI)
then have Ai: t ∈ A i if i≥j for i
  using Asub [OF that] by blast
then have fj1: f t ≤ (j − 1/3)∗e
  by (simp add: A_def)
then have Anj: t ∉ A i if i<j for i
  using  Aj ‹i<j› not_less_Least by (fastforce simp add: j_def)
have j1: 1 ≤ j
  using A0 Aj j_def not_less_eq_eq by (fastforce simp add: j_def)
then have Anj: t ∉ A (j−1)
  using Least_le by (fastforce simp add: j_def)
then have fj2: (j − 4/3)∗e < f t
  using j1 t  by (simp add: A_def of_nat_diff)
have xf_le1: ⋀i. xf i t ≤ 1
  using xf01 t by force
have g t = e ∗ (∑ i≤n. xf i t)
  by (simp add: g_def flip: distrib_left)
also have ... = e ∗ (∑ i ∈ {..<j} ∪ {j..n}. xf i t)
  by (simp add: ivl_disj_un_one(4) jn)
also have ... = e ∗ (∑ i<j. xf i t) + e ∗ (∑ i=j..n. xf i t)
  by (simp add: distrib_left ivl_disj_int sum.union_disjoint)
also have ... ≤ e∗j + e ∗ ((Suc n − j)∗e/n)
proof (intro add_mono mult_left_mono)
  show (∑ i<j. xf i t) ≤ j
  by (rule sum_bounded_above [OF xf_le1, where A = lessThan j, simplified])
  have xf i t ≤ e/n if i≥j for i
    using xfA [OF Ai] that by (simp add: less_eq_real_def)
  then show (∑ i = j..n. xf i t) ≤ real (Suc n − j) ∗ e / real n
    using sum_bounded_above [of {j..n} λi. xf i t]
    by fastforce
qed (use e in auto)
also have ... ≤ j∗e + e∗(n − j + 1)∗e/n
  using ‹1 ≤ n› e  by (simp add: field_simps del: of_nat_Suc)
also have ... ≤ j∗e + e∗e
  using ‹1 ≤ n› e j1 by (simp add: field_simps del: of_nat_Suc)
also have ... < (j + 1/3)∗e
  using e by (auto simp: field_simps)
finally have gj1: g t < (j + 1 / 3) ∗ e .
have gj2: (j − 4/3)∗e < g t
proof (cases 2 ≤ j)
  case False
  then have j=1 using j1 by simp
  with t gge0 e show ?thesis by force
next
  case True
  then have (j − 4/3)∗e < (j−1)∗e − e^2
    using e by (auto simp: of_nat_diff algebra_simps power2_eq_square)
  also have ... < (j−1)∗e − ((j − 1)/n) ∗ e^2
    using e True jn by (simp add: power2_eq_square field_simps)
```

```
    also have ... = e * (j-1) * (1 - e/n)
      by (simp add: power2_eq_square field_simps)
    also have ... ≤ e * (∑ i≤j-2. xf i t)
    proof -
      { fix i
        assume i+2 ≤ j
        then obtain d where i+2+d = j
          using le_Suc_ex that by blast
        then have t ∈ B i
          using Anj e ge_fx [OF t] ‹1 ≤ n› fpos [OF t] t
          unfolding A_def B_def
           by (auto simp add: field_simps of_nat_diff not_le intro: order_trans
[of _ e * 2 + e * d * 3 + e * i * 3])
        then have xf i t > 1 - e/n
          by (rule xfB)
      }
     moreover have real (j - Suc 0) * (1 - e / real n) ≤ real (card {..j - 2})
* (1 - e / real n)
        using Suc_diff_le True by fastforce
      ultimately show ?thesis
       using e True by (auto intro: order_trans [OF _ sum_bounded_below [OF
less_imp_le]])
    qed
    also have ... ≤ g t
      using jn e xf01 t
     by (auto intro!: Groups_Big.sum_mono2 simp add: g_def zero_le_mult_iff
image_subset_iff sum_nonneg)
    finally show ?thesis .
  qed
  have |f t - g t| < 2 * e
    using fj1 fj2 gj1 gj2 by (simp add: abs_less_iff field_simps)
 }
 then show ?thesis
   by (rule_tac x=g in bexI) (auto intro: gR)
qed
```

The “unpretentious” formulation

```
proposition (in function_ring_on) Stone_Weierstrass_basic:
 assumes f: continuous_on S f and e: e > 0
 shows ∃ g ∈ R. ∀ x∈S. |f x - g x| < e
proof -
 have ∃ g ∈ R. ∀ x∈S. |(f x + normf f) - g x| < 2 * min (e/2) (1/4)
 proof (rule Stone_Weierstrass_special)
   show continuous_on S (λx. f x + normf f)
    by (force intro: Limits.continuous_on_add [OF f Topological_Spaces.continuous_on_const])
   show ⋀x. x ∈ S ⟹ 0 ≤ f x + normf f
     using normf_upper [OF f] by force
 qed (use e in auto)
 then obtain g where g ∈ R ∀ x∈S. |g x - (f x + normf f)| < e
```

```
    by force
  then show ?thesis
    by (rule_tac x=λx. g x − normf f in bexI) (auto simp: algebra_simps intro:
diff const)
qed


theorem (in function_ring_on) Stone_Weierstrass:
  assumes f: continuous_on S f
  shows ∃ F∈UNIV → R. LIM n sequentially. F n :> uniformly_on S f
proof −
  define h where h ≡ λn::nat. SOME g. g ∈ R ∧ (∀ x∈S. |f x − g x| < 1 / (1 +
n))
  show ?thesis
  proof
    { fix e::real
      assume e: 0 < e
      then obtain N::nat where N: 0 < N 0 < inverse N inverse N < e
        by (auto simp: real_arch_inverse [of e])
      { fix n :: nat and x :: 'a and g :: 'a ⇒ real
        assume n: N ≤ n  ∀ x∈S. |f x − g x| < 1 / (1 + real n)
        assume x: x ∈ S
        have ¬ real (Suc n) < inverse e
          using ‹N ≤ n› N using less_imp_inverse_less by force
        then have 1 / (1 + real n) ≤ e
          using e by (simp add: field_simps)
        then have |f x − g x| < e
          using n(2) x by auto
      }
      then have ∀ F n in sequentially. ∀ x∈S. |f x − h n x| < e
        unfolding h_def
        by (force intro: someI2_bex [OF Stone_Weierstrass_basic [OF f]] eventu-
ally_sequentiallyI [of N])
    }
    then show uniform_limit S h f sequentially
    unfolding uniform_limit_iff by (auto simp: dist_norm abs_minus_commute)
    show h ∈ UNIV → R
      unfolding h_def by (force intro: someI2_bex [OF Stone_Weierstrass_basic
[OF f]])
  qed
qed
```

A HOL Light formulation

```
corollary Stone_Weierstrass_HOL:
  fixes R :: ('a::t2_space ⇒ real) set and S :: 'a set
  assumes compact S  ⋀c. P(λx. c::real)
          ⋀f. P f ⟹ continuous_on S f
          ⋀f g. P(f) ∧ P(g) ⟹ P(λx. f x + g x)  ⋀f g. P(f) ∧ P(g) ⟹ P(λx. f
x * g x)
```

$\bigwedge x\ y.\ x \in S \wedge y \in S \wedge x \neq y \Longrightarrow \exists f.\ P(f) \wedge f\ x \neq f\ y$
continuous_on S f
$0 < e$
shows $\exists g.\ P(g) \wedge (\forall x \in S.\ |f\ x - g\ x| < e)$
proof −
interpret *PR*: *function_ring_on Collect P*
by *unfold_locales* (*use assms* **in** *auto*)
show *?thesis*
using *PR.Stone_Weierstrass_basic* [*OF* ‹*continuous_on S f*› ‹$0 < e$›]
by *blast*
qed

5.19.4 Polynomial functions

inductive *real_polynomial_function* :: ('*a*::*real_normed_vector* ⇒ *real*) ⇒ *bool*
where
linear: *bounded_linear f* ⟹ *real_polynomial_function f*
| *const*: *real_polynomial_function* (λ*x*. *c*)
| *add*: ⟦*real_polynomial_function f*; *real_polynomial_function g*⟧ ⟹ *real_polynomial_function* (λ*x*. *f x* + *g x*)
| *mult*: ⟦*real_polynomial_function f*; *real_polynomial_function g*⟧ ⟹ *real_polynomial_function* (λ*x*. *f x* ∗ *g x*)

declare *real_polynomial_function.intros* [*intro*]

definition *polynomial_function* :: ('*a*::*real_normed_vector* ⇒ '*b*::*real_normed_vector*) ⇒ *bool*
where
polynomial_function p ≡ (∀*f*. *bounded_linear f* ⟶ *real_polynomial_function* (*f o p*))

lemma *real_polynomial_function_eq*: *real_polynomial_function p* = *polynomial_function p*
unfolding *polynomial_function_def*
proof
assume *real_polynomial_function p*
then show ∀*f*. *bounded_linear f* ⟶ *real_polynomial_function* (*f* ∘ *p*)
proof (*induction p rule*: *real_polynomial_function.induct*)
case (*linear h*) **then show** *?case*
by (*auto simp*: *bounded_linear_compose real_polynomial_function.linear*)
next
case (*const h*) **then show** *?case*
by (*simp add*: *real_polynomial_function.const*)
next
case (*add h*) **then show** *?case*
by (*force simp add*: *bounded_linear_def linear_add real_polynomial_function.add*)
next
case (*mult h*) **then show** *?case*
by (*force simp add*: *real_bounded_linear const real_polynomial_function.mult*)

qed
next
assume [*rule_format*, *OF bounded_linear_ident*]: $\forall f.$ *bounded_linear* $f \longrightarrow$ *real_polynomial_function* $(f \circ p)$
then show *real_polynomial_function p*
by (*simp add*: *o_def*)
qed

lemma *polynomial_function_const* [*iff*]: *polynomial_function* $(\lambda x.\ c)$
by (*simp add*: *polynomial_function_def o_def const*)

lemma *polynomial_function_bounded_linear*:
bounded_linear f $\Longrightarrow$ *polynomial_function f*
by (*simp add*: *polynomial_function_def o_def bounded_linear_compose real_polynomial_function.linear*)

lemma *polynomial_function_id* [*iff*]: *polynomial_function*$(\lambda x.\ x)$
by (*simp add*: *polynomial_function_bounded_linear*)

lemma *polynomial_function_add* [*intro*]:
$[\![$*polynomial_function f*; *polynomial_function g*$]\!] \Longrightarrow$ *polynomial_function* $(\lambda x.$ $f\ x + g\ x)$
by (*auto simp*: *polynomial_function_def bounded_linear_def linear_add real_polynomial_function.add o_def*)

lemma *polynomial_function_mult* [*intro*]:
assumes *f*: *polynomial_function f* **and** *g*: *polynomial_function g*
shows *polynomial_function* $(\lambda x.\ f\ x *_R g\ x)$
proof −
have *real_polynomial_function* $(\lambda x.\ h\ (g\ x))$ **if** *bounded_linear h* **for** *h*
using *g that* **unfolding** *polynomial_function_def o_def bounded_linear_def*
by (*auto simp*: *real_polynomial_function_eq*)
moreover have *real_polynomial_function f*
by (*simp add*: *f real_polynomial_function_eq*)
ultimately show *?thesis*
unfolding *polynomial_function_def bounded_linear_def o_def*
by (*auto simp*: *linear.scaleR*)
qed

lemma *polynomial_function_cmul* [*intro*]:
assumes *f*: *polynomial_function f*
shows *polynomial_function* $(\lambda x.\ c *_R f\ x)$
by (*rule polynomial_function_mult* [*OF polynomial_function_const f*])

lemma *polynomial_function_minus* [*intro*]:
assumes *f*: *polynomial_function f*
shows *polynomial_function* $(\lambda x.\ -\ f\ x)$
using *polynomial_function_cmul* [*OF f*, *of* -1] **by** *simp*

lemma *polynomial_function_diff* [*intro*]:

⟦*polynomial_function f*; *polynomial_function g*⟧ ⟹ *polynomial_function* (λ*x*. *f x* − *g x*)
unfolding *add_uminus_conv_diff* [*symmetric*]
by (*metis polynomial_function_add polynomial_function_minus*)

lemma *polynomial_function_sum* [*intro*]:
⟦*finite I*; ⋀*i*. *i* ∈ *I* ⟹ *polynomial_function* (λ*x*. *f x i*)⟧ ⟹ *polynomial_function* (λ*x*. *sum* (*f x*) *I*)
by (*induct I rule*: *finite_induct*) *auto*

lemma *real_polynomial_function_minus* [*intro*]:
real_polynomial_function f ⟹ *real_polynomial_function* (λ*x*. − *f x*)
using *polynomial_function_minus* [*of f*]
by (*simp add*: *real_polynomial_function_eq*)

lemma *real_polynomial_function_diff* [*intro*]:
⟦*real_polynomial_function f*; *real_polynomial_function g*⟧ ⟹ *real_polynomial_function* (λ*x*. *f x* − *g x*)
using *polynomial_function_diff* [*of f*]
by (*simp add*: *real_polynomial_function_eq*)

lemma *real_polynomial_function_divide* [*intro*]:
assumes *real_polynomial_function p* **shows** *real_polynomial_function* (λ*x*. *p x* / *c*)
proof −
have *real_polynomial_function* (λ*x*. *p x* ∗ *Fields.inverse c*)
using *assms* **by** *auto*
then show *?thesis*
by (*simp add*: *divide_inverse*)
qed

lemma *real_polynomial_function_sum* [*intro*]:
⟦*finite I*; ⋀*i*. *i* ∈ *I* ⟹ *real_polynomial_function* (λ*x*. *f x i*)⟧ ⟹ *real_polynomial_function* (λ*x*. *sum* (*f x*) *I*)
using *polynomial_function_sum* [*of I f*]
by (*simp add*: *real_polynomial_function_eq*)

lemma *real_polynomial_function_prod* [*intro*]:
⟦*finite I*; ⋀*i*. *i* ∈ *I* ⟹ *real_polynomial_function* (λ*x*. *f x i*)⟧ ⟹ *real_polynomial_function* (λ*x*. *prod* (*f x*) *I*)
by (*induct I rule*: *finite_induct*) *auto*

lemma *real_polynomial_function_gchoose*:
obtains *p* **where** *real_polynomial_function p* ⋀*x*. *x gchoose r* = *p x*
proof
show *real_polynomial_function* (λ*x*. (∏ *i* = *0*..<*r*. *x* − *real i*) / *fact r*)
by *force*
qed (*simp add*: *gbinomial_prod_rev*)

lemma *real_polynomial_function_power* [*intro*]:
  real_polynomial_function f ⟹ *real_polynomial_function* (λ*x*. *f x*^*n*)
 by (*induct n*) (*simp_all add*: *const mult*)

lemma *real_polynomial_function_compose* [*intro*]:
 assumes *f*: *polynomial_function f* **and** *g*: *real_polynomial_function g*
  shows *real_polynomial_function* (*g o f*)
 using *g*
proof (*induction g rule*: *real_polynomial_function.induct*)
 case (*linear f*)
 then show *?case*
  using *f polynomial_function_def* **by** *blast*
next
 case (*add f g*)
 then show *?case*
  using *f add* **by** (*auto simp*: *polynomial_function_def*)
next
 case (*mult f g*)
 then show *?case*
 using *f mult* **by** (*auto simp*: *polynomial_function_def*)
qed *auto*

lemma *polynomial_function_compose* [*intro*]:
 assumes *f*: *polynomial_function f* **and** *g*: *polynomial_function g*
  shows *polynomial_function* (*g o f*)
 using *g real_polynomial_function_compose* [*OF f*]
 by (*auto simp*: *polynomial_function_def o_def*)

lemma *sum_max_0*:
 fixes *x*::*real*
 shows (∑ *i*≤*max m n*. *x*^*i* ∗ (*if i* ≤ *m then a i else 0*)) = (∑ *i*≤*m*. *x*^*i* ∗ *a i*)
proof −
 have (∑ *i*≤*max m n*. *x*^*i* ∗ (*if i* ≤ *m then a i else 0*)) = (∑ *i*≤*max m n*. (*if i* ≤ *m then x*^*i* ∗ *a i else 0*))
  by (*auto simp*: *algebra_simps intro*: *sum.cong*)
 also have ... = (∑ *i*≤*m*. (*if i* ≤ *m then x*^*i* ∗ *a i else 0*))
  by (*rule sum.mono_neutral_right*) *auto*
 also have ... = (∑ *i*≤*m*. *x*^*i* ∗ *a i*)
  by (*auto simp*: *algebra_simps intro*: *sum.cong*)
 finally show *?thesis* **.**
qed

lemma *real_polynomial_function_imp_sum*:
 assumes *real_polynomial_function f*
  shows ∃ *a n*::*nat*. *f* = (λ*x*. ∑ *i*≤*n*. *a i* ∗ *x*^*i*)
using *assms*
proof (*induct f*)
 case (*linear f*)
 then obtain *c* **where** *f*: *f* = (λ*x*. *x* ∗ *c*)

```
    by (auto simp add: real_bounded_linear)
  have x * c = (∑ i≤1. (if i = 0 then 0 else c) * x^i) for x
    by (simp add: mult_ac)
  with f show ?case
    by fastforce
next
  case (const c)
  have c = (∑ i≤0. c * x^i) for x
    by auto
  then show ?case
    by fastforce
  case (add f1 f2)
  then obtain a1 n1 a2 n2 where
    f1 = (λx. ∑ i≤n1. a1 i * x^i) f2 = (λx. ∑ i≤n2. a2 i * x^i)
    by auto
  then have f1 x + f2 x = (∑ i≤max n1 n2. ((if i ≤ n1 then a1 i else 0) + (if i
≤ n2 then a2 i else 0)) * x^i)
      for x
     using sum_max_0 [where m=n1 and n=n2] sum_max_0 [where m=n2
and n=n1]
    by (simp add: sum.distrib algebra_simps max.commute)
  then show ?case
    by force
  case (mult f1 f2)
  then obtain a1 n1 a2 n2 where
    f1 = (λx. ∑ i≤n1. a1 i * x^i) f2 = (λx. ∑ i≤n2. a2 i * x^i)
    by auto
  then obtain b1 b2 where
    f1 = (λx. ∑ i≤n1. b1 i * x^i) f2 = (λx. ∑ i≤n2. b2 i * x^i)
    b1 = (λi. if i≤n1 then a1 i else 0) b2 = (λi. if i≤n2 then a2 i else 0)
    by auto
  then have f1 x * f2 x = (∑ i≤n1 + n2. (∑ k≤i. b1 k * b2 (i − k)) * x ^ i)
for x
   using polynomial_product [of n1 b1 n2 b2] by (simp add: Set_Interval.atLeast0AtMost)
  then show ?case
    by force
qed

lemma real_polynomial_function_iff_sum:
     real_polynomial_function f ⟷ (∃ a n. f = (λx. ∑ i≤n. a i * x^i))  (is ?lhs
= ?rhs)
proof
  assume ?lhs then show ?rhs
    by (metis real_polynomial_function_imp_sum)
next
  assume ?rhs then show ?lhs
   by (auto simp: linear mult const real_polynomial_function_power real_polynomial_function_sum)
qed
```

lemma *polynomial_function_iff_Basis_inner*:
fixes *f* :: *'a*::*real_normed_vector* ⇒ *'b*::*euclidean_space*
shows *polynomial_function f* ⟷ (∀ *b*∈*Basis*. *real_polynomial_function* (λ*x*. *inner* (*f x*) *b*))
(**is** *?lhs* = *?rhs*)
unfolding *polynomial_function_def*
proof (*intro iffI allI impI*)
assume ∀ *h*. *bounded_linear h* ⟶ *real_polynomial_function* (*h* ∘ *f*)
then show *?rhs*
by (*force simp add*: *bounded_linear_inner_left o_def*)
next
fix *h* :: *'b* ⇒ *real*
assume *rp*: ∀ *b*∈*Basis*. *real_polynomial_function* (λ*x*. *f x* ⋅ *b*) **and** *h*: *bounded_linear h*
have *real_polynomial_function* (*h* ∘ (λ*x*. ∑ *b*∈*Basis*. (*f x* ⋅ *b*) ∗*R* *b*))
using *rp*
by (*force simp*: *real_polynomial_function_eq polynomial_function_mult*
intro!: *real_polynomial_function_compose* [*OF* _ *linear* [*OF h*]])
then show *real_polynomial_function* (*h* ∘ *f*)
by (*simp add*: *euclidean_representation_sum_fun*)
qed

5.19.5 Stone-Weierstrass theorem for polynomial functions

First, we need to show that they are continuous, differentiable and separable.

lemma *continuous_real_polymonial_function*:
assumes *real_polynomial_function f*
shows *continuous* (*at x*) *f*
using *assms*
by (*induct f*) (*auto simp*: *linear_continuous_at*)

lemma *continuous_polymonial_function*:
fixes *f* :: *'a*::*real_normed_vector* ⇒ *'b*::*euclidean_space*
assumes *polynomial_function f*
shows *continuous* (*at x*) *f*
proof (*rule euclidean_isCont*)
show ⋀*b*. *b* ∈ *Basis* ⟹ *isCont* (λ*x*. (*f x* ⋅ *b*) ∗*R* *b*) *x*
using *assms continuous_real_polymonial_function*
by (*force simp*: *polynomial_function_iff_Basis_inner intro*: *isCont_scaleR*)
qed

lemma *continuous_on_polymonial_function*:
fixes *f* :: *'a*::*real_normed_vector* ⇒ *'b*::*euclidean_space*
assumes *polynomial_function f*
shows *continuous_on S f*
using *continuous_polymonial_function* [*OF assms*] *continuous_at_imp_continuous_on*
by *blast*

lemma *has_real_derivative_polynomial_function*:

```
  assumes real_polynomial_function p
    shows ∃p'. real_polynomial_function p' ∧
                (∀x. (p has_real_derivative (p' x)) (at x))
using assms
proof (induct p)
  case (linear p)
  then show ?case
    by (force simp: real_bounded_linear const intro!: derivative_eq_intros)
next
  case (const c)
  show ?case
    by (rule_tac x=λx. 0 in exI) auto
  case (add f1 f2)
  then obtain p1 p2 where
    real_polynomial_function p1 ⋀x. (f1 has_real_derivative p1 x) (at x)
    real_polynomial_function p2 ⋀x. (f2 has_real_derivative p2 x) (at x)
    by auto
  then show ?case
    by (rule_tac x=λx. p1 x + p2 x in exI) (auto intro!: derivative_eq_intros)
  case (mult f1 f2)
  then obtain p1 p2 where
    real_polynomial_function p1 ⋀x. (f1 has_real_derivative p1 x) (at x)
    real_polynomial_function p2 ⋀x. (f2 has_real_derivative p2 x) (at x)
    by auto
  then show ?case
    using mult
     by (rule_tac x=λx. f1 x * p2 x + f2 x * p1 x in exI) (auto intro!: deriva-
tive_eq_intros)
qed

lemma has_vector_derivative_polynomial_function:
  fixes p :: real ⇒ 'a::euclidean_space
  assumes polynomial_function p
  obtains p' where polynomial_function p' ⋀x. (p has_vector_derivative (p' x))
(at x)
proof −
  { fix b :: 'a
    assume b ∈ Basis
    then
    obtain p' where p': real_polynomial_function p' and pd: ⋀x. ((λx. p x • b)
has_real_derivative p' x) (at x)
     using assms [unfolded polynomial_function_iff_Basis_inner] has_real_derivative_polynomial_func
      by blast
    have polynomial_function (λx. p' x *R b)
      using ‹b ∈ Basis› p' const [where 'a=real and c=0]
      by (simp add: polynomial_function_iff_Basis_inner inner_Basis)
   then have ∃q. polynomial_function q ∧ (∀x. ((λu. (p u • b) *R b) has_vector_derivative
q x) (at x))
      by (fastforce intro: derivative_eq_intros pd)
```

```
  }
  then obtain qf where qf:
      ⋀b. b ∈ Basis ⟹ polynomial_function (qf b)
      ⋀b x. b ∈ Basis ⟹ ((λu. (p u • b) *R b) has_vector_derivative qf b x) (at x)
    by metis
  show ?thesis
  proof
    show ⋀x. (p has_vector_derivative (∑ b∈Basis. qf b x)) (at x)
      apply (subst euclidean_representation_sum_fun [of p, symmetric])
      by (auto intro: has_vector_derivative_sum qf)
  qed (force intro: qf)
qed

lemma real_polynomial_function_separable:
  fixes x :: 'a::euclidean_space
  assumes x ≠ y shows ∃f. real_polynomial_function f ∧ f x ≠ f y
proof −
  have real_polynomial_function (λu. ∑ b∈Basis. (inner (x−u) b)^2)
  proof (rule real_polynomial_function_sum)
    show ⋀i. i ∈ Basis ⟹ real_polynomial_function (λu. ((x − u) • i)^2)
      by (auto simp: algebra_simps real_polynomial_function_diff const linear
bounded_linear_inner_left)
  qed auto
  moreover have (∑ b∈Basis. ((x − y) • b)^2) ≠ 0
   using assms by (force simp add: euclidean_eq_iff [of x y] sum_nonneg_eq_0_iff
algebra_simps)
  ultimately show ?thesis
    by auto
qed

lemma Stone_Weierstrass_real_polynomial_function:
  fixes f :: 'a::euclidean_space ⇒ real
  assumes compact S continuous_on S f 0 < e
  obtains g where real_polynomial_function g ⋀x. x ∈ S ⟹ |f x − g x| < e
proof −
  interpret PR: function_ring_on Collect real_polynomial_function
  proof unfold_locales
 qed (use assms continuous_on_polymonial_function real_polynomial_function_eq

      in ‹auto intro: real_polynomial_function_separable›)
  show ?thesis
    using PR.Stone_Weierstrass_basic [OF ‹continuous_on S f› ‹0 < e›] that by
blast
qed

theorem Stone_Weierstrass_polynomial_function:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes S: compact S
      and f: continuous_on S f
```

```
    and e: 0 < e
  shows ∃ g. polynomial_function g ∧ (∀ x ∈ S. norm(f x − g x) < e)
proof −
 { fix b :: 'b
   assume b ∈ Basis
  have ∃ p. real_polynomial_function p ∧ (∀ x ∈ S. |f x • b − p x| < e / DIM('b))
   proof (rule Stone_Weierstrass_real_polynomial_function [OF S _, of λx. f x
• b e / card Basis])
     show continuous_on S (λx. f x • b)
       using f by (auto intro: continuous_intros)
   qed (use e in auto)
 }
 then obtain pf where pf:
     ⋀b. b ∈ Basis ⟹ real_polynomial_function (pf b) ∧ (∀ x ∈ S. |f x • b − pf
b x| < e / DIM('b))
   by metis
 let ?g = λx. ∑ b∈Basis. pf b x *R b
 { fix x
   assume x ∈ S
   have norm (∑ b∈Basis. (f x • b) *R b − pf b x *R b) ≤ (∑ b∈Basis. norm ((f
x • b) *R b − pf b x *R b))
     by (rule norm_sum)
   also have ... < of_nat DIM('b) * (e / DIM('b))
   proof (rule sum_bounded_above_strict)
    show ⋀i. i ∈ Basis ⟹ norm ((f x • i) *R i − pf i x *R i) < e / real DIM('b)
      by (simp add: Real_Vector_Spaces.scaleR_diff_left [symmetric] pf ‹x ∈ S›)
   qed (rule DIM_positive)
   also have ... = e
     by (simp add: field_simps)
   finally have norm (∑ b∈Basis. (f x • b) *R b − pf b x *R b) < e .
 }
 then have ∀ x∈S. norm ((∑ b∈Basis. (f x • b) *R b) − ?g x) < e
   by (auto simp flip: sum_subtractf)
 moreover
 have polynomial_function ?g
   using pf by (simp add: polynomial_function_sum polynomial_function_mult
real_polynomial_function_eq)
 ultimately show ?thesis
   using euclidean_representation_sum_fun [of f] by (metis (no_types, lifting))
qed

proposition Stone_Weierstrass_uniform_limit:
 fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
 assumes S: compact S
   and f: continuous_on S f
 obtains g where uniform_limit S g f sequentially ⋀n. polynomial_function (g
n)
proof −
 have pos: inverse (Suc n) > 0 for n by auto
```

obtain *g* **where** *g*: $\bigwedge n.$ *polynomial_function* (*g n*) $\bigwedge x\ n.\ x \in S \Longrightarrow$ *norm*(*f x* − *g n x*) < *inverse* (*Suc n*)
using *Stone_Weierstrass_polynomial_function*[*OF S f pos*]
by *metis*
have *uniform_limit S g f sequentially*
proof (*rule uniform_limitI*)
fix *e*::*real* **assume** *0* < *e*
with *LIMSEQ_inverse_real_of_nat* **have** $\forall_F\ n$ *in sequentially*. *inverse* (*Suc n*) < *e*
by (*rule order_tendstoD*)
moreover have $\forall_F\ n$ *in sequentially*. $\forall x{\in}S$. *dist* (*g n x*) (*f x*) < *inverse* (*Suc n*)
using *g* **by** (*simp add*: *dist_norm norm_minus_commute*)
ultimately show $\forall_F\ n$ *in sequentially*. $\forall x{\in}S$. *dist* (*g n x*) (*f x*) < *e*
by (*eventually_elim*) *auto*
qed
then show *?thesis* **using** *g*(*1*) **..**
qed

5.19.6 Polynomial functions as paths

One application is to pick a smooth approximation to a path, or just pick a smooth path anyway in an open connected set

lemma *path_polynomial_function*:
fixes *g* :: *real* ⇒ *'b*::*euclidean_space*
shows *polynomial_function g* ⟹ *path g*
by (*simp add*: *path_def continuous_on_polymonial_function*)

lemma *path_approx_polynomial_function*:
fixes *g* :: *real* ⇒ *'b*::*euclidean_space*
assumes *path g 0* < *e*
obtains *p* **where** *polynomial_function p pathstart p* = *pathstart g pathfinish p* = *pathfinish g*
$\bigwedge t.\ t \in \{0..1\} \Longrightarrow$ *norm*(*p t* − *g t*) < *e*
proof −
obtain *q* **where** *poq*: *polynomial_function q* **and** *noq*: $\bigwedge x.\ x \in \{0..1\} \Longrightarrow$ *norm* (*g x* − *q x*) < *e*/*4*
using *Stone_Weierstrass_polynomial_function* [*of* {*0..1*} *g e*/*4*] *assms*
by (*auto simp*: *path_def*)
define *pf* **where** *pf* ≡ λ*t*. *q t* + (*g 0* − *q 0*) + *t* ∗$_R$ (*g 1* − *q 1* − (*g 0* − *q 0*))
show *thesis*
proof
show *polynomial_function pf*
by (*force simp add*: *poq pf_def*)
show *norm* (*pf t* − *g t*) < *e*
if *t* ∈ {*0..1*} **for** *t*
proof −
have ∗: *norm* (((*q t* − *g t*) + (*g 0* − *q 0*)) + (*t* ∗$_R$ (*g 1* − *q 1*) + *t* ∗$_R$ (*q 0* − *g 0*))) < (*e*/*4* + *e*/*4*) + (*e*/*4*+*e*/*4*)

```
      proof (intro Real_Vector_Spaces.norm_add_less)
        show norm (q t − g t) < e / 4
          by (metis noq norm_minus_commute that)
        show norm (t *R (g 1 − q 1)) < e / 4
          using noq that le_less_trans [OF mult_left_le_one_le noq]
          by auto
        show norm (t *R (q 0 − g 0)) < e / 4
          using noq that le_less_trans [OF mult_left_le_one_le noq]
          by simp (metis norm_minus_commute order_refl zero_le_one)
      qed (use noq norm_minus_commute that in auto)
      then show ?thesis
        by (auto simp add: algebra_simps pf_def)
    qed
  qed (auto simp add: path_defs pf_def)
qed

proposition connected_open_polynomial_connected:
  fixes S :: 'a::euclidean_space set
  assumes S: open S connected S
      and x ∈ S y ∈ S
    shows ∃g. polynomial_function g ∧ path_image g ⊆ S ∧ pathstart g = x ∧
pathfinish g = y
proof −
  have path_connected S using assms
    by (simp add: connected_open_path_connected)
  with ‹x ∈ S› ‹y ∈ S› obtain p where p: path p path_image p ⊆ S pathstart p
= x pathfinish p = y
    by (force simp: path_connected_def)
  have ∃e. 0 < e ∧ (∀x ∈ path_image p. ball x e ⊆ S)
  proof (cases S = UNIV)
    case True then show ?thesis
      by (simp add: gt_ex)
  next
    case False
    show ?thesis
    proof (intro exI conjI ballI)
      show ⋀x. x ∈ path_image p ⟹ ball x (setdist (path_image p) (−S)) ⊆ S
        using setdist_le_dist [of _ path_image p _ −S] by fastforce
      show 0 < setdist (path_image p) (− S)
        using S p False
       by (fastforce simp add: setdist_gt_0_compact_closed compact_path_image
open_closed)
    qed
  qed
  then obtain e where 0 < eand eb: ⋀x. x ∈ path_image p ⟹ ball x e ⊆ S
    by auto
  obtain pf where polynomial_function pf and pf: pathstart pf = pathstart p
pathfinish pf = pathfinish p
                  and pf_e: ⋀t. t ∈ {0..1} ⟹ norm(pf t − p t) < e
```

```
    using path_approx_polynomial_function [OF ‹path p› ‹0 < e›] by blast
  show ?thesis
  proof (intro exI conjI)
    show polynomial_function pf
      by fact
    show pathstart pf = x pathfinish pf = y
      by (simp_all add: p pf)
    show path_image pf ⊆ S
      unfolding path_image_def
    proof clarsimp
      fix x'::real
      assume 0 ≤ x' x' ≤ 1
      then have dist (p x') (pf x') < e
        by (metis atLeastAtMost_iff dist_commute dist_norm pf_e)
      then show pf x' ∈ S
          by (metis ‹0 ≤ x'› ‹x' ≤ 1› atLeastAtMost_iff eb imageI mem_ball
path_image_def subset_iff)
    qed
  qed
qed

lemma differentiable_componentwise_within:
   f differentiable (at a within S) ⟷
    (∀ i ∈ Basis. (λx. f x • i) differentiable at a within S)
proof −
  { assume ∀ i∈Basis. ∃ D. ((λx. f x • i) has_derivative D) (at a within S)
    then obtain f' where f':
          ⋀i. i ∈ Basis ⟹ ((λx. f x • i) has_derivative f' i) (at a within S)
      by metis
    have eq: (λx. (∑ j∈Basis. f' j x *R j) • i) = f' i if i ∈ Basis for i
      using that by (simp add: inner_add_left inner_add_right)
    have ∃ D. ∀ i∈Basis. ((λx. f x • i) has_derivative (λx. D x • i)) (at a within S)
      apply (rule_tac x=λx::'a. (∑ j∈Basis. f' j x *R j) :: 'b in exI)
      apply (simp add: eq f')
      done
  }
  then show ?thesis
    apply (simp add: differentiable_def)
    using has_derivative_componentwise_within
    by blast
qed

lemma polynomial_function_inner [intro]:
  fixes i :: 'a::euclidean_space
  shows polynomial_function g ⟹ polynomial_function (λx. g x • i)
  apply (subst euclidean_representation [where x=i, symmetric])
  apply (force simp: inner_sum_right polynomial_function_iff_Basis_inner poly-
nomial_function_sum)
  done
```

Differentiability of real and vector polynomial functions.

lemma *differentiable_at_real_polynomial_function*:
  real_polynomial_function f $\Longrightarrow$ *f differentiable* (*at a within S*)
 by (*induction f rule*: *real_polynomial_function.induct*)
   (*simp_all add*: *bounded_linear_imp_differentiable*)

lemma *differentiable_on_real_polynomial_function*:
  real_polynomial_function p $\Longrightarrow$ *p differentiable_on S*
by (*simp add*: *differentiable_at_imp_differentiable_on differentiable_at_real_polynomial_function*)

lemma *differentiable_at_polynomial_function*:
 fixes *f* :: _ $\Rightarrow$ *'a*::*euclidean_space*
 shows *polynomial_function f* $\Longrightarrow$ *f differentiable* (*at a within S*)
 by (*metis differentiable_at_real_polynomial_function polynomial_function_iff_Basis_inner differentiable_componentwise_within*)

lemma *differentiable_on_polynomial_function*:
 fixes *f* :: _ $\Rightarrow$ *'a*::*euclidean_space*
 shows *polynomial_function f* $\Longrightarrow$ *f differentiable_on S*
by (*simp add*: *differentiable_at_polynomial_function differentiable_on_def*)

lemma *vector_eq_dot_span*:
 assumes *x* $\in$ *span B y* $\in$ *span B* **and** *i*: $\bigwedge$*i*. *i* $\in$ *B* $\Longrightarrow$ *i* $\cdot$ *x* = *i* $\cdot$ *y*
 shows *x* = *y*
proof −
 have $\bigwedge$*i*. *i* $\in$ *B* $\Longrightarrow$ *orthogonal* (*x* − *y*) *i*
   by (*simp add*: *i inner_commute inner_diff_right orthogonal_def*)
 moreover have *x* − *y* $\in$ *span B*
   by (*simp add*: *assms span_diff*)
 ultimately have *x* − *y* = *0*
   using *orthogonal_to_span orthogonal_self* **by** *blast*
   then show *?thesis* **by** *simp*
qed

lemma *orthonormal_basis_expand*:
 assumes *B*: *pairwise orthogonal B*
     and *1*: $\bigwedge$*i*. *i* $\in$ *B* $\Longrightarrow$ *norm i* = *1*
     and *x* $\in$ *span B*
     and *finite B*
   shows ($\sum$ *i*$\in$*B*. (*x* $\cdot$ *i*) $*_R$ *i*) = *x*
proof (*rule vector_eq_dot_span* [*OF* _ ‹*x* $\in$ *span B*›])
 show ($\sum$ *i*$\in$*B*. (*x* $\cdot$ *i*) $*_R$ *i*) $\in$ *span B*
   by (*simp add*: *span_clauses span_sum*)
 show *i* $\cdot$ ($\sum$ *i*$\in$*B*. (*x* $\cdot$ *i*) $*_R$ *i*) = *i* $\cdot$ *x* **if** *i* $\in$ *B* **for** *i*
 proof −
   have [*simp*]: *i* $\cdot$ *j* = (*if j* = *i then 1 else 0*) **if** *j* $\in$ *B* **for** *j*
     using *B 1 that* ‹*i* $\in$ *B*›
     by (*force simp*: *norm_eq_1 orthogonal_def pairwise_def*)
   have *i* $\cdot$ ($\sum$ *i*$\in$*B*. (*x* $\cdot$ *i*) $*_R$ *i*) = ($\sum$ *j*$\in$*B*. *x* $\cdot$ *j* $*$ (*i* $\cdot$ *j*))

```
    by (simp add: inner_sum_right)
  also have ... = (∑j∈B. if j = i then x • i else 0)
    by (rule sum.cong; simp)
  also have ... = i • x
    by (simp add: ‹finite B› that inner_commute)
  finally show ?thesis .
 qed
qed


theorem Stone_Weierstrass_polynomial_function_subspace:
 fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
 assumes compact S
    and contf: continuous_on S f
    and 0 < e
    and subspace T f ‘ S ⊆ T
  obtains g where polynomial_function g g ‘ S ⊆ T
                  ⋀x. x ∈ S ⟹ norm(f x − g x) < e
proof −
 obtain B where B ⊆ T and orthB: pairwise orthogonal B
           and B1: ⋀x. x ∈ B ⟹ norm x = 1
           and independent B and cardB: card B = dim T
           and spanB: span B = T
  using orthonormal_basis_subspace ‹subspace T› by metis
 then have finite B
  by (simp add: independent_imp_finite)
 then obtain n::nat and b where B = b ‘ {i. i < n} inj_on b {i. i < n}
  using finite_imp_nat_seg_image_inj_on by metis
 with cardB have n = card B dim T = n
  by (auto simp: card_image)
 have fx: (∑ i∈B. (f x • i) *R i) = f x if x ∈ S for x
   by (metis (no_types, lifting) B1 ‹finite B› assms(5) image_subset_iff orthB
orthonormal_basis_expand spanB sum.cong that)
 have cont: continuous_on S (λx. ∑ i∈B. (f x • i) *R i)
  by (intro continuous_intros contf)
 obtain g where polynomial_function g
         and g: ⋀x. x ∈ S ⟹ norm ((∑ i∈B. (f x • i) *R i) − g x) < e / (n+2)
   using Stone_Weierstrass_polynomial_function [OF ‹compact S› cont, of e /
real (n + 2)] ‹0 < e›
  by auto
 with fx have g: ⋀x. x ∈ S ⟹ norm (f x − g x) < e / (n+2)
  by auto
 show ?thesis
 proof
  show polynomial_function (λx. ∑ i∈B. (g x • i) *R i)
   using ‹polynomial_function g› by (force intro: ‹finite B›)
  show (λx. ∑ i∈B. (g x • i) *R i) ‘ S ⊆ T
   using ‹B ⊆ T›
   by (blast intro: subspace_sum subspace_mul ‹subspace T›)
```

```
    show norm (f x − (∑ i∈B. (g x · i) *R i)) < e if x ∈ S for x
    proof −
      have orth': pairwise (λi j. orthogonal ((f x · i) *R i − (g x · i) *R i)
                                             ((f x · j) *R j − (g x · j) *R j)) B
        by (auto simp: orthogonal_def inner_diff_right inner_diff_left intro: pair-
wise_mono [OF orthB])
      then have (norm (∑ i∈B. (f x · i) *R i − (g x · i) *R i))² =
                 (∑ i∈B. (norm ((f x · i) *R i − (g x · i) *R i))²)
        by (simp add:  norm_sum_Pythagorean [OF ‹finite B› orth'])
      also have ... = (∑ i∈B. (norm (((f x − g x) · i) *R i))²)
        by (simp add: algebra_simps)
      also have ... ≤ (∑ i∈B. (norm (f x − g x))²)
      proof −
        have ⋀i. i ∈ B ⟹ ((f x − g x) · i)² ≤ (norm (f x − g x))²
          by (metis B1 Cauchy_Schwarz_ineq inner_commute mult.left_neutral
norm_eq_1 power2_norm_eq_inner)
        then show ?thesis
          by (intro sum_mono) (simp add: sum_mono B1)
      qed
      also have ... = n * norm (f x − g x)^2
        by (simp add: ‹n = card B›)
      also have ... ≤ n * (e / (n+2))^2
      proof (rule mult_left_mono)
        show (norm (f x − g x))² ≤ (e / real (n + 2))²
        by (meson dual_order.order_iff_strict g norm_ge_zero power_mono that)
      qed auto
      also have ... ≤ e^2 / (n+2)
        using ‹0 < e› by (simp add: divide_simps power2_eq_square)
      also have ... < e^2
        using ‹0 < e› by (simp add: divide_simps)
      finally have (norm (∑ i∈B. (f x · i) *R i − (g x · i) *R i))² < e^2 .
      then have (norm (∑ i∈B. (f x · i) *R i − (g x · i) *R i)) < e
        by (simp add: ‹0 < e› norm_lt_square power2_norm_eq_inner)
      then show ?thesis
        using fx that by (simp add: sum_subtractf)
    qed
  qed
qed


hide_fact linear add mult const

end
```

Chapter 6

Measure and Integration Theory

theory *Sigma_Algebra*
imports
Complex_Main
HOL−Library.Countable_Set
HOL−Library.FuncSet
HOL−Library.Indicator_Function
HOL−Library.Extended_Nonnegative_Real
HOL−Library.Disjoint_Sets
begin

6.1 Sigma Algebra

Sigma algebras are an elementary concept in measure theory. To measure — that is to integrate — functions, we first have to measure sets. Unfortunately, when dealing with a large universe, it is often not possible to consistently assign a measure to every subset. Therefore it is necessary to define the set of measurable subsets of the universe. A sigma algebra is such a set that has three very natural and desirable properties.

6.1.1 Families of sets

locale *subset_class* =
fixes Ω :: *'a set* **and** M :: *'a set set*
assumes *space_closed*: $M \subseteq Pow\ \Omega$

lemma (**in** *subset_class*) *sets_into_space*: $x \in M \Longrightarrow x \subseteq \Omega$
by (*metis PowD contra_subsetD space_closed*)

Semiring of sets

locale *semiring_of_sets* = *subset_class* +

assumes *empty_sets*[*iff*]: $\{\} \in M$
assumes *Int*[*intro*]: $\bigwedge a\ b.\ a \in M \Longrightarrow b \in M \Longrightarrow a \cap b \in M$
assumes *Diff_cover*:
$\bigwedge a\ b.\ a \in M \Longrightarrow b \in M \Longrightarrow \exists\, C \subseteq M.\ \textit{finite}\ C \wedge \textit{disjoint}\ C \wedge a - b = \bigcup C$

lemma (**in** *semiring_of_sets*) *finite_INT*[*intro*]:
assumes *finite* I $I \neq \{\}$ $\bigwedge i.\ i \in I \Longrightarrow A\ i \in M$
shows $(\bigcap i \in I.\ A\ i) \in M$
using *assms* **by** (*induct rule*: *finite_ne_induct*) *auto*

lemma (**in** *semiring_of_sets*) *Int_space_eq1* [*simp*]: $x \in M \Longrightarrow \Omega \cap x = x$
by (*metis Int_absorb1 sets_into_space*)

lemma (**in** *semiring_of_sets*) *Int_space_eq2* [*simp*]: $x \in M \Longrightarrow x \cap \Omega = x$
by (*metis Int_absorb2 sets_into_space*)

lemma (**in** *semiring_of_sets*) *sets_Collect_conj*:
assumes $\{x \in \Omega.\ P\ x\} \in M$ $\{x \in \Omega.\ Q\ x\} \in M$
shows $\{x \in \Omega.\ Q\ x \wedge P\ x\} \in M$
proof −
have $\{x \in \Omega.\ Q\ x \wedge P\ x\} = \{x \in \Omega.\ Q\ x\} \cap \{x \in \Omega.\ P\ x\}$
by *auto*
with *assms* **show** *?thesis* **by** *auto*
qed

lemma (**in** *semiring_of_sets*) *sets_Collect_finite_All′*:
assumes $\bigwedge i.\ i \in S \Longrightarrow \{x \in \Omega.\ P\ i\ x\} \in M$ *finite* S $S \neq \{\}$
shows $\{x \in \Omega.\ \forall\, i \in S.\ P\ i\ x\} \in M$
proof −
have $\{x \in \Omega.\ \forall\, i \in S.\ P\ i\ x\} = (\bigcap i \in S.\ \{x \in \Omega.\ P\ i\ x\})$
using ‹$S \neq \{\}$› **by** *auto*
with *assms* **show** *?thesis* **by** *auto*
qed

Ring of sets

locale *ring_of_sets* = *semiring_of_sets* +
assumes *Un* [*intro*]: $\bigwedge a\ b.\ a \in M \Longrightarrow b \in M \Longrightarrow a \cup b \in M$

lemma (**in** *ring_of_sets*) *finite_Union* [*intro*]:
finite $X \Longrightarrow X \subseteq M \Longrightarrow \bigcup X \in M$
by (*induct set*: *finite*) (*auto simp add*: *Un*)

lemma (**in** *ring_of_sets*) *finite_UN*[*intro*]:
assumes *finite* I **and** $\bigwedge i.\ i \in I \Longrightarrow A\ i \in M$
shows $(\bigcup i \in I.\ A\ i) \in M$
using *assms* **by** *induct auto*

lemma (**in** *ring_of_sets*) *Diff* [*intro*]:

assumes $a \in M$ $b \in M$ **shows** $a - b \in M$
using *Diff_cover*[*OF assms*] **by** *auto*

lemma *ring_of_setsI*:
assumes *space_closed*: $M \subseteq Pow\ \Omega$
assumes *empty_sets*[*iff*]: $\{\} \in M$
assumes *Un*[*intro*]: $\bigwedge a\ b.\ a \in M \Longrightarrow b \in M \Longrightarrow a \cup b \in M$
assumes *Diff*[*intro*]: $\bigwedge a\ b.\ a \in M \Longrightarrow b \in M \Longrightarrow a - b \in M$
shows *ring_of_sets* $\Omega\ M$
proof
fix $a\ b$ **assume** *ab*: $a \in M\ b \in M$
from *ab* **show** $\exists\, C \subseteq M.\ finite\ C \wedge disjoint\ C \wedge a - b = \bigcup C$
by (*intro exI*[*of* _ $\{a - b\}$]) (*auto simp*: *disjoint_def*)
have $a \cap b = a - (a - b)$ **by** *auto*
also have $\ldots \in M$ **using** *ab* **by** *auto*
finally show $a \cap b \in M$ **.**
qed *fact*+

lemma *ring_of_sets_iff*: *ring_of_sets* $\Omega\ M \longleftrightarrow M \subseteq Pow\ \Omega \wedge \{\} \in M \wedge (\forall\, a{\in}M.\ \forall\, b{\in}M.\ a \cup b \in M) \wedge (\forall\, a{\in}M.\ \forall\, b{\in}M.\ a - b \in M)$
proof
assume *ring_of_sets* $\Omega\ M$
then interpret *ring_of_sets* $\Omega\ M$ **.**
show $M \subseteq Pow\ \Omega \wedge \{\} \in M \wedge (\forall\, a{\in}M.\ \forall\, b{\in}M.\ a \cup b \in M) \wedge (\forall\, a{\in}M.\ \forall\, b{\in}M.\ a - b \in M)$
using *space_closed* **by** *auto*
qed (*auto intro*!: *ring_of_setsI*)

lemma (**in** *ring_of_sets*) *insert_in_sets*:
assumes $\{x\} \in M$ $A \in M$ **shows** *insert* $x\ A \in M$
proof −
have $\{x\} \cup A \in M$ **using** *assms* **by** (*rule Un*)
thus *?thesis* **by** *auto*
qed

lemma (**in** *ring_of_sets*) *sets_Collect_disj*:
assumes $\{x{\in}\Omega.\ P\ x\} \in M$ $\{x{\in}\Omega.\ Q\ x\} \in M$
shows $\{x{\in}\Omega.\ Q\ x \vee P\ x\} \in M$
proof −
have $\{x{\in}\Omega.\ Q\ x \vee P\ x\} = \{x{\in}\Omega.\ Q\ x\} \cup \{x{\in}\Omega.\ P\ x\}$
by *auto*
with *assms* **show** *?thesis* **by** *auto*
qed

lemma (**in** *ring_of_sets*) *sets_Collect_finite_Ex*:
assumes $\bigwedge i.\ i \in S \Longrightarrow \{x{\in}\Omega.\ P\ i\ x\} \in M\ finite\ S$
shows $\{x{\in}\Omega.\ \exists\, i{\in}S.\ P\ i\ x\} \in M$
proof −
have $\{x{\in}\Omega.\ \exists\, i{\in}S.\ P\ i\ x\} = (\bigcup i{\in}S.\ \{x{\in}\Omega.\ P\ i\ x\})$

by *auto*
with *assms* **show** *?thesis* **by** *auto*
qed

Algebra of sets

locale *algebra* = *ring_of_sets* +
assumes *top* [*iff*]: $\Omega \in M$

lemma (**in** *algebra*) *compl_sets* [*intro*]:
$a \in M \Longrightarrow \Omega - a \in M$
by *auto*

proposition *algebra_iff_Un*:
algebra Ω $M \longleftrightarrow$
$M \subseteq Pow\ \Omega \wedge$
$\{\} \in M \wedge$
$(\forall a \in M.\ \Omega - a \in M) \wedge$
$(\forall a \in M.\ \forall\ b \in M.\ a \cup b \in M)$ (**is** _ $\longleftrightarrow$ *?Un*)
proof
assume *algebra* Ω M
then interpret *algebra* Ω M **.**
show *?Un* **using** *sets_into_space* **by** *auto*
next
assume *?Un*
then have $\Omega \in M$ **by** *auto*
interpret *ring_of_sets* Ω M
proof (*rule ring_of_setsI*)
show Ω: $M \subseteq Pow\ \Omega\ \{\} \in M$
using ‹*?Un*› **by** *auto*
fix a b **assume** a: $a \in M$ **and** b: $b \in M$
then show $a \cup b \in M$ **using** ‹*?Un*› **by** *auto*
have $a - b = \Omega - ((\Omega - a) \cup b)$
using Ω a b **by** *auto*
then show $a - b \in M$
using a b ‹*?Un*› **by** *auto*
qed
show *algebra* Ω M **proof qed** *fact*
qed

proposition *algebra_iff_Int*:
algebra Ω $M \longleftrightarrow$
$M \subseteq Pow\ \Omega$ & $\{\} \in M$ &
$(\forall a \in M.\ \Omega - a \in M)$ &
$(\forall a \in M.\ \forall\ b \in M.\ a \cap b \in M)$ (**is** _ $\longleftrightarrow$ *?Int*)
proof
assume *algebra* Ω M
then interpret *algebra* Ω M **.**
show *?Int* **using** *sets_into_space* **by** *auto*

next
assume *?Int*
show *algebra* Ω *M*
proof (*unfold algebra_iff_Un*, *intro conjI ballI*)
show Ω: *M* ⊆ *Pow* Ω {} ∈ *M*
using ‹*?Int*› **by** *auto*
from ‹*?Int*› **show** ⋀*a*. *a* ∈ *M* ⟹ Ω − *a* ∈ *M* **by** *auto*
fix *a b* **assume** *M*: *a* ∈ *M b* ∈ *M*
hence *a* ∪ *b* = Ω − ((Ω − *a*) ∩ (Ω − *b*))
using Ω **by** *blast*
also have ... ∈ *M*
using *M* ‹*?Int*› **by** *auto*
finally show *a* ∪ *b* ∈ *M* **.**
qed
qed

lemma (**in** *algebra*) *sets_Collect_neg*:
assumes {*x*∈Ω. *P x*} ∈ *M*
shows {*x*∈Ω. ¬ *P x*} ∈ *M*
proof −
have {*x*∈Ω. ¬ *P x*} = Ω − {*x*∈Ω. *P x*} **by** *auto*
with *assms* **show** *?thesis* **by** *auto*
qed

lemma (**in** *algebra*) *sets_Collect_imp*:
{*x*∈Ω. *P x*} ∈ *M* ⟹ {*x*∈Ω. *Q x*} ∈ *M* ⟹ {*x*∈Ω. *Q x* ⟶ *P x*} ∈ *M*
unfolding *imp_conv_disj* **by** (*intro sets_Collect_disj sets_Collect_neg*)

lemma (**in** *algebra*) *sets_Collect_const*:
{*x*∈Ω. *P*} ∈ *M*
by (*cases P*) *auto*

lemma *algebra_single_set*:
X ⊆ *S* ⟹ *algebra S* { {}, *X*, *S* − *X*, *S* }
by (*auto simp*: *algebra_iff_Int*)

Restricted algebras

abbreviation (**in** *algebra*)
restricted_space A ≡ ((∩) *A*) ‘ *M*

lemma (**in** *algebra*) *restricted_algebra*:
assumes *A* ∈ *M* **shows** *algebra A* (*restricted_space A*)
using *assms* **by** (*auto simp*: *algebra_iff_Int*)

Sigma Algebras

locale *sigma_algebra* = *algebra* +
assumes *countable_nat_UN* [*intro*]: ⋀*A*. *range A* ⊆ *M* ⟹ (⋃ *i*::*nat*. *A i*) ∈ *M*

lemma (**in** *algebra*) *is_sigma_algebra*:
assumes *finite M*
shows *sigma_algebra* Ω *M*
proof
fix *A* :: *nat* ⇒ *'a set* **assume** *range A* ⊆ *M*
then have (⋃ *i*. *A i*) = (⋃ *s*∈*M* ∩ *range A*. *s*)
by *auto*
also have (⋃ *s*∈*M* ∩ *range A*. *s*) ∈ *M*
using ‹*finite M*› **by** *auto*
finally show (⋃ *i*. *A i*) ∈ *M* **.**
qed

lemma *countable_UN_eq*:
fixes *A* :: *'i*::*countable* ⇒ *'a set*
shows (*range A* ⊆ *M* ⟶ (⋃ *i*. *A i*) ∈ *M*) ⟷
(*range* (*A* ∘ *from_nat*) ⊆ *M* ⟶ (⋃ *i*. (*A* ∘ *from_nat*) *i*) ∈ *M*)
proof −
let *?A'* = *A* ∘ *from_nat*
have ∗: (⋃ *i*. *?A'* *i*) = (⋃ *i*. *A i*) (**is** *?l* = *?r*)
proof *safe*
fix *x i* **assume** *x* ∈ *A i* **thus** *x* ∈ *?l*
by (*auto intro*!: *exI*[*of* _ *to_nat i*])
next
fix *x i* **assume** *x* ∈ *?A'* *i* **thus** *x* ∈ *?r*
by (*auto intro*!: *exI*[*of* _ *from_nat i*])
qed
have *A* ‘ *range from_nat* = *range A*
using *surj_from_nat* **by** *simp*
then have ∗∗: *range ?A'* = *range A*
by (*simp only*: *image_comp* [*symmetric*])
show *?thesis* **unfolding** ∗ ∗∗ **..**
qed

lemma (**in** *sigma_algebra*) *countable_Union* [*intro*]:
assumes *countable X X* ⊆ *M* **shows** ⋃ *X* ∈ *M*
proof *cases*
assume *X* ≠ {}
hence ⋃ *X* = (⋃ *n*. *from_nat_into X n*)
using *assms* **by** (*auto cong del*: *SUP_cong*)
also have ... ∈ *M* **using** *assms*
by (*auto intro*!: *countable_nat_UN*) (*metis* ‹*X* ≠ {}› *from_nat_into subsetD*)
finally show *?thesis* **.**
qed *simp*

lemma (**in** *sigma_algebra*) *countable_UN*[*intro*]:
fixes *A* :: *'i*::*countable* ⇒ *'a set*
assumes *A*‘*X* ⊆ *M*
shows (⋃ *x*∈*X*. *A x*) ∈ *M*
proof −

let *?A* = *λi. if i ∈ X then A i else {}*
from *assms* **have** *range ?A ⊆ M* **by** *auto*
with *countable_nat_UN*[*of ?A ∘ from_nat*] *countable_UN_eq*[*of ?A M*]
have *(⋃x. ?A x) ∈ M* **by** *auto*
moreover have *(⋃x. ?A x) = (⋃x∈X. A x)* **by** (*auto split*: *if_split_asm*)
ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *sigma_algebra*) *countable_UN′*:
fixes *A* :: *'i ⇒ 'a set*
assumes *X*: *countable X*
assumes *A*: *A‘X ⊆ M*
shows *(⋃x∈X. A x) ∈ M*
proof −
have *(⋃x∈X. A x) = (⋃i∈to_nat_on X ‘ X. A (from_nat_into X i))*
using *X* **by** *auto*
also have *... ∈ M*
using *A X*
by (*intro countable_UN*) *auto*
finally show *?thesis* **.**
qed

lemma (**in** *sigma_algebra*) *countable_UN″*:
⟦ countable X; *⋀x y. x ∈ X ⟹ A x ∈ M ⟧ ⟹ (⋃x∈X. A x) ∈ M*
by(*erule countable_UN′*)(*auto*)

lemma (**in** *sigma_algebra*) *countable_INT* [*intro*]:
fixes *A* :: *'i*::*countable ⇒ 'a set*
assumes *A*: *A‘X ⊆ M X ≠ {}*
shows *(⋂i∈X. A i) ∈ M*
proof −
from *A* **have** *∀i∈X. A i ∈ M* **by** *fast*
hence *Ω − (⋃i∈X. Ω − A i) ∈ M* **by** *blast*
moreover
have *(⋂i∈X. A i) = Ω − (⋃i∈X. Ω − A i)* **using** *space_closed A*
by *blast*
ultimately show *?thesis* **by** *metis*
qed

lemma (**in** *sigma_algebra*) *countable_INT′*:
fixes *A* :: *'i ⇒ 'a set*
assumes *X*: *countable X X ≠ {}*
assumes *A*: *A‘X ⊆ M*
shows *(⋂x∈X. A x) ∈ M*
proof −
have *(⋂x∈X. A x) = (⋂i∈to_nat_on X ‘ X. A (from_nat_into X i))*
using *X* **by** *auto*
also have *... ∈ M*
using *A X*

by (*intro countable_INT*) *auto*
finally show *?thesis* **.**
qed

lemma (**in** *sigma_algebra*) *countable_INT''*:
$UNIV \in M \Longrightarrow countable\ I \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow F\ i \in M) \Longrightarrow (\bigcap i{\in}I.\ F\ i) \in M$
by (*cases* $I = \{\}$) (*auto intro*: *countable_INT'*)

lemma (**in** *sigma_algebra*) *countable*:
assumes $\bigwedge a.\ a \in A \Longrightarrow \{a\} \in M$ *countable* A
shows $A \in M$
proof −
have $(\bigcup a{\in}A.\ \{a\}) \in M$
using *assms* **by** (*intro countable_UN'*) *auto*
also have $(\bigcup a{\in}A.\ \{a\}) = A$ **by** *auto*
finally show *?thesis* **by** *auto*
qed

lemma *ring_of_sets_Pow*: *ring_of_sets sp* (*Pow sp*)
by (*auto simp*: *ring_of_sets_iff*)

lemma *algebra_Pow*: *algebra sp* (*Pow sp*)
by (*auto simp*: *algebra_iff_Un*)

lemma *sigma_algebra_iff*:
sigma_algebra Ω $M \longleftrightarrow$
algebra Ω $M \wedge (\forall A.\ range\ A \subseteq M \longrightarrow (\bigcup i{::}nat.\ A\ i) \in M)$
by (*simp add*: *sigma_algebra_def sigma_algebra_axioms_def*)

lemma *sigma_algebra_Pow*: *sigma_algebra sp* (*Pow sp*)
by (*auto simp*: *sigma_algebra_iff algebra_iff_Int*)

lemma (**in** *sigma_algebra*) *sets_Collect_countable_All*:
assumes $\bigwedge i.\ \{x{\in}\Omega.\ P\ i\ x\} \in M$
shows $\{x{\in}\Omega.\ \forall i{::}'i{::}countable.\ P\ i\ x\} \in M$
proof −
have $\{x{\in}\Omega.\ \forall i{::}'i{::}countable.\ P\ i\ x\} = (\bigcap i.\ \{x{\in}\Omega.\ P\ i\ x\})$ **by** *auto*
with *assms* **show** *?thesis* **by** *auto*
qed

lemma (**in** *sigma_algebra*) *sets_Collect_countable_Ex*:
assumes $\bigwedge i.\ \{x{\in}\Omega.\ P\ i\ x\} \in M$
shows $\{x{\in}\Omega.\ \exists i{::}'i{::}countable.\ P\ i\ x\} \in M$
proof −
have $\{x{\in}\Omega.\ \exists i{::}'i{::}countable.\ P\ i\ x\} = (\bigcup i.\ \{x{\in}\Omega.\ P\ i\ x\})$ **by** *auto*
with *assms* **show** *?thesis* **by** *auto*
qed

lemma (**in** *sigma_algebra*) *sets_Collect_countable_Ex'*:

assumes $\bigwedge i.\ i \in I \Longrightarrow \{x{\in}\Omega.\ P\ i\ x\} \in M$
assumes *countable* I
shows $\{x{\in}\Omega.\ \exists\, i{\in}I.\ P\ i\ x\} \in M$
proof −
have $\{x{\in}\Omega.\ \exists\, i{\in}I.\ P\ i\ x\} = (\bigcup i{\in}I.\ \{x{\in}\Omega.\ P\ i\ x\})$ **by** *auto*
with *assms* **show** *?thesis*
by (*auto intro*!: *countable_UN'*)
qed

lemma (**in** *sigma_algebra*) *sets_Collect_countable_All'*:
assumes $\bigwedge i.\ i \in I \Longrightarrow \{x{\in}\Omega.\ P\ i\ x\} \in M$
assumes *countable* I
shows $\{x{\in}\Omega.\ \forall\, i{\in}I.\ P\ i\ x\} \in M$
proof −
have $\{x{\in}\Omega.\ \forall\, i{\in}I.\ P\ i\ x\} = (\bigcap i{\in}I.\ \{x{\in}\Omega.\ P\ i\ x\}) \cap \Omega$ **by** *auto*
with *assms* **show** *?thesis*
by (*cases* $I = \{\}$) (*auto intro*!: *countable_INT'*)
qed

lemma (**in** *sigma_algebra*) *sets_Collect_countable_Ex1'*:
assumes $\bigwedge i.\ i \in I \Longrightarrow \{x{\in}\Omega.\ P\ i\ x\} \in M$
assumes *countable* I
shows $\{x{\in}\Omega.\ \exists!\, i{\in}I.\ P\ i\ x\} \in M$
proof −
have $\{x{\in}\Omega.\ \exists!\, i{\in}I.\ P\ i\ x\} = \{x{\in}\Omega.\ \exists\, i{\in}I.\ P\ i\ x \wedge (\forall\, j{\in}I.\ P\ j\ x \longrightarrow i = j)\}$
by *auto*
with *assms* **show** *?thesis*
by (*auto intro*!: *sets_Collect_countable_All' sets_Collect_countable_Ex' sets_Collect_conj sets_Collect_imp sets_Collect_const*)
qed

lemmas (**in** *sigma_algebra*) *sets_Collect* =
sets_Collect_imp sets_Collect_disj sets_Collect_conj sets_Collect_neg sets_Collect_const sets_Collect_countable_All sets_Collect_countable_Ex sets_Collect_countable_All

lemma (**in** *sigma_algebra*) *sets_Collect_countable_Ball*:
assumes $\bigwedge i.\ \{x{\in}\Omega.\ P\ i\ x\} \in M$
shows $\{x{\in}\Omega.\ \forall\, i{::}'i{::}countable{\in}X.\ P\ i\ x\} \in M$
unfolding *Ball_def* **by** (*intro sets_Collect assms*)

lemma (**in** *sigma_algebra*) *sets_Collect_countable_Bex*:
assumes $\bigwedge i.\ \{x{\in}\Omega.\ P\ i\ x\} \in M$
shows $\{x{\in}\Omega.\ \exists\, i{::}'i{::}countable{\in}X.\ P\ i\ x\} \in M$
unfolding *Bex_def* **by** (*intro sets_Collect assms*)

lemma *sigma_algebra_single_set*:
assumes $X \subseteq S$
shows *sigma_algebra* S $\{\ \{\},\ X,\ S - X,\ S\ \}$
using *algebra.is_sigma_algebra*[*OF algebra_single_set*[*OF* ‹$X \subseteq S$›]] **by** *simp*

Binary Unions

definition *binary* :: *'a* ⇒ *'a* ⇒ *nat* ⇒ *'a*
  where *binary a b* = (λ*x*. *b*)(*0* := *a*)

lemma *range_binary_eq*: *range*(*binary a b*) = {*a*,*b*}
  by (*auto simp add*: *binary_def*)

lemma *Un_range_binary*: *a* ∪ *b* = (⋃ *i*::*nat*. *binary a b i*)
  by (*simp add*: *range_binary_eq cong del*: *SUP_cong_simp*)

lemma *Int_range_binary*: *a* ∩ *b* = (⋂ *i*::*nat*. *binary a b i*)
  by (*simp add*: *range_binary_eq cong del*: *INF_cong_simp*)

lemma *sigma_algebra_iff2*:
  sigma_algebra Ω *M* ⟷
    M ⊆ *Pow* Ω ∧ {} ∈ *M* ∧ (∀ *s* ∈ *M*. Ω − *s* ∈ *M*)
    ∧ (∀ *A*. *range A* ⊆ *M* ⟶(⋃ *i*::*nat*. *A i*) ∈ *M*) (**is** *?P* ⟷ *?R* ∧ *?S* ∧ *?V* ∧ *?W*)
proof
  assume *?P*
  then interpret *sigma_algebra* Ω *M* **.**
  from *space_closed* **show** *?R* ∧ *?S* ∧ *?V* ∧ *?W*
    by *auto*
next
  assume *?R* ∧ *?S* ∧ *?V* ∧ *?W*
  then have *?R ?S ?V ?W*
    by *simp_all*
  show *?P*
  proof (*rule sigma_algebra.intro*)
    show *sigma_algebra_axioms M*
      by *standard* (*use* ‹*?W*› **in** *simp*)
    from ‹*?W*› **have** ∗: *range* (*binary a b*) ⊆ *M* ⟹ ⋃ (*range* (*binary a b*)) ∈ *M*
for *a b*
      by *auto*
    show *algebra* Ω *M*
      unfolding *algebra_iff_Un* **using** ‹*?R*› ‹*?S*› ‹*?V*› ∗
      by (*auto simp add*: *range_binary_eq*)
  qed
qed

Initial Sigma Algebra

Sigma algebras can naturally be created as the closure of any set of M with regard to the properties just postulated.

inductive_set *sigma_sets* :: *'a set* ⇒ *'a set set* ⇒ *'a set set*
  for *sp* :: *'a set* **and** *A* :: *'a set set*
  where
    Basic[*intro*, *simp*]: *a* ∈ *A* ⟹ *a* ∈ *sigma_sets sp A*

| *Empty*: $\{\} \in$ *sigma_sets sp A*
| *Compl*: $a \in$ *sigma_sets sp A* $\Longrightarrow$ *sp* $-$ *a* $\in$ *sigma_sets sp A*
| *Union*: $(\bigwedge i{::}nat.\ a\ i \in$ *sigma_sets sp A*$) \Longrightarrow (\bigcup i.\ a\ i) \in$ *sigma_sets sp A*

lemma (**in** *sigma_algebra*) *sigma_sets_subset*:
assumes *a*: $a \subseteq M$
shows *sigma_sets* $\Omega\ a \subseteq M$
proof
fix *x*
assume $x \in$ *sigma_sets* $\Omega\ a$
from *this* **show** $x \in M$
by (*induct rule*: *sigma_sets.induct*, *auto*) (*metis a subsetD*)
qed

lemma *sigma_sets_into_sp*: $A \subseteq$ *Pow sp* $\Longrightarrow x \in$ *sigma_sets sp A* $\Longrightarrow x \subseteq$ *sp*
by (*erule sigma_sets.induct*, *auto*)

lemma *sigma_algebra_sigma_sets*:
$a \subseteq$ *Pow* $\Omega \Longrightarrow$ *sigma_algebra* Ω (*sigma_sets* $\Omega\ a$)
by (*auto simp add*: *sigma_algebra_iff2 dest*: *sigma_sets_into_sp*
intro!: *sigma_sets.Union sigma_sets.Empty sigma_sets.Compl*)

lemma *sigma_sets_least_sigma_algebra*:
assumes $A \subseteq$ *Pow S*
shows *sigma_sets S A* $= \bigcap \{B.\ A \subseteq B \wedge$ *sigma_algebra S B*$\}$
proof *safe*
fix *B X* **assume** $A \subseteq B$ **and** *sa*: *sigma_algebra S B*
and *X*: $X \in$ *sigma_sets S A*
from *sigma_algebra.sigma_sets_subset*[*OF sa*, *simplified*, *OF* ‹$A \subseteq B$›] *X*
show $X \in B$ **by** *auto*
next
fix *X* **assume** $X \in \bigcap \{B.\ A \subseteq B \wedge$ *sigma_algebra S B*$\}$
then have [*intro*!]: $\bigwedge B.\ A \subseteq B \Longrightarrow$ *sigma_algebra S B* $\Longrightarrow X \in B$
by *simp*
have $A \subseteq$ *sigma_sets S A* **using** *assms* **by** *auto*
moreover have *sigma_algebra S* (*sigma_sets S A*)
using *assms* **by** (*intro sigma_algebra_sigma_sets*[*of A*]) *auto*
ultimately show $X \in$ *sigma_sets S A* **by** *auto*
qed

lemma *sigma_sets_top*: *sp* $\in$ *sigma_sets sp A*
by (*metis Diff_empty sigma_sets.Compl sigma_sets.Empty*)

lemma *binary_in_sigma_sets*:
binary a b i $\in$ *sigma_sets sp A* **if** $a \in$ *sigma_sets sp A* **and** $b \in$ *sigma_sets sp A*
using *that* **by** (*simp add*: *binary_def*)

lemma *sigma_sets_Un*:
$a \cup b \in$ *sigma_sets sp A* **if** $a \in$ *sigma_sets sp A* **and** $b \in$ *sigma_sets sp A*

using *that* **by** (*simp add*: *Un_range_binary binary_in_sigma_sets Union*)

lemma *sigma_sets_Inter*:
assumes *Asb*: $A \subseteq Pow\ sp$
shows $(\bigwedge i{::}nat.\ a\ i \in sigma\_sets\ sp\ A) \Longrightarrow (\bigcap i.\ a\ i) \in sigma\_sets\ sp\ A$
proof −
assume *ai*: $\bigwedge i{::}nat.\ a\ i \in sigma\_sets\ sp\ A$
hence $\bigwedge i{::}nat.\ sp{-}(a\ i) \in sigma\_sets\ sp\ A$
by (*rule sigma_sets.Compl*)
hence $(\bigcup i.\ sp{-}(a\ i)) \in sigma\_sets\ sp\ A$
by (*rule sigma_sets.Union*)
hence $sp{-}(\bigcup i.\ sp{-}(a\ i)) \in sigma\_sets\ sp\ A$
by (*rule sigma_sets.Compl*)
also have $sp{-}(\bigcup i.\ sp{-}(a\ i)) = sp\ Int\ (\bigcap i.\ a\ i)$
by *auto*
also have $... = (\bigcap i.\ a\ i)$ **using** *ai*
by (*blast dest*: *sigma_sets_into_sp* [*OF Asb*])
finally show *?thesis* **.**
qed

lemma *sigma_sets_INTER*:
assumes *Asb*: $A \subseteq Pow\ sp$
and *ai*: $\bigwedge i{::}nat.\ i \in S \Longrightarrow a\ i \in sigma\_sets\ sp\ A$ **and** *non*: $S \neq \{\}$
shows $(\bigcap i{\in}S.\ a\ i) \in sigma\_sets\ sp\ A$
proof −
from *ai* **have** $\bigwedge i.\ (if\ i{\in}S\ then\ a\ i\ else\ sp) \in sigma\_sets\ sp\ A$
by (*simp add*: *sigma_sets.intros*(*2*−) *sigma_sets_top*)
hence $(\bigcap i.\ (if\ i{\in}S\ then\ a\ i\ else\ sp)) \in sigma\_sets\ sp\ A$
by (*rule sigma_sets_Inter* [*OF Asb*])
also have $(\bigcap i.\ (if\ i{\in}S\ then\ a\ i\ else\ sp)) = (\bigcap i{\in}S.\ a\ i)$
by *auto* (*metis ai non sigma_sets_into_sp subset_empty subset_iff Asb*)+
finally show *?thesis* **.**
qed

lemma *sigma_sets_UNION*:
countable B $\Longrightarrow (\bigwedge b.\ b \in B \Longrightarrow b \in sigma\_sets\ X\ A) \Longrightarrow \bigcup\ B \in sigma\_sets\ X\ A$
using *from_nat_into* [*of B*] *range_from_nat_into* [*of B*] *sigma_sets.Union* [*of from_nat_into B X A*]
by (*cases* $B = \{\}$) (*simp_all add*: *sigma_sets.Empty cong del*: *SUP_cong*)

lemma (**in** *sigma_algebra*) *sigma_sets_eq*:
$sigma\_sets\ \Omega\ M = M$
proof
show $M \subseteq sigma\_sets\ \Omega\ M$
by (*metis Set.subsetI sigma_sets.Basic*)
next
show $sigma\_sets\ \Omega\ M \subseteq M$
by (*metis sigma_sets_subset subset_refl*)

qed

lemma *sigma_sets_eqI*:
assumes *A*: $\bigwedge a.\ a \in A \Longrightarrow a \in$ *sigma_sets M B*
assumes *B*: $\bigwedge b.\ b \in B \Longrightarrow b \in$ *sigma_sets M A*
shows *sigma_sets M A* = *sigma_sets M B*
proof (*intro set_eqI iffI*)
fix *a* **assume** *a* ∈ *sigma_sets M A*
from *this A* **show** *a* ∈ *sigma_sets M B*
by *induct* (*auto intro*!: *sigma_sets.intros*(2−) *del*: *sigma_sets.Basic*)
next
fix *b* **assume** *b* ∈ *sigma_sets M B*
from *this B* **show** *b* ∈ *sigma_sets M A*
by *induct* (*auto intro*!: *sigma_sets.intros*(2−) *del*: *sigma_sets.Basic*)
qed

lemma *sigma_sets_subseteq*: **assumes** *A* ⊆ *B* **shows** *sigma_sets X A* ⊆ *sigma_sets X B*
proof
fix *x* **assume** *x* ∈ *sigma_sets X A* **then show** *x* ∈ *sigma_sets X B*
by *induct* (*insert* ‹*A* ⊆ *B*›, *auto intro*: *sigma_sets.intros*(2−))
qed

lemma *sigma_sets_mono*: **assumes** *A* ⊆ *sigma_sets X B* **shows** *sigma_sets X A* ⊆ *sigma_sets X B*
proof
fix *x* **assume** *x* ∈ *sigma_sets X A* **then show** *x* ∈ *sigma_sets X B*
by *induct* (*insert* ‹*A* ⊆ *sigma_sets X B*›, *auto intro*: *sigma_sets.intros*(2−))
qed

lemma *sigma_sets_mono'*: **assumes** *A* ⊆ *B* **shows** *sigma_sets X A* ⊆ *sigma_sets X B*
proof
fix *x* **assume** *x* ∈ *sigma_sets X A* **then show** *x* ∈ *sigma_sets X B*
by *induct* (*insert* ‹*A* ⊆ *B*›, *auto intro*: *sigma_sets.intros*(2−))
qed

lemma *sigma_sets_superset_generator*: *A* ⊆ *sigma_sets X A*
by (*auto intro*: *sigma_sets.Basic*)

lemma (**in** *sigma_algebra*) *restriction_in_sets*:
fixes *A* :: *nat* ⇒ *'a set*
assumes *S* ∈ *M*
and ∗: *range A* ⊆ (λ*A*. *S* ∩ *A*) ‘ *M* (**is** _ ⊆ *?r*)
shows *range A* ⊆ *M* (⋃ *i*. *A i*) ∈ (λ*A*. *S* ∩ *A*) ‘ *M*
proof −
{ fix *i* **have** *A i* ∈ *?r* **using** ∗ **by** *auto*
hence ∃ *B*. *A i* = *B* ∩ *S* ∧ *B* ∈ *M* **by** *auto*
hence *A i* ⊆ *S A i* ∈ *M* **using** ‹*S* ∈ *M*› **by** *auto* **}**

```
  thus range A ⊆ M (⋃i. A i) ∈ (λA. S ∩ A) ‘ M
    by (auto intro!: image_eqI[of _ _ (⋃i. A i)])
qed

lemma (in sigma_algebra) restricted_sigma_algebra:
  assumes S ∈ M
  shows sigma_algebra S (restricted_space S)
  unfolding sigma_algebra_def sigma_algebra_axioms_def
proof safe
  show algebra S (restricted_space S) using restricted_algebra[OF assms] .
next
  fix A :: nat ⇒ 'a set assume range A ⊆ restricted_space S
  from restriction_in_sets[OF assms this[simplified]]
  show (⋃i. A i) ∈ restricted_space S by simp
qed

lemma sigma_sets_Int:
  assumes A ∈ sigma_sets sp st A ⊆ sp
  shows (∩) A ‘ sigma_sets sp st = sigma_sets A ((∩) A ‘ st)
proof (intro equalityI subsetI)
  fix x assume x ∈ (∩) A ‘ sigma_sets sp st
  then obtain y where y ∈ sigma_sets sp st x = y ∩ A by auto
  then have x ∈ sigma_sets (A ∩ sp) ((∩) A ‘ st)
  proof (induct arbitrary: x)
    case (Compl a)
    then show ?case
      by (force intro!: sigma_sets.Compl simp: Diff_Int_distrib ac_simps)
  next
    case (Union a)
    then show ?case
      by (auto intro!: sigma_sets.Union
            simp add: UN_extend_simps simp del: UN_simps)
  qed (auto intro!: sigma_sets.intros(2−))
  then show x ∈ sigma_sets A ((∩) A ‘ st)
    using ‹A ⊆ sp› by (simp add: Int_absorb2)
next
  fix x assume x ∈ sigma_sets A ((∩) A ‘ st)
  then show x ∈ (∩) A ‘ sigma_sets sp st
  proof induct
    case (Compl a)
    then obtain x where a = A ∩ x x ∈ sigma_sets sp st by auto
    then show ?case using ‹A ⊆ sp›
      by (force simp add: image_iff intro!: bexI[of _ sp − x] sigma_sets.Compl)
  next
    case (Union a)
    then have ∀i. ∃x. x ∈ sigma_sets sp st ∧ a i = A ∩ x
      by (auto simp: image_iff Bex_def)
    then obtain f where ∀x. f x ∈ sigma_sets sp st ∧ a x = A ∩ f x
      by metis
```

```
    then show ?case
      by (auto intro!: bexI[of _ (⋃x. f x)] sigma_sets.Union
            simp add: image_iff)
  qed (auto intro!: sigma_sets.intros(2−))
qed

lemma sigma_sets_empty_eq: sigma_sets A {} = {{}, A}
proof (intro set_eqI iffI)
  fix a assume a ∈ sigma_sets A {} then show a ∈ {{}, A}
    by induct blast+
qed (auto intro: sigma_sets.Empty sigma_sets_top)

lemma sigma_sets_single[simp]: sigma_sets A {A} = {{}, A}
proof (intro set_eqI iffI)
  fix x assume x ∈ sigma_sets A {A}
  then show x ∈ {{}, A}
    by induct blast+
next
  fix x assume x ∈ {{}, A}
  then show x ∈ sigma_sets A {A}
    by (auto intro: sigma_sets.Empty sigma_sets_top)
qed

lemma sigma_sets_sigma_sets_eq:
  M ⊆ Pow S ⟹ sigma_sets S (sigma_sets S M) = sigma_sets S M
  by (rule sigma_algebra.sigma_sets_eq[OF sigma_algebra_sigma_sets, of M S])
auto

lemma sigma_sets_singleton:
  assumes X ⊆ S
  shows sigma_sets S { X } = { {}, X, S − X, S }
proof −
  interpret sigma_algebra S { {}, X, S − X, S }
    by (rule sigma_algebra_single_set) fact
  have sigma_sets S { X } ⊆ sigma_sets S { {}, X, S − X, S }
    by (rule sigma_sets_subseteq) simp
  moreover have ... = { {}, X, S − X, S }
    using sigma_sets_eq by simp
  moreover
  { fix A assume A ∈ { {}, X, S − X, S }
    then have A ∈ sigma_sets S { X }
      by (auto intro: sigma_sets.intros(2−) sigma_sets_top) }
  ultimately have sigma_sets S { X } = sigma_sets S { {}, X, S − X, S }
    by (intro antisym) auto
  with sigma_sets_eq show ?thesis by simp
qed

lemma restricted_sigma:
  assumes S: S ∈ sigma_sets Ω M and M: M ⊆ Pow Ω
```

```
  shows algebra.restricted_space (sigma_sets Ω M) S =
    sigma_sets S (algebra.restricted_space M S)
proof −
  from S sigma_sets_into_sp[OF M]
  have S ∈ sigma_sets Ω M S ⊆ Ω by auto
  from sigma_sets_Int[OF this]
  show ?thesis by simp
qed

lemma sigma_sets_vimage_commute:
  assumes X: X ∈ Ω → Ω'
  shows {X −‘ A ∩ Ω |A. A ∈ sigma_sets Ω' M'}
      = sigma_sets Ω {X −‘ A ∩ Ω |A. A ∈ M'} (is ?L = ?R)
proof
  show ?L ⊆ ?R
  proof clarify
    fix A assume A ∈ sigma_sets Ω' M'
    then show X −‘ A ∩ Ω ∈ ?R
    proof induct
      case Empty then show ?case
        by (auto intro!: sigma_sets.Empty)
    next
      case (Compl B)
      have [simp]: X −‘ (Ω' − B) ∩ Ω = Ω − (X −‘ B ∩ Ω)
        by (auto simp add: funcset_mem [OF X])
      with Compl show ?case
        by (auto intro!: sigma_sets.Compl)
    next
      case (Union F)
      then show ?case
       by (auto simp add: vimage_UN UN_extend_simps(4) simp del: UN_simps
              intro!: sigma_sets.Union)
    qed auto
  qed
  show ?R ⊆ ?L
  proof clarify
    fix A assume A ∈ ?R
    then show ∃ B. A = X −‘ B ∩ Ω ∧ B ∈ sigma_sets Ω' M'
    proof induct
      case (Basic B) then show ?case by auto
    next
      case Empty then show ?case
        by (auto intro!: sigma_sets.Empty exI[of _ {}])
    next
      case (Compl B)
      then obtain A where A: B = X −‘ A ∩ Ω A ∈ sigma_sets Ω' M' by auto
      then have [simp]: Ω − B = X −‘ (Ω' − A) ∩ Ω
        by (auto simp add: funcset_mem [OF X])
      with A(2) show ?case
```

```
      by (auto intro: sigma_sets.Compl)
    next
      case (Union F)
      then have ∀ i. ∃ B. F i = X -‘ B ∩ Ω ∧ B ∈ sigma_sets Ω' M' by auto
      then obtain A where ∀ x. F x = X -‘ A x ∩ Ω ∧ A x ∈ sigma_sets Ω' M'
        by metis
      then show ?case
        by (auto simp: vimage_UN[symmetric] intro: sigma_sets.Union)
    qed
  qed
qed

lemma (in ring_of_sets) UNION_in_sets:
  fixes A:: nat ⇒ 'a set
  assumes A: range A ⊆ M
  shows  (⋃ i∈{0..<n}. A i) ∈ M
proof (induct n)
  case 0 show ?case by simp
next
  case (Suc n)
  thus ?case
    by (simp add: atLeastLessThanSuc) (metis A Un UNIV_I image_subset_iff)
qed

lemma (in ring_of_sets) range_disjointed_sets:
  assumes A: range A ⊆ M
  shows  range (disjointed A) ⊆ M
proof (auto simp add: disjointed_def)
  fix n
  show A n - (⋃ i∈{0..<n}. A i) ∈ M using UNION_in_sets
    by (metis A Diff UNIV_I image_subset_iff)
qed

lemma (in algebra) range_disjointed_sets':
  range A ⊆ M ⟹ range (disjointed A) ⊆ M
  using range_disjointed_sets .

lemma sigma_algebra_disjoint_iff:
  sigma_algebra Ω M ⟷ algebra Ω M ∧
    (∀ A. range A ⊆ M ⟶ disjoint_family A ⟶ (⋃ i::nat. A i) ∈ M)
proof (auto simp add: sigma_algebra_iff)
  fix A :: nat ⇒ 'a set
  assume M: algebra Ω M
     and A: range A ⊆ M
     and UnA: ∀ A. range A ⊆ M ⟶ disjoint_family A ⟶ (⋃ i::nat. A i) ∈ M
  hence range (disjointed A) ⊆ M ⟶
        disjoint_family (disjointed A) ⟶
        (⋃ i. disjointed A i) ∈ M by blast
  hence (⋃ i. disjointed A i) ∈ M
```

by (*simp add*: *algebra.range_disjointed_sets'*[*of* Ω] *M A disjoint_family_disjointed*)
thus (⋃ *i*::*nat*. *A i*) ∈ *M* **by** (*simp add*: *UN_disjointed_eq*)
qed

Ring generated by a semiring

definition (**in** *semiring_of_sets*) *generated_ring* :: *'a set set* **where**
generated_ring = { ⋃ *C* | *C*. *C* ⊆ *M* ∧ *finite C* ∧ *disjoint C* }

lemma (**in** *semiring_of_sets*) *generated_ringE*[*elim?*]:
assumes *a* ∈ *generated_ring*
obtains *C* **where** *finite C disjoint C C* ⊆ *M a* = ⋃ *C*
using *assms* **unfolding** *generated_ring_def* **by** *auto*

lemma (**in** *semiring_of_sets*) *generated_ringI*[*intro?*]:
assumes *finite C disjoint C C* ⊆ *M a* = ⋃ *C*
shows *a* ∈ *generated_ring*
using *assms* **unfolding** *generated_ring_def* **by** *auto*

lemma (**in** *semiring_of_sets*) *generated_ringI_Basic*:
A ∈ *M* ⟹ *A* ∈ *generated_ring*
by (*rule generated_ringI*[*of* {*A*}]) (*auto simp*: *disjoint_def*)

lemma (**in** *semiring_of_sets*) *generated_ring_disjoint_Un*[*intro*]:
assumes *a*: *a* ∈ *generated_ring* **and** *b*: *b* ∈ *generated_ring*
and *a* ∩ *b* = {}
shows *a* ∪ *b* ∈ *generated_ring*
proof −
from *a b* **obtain** *Ca Cb*
where *Ca*: *finite Ca disjoint Ca Ca* ⊆ *M a* = ⋃ *Ca*
and *Cb*: *finite Cb disjoint Cb Cb* ⊆ *M b* = ⋃ *Cb*
using *generated_ringE* **by** *metis*
show *?thesis*
proof
from ‹*a* ∩ *b* = {}› *Ca Cb* **show** *disjoint* (*Ca* ∪ *Cb*)
by (*auto intro*!: *disjoint_union*)
qed (*use Ca Cb* **in** *auto*)
qed

lemma (**in** *semiring_of_sets*) *generated_ring_empty*: {} ∈ *generated_ring*
by (*auto simp*: *generated_ring_def disjoint_def*)

lemma (**in** *semiring_of_sets*) *generated_ring_disjoint_Union*:
assumes *finite A* **shows** *A* ⊆ *generated_ring* ⟹ *disjoint A* ⟹ ⋃ *A* ∈ *generated_ring*
using *assms* **by** (*induct A*) (*auto simp*: *disjoint_def intro*!: *generated_ring_disjoint_Un generated_ring_empty*)

lemma (**in** *semiring_of_sets*) *generated_ring_disjoint_UNION*:

finite I $\Longrightarrow$ *disjoint* (*A* ‘ *I*) $\Longrightarrow$ ($\bigwedge$*i*. *i* $\in$ *I* $\Longrightarrow$ *A i* $\in$ *generated_ring*) $\Longrightarrow$ $\bigcup$(*A* ‘ *I*) $\in$ *generated_ring*
by (*intro generated_ring_disjoint_Union*) *auto*

lemma (**in** *semiring_of_sets*) *generated_ring_Int*:
assumes *a*: *a* $\in$ *generated_ring* **and** *b*: *b* $\in$ *generated_ring*
shows *a* $\cap$ *b* $\in$ *generated_ring*
proof −
from *a b* **obtain** *Ca Cb*
where *Ca*: *finite Ca disjoint Ca Ca* $\subseteq$ *M a* = $\bigcup$ *Ca*
and *Cb*: *finite Cb disjoint Cb Cb* $\subseteq$ *M b* = $\bigcup$ *Cb*
using *generated_ringE* **by** *metis*
define *C* **where** *C* = (λ(*a*,*b*). *a* $\cap$ *b*)‘ (*Ca*×*Cb*)
show *?thesis*
proof
show *disjoint C*
proof (*simp add*: *disjoint_def C_def*, *intro ballI impI*)
fix *a1 b1 a2 b2* **assume** *sets*: *a1* $\in$ *Ca b1* $\in$ *Cb a2* $\in$ *Ca b2* $\in$ *Cb*
assume *a1* $\cap$ *b1* $\neq$ *a2* $\cap$ *b2*
then have *a1* $\neq$ *a2* $\vee$ *b1* $\neq$ *b2* **by** *auto*
then show (*a1* $\cap$ *b1*) $\cap$ (*a2* $\cap$ *b2*) = {}
proof
assume *a1* $\neq$ *a2*
with *sets Ca* **have** *a1* $\cap$ *a2* = {}
by (*auto simp*: *disjoint_def*)
then show *?thesis* **by** *auto*
next
assume *b1* $\neq$ *b2*
with *sets Cb* **have** *b1* $\cap$ *b2* = {}
by (*auto simp*: *disjoint_def*)
then show *?thesis* **by** *auto*
qed
qed
qed (*use Ca Cb* **in** ‹*auto simp*: *C_def*›)
qed

lemma (**in** *semiring_of_sets*) *generated_ring_Inter*:
assumes *finite A A* $\neq$ {} **shows** *A* $\subseteq$ *generated_ring* $\Longrightarrow$ $\bigcap$ *A* $\in$ *generated_ring*
using *assms* **by** (*induct A rule*: *finite_ne_induct*) (*auto intro*: *generated_ring_Int*)

lemma (**in** *semiring_of_sets*) *generated_ring_INTER*:
finite I $\Longrightarrow$ *I* $\neq$ {} $\Longrightarrow$ ($\bigwedge$*i*. *i* $\in$ *I* $\Longrightarrow$ *A i* $\in$ *generated_ring*) $\Longrightarrow$ $\bigcap$(*A* ‘ *I*) $\in$ *generated_ring*
by (*intro generated_ring_Inter*) *auto*

lemma (**in** *semiring_of_sets*) *generating_ring*:
ring_of_sets Ω *generated_ring*
proof (*rule ring_of_setsI*)
let *?R* = *generated_ring*

```
  show ?R ⊆ Pow Ω
   using sets_into_space by (auto simp: generated_ring_def generated_ring_empty)
  show {} ∈ ?R by (rule generated_ring_empty)

  {
    fix a b assume a ∈ ?R b ∈ ?R
    then obtain Ca Cb
      where Ca: finite Ca disjoint Ca Ca ⊆ M a = ⋃ Ca
        and Cb: finite Cb disjoint Cb Cb ⊆ M b = ⋃ Cb
      using generated_ringE by metis
    show a − b ∈ ?R
    proof cases
      assume Cb = {} with Cb ‹a ∈ ?R› show ?thesis
        by simp
    next
      assume Cb ≠ {}
      with Ca Cb have a − b = (⋃a'∈Ca. ⋂b'∈Cb. a' − b') by auto
      also have ... ∈ ?R
      proof (intro generated_ring_INTER generated_ring_disjoint_UNION)
        fix a b assume a ∈ Ca b ∈ Cb
        with Ca Cb Diff_cover[of a b] show a − b ∈ ?R
          by (auto simp add: generated_ring_def)
            (metis DiffI Diff_eq_empty_iff empty_iff)
      next
        show disjoint ((λa'. ⋂b'∈Cb. a' − b') ‘Ca)
          using Ca by (auto simp add: disjoint_def ‹Cb ≠ {}›)
      next
        show finite Ca finite Cb Cb ≠ {} by fact+
      qed
      finally show a − b ∈ ?R .
    qed
  }
  note Diff = this

  fix a b assume sets: a ∈ ?R b ∈ ?R
  have a ∪ b = (a − b) ∪ (a ∩ b) ∪ (b − a) by auto
  also have ... ∈ ?R
    by (intro sets generated_ring_disjoint_Un generated_ring_Int Diff) auto
  finally show a ∪ b ∈ ?R .
qed

lemma (in semiring_of_sets) sigma_sets_generated_ring_eq: sigma_sets Ω generated_ring = sigma_sets Ω M
proof
  interpret M: sigma_algebra Ω sigma_sets Ω M
    using space_closed by (rule sigma_algebra_sigma_sets)
  show sigma_sets Ω generated_ring ⊆ sigma_sets Ω M
    by (blast intro!: sigma_sets_mono elim: generated_ringE)
qed (auto intro!: generated_ringI_Basic sigma_sets_mono)
```

A Two-Element Series

definition *binaryset* :: *'a set ⇒ 'a set ⇒ nat ⇒ 'a set*
where *binaryset A B* = (*λx*. {})(*0* := *A*, *Suc 0* := *B*)

lemma *range_binaryset_eq*: *range*(*binaryset A B*) = {*A*,*B*,{}}
apply (*simp add*: *binaryset_def*)
apply (*rule set_eqI*)
apply (*auto simp add*: *image_iff*)
done

lemma *UN_binaryset_eq*: (⋃ *i*. *binaryset A B i*) = *A* ∪ *B*
by (*simp add*: *range_binaryset_eq cong del*: *SUP_cong_simp*)

Closed CDI

definition *closed_cdi* :: *'a set ⇒ 'a set set ⇒ bool* **where**
closed_cdi Ω *M* ⟷
M ⊆ *Pow* Ω &
(∀ *s* ∈ *M*. Ω − *s* ∈ *M*) &
(∀ *A*. (*range A* ⊆ *M*) & (*A 0* = {}) & (∀ *n*. *A n* ⊆ *A* (*Suc n*)) ⟶
(⋃ *i*. *A i*) ∈ *M*) &
(∀ *A*. (*range A* ⊆ *M*) & *disjoint_family A* ⟶ (⋃ *i*::*nat*. *A i*) ∈ *M*)

inductive_set
smallest_ccdi_sets :: *'a set ⇒ 'a set set ⇒ 'a set set*
for Ω *M*
where
Basic [*intro*]:
a ∈ *M* ⟹ *a* ∈ *smallest_ccdi_sets* Ω *M*
| *Compl* [*intro*]:
a ∈ *smallest_ccdi_sets* Ω *M* ⟹ Ω − *a* ∈ *smallest_ccdi_sets* Ω *M*
| *Inc*:
range A ∈ *Pow*(*smallest_ccdi_sets* Ω *M*) ⟹ *A 0* = {} ⟹ (⋀*n*. *A n* ⊆ *A* (*Suc n*))
⟹ (⋃ *i*. *A i*) ∈ *smallest_ccdi_sets* Ω *M*
| *Disj*:
range A ∈ *Pow*(*smallest_ccdi_sets* Ω *M*) ⟹ *disjoint_family A*
⟹ (⋃ *i*::*nat*. *A i*) ∈ *smallest_ccdi_sets* Ω *M*

lemma (**in** *subset_class*) *smallest_closed_cdi1*: *M* ⊆ *smallest_ccdi_sets* Ω *M*
by *auto*

lemma (**in** *subset_class*) *smallest_ccdi_sets*: *smallest_ccdi_sets* Ω *M* ⊆ *Pow* Ω
apply (*rule subsetI*)
apply (*erule smallest_ccdi_sets.induct*)
apply (*auto intro*: *range_subsetD dest*: *sets_into_space*)
done

lemma (**in** *subset_class*) *smallest_closed_cdi2*: *closed_cdi* Ω (*smallest_ccdi_sets*

Ω M)
apply (*auto simp add*: *closed_cdi_def smallest_ccdi_sets*)
apply (*blast intro*: *smallest_ccdi_sets.Inc smallest_ccdi_sets.Disj*) +
done

lemma *closed_cdi_subset*: *closed_cdi* Ω $M \Longrightarrow M \subseteq Pow\ \Omega$
by (*simp add*: *closed_cdi_def*)

lemma *closed_cdi_Compl*: *closed_cdi* Ω $M \Longrightarrow s \in M \Longrightarrow \Omega - s \in M$
by (*simp add*: *closed_cdi_def*)

lemma *closed_cdi_Inc*:
closed_cdi Ω $M \Longrightarrow$ *range* $A \subseteq M \Longrightarrow A\ 0 = \{\} \Longrightarrow (!!n.\ A\ n \subseteq A\ (Suc\ n))$
$\Longrightarrow (\bigcup i.\ A\ i) \in M$
by (*simp add*: *closed_cdi_def*)

lemma *closed_cdi_Disj*:
closed_cdi Ω $M \Longrightarrow$ *range* $A \subseteq M \Longrightarrow$ *disjoint_family* $A \Longrightarrow (\bigcup i{::}nat.\ A\ i) \in$
M
by (*simp add*: *closed_cdi_def*)

lemma *closed_cdi_Un*:
assumes *cdi*: *closed_cdi* Ω M **and** *empty*: $\{\} \in M$
and A: $A \in M$ **and** B: $B \in M$
and *disj*: $A \cap B = \{\}$
shows $A \cup B \in M$
proof −
have *ra*: *range* (*binaryset A B*) $\subseteq M$
by (*simp add*: *range_binaryset_eq empty A B*)
have *di*: *disjoint_family* (*binaryset A B*) **using** *disj*
by (*simp add*: *disjoint_family_on_def binaryset_def Int_commute*)
from *closed_cdi_Disj* [*OF cdi ra di*]
show *?thesis*
by (*simp add*: *UN_binaryset_eq*)
qed

lemma (**in** *algebra*) *smallest_ccdi_sets_Un*:
assumes A: $A \in$ *smallest_ccdi_sets* Ω M **and** B: $B \in$ *smallest_ccdi_sets* Ω M
and *disj*: $A \cap B = \{\}$
shows $A \cup B \in$ *smallest_ccdi_sets* Ω M
proof −
have *ra*: *range* (*binaryset A B*) $\in$ *Pow* (*smallest_ccdi_sets* Ω M)
by (*simp add*: *range_binaryset_eq A B smallest_ccdi_sets.Basic*)
have *di*: *disjoint_family* (*binaryset A B*) **using** *disj*
by (*simp add*: *disjoint_family_on_def binaryset_def Int_commute*)
from *Disj* [*OF ra di*]
show *?thesis*
by (*simp add*: *UN_binaryset_eq*)
qed

lemma (**in** *algebra*) *smallest_ccdi_sets_Int1*:
assumes *a*: $a \in M$
shows $b \in$ *smallest_ccdi_sets* $\Omega\ M \Longrightarrow a \cap b \in$ *smallest_ccdi_sets* $\Omega\ M$
proof (*induct rule*: *smallest_ccdi_sets.induct*)
case (*Basic x*)
thus *?case*
by (*metis a Int smallest_ccdi_sets.Basic*)
next
case (*Compl x*)
have $a \cap (\Omega - x) = \Omega - ((\Omega - a) \cup (a \cap x))$
by *blast*
also have ... $\in$ *smallest_ccdi_sets* $\Omega\ M$
by (*metis smallest_ccdi_sets.Compl a Compl*(*2*) *Diff_Int2 Diff_Int_distrib2*
Diff_disjoint Int_Diff Int_empty_right smallest_ccdi_sets_Un
smallest_ccdi_sets.Basic smallest_ccdi_sets.Compl)
finally show *?case* **.**
next
case (*Inc A*)
have *1*: $(\bigcup i.\ (\lambda i.\ a \cap A\ i)\ i) = a \cap (\bigcup i.\ A\ i)$
by *blast*
have *range* $(\lambda i.\ a \cap A\ i) \in$ *Pow*(*smallest_ccdi_sets* $\Omega\ M$) **using** *Inc*
by *blast*
moreover have $(\lambda i.\ a \cap A\ i)\ 0 = \{\}$
by (*simp add*: *Inc*)
moreover have $!!n.\ (\lambda i.\ a \cap A\ i)\ n \subseteq (\lambda i.\ a \cap A\ i)\ (Suc\ n)$ **using** *Inc*
by *blast*
ultimately have *2*: $(\bigcup i.\ (\lambda i.\ a \cap A\ i)\ i) \in$ *smallest_ccdi_sets* $\Omega\ M$
by (*rule smallest_ccdi_sets.Inc*)
show *?case*
by (*metis 1 2*)
next
case (*Disj A*)
have *1*: $(\bigcup i.\ (\lambda i.\ a \cap A\ i)\ i) = a \cap (\bigcup i.\ A\ i)$
by *blast*
have *range* $(\lambda i.\ a \cap A\ i) \in$ *Pow*(*smallest_ccdi_sets* $\Omega\ M$) **using** *Disj*
by *blast*
moreover have *disjoint_family* $(\lambda i.\ a \cap A\ i)$ **using** *Disj*
by (*auto simp add*: *disjoint_family_on_def*)
ultimately have *2*: $(\bigcup i.\ (\lambda i.\ a \cap A\ i)\ i) \in$ *smallest_ccdi_sets* $\Omega\ M$
by (*rule smallest_ccdi_sets.Disj*)
show *?case*
by (*metis 1 2*)
qed

lemma (**in** *algebra*) *smallest_ccdi_sets_Int*:
assumes *b*: $b \in$ *smallest_ccdi_sets* $\Omega\ M$
shows $a \in$ *smallest_ccdi_sets* $\Omega\ M \Longrightarrow a \cap b \in$ *smallest_ccdi_sets* $\Omega\ M$

proof (*induct rule*: *smallest_ccdi_sets.induct*)
case (*Basic x*)
thus *?case*
by (*metis b smallest_ccdi_sets_Int1*)
next
case (*Compl x*)
have $(\Omega - x) \cap b = \Omega - (x \cap b \cup (\Omega - b))$
by *blast*
also have ... $\in$ *smallest_ccdi_sets* Ω *M*
by (*metis Compl*(*2*) *Diff_disjoint Int_Diff Int_commute Int_empty_right b*
smallest_ccdi_sets.Compl smallest_ccdi_sets_Un)
finally show *?case* **.**
next
case (*Inc A*)
have *1*: $(\bigcup i.\ (\lambda i.\ A\ i \cap b)\ i) = (\bigcup i.\ A\ i) \cap b$
by *blast*
have *range* $(\lambda i.\ A\ i \cap b) \in$ *Pow*(*smallest_ccdi_sets* Ω *M*) **using** *Inc*
by *blast*
moreover have $(\lambda i.\ A\ i \cap b)\ 0 = \{\}$
by (*simp add*: *Inc*)
moreover have $!!n.\ (\lambda i.\ A\ i \cap b)\ n \subseteq (\lambda i.\ A\ i \cap b)\ (Suc\ n)$ **using** *Inc*
by *blast*
ultimately have *2*: $(\bigcup i.\ (\lambda i.\ A\ i \cap b)\ i) \in$ *smallest_ccdi_sets* Ω *M*
by (*rule smallest_ccdi_sets.Inc*)
show *?case*
by (*metis 1 2*)
next
case (*Disj A*)
have *1*: $(\bigcup i.\ (\lambda i.\ A\ i \cap b)\ i) = (\bigcup i.\ A\ i) \cap b$
by *blast*
have *range* $(\lambda i.\ A\ i \cap b) \in$ *Pow*(*smallest_ccdi_sets* Ω *M*) **using** *Disj*
by *blast*
moreover have *disjoint_family* $(\lambda i.\ A\ i \cap b)$ **using** *Disj*
by (*auto simp add*: *disjoint_family_on_def*)
ultimately have *2*: $(\bigcup i.\ (\lambda i.\ A\ i \cap b)\ i) \in$ *smallest_ccdi_sets* Ω *M*
by (*rule smallest_ccdi_sets.Disj*)
show *?case*
by (*metis 1 2*)
qed

lemma (**in** *algebra*) *sigma_property_disjoint_lemma*:
assumes *sbC*: $M \subseteq C$
and *ccdi*: *closed_cdi* Ω *C*
shows *sigma_sets* Ω *M* $\subseteq$ *C*
proof −
have *smallest_ccdi_sets* Ω *M* $\in \{B\ .\ M \subseteq B \wedge$ *sigma_algebra* Ω *B*$\}$
apply (*auto simp add*: *sigma_algebra_disjoint_iff algebra_iff_Int*
smallest_ccdi_sets_Int)
apply (*metis Union_Pow_eq Union_upper subsetD smallest_ccdi_sets*)

```
    apply (blast intro: smallest_ccdi_sets.Disj)
    done
  hence sigma_sets (Ω) (M) ⊆ smallest_ccdi_sets Ω M
    by clarsimp
       (drule sigma_algebra.sigma_sets_subset [where a=M], auto)
  also have ... ⊆ C
    proof
      fix x
      assume x: x ∈ smallest_ccdi_sets Ω M
      thus x ∈ C
        proof (induct rule: smallest_ccdi_sets.induct)
          case (Basic x)
          thus ?case
            by (metis Basic subsetD sbC)
        next
          case (Compl x)
          thus ?case
            by (blast intro: closed_cdi_Compl [OF ccdi, simplified])
        next
          case (Inc A)
          thus ?case
              by (auto intro: closed_cdi_Inc [OF ccdi, simplified])
        next
          case (Disj A)
          thus ?case
              by (auto intro: closed_cdi_Disj [OF ccdi, simplified])
        qed
    qed
  finally show ?thesis .
qed

lemma (in algebra) sigma_property_disjoint:
  assumes sbC: M ⊆ C
      and compl: !!s. s ∈ C ∩ sigma_sets (Ω) (M) ⟹ Ω − s ∈ C
      and inc: !!A. range A ⊆ C ∩ sigma_sets (Ω) (M)
                    ⟹ A 0 = {} ⟹ (!!n. A n ⊆ A (Suc n))
                    ⟹ (⋃ i. A i) ∈ C
      and disj: !!A. range A ⊆ C ∩ sigma_sets (Ω) (M)
                     ⟹ disjoint_family A ⟹ (⋃ i::nat. A i) ∈ C
  shows sigma_sets (Ω) (M) ⊆ C
proof −
  have sigma_sets (Ω) (M) ⊆ C ∩ sigma_sets (Ω) (M)
    proof (rule sigma_property_disjoint_lemma)
      show M ⊆ C ∩ sigma_sets (Ω) (M)
        by (metis Int_greatest Set.subsetI sbC sigma_sets.Basic)
    next
      show closed_cdi Ω (C ∩ sigma_sets (Ω) (M))
        by (simp add: closed_cdi_def compl inc disj)
           (metis PowI Set.subsetI le_infI2 sigma_sets_into_sp space_closed
```

IntE sigma_sets.Compl range_subsetD sigma_sets.Union)
qed
thus *?thesis*
by *blast*
qed

Dynkin systems

locale *Dynkin_system* = *subset_class* +
assumes *space*: $\Omega \in M$
and *compl*[*intro*!]: $\bigwedge A.\ A \in M \Longrightarrow \Omega - A \in M$
and *UN*[*intro*!]: $\bigwedge A.$ *disjoint_family* $A \Longrightarrow$ *range* $A \subseteq M$
$\Longrightarrow (\bigcup i{::}nat.\ A\ i) \in M$

lemma (**in** *Dynkin_system*) *empty*[*intro*, *simp*]: $\{\} \in M$
using *space compl*[*of* Ω] **by** *simp*

lemma (**in** *Dynkin_system*) *diff*:
assumes *sets*: $D \in M$ $E \in M$ **and** $D \subseteq E$
shows $E - D \in M$
proof −
let *?f* = $\lambda x.$ *if* $x = 0$ *then* D *else if* $x =$ *Suc 0 then* $\Omega - E$ *else* $\{\}$
have *range ?f* = $\{D, \Omega - E, \{\}\}$
by (*auto simp*: *image_iff*)
moreover have $D \cup (\Omega - E) = (\bigcup i.$ *?f* $i)$
by (*auto simp*: *image_iff split*: *if_split_asm*)
moreover
have *disjoint_family ?f* **unfolding** *disjoint_family_on_def*
using ‹$D \in M$›[*THEN sets_into_space*] ‹$D \subseteq E$› **by** *auto*
ultimately have $\Omega - (D \cup (\Omega - E)) \in M$
using *sets UN* **by** *auto fastforce*
also have $\Omega - (D \cup (\Omega - E)) = E - D$
using *assms sets_into_space* **by** *auto*
finally show *?thesis* **.**
qed

lemma *Dynkin_systemI*:
assumes $\bigwedge A.\ A \in M \Longrightarrow A \subseteq \Omega$ $\Omega \in M$
assumes $\bigwedge A.\ A \in M \Longrightarrow \Omega - A \in M$
assumes $\bigwedge A.$ *disjoint_family* $A \Longrightarrow$ *range* $A \subseteq M$
$\Longrightarrow (\bigcup i{::}nat.\ A\ i) \in M$
shows *Dynkin_system* Ω M
using *assms* **by** (*auto simp*: *Dynkin_system_def Dynkin_system_axioms_def subset_class_def*)

lemma *Dynkin_systemI'*:
assumes *1*: $\bigwedge A.\ A \in M \Longrightarrow A \subseteq \Omega$
assumes *empty*: $\{\} \in M$
assumes *Diff*: $\bigwedge A.\ A \in M \Longrightarrow \Omega - A \in M$

assumes *2*: $\bigwedge A.$ *disjoint_family* $A \Longrightarrow$ *range* $A \subseteq M$
$\Longrightarrow (\bigcup i{::}nat.\ A\ i) \in M$
shows *Dynkin_system* $\Omega\ M$
proof −
from *Diff*[*OF empty*] **have** $\Omega \in M$ **by** *auto*
from *1 this Diff 2* **show** *?thesis*
by (*intro Dynkin_systemI*) *auto*
qed

lemma *Dynkin_system_trivial*:
shows *Dynkin_system* A (*Pow* A)
by (*rule Dynkin_systemI*) *auto*

lemma *sigma_algebra_imp_Dynkin_system*:
assumes *sigma_algebra* $\Omega\ M$ **shows** *Dynkin_system* $\Omega\ M$
proof −
interpret *sigma_algebra* $\Omega\ M$ **by** *fact*
show *?thesis* **using** *sets_into_space* **by** (*fastforce intro*!: *Dynkin_systemI*)
qed

Intersection sets systems

definition *Int_stable* :: $'a\ set\ set \Rightarrow bool$ **where**
Int_stable $M \longleftrightarrow (\forall\ a \in M.\ \forall\ b \in M.\ a \cap b \in M)$

lemma (**in** *algebra*) *Int_stable*: *Int_stable* M
unfolding *Int_stable_def* **by** *auto*

lemma *Int_stableI_image*:
$(\bigwedge i\ j.\ i \in I \Longrightarrow j \in I \Longrightarrow \exists k{\in}I.\ A\ i \cap A\ j = A\ k) \Longrightarrow$ *Int_stable* $(A\ `\ I)$
by (*auto simp*: *Int_stable_def image_def*)

lemma *Int_stableI*:
$(\bigwedge a\ b.\ a \in A \Longrightarrow b \in A \Longrightarrow a \cap b \in A) \Longrightarrow$ *Int_stable* A
unfolding *Int_stable_def* **by** *auto*

lemma *Int_stableD*:
Int_stable $M \Longrightarrow a \in M \Longrightarrow b \in M \Longrightarrow a \cap b \in M$
unfolding *Int_stable_def* **by** *auto*

lemma (**in** *Dynkin_system*) *sigma_algebra_eq_Int_stable*:
sigma_algebra $\Omega\ M \longleftrightarrow$ *Int_stable* M
proof
assume *sigma_algebra* $\Omega\ M$ **then show** *Int_stable* M
unfolding *sigma_algebra_def* **using** *algebra.Int_stable* **by** *auto*
next
assume *Int_stable* M
show *sigma_algebra* $\Omega\ M$
unfolding *sigma_algebra_disjoint_iff algebra_iff_Un*

proof (*intro conjI ballI allI impI*)
show $M \subseteq Pow\ (\Omega)$ **using** *sets_into_space* **by** *auto*
next
fix $A\ B$ **assume** $A \in M\ B \in M$
then have $A \cup B = \Omega - ((\Omega - A) \cap (\Omega - B))$
$\Omega - A \in M\ \Omega - B \in M$
using *sets_into_space* **by** *auto*
then show $A \cup B \in M$
using ‹*Int_stable M*› **unfolding** *Int_stable_def* **by** *auto*
qed *auto*
qed

Smallest Dynkin systems

definition *Dynkin* :: *'a set* ⇒ *'a set set* ⇒ *'a set set* **where**
Dynkin $\Omega\ M = (\bigcap \{D.\ Dynkin\_system\ \Omega\ D \wedge M \subseteq D\})$

lemma *Dynkin_system_Dynkin*:
assumes $M \subseteq Pow\ (\Omega)$
shows *Dynkin_system* Ω (*Dynkin* $\Omega\ M$)
proof (*rule Dynkin_systemI*)
fix A **assume** $A \in Dynkin\ \Omega\ M$
moreover
{ fix D **assume** $A \in D$ **and** *d*: *Dynkin_system* $\Omega\ D$
then have $A \subseteq \Omega$ **by** (*auto simp*: *Dynkin_system_def subset_class_def*) **}**
moreover have $\{D.\ Dynkin\_system\ \Omega\ D \wedge M \subseteq D\} \neq \{\}$
using *assms Dynkin_system_trivial* **by** *fastforce*
ultimately show $A \subseteq \Omega$
unfolding *Dynkin_def* **using** *assms*
by *auto*
next
show $\Omega \in Dynkin\ \Omega\ M$
unfolding *Dynkin_def* **using** *Dynkin_system.space* **by** *fastforce*
next
fix A **assume** $A \in Dynkin\ \Omega\ M$
then show $\Omega - A \in Dynkin\ \Omega\ M$
unfolding *Dynkin_def* **using** *Dynkin_system.compl* **by** *force*
next
fix A :: *nat* ⇒ *'a set*
assume *A*: *disjoint_family A range* $A \subseteq Dynkin\ \Omega\ M$
show $(\bigcup i.\ A\ i) \in Dynkin\ \Omega\ M$ **unfolding** *Dynkin_def*
proof (*simp*, *safe*)
fix D **assume** *Dynkin_system* $\Omega\ D\ M \subseteq D$
with A **have** $(\bigcup i.\ A\ i) \in D$
by (*intro Dynkin_system.UN*) (*auto simp*: *Dynkin_def*)
then show $(\bigcup i.\ A\ i) \in D$ **by** *auto*
qed
qed

lemma *Dynkin_Basic*[*intro*]: $A \in M \Longrightarrow A \in Dynkin\ \Omega\ M$
unfolding *Dynkin_def* **by** *auto*

lemma (**in** *Dynkin_system*) *restricted_Dynkin_system*:
assumes $D \in M$
shows *Dynkin_system* Ω $\{Q.\ Q \subseteq \Omega \wedge Q \cap D \in M\}$
proof (*rule Dynkin_systemI*, *simp_all*)
have $\Omega \cap D = D$
using ‹$D \in M$› *sets_into_space* **by** *auto*
then show $\Omega \cap D \in M$
using ‹$D \in M$› **by** *auto*
next
fix A **assume** $A \subseteq \Omega \wedge A \cap D \in M$
moreover have $(\Omega - A) \cap D = (\Omega - (A \cap D)) - (\Omega - D)$
by *auto*
ultimately show $(\Omega - A) \cap D \in M$
using ‹$D \in M$› **by** (*auto intro*: *diff*)
next
fix $A :: nat \Rightarrow 'a\ set$
assume *disjoint_family A range* $A \subseteq \{Q.\ Q \subseteq \Omega \wedge Q \cap D \in M\}$
then have $\bigwedge i.\ A\ i \subseteq \Omega$ *disjoint_family* $(\lambda i.\ A\ i \cap D)$
range $(\lambda i.\ A\ i \cap D) \subseteq M$ $(\bigcup x.\ A\ x) \cap D = (\bigcup x.\ A\ x \cap D)$
by ((*fastforce simp*: *disjoint_family_on_def*)+)
then show $(\bigcup x.\ A\ x) \subseteq \Omega \wedge (\bigcup x.\ A\ x) \cap D \in M$
by (*auto simp del*: *UN_simps*)
qed

lemma (**in** *Dynkin_system*) *Dynkin_subset*:
assumes $N \subseteq M$
shows *Dynkin* Ω $N \subseteq M$
proof −
have *Dynkin_system* Ω M **..**
then have *Dynkin_system* Ω M
using *assms* **unfolding** *Dynkin_system_def Dynkin_system_axioms_def subset_class_def* **by** *simp*
with ‹$N \subseteq M$› **show** *?thesis* **by** (*auto simp add*: *Dynkin_def*)
qed

lemma *sigma_eq_Dynkin*:
assumes *sets*: $M \subseteq Pow\ \Omega$
assumes *Int_stable M*
shows *sigma_sets* Ω M = *Dynkin* Ω M
proof −
have *Dynkin* Ω $M \subseteq$ *sigma_sets* (Ω) (M)
using *sigma_algebra_imp_Dynkin_system*
unfolding *Dynkin_def sigma_sets_least_sigma_algebra*[*OF sets*] **by** *auto*
moreover
interpret *Dynkin_system* Ω *Dynkin* Ω M
using *Dynkin_system_Dynkin*[*OF sets*] **.**

```
  have sigma_algebra Ω (Dynkin Ω M)
    unfolding sigma_algebra_eq_Int_stable Int_stable_def
  proof (intro ballI)
    fix A B assume A ∈ Dynkin Ω M B ∈ Dynkin Ω M
    let ?D = λE. {Q. Q ⊆ Ω ∧ Q ∩ E ∈ Dynkin Ω M}
    have M ⊆ ?D B
    proof
      fix E assume E ∈ M
      then have M ⊆ ?D E E ∈ Dynkin Ω M
        using sets_into_space ‹Int_stable M› by (auto simp: Int_stable_def)
      then have Dynkin Ω M ⊆ ?D E
        using restricted_Dynkin_system ‹E ∈ Dynkin Ω M›
        by (intro Dynkin_system.Dynkin_subset) simp_all
      then have B ∈ ?D E
        using ‹B ∈ Dynkin Ω M› by auto
      then have E ∩ B ∈ Dynkin Ω M
        by (subst Int_commute) simp
      then show E ∈ ?D B
        using sets ‹E ∈ M› by auto
    qed
    then have Dynkin Ω M ⊆ ?D B
      using restricted_Dynkin_system ‹B ∈ Dynkin Ω M›
      by (intro Dynkin_system.Dynkin_subset) simp_all
    then show A ∩ B ∈ Dynkin Ω M
      using ‹A ∈ Dynkin Ω M› sets_into_space by auto
  qed
  from sigma_algebra.sigma_sets_subset[OF this, of M]
  have sigma_sets (Ω) (M) ⊆ Dynkin Ω M by auto
  ultimately have sigma_sets (Ω) (M) = Dynkin Ω M by auto
  then show ?thesis
    by (auto simp: Dynkin_def)
qed

lemma (in Dynkin_system) Dynkin_idem:
  Dynkin Ω M = M
proof −
  have Dynkin Ω M = M
  proof
    show M ⊆ Dynkin Ω M
      using Dynkin_Basic by auto
    show Dynkin Ω M ⊆ M
      by (intro Dynkin_subset) auto
  qed
  then show ?thesis
    by (auto simp: Dynkin_def)
qed

lemma (in Dynkin_system) Dynkin_lemma:
  assumes Int_stable E
```

and E: $E \subseteq M$ $M \subseteq$ *sigma_sets* Ω E
shows *sigma_sets* Ω $E = M$
proof −
have $E \subseteq$ *Pow* Ω
using E *sets_into_space* **by** *force*
then have $*$: *sigma_sets* Ω E = *Dynkin* Ω E
using ‹*Int_stable* E› **by** (*rule sigma_eq_Dynkin*)
then have *Dynkin* Ω $E = M$
using *assms Dynkin_subset*[*OF* $E(1)$] **by** *simp*
with $*$ **show** *?thesis*
using *assms* **by** (*auto simp*: *Dynkin_def*)
qed

Induction rule for intersection-stable generators

The reason to introduce Dynkin-systems is the following induction rules for σ-algebras generated by a generator closed under intersection.

proposition *sigma_sets_induct_disjoint*[*consumes 3*, *case_names basic empty compl union*]:
assumes *Int_stable* G
and *closed*: $G \subseteq$ *Pow* Ω
and A: $A \in$ *sigma_sets* Ω G
assumes *basic*: $\bigwedge A.\ A \in G \Longrightarrow P\ A$
and *empty*: $P\ \{\}$
and *compl*: $\bigwedge A.\ A \in$ *sigma_sets* Ω $G \Longrightarrow P\ A \Longrightarrow P\ (\Omega - A)$
and *union*: $\bigwedge A.$ *disjoint_family* $A \Longrightarrow$ *range* $A \subseteq$ *sigma_sets* Ω $G \Longrightarrow (\bigwedge i.\ P\ (A\ i)) \Longrightarrow P\ (\bigcup i{::}nat.\ A\ i)$
shows $P\ A$
proof −
let *?D* = $\{\ A \in$ *sigma_sets* Ω $G.\ P\ A\ \}$
interpret *sigma_algebra* Ω *sigma_sets* Ω G
using *closed* **by** (*rule sigma_algebra_sigma_sets*)
from *compl*[*OF* _ *empty*] *closed* **have** *space*: $P\ \Omega$ **by** *simp*
interpret *Dynkin_system* Ω *?D*
by *standard* (*auto dest*: *sets_into_space intro*!: *space compl union*)
have *sigma_sets* Ω G = *?D*
by (*rule Dynkin_lemma*) (*auto simp*: *basic* ‹*Int_stable* G›)
with A **show** *?thesis* **by** *auto*
qed

6.1.2 Measure type

definition *positive* :: *'a set set* ⇒ (*'a set* ⇒ *ennreal*) ⇒ *bool* **where**
positive M μ ⟷ μ $\{\}$ = *0*

definition *countably_additive* :: *'a set set* ⇒ (*'a set* ⇒ *ennreal*) ⇒ *bool* **where**
countably_additive M f ⟷
(∀ A. *range* $A \subseteq M$ ⟶ *disjoint_family* A ⟶ $(\bigcup i.\ A\ i) \in M$ ⟶
$(\sum i.\ f\ (A\ i)) = f\ (\bigcup i.\ A\ i)$)

definition *measure_space* :: *'a set* ⇒ *'a set set* ⇒ (*'a set* ⇒ *ennreal*) ⇒ *bool*
where
measure_space Ω *A* μ ⟷
sigma_algebra Ω *A* ∧ *positive A* μ ∧ *countably_additive A* μ

typedef *'a measure* =
{(Ω::*'a set*, *A*, μ). (∀ *a*∈−*A*. μ *a* = *0*) ∧ *measure_space* Ω *A* μ }
proof
have *sigma_algebra UNIV* {{}, *UNIV*}
by (*auto simp*: *sigma_algebra_iff2*)
then show (*UNIV*, {{}, *UNIV*}, λ*A*. *0*) ∈ {(Ω, *A*, μ). (∀ *a*∈−*A*. μ *a* = *0*) ∧ *measure_space* Ω *A* μ}
by (*auto simp*: *measure_space_def positive_def countably_additive_def*)
qed

definition *space* :: *'a measure* ⇒ *'a set* **where**
space M = *fst* (*Rep_measure M*)

definition *sets* :: *'a measure* ⇒ *'a set set* **where**
sets M = *fst* (*snd* (*Rep_measure M*))

definition *emeasure* :: *'a measure* ⇒ *'a set* ⇒ *ennreal* **where**
emeasure M = *snd* (*snd* (*Rep_measure M*))

definition *measure* :: *'a measure* ⇒ *'a set* ⇒ *real* **where**
measure M A = *enn2real* (*emeasure M A*)

declare [[*coercion sets*]]

declare [[*coercion measure*]]

declare [[*coercion emeasure*]]

lemma *measure_space*: *measure_space* (*space M*) (*sets M*) (*emeasure M*)
by (*cases M*) (*auto simp*: *space_def sets_def emeasure_def Abs_measure_inverse*)

interpretation *sets*: *sigma_algebra space M sets M* **for** *M* :: *'a measure*
using *measure_space*[*of M*] **by** (*auto simp*: *measure_space_def*)

definition *measure_of* :: *'a set* ⇒ *'a set set* ⇒ (*'a set* ⇒ *ennreal*) ⇒ *'a measure*
where
measure_of Ω *A* μ =
Abs_measure (Ω, *if A* ⊆ *Pow* Ω *then sigma_sets* Ω *A else* {{}, Ω},
λ*a*. *if a* ∈ *sigma_sets* Ω *A* ∧ *measure_space* Ω (*sigma_sets* Ω *A*) μ *then* μ *a* *else 0*)

abbreviation *sigma* Ω *A* ≡ *measure_of* Ω *A* (λ*x*. *0*)

lemma *measure_space_0*: $A \subseteq Pow\ \Omega \Longrightarrow measure\_space\ \Omega\ (sigma\_sets\ \Omega\ A)\ (\lambda x.\ 0)$
unfolding *measure_space_def*
by (*auto intro*!: *sigma_algebra_sigma_sets simp*: *positive_def countably_additive_def*)

lemma *sigma_algebra_trivial*: *sigma_algebra* Ω $\{\{\}, \Omega\}$
by *unfold_locales*(*fastforce intro*: *exI*[**where** x=$\{\{\}\}$] *exI*[**where** x=$\{\Omega\}$])+

lemma *measure_space_0'*: *measure_space* Ω $\{\{\}, \Omega\}$ $(\lambda x.\ 0)$
by(*simp add*: *measure_space_def positive_def countably_additive_def sigma_algebra_trivial*)

lemma *measure_space_closed*:
assumes *measure_space* Ω M μ
shows $M \subseteq Pow\ \Omega$
proof −
interpret *sigma_algebra* Ω M **using** *assms* **by**(*simp add*: *measure_space_def*)
show *?thesis* **by**(*rule space_closed*)
qed

lemma (**in** *ring_of_sets*) *positive_cong_eq*:
$(\bigwedge a.\ a \in M \Longrightarrow \mu'\ a = \mu\ a) \Longrightarrow positive\ M\ \mu' = positive\ M\ \mu$
by (*auto simp add*: *positive_def*)

lemma (**in** *sigma_algebra*) *countably_additive_eq*:
$(\bigwedge a.\ a \in M \Longrightarrow \mu'\ a = \mu\ a) \Longrightarrow countably\_additive\ M\ \mu' = countably\_additive\ M\ \mu$
unfolding *countably_additive_def*
by (*intro arg_cong*[**where** f=*All*] *ext*) (*auto simp add*: *countably_additive_def subset_eq*)

lemma *measure_space_eq*:
assumes *closed*: $A \subseteq Pow\ \Omega$ **and** *eq*: $\bigwedge a.\ a \in sigma\_sets\ \Omega\ A \Longrightarrow \mu\ a = \mu'\ a$
shows *measure_space* Ω (*sigma_sets* Ω A) μ = *measure_space* Ω (*sigma_sets* Ω A) μ'
proof −
interpret *sigma_algebra* Ω *sigma_sets* Ω A **using** *closed* **by** (*rule sigma_algebra_sigma_sets*)
from *positive_cong_eq*[*OF eq*, *of* $\lambda i.\ i$] *countably_additive_eq*[*OF eq*, *of* $\lambda i.\ i$]
show *?thesis*
by (*auto simp*: *measure_space_def*)
qed

lemma *measure_of_eq*:
assumes *closed*: $A \subseteq Pow\ \Omega$ **and** *eq*: $(\bigwedge a.\ a \in sigma\_sets\ \Omega\ A \Longrightarrow \mu\ a = \mu'\ a)$
shows *measure_of* Ω A μ = *measure_of* Ω A μ'
proof −
have *measure_space* Ω (*sigma_sets* Ω A) μ = *measure_space* Ω (*sigma_sets* Ω A) μ'
using *assms* **by** (*rule measure_space_eq*)
with *eq* **show** *?thesis*

by (*auto simp add*: *measure_of_def intro*!: *arg_cong*[**where** *f*=*Abs_measure*])
qed

lemma
shows *space_measure_of_conv*: *space* (*measure_of* Ω *A* μ) = Ω (**is** *?space*)
and *sets_measure_of_conv*:
sets (*measure_of* Ω *A* μ) = (*if A* ⊆ *Pow* Ω *then sigma_sets* Ω *A else* {{}, Ω})
(**is** *?sets*)
and *emeasure_measure_of_conv*:
emeasure (*measure_of* Ω *A* μ) =
(λ*B*. *if B* ∈ *sigma_sets* Ω *A* ∧ *measure_space* Ω (*sigma_sets* Ω *A*) μ *then* μ *B*
else 0) (**is** *?emeasure*)
proof −
have *?space* ∧ *?sets* ∧ *?emeasure*
proof(*cases measure_space* Ω (*sigma_sets* Ω *A*) μ)
case *True*
from *measure_space_closed*[*OF this*] *sigma_sets_superset_generator*[*of A* Ω]
have *A* ⊆ *Pow* Ω **by** *simp*
hence *measure_space* Ω (*sigma_sets* Ω *A*) μ = *measure_space* Ω (*sigma_sets*
Ω *A*)
(λ*a*. *if a* ∈ *sigma_sets* Ω *A then* μ *a else 0*)
by(*rule measure_space_eq*) *auto*
with *True* ‹*A* ⊆ *Pow* Ω› **show** *?thesis*
by(*simp add*: *measure_of_def space_def sets_def emeasure_def Abs_measure_inverse*)
next
case *False* **thus** *?thesis*
by(*cases A* ⊆ *Pow* Ω)(*simp_all add*: *Abs_measure_inverse measure_of_def*
sets_def space_def emeasure_def measure_space_0 measure_space_0')
qed
thus *?space ?sets ?emeasure* **by** *simp_all*
qed

lemma [*simp*]:
assumes *A*: *A* ⊆ *Pow* Ω
shows *sets_measure_of*: *sets* (*measure_of* Ω *A* μ) = *sigma_sets* Ω *A*
and *space_measure_of*: *space* (*measure_of* Ω *A* μ) = Ω
using *assms*
by(*simp_all add*: *sets_measure_of_conv space_measure_of_conv*)

lemma *space_in_measure_of*[*simp*]: Ω ∈ *sets* (*measure_of* Ω *M* μ)
by (*subst sets_measure_of_conv*) (*auto simp*: *sigma_sets_top*)

lemma (**in** *sigma_algebra*) *sets_measure_of_eq*[*simp*]: *sets* (*measure_of* Ω *M* μ)
= *M*
using *space_closed* **by** (*auto intro*!: *sigma_sets_eq*)

lemma (**in** *sigma_algebra*) *space_measure_of_eq*[*simp*]: *space* (*measure_of* Ω *M*
μ) = Ω
by (*rule space_measure_of_conv*)

lemma *measure_of_subset*: $M \subseteq Pow\ \Omega \Longrightarrow M' \subseteq M \Longrightarrow sets\ (measure\_of\ \Omega\ M'\ \mu) \subseteq sets\ (measure\_of\ \Omega\ M\ \mu')$
by (*auto intro*!: *sigma_sets_subseteq*)

lemma *emeasure_sigma*: $emeasure\ (sigma\ \Omega\ A) = (\lambda x.\ 0)$
unfolding *measure_of_def emeasure_def*
by (*subst Abs_measure_inverse*)
(*auto simp*: *measure_space_def positive_def countably_additive_def*
intro!: *sigma_algebra_sigma_sets sigma_algebra_trivial*)

lemma *sigma_sets_mono''*:
assumes $A \in sigma\_sets\ C\ D$
assumes $B \subseteq D$
assumes $D \subseteq Pow\ C$
shows $sigma\_sets\ A\ B \subseteq sigma\_sets\ C\ D$
proof
fix x **assume** $x \in sigma\_sets\ A\ B$
thus $x \in sigma\_sets\ C\ D$
proof *induct*
case (*Basic a*) **with** *assms* **have** $a \in D$ **by** *auto*
thus *?case* **..**
next
case *Empty* **show** *?case* **by** (*rule sigma_sets.Empty*)
next
from *assms* **have** $A \in sets\ (sigma\ C\ D)$ **by** (*subst sets_measure_of*[*OF* ‹$D \subseteq Pow\ C$›])
moreover case (*Compl a*) **hence** $a \in sets\ (sigma\ C\ D)$ **by** (*subst sets_measure_of*[*OF* ‹$D \subseteq Pow\ C$›])
ultimately have $A - a \in sets\ (sigma\ C\ D)$ **..**
thus *?case* **by** (*subst* (*asm*) *sets_measure_of*[*OF* ‹$D \subseteq Pow\ C$›])
next
case (*Union a*)
thus *?case* **by** (*intro sigma_sets.Union*)
qed
qed

lemma *in_measure_of*[*intro*, *simp*]: $M \subseteq Pow\ \Omega \Longrightarrow A \in M \Longrightarrow A \in sets\ (measure\_of\ \Omega\ M\ \mu)$
by *auto*

lemma *space_empty_iff*: $space\ N = \{\} \longleftrightarrow sets\ N = \{\{\}\}$
by (*metis Pow_empty Sup_bot_conv*(*1*) *cSup_singleton empty_iff*
sets.sigma_sets_eq sets.space_closed sigma_sets_top subset_singletonD)

Constructing simple *'a measure*

proposition *emeasure_measure_of*:
assumes *M*: $M = measure\_of\ \Omega\ A\ \mu$

assumes *ms*: $A \subseteq Pow\ \Omega$ *positive* (*sets M*) μ *countably_additive* (*sets M*) μ
assumes *X*: $X \in sets\ M$
shows *emeasure M X* = $\mu\ X$
proof −
interpret *sigma_algebra* Ω *sigma_sets* $\Omega\ A$ **by** (*rule sigma_algebra_sigma_sets*) *fact*
have *measure_space* Ω (*sigma_sets* $\Omega\ A$) μ
using *ms M* **by** (*simp add*: *measure_space_def sigma_algebra_sigma_sets*)
thus *?thesis* **using** *X ms*
by(*simp add*: *M emeasure_measure_of_conv sets_measure_of_conv*)
qed

lemma *emeasure_measure_of_sigma*:
assumes *ms*: *sigma_algebra* Ω *M positive M* μ *countably_additive M* μ
assumes *A*: $A \in M$
shows *emeasure* (*measure_of* Ω *M* μ) *A* = $\mu\ A$
proof −
interpret *sigma_algebra* Ω *M* **by** *fact*
have *measure_space* Ω (*sigma_sets* Ω *M*) μ
using *ms sigma_sets_eq* **by** (*simp add*: *measure_space_def*)
thus *?thesis* **by**(*simp add*: *emeasure_measure_of_conv A*)
qed

lemma *measure_cases*[*cases type*: *measure*]:
obtains (*measure*) $\Omega\ A\ \mu$ **where** *x* = *Abs_measure* (Ω, *A*, μ) $\forall a \in -A.\ \mu\ a = 0$ *measure_space* $\Omega\ A\ \mu$
by *atomize_elim* (*cases x*, *auto*)

lemma *sets_le_imp_space_le*: *sets A* $\subseteq$ *sets B* $\Longrightarrow$ *space A* $\subseteq$ *space B*
by (*auto dest*: *sets.sets_into_space*)

lemma *sets_eq_imp_space_eq*: *sets M* = *sets M′* $\Longrightarrow$ *space M* = *space M′*
by (*auto intro*!: *antisym sets_le_imp_space_le*)

lemma *emeasure_notin_sets*: *A* $\notin$ *sets M* $\Longrightarrow$ *emeasure M A* = *0*
by (*cases M*) (*auto simp*: *sets_def emeasure_def Abs_measure_inverse measure_space_def*)

lemma *emeasure_neq_0_sets*: *emeasure M A* $\neq$ *0* $\Longrightarrow$ *A* $\in$ *sets M*
using *emeasure_notin_sets*[*of A M*] **by** *blast*

lemma *measure_notin_sets*: *A* $\notin$ *sets M* $\Longrightarrow$ *measure M A* = *0*
by (*simp add*: *measure_def emeasure_notin_sets zero_ennreal.rep_eq*)

lemma *measure_eqI*:
fixes *M N* :: *'a measure*
assumes *sets M* = *sets N* **and** *eq*: $\bigwedge A.\ A \in sets\ M \Longrightarrow emeasure\ M\ A = emeasure\ N\ A$
shows *M* = *N*

proof (*cases M N rule*: *measure_cases*[*case_product measure_cases*])
case (*measure_measure* Ω *A* μ Ω′ *A*′ μ′)
interpret *M*: *sigma_algebra* Ω *A* **using** *measure_measure* **by** (*auto simp*: *measure_space_def*)
interpret *N*: *sigma_algebra* Ω′ *A*′ **using** *measure_measure* **by** (*auto simp*: *measure_space_def*)
have *A* = *sets M A*′ = *sets N*
using *measure_measure* **by** (*simp_all add*: *sets_def Abs_measure_inverse*)
with ‹*sets M* = *sets N*› **have** *AA*′: *A* = *A*′ **by** *simp*
moreover from *M.top N.top M.space_closed N.space_closed AA*′ **have** Ω = Ω′
by *auto*
moreover { **fix** *B* **have** μ *B* = μ′ *B*
proof *cases*
assume *B* ∈ *A*
with *eq* ‹*A* = *sets M*› **have** *emeasure M B* = *emeasure N B* **by** *simp*
with *measure_measure* **show** μ *B* = μ′ *B*
by (*simp add*: *emeasure_def Abs_measure_inverse*)
next
assume *B* ∉ *A*
with ‹*A* = *sets M*› ‹*A*′ = *sets N*› ‹*A* = *A*′› **have** *B* ∉ *sets M B* ∉ *sets N*
by *auto*
then have *emeasure M B* = *0 emeasure N B* = *0*
by (*simp_all add*: *emeasure_notin_sets*)
with *measure_measure* **show** μ *B* = μ′ *B*
by (*simp add*: *emeasure_def Abs_measure_inverse*)
qed }
then have μ = μ′ **by** *auto*
ultimately show *M* = *N*
by (*simp add*: *measure_measure*)
qed

lemma *sigma_eqI*:
assumes [*simp*]: *M* ⊆ *Pow* Ω *N* ⊆ *Pow* Ω *sigma_sets* Ω *M* = *sigma_sets* Ω *N*
shows *sigma* Ω *M* = *sigma* Ω *N*
by (*rule measure_eqI*) (*simp_all add*: *emeasure_sigma*)

Measurable functions

definition *measurable* :: *'a measure* ⇒ *'b measure* ⇒ (*'a* ⇒ *'b*) *set*
(**infixr** →$_M$ *60*) **where**
measurable A B = {*f* ∈ *space A* → *space B*. ∀ *y* ∈ *sets B*. *f* −‘ *y* ∩ *space A* ∈ *sets A*}

lemma *measurableI*:
(⋀*x*. *x* ∈ *space M* ⟹ *f x* ∈ *space N*) ⟹ (⋀*A*. *A* ∈ *sets N* ⟹ *f* −‘ *A* ∩ *space M* ∈ *sets M*) ⟹
f ∈ *measurable M N*
by (*auto simp*: *measurable_def*)

lemma *measurable_space*:
 $f \in$ *measurable* $M\ A \Longrightarrow x \in$ *space* $M \Longrightarrow f\ x \in$ *space* A
 unfolding *measurable_def* **by** *auto*

lemma *measurable_sets*:
 $f \in$ *measurable* $M\ A \Longrightarrow S \in$ *sets* $A \Longrightarrow f -` S \cap$ *space* $M \in$ *sets* M
 unfolding *measurable_def* **by** *auto*

lemma *measurable_sets_Collect*:
 assumes *f*: $f \in$ *measurable* $M\ N$ **and** *P*: $\{x \in$ *space* $N.\ P\ x\} \in$ *sets* N **shows** $\{x \in$ *space* $M.\ P\ (f\ x)\} \in$ *sets* M
proof −
 have $f -` \{x \in$ *space* $N.\ P\ x\} \cap$ *space* $M = \{x \in$ *space* $M.\ P\ (f\ x)\}$
  using *measurable_space*[*OF f*] **by** *auto*
 with *measurable_sets*[*OF f P*] **show** *?thesis*
  by *simp*
qed

lemma *measurable_sigma_sets*:
 assumes *B*: *sets* $N =$ *sigma_sets* $\Omega\ A\ A \subseteq$ *Pow* Ω
  and *f*: $f \in$ *space* $M \to \Omega$
  and *ba*: $\bigwedge y.\ y \in A \Longrightarrow (f -` y) \cap$ *space* $M \in$ *sets* M
 shows $f \in$ *measurable* $M\ N$
proof −
 interpret *A*: *sigma_algebra* Ω *sigma_sets* $\Omega\ A$ **using** *B*(*2*) **by** (*rule sigma_algebra_sigma_sets*)
 from *B sets.top*[*of N*] *A.top sets.space_closed*[*of N*] *A.space_closed* **have** Ω: Ω = *space* N **by** *force*

 { fix *X* **assume** $X \in$ *sigma_sets* $\Omega\ A$
  then have $f -` X \cap$ *space* $M \in$ *sets* $M \wedge X \subseteq \Omega$
   proof *induct*
    case (*Basic a*) **then show** *?case*
     by (*auto simp add*: *ba*) (*metis B*(*2*) *subsetD PowD*)
   next
    case (*Compl a*)
    have [*simp*]: $f -` \Omega \cap$ *space* $M =$ *space* M
     by (*auto simp add*: *funcset_mem* [*OF f*])
    then show *?case*
     by (*auto simp add*: *vimage_Diff Diff_Int_distrib2 sets.compl_sets Compl*)
   next
    case (*Union a*)
    then show *?case*
     by (*simp add*: *vimage_UN*, *simp only*: *UN_extend_simps*(*4*)) *blast*
   qed *auto* **}**
 with *f* **show** *?thesis*
  by (*auto simp add*: *measurable_def B* Ω)
qed

lemma *measurable_measure_of*:

assumes *B*: $N \subseteq Pow\ \Omega$
and *f*: $f \in space\ M \to \Omega$
and *ba*: $\bigwedge y.\ y \in N \Longrightarrow (f\ -`\ y) \cap space\ M \in sets\ M$
shows $f \in measurable\ M\ (measure\_of\ \Omega\ N\ \mu)$
proof −
have $sets\ (measure\_of\ \Omega\ N\ \mu) = sigma\_sets\ \Omega\ N$
using *B* **by** (*rule sets_measure_of*)
from *this assms* **show** *?thesis* **by** (*rule measurable_sigma_sets*)
qed

lemma *measurable_iff_measure_of*:
assumes $N \subseteq Pow\ \Omega\ f \in space\ M \to \Omega$
shows $f \in measurable\ M\ (measure\_of\ \Omega\ N\ \mu) \longleftrightarrow (\forall A{\in}N.\ f\ -`\ A \cap space\ M \in sets\ M)$
by (*metis assms in_measure_of measurable_measure_of assms measurable_sets*)

lemma *measurable_cong_sets*:
assumes *sets*: $sets\ M = sets\ M'\ sets\ N = sets\ N'$
shows $measurable\ M\ N = measurable\ M'\ N'$
using *sets*[*THEN sets_eq_imp_space_eq*] *sets* **by** (*simp add*: *measurable_def*)

lemma *measurable_cong*:
assumes $\bigwedge w.\ w \in space\ M \Longrightarrow f\ w = g\ w$
shows $f \in measurable\ M\ M' \longleftrightarrow g \in measurable\ M\ M'$
unfolding *measurable_def* **using** *assms*
by (*simp cong*: *vimage_inter_cong Pi_cong*)

lemma *measurable_cong'*:
assumes $\bigwedge w.\ w \in space\ M =simp=> f\ w = g\ w$
shows $f \in measurable\ M\ M' \longleftrightarrow g \in measurable\ M\ M'$
unfolding *measurable_def* **using** *assms*
by (*simp cong*: *vimage_inter_cong Pi_cong add*: *simp_implies_def*)

lemma *measurable_cong_simp*:
$M = N \Longrightarrow M' = N' \Longrightarrow (\bigwedge w.\ w \in space\ M \Longrightarrow f\ w = g\ w) \Longrightarrow$
$f \in measurable\ M\ M' \longleftrightarrow g \in measurable\ N\ N'$
by (*metis measurable_cong*)

lemma *measurable_compose*:
assumes *f*: $f \in measurable\ M\ N$ **and** *g*: $g \in measurable\ N\ L$
shows $(\lambda x.\ g\ (f\ x)) \in measurable\ M\ L$
proof −
have $\bigwedge A.\ (\lambda x.\ g\ (f\ x))\ -`\ A \cap space\ M = f\ -`\ (g\ -`\ A \cap space\ N) \cap space\ M$
using *measurable_space*[*OF f*] **by** *auto*
with *measurable_space*[*OF f*] *measurable_space*[*OF g*] **show** *?thesis*
by (*auto intro*: *measurable_sets*[*OF f*] *measurable_sets*[*OF g*]
simp del: *vimage_Int simp add*: *measurable_def*)
qed

lemma *measurable_comp*:
f ∈ *measurable M N* ⟹ *g* ∈ *measurable N L* ⟹ *g* ∘ *f* ∈ *measurable M L*
using *measurable_compose*[*of f M N g L*] **by** (*simp add*: *comp_def*)

lemma *measurable_const*:
c ∈ *space M'* ⟹ (λ*x*. *c*) ∈ *measurable M M'*
by (*auto simp add*: *measurable_def*)

lemma *measurable_ident*: *id* ∈ *measurable M M*
by (*auto simp add*: *measurable_def*)

lemma *measurable_id*: (λ*x*. *x*) ∈ *measurable M M*
by (*simp add*: *measurable_def*)

lemma *measurable_ident_sets*:
assumes *eq*: *sets M* = *sets M'* **shows** (λ*x*. *x*) ∈ *measurable M M'*
using *measurable_ident*[*of M*]
unfolding *id_def measurable_def eq sets_eq_imp_space_eq*[*OF eq*] **.**

lemma *sets_Least*:
assumes *meas*: ⋀*i*::*nat*. {*x*∈*space M*. *P i x*} ∈ *M*
shows (λ*x*. *LEAST j*. *P j x*) −‘ *A* ∩ *space M* ∈ *sets M*
proof −
{ fix *i* **have** (λ*x*. *LEAST j*. *P j x*) −‘ {*i*} ∩ *space M* ∈ *sets M*
proof *cases*
assume *i*: (*LEAST j*. *False*) = *i*
have (λ*x*. *LEAST j*. *P j x*) −‘ {*i*} ∩ *space M* =
{*x*∈*space M*. *P i x*} ∩ (*space M* − (⋃*j*<*i*. {*x*∈*space M*. *P j x*})) ∪ (*space M* − (⋃ *i*. {*x*∈*space M*. *P i x*}))
by (*simp add*: *set_eq_iff*, *safe*)
(*insert i*, *auto dest*: *Least_le intro*: *LeastI intro*!: *Least_equality*)
with *meas* **show** *?thesis*
by (*auto intro*!: *sets.Int*)
next
assume *i*: (*LEAST j*. *False*) ≠ *i*
then have (λ*x*. *LEAST j*. *P j x*) −‘ {*i*} ∩ *space M* =
{*x*∈*space M*. *P i x*} ∩ (*space M* − (⋃*j*<*i*. {*x*∈*space M*. *P j x*}))
proof (*simp add*: *set_eq_iff*, *safe*)
fix *x* **assume** *neq*: (*LEAST j*. *False*) ≠ (*LEAST j*. *P j x*)
have ∃ *j*. *P j x*
by (*rule ccontr*) (*insert neq*, *auto*)
then show *P* (*LEAST j*. *P j x*) *x* **by** (*rule LeastI_ex*)
qed (*auto dest*: *Least_le intro*!: *Least_equality*)
with *meas* **show** *?thesis*
by *auto*
qed }
then have (⋃*i*∈*A*. (λ*x*. *LEAST j*. *P j x*) −‘ {*i*} ∩ *space M*) ∈ *sets M*
by (*intro sets.countable_UN*) *auto*
moreover have (⋃*i*∈*A*. (λ*x*. *LEAST j*. *P j x*) −‘ {*i*} ∩ *space M*) =

$(\lambda x.\ LEAST\ j.\ P\ j\ x) -` A \cap space\ M$ **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *measurable_mono1*:
$M' \subseteq Pow\ \Omega \Longrightarrow M \subseteq M' \Longrightarrow$
measurable (*measure_of* $\Omega\ M\ \mu$) $N \subseteq$ *measurable* (*measure_of* $\Omega\ M'\ \mu'$) N
using *measure_of_subset*[*of* $M'\ \Omega\ M$] **by** (*auto simp add*: *measurable_def*)

Counting space

definition *count_space* :: *'a set* ⇒ *'a measure* **where**
count_space Ω = *measure_of* Ω (*Pow* Ω) ($\lambda A.$ *if finite A then of_nat* (*card A*) *else* ∞)

lemma
shows *space_count_space*[*simp*]: *space* (*count_space* Ω) = Ω
and *sets_count_space*[*simp*]: *sets* (*count_space* Ω) = *Pow* Ω
using *sigma_sets_into_sp*[*of Pow* Ω Ω]
by (*auto simp*: *count_space_def*)

lemma *measurable_count_space_eq1*[*simp*]:
$f \in$ *measurable* (*count_space A*) $M \longleftrightarrow f \in A \to$ *space M*
unfolding *measurable_def* **by** *simp*

lemma *measurable_compose_countable'*:
assumes *f*: $\bigwedge i.\ i \in I \Longrightarrow (\lambda x.\ f\ i\ x) \in$ *measurable M N*
and *g*: $g \in$ *measurable M* (*count_space I*) **and** *I*: *countable I*
shows $(\lambda x.\ f\ (g\ x)\ x) \in$ *measurable M N*
unfolding *measurable_def*
proof *safe*
fix *x* **assume** $x \in$ *space M* **then show** $f\ (g\ x)\ x \in$ *space N*
using *measurable_space*[*OF f*] *g*[*THEN measurable_space*] **by** *auto*
next
fix *A* **assume** *A*: $A \in$ *sets N*
have $(\lambda x.\ f\ (g\ x)\ x) -` A \cap space\ M = (\bigcup i \in I.\ (g -` \{i\} \cap space\ M) \cap (f\ i -` A \cap space\ M))$
using *measurable_space*[*OF g*] **by** *auto*
also have $\ldots \in$ *sets M*
using *f*[*THEN measurable_sets*, *OF _ A*] *g*[*THEN measurable_sets*]
by (*auto intro*!: *sets.countable_UN' I intro*: *sets.Int*[*OF measurable_sets measurable_sets*])
finally show $(\lambda x.\ f\ (g\ x)\ x) -` A \cap space\ M \in$ *sets M* **.**
qed

lemma *measurable_count_space_eq_countable*:
assumes *countable A*
shows $f \in$ *measurable M* (*count_space A*) $\longleftrightarrow (f \in space\ M \to A \wedge (\forall a \in A.\ f -` \{a\} \cap space\ M \in sets\ M))$

proof −
{ fix X **assume** $X \subseteq A$ $f \in$ *space* $M \to A$
with ‹*countable* A› **have** $f -` X \cap$ *space* $M = (\bigcup a{\in}X.\ f -` \{a\} \cap$ *space* $M)$ *countable* X
by (*auto dest*: *countable_subset*)
moreover assume $\forall\, a{\in}A.\ f -` \{a\} \cap$ *space* $M \in$ *sets* M
ultimately have $f -` X \cap$ *space* $M \in$ *sets* M
using ‹$X \subseteq A$› **by** (*auto intro*!: *sets.countable_UN'* *simp del*: *UN_simps*) **}**
then show *?thesis*
unfolding *measurable_def* **by** *auto*
qed

lemma *measurable_count_space_eq2*:
finite $A \Longrightarrow f \in$ *measurable* M (*count_space* A) $\longleftrightarrow$ ($f \in$ *space* $M \to A \wedge$ ($\forall\, a{\in}A.\ f -` \{a\} \cap$ *space* $M \in$ *sets* M))
by (*intro measurable_count_space_eq_countable countable_finite*)

lemma *measurable_count_space_eq2_countable*:
fixes $f :: 'a \Rightarrow 'c{::}$*countable*
shows $f \in$ *measurable* M (*count_space* A) $\longleftrightarrow$ ($f \in$ *space* $M \to A \wedge (\forall\, a{\in}A.\ f -` \{a\} \cap$ *space* $M \in$ *sets* M))
by (*intro measurable_count_space_eq_countable countableI_type*)

lemma *measurable_compose_countable*:
assumes *f*: $\bigwedge i{::}'i{::}$*countable*. $(\lambda x.\ f\ i\ x) \in$ *measurable* $M\ N$ **and** *g*: $g \in$ *measurable* M (*count_space UNIV*)
shows $(\lambda x.\ f\ (g\ x)\ x) \in$ *measurable* $M\ N$
by (*rule measurable_compose_countable'*[*OF assms*]) *auto*

lemma *measurable_count_space_const*:
$(\lambda x.\ c) \in$ *measurable* M (*count_space UNIV*)
by (*simp add*: *measurable_const*)

lemma *measurable_count_space*:
$f \in$ *measurable* (*count_space* A) (*count_space UNIV*)
by *simp*

lemma *measurable_compose_rev*:
assumes *f*: $f \in$ *measurable* $L\ N$ **and** *g*: $g \in$ *measurable* $M\ L$
shows $(\lambda x.\ f\ (g\ x)) \in$ *measurable* $M\ N$
using *measurable_compose*[*OF g f*] **.**

lemma *measurable_empty_iff*:
space $N = \{\} \Longrightarrow f \in$ *measurable* $M\ N \longleftrightarrow$ *space* $M = \{\}$
by (*auto simp add*: *measurable_def Pi_iff*)

Extend measure

definition *extend_measure* :: *'a set* ⇒ *'b set* ⇒ (*'b* ⇒ *'a set*) ⇒ (*'b* ⇒ *ennreal*) ⇒ *'a measure*
where
extend_measure Ω *I G* μ =
(*if* (∃μ'. (∀*i*∈*I*. μ' (*G i*) = μ *i*) ∧ *measure_space* Ω (*sigma_sets* Ω (*G*'*I*)) μ') ∧ ¬ (∀*i*∈*I*. μ *i* = *0*)
then measure_of Ω (*G*'*I*) (*SOME* μ'. (∀*i*∈*I*. μ' (*G i*) = μ *i*) ∧ *measure_space* Ω (*sigma_sets* Ω (*G*'*I*)) μ')
else measure_of Ω (*G*'*I*) (λ_. *0*))

lemma *space_extend_measure*: *G* ' *I* ⊆ *Pow* Ω ⟹ *space* (*extend_measure* Ω *I G* μ) = Ω
unfolding *extend_measure_def* **by** *simp*

lemma *sets_extend_measure*: *G* ' *I* ⊆ *Pow* Ω ⟹ *sets* (*extend_measure* Ω *I G* μ) = *sigma_sets* Ω (*G*'*I*)
unfolding *extend_measure_def* **by** *simp*

lemma *emeasure_extend_measure*:
assumes *M*: *M* = *extend_measure* Ω *I G* μ
and *eq*: ⋀*i*. *i* ∈ *I* ⟹ μ' (*G i*) = μ *i*
and *ms*: *G* ' *I* ⊆ *Pow* Ω *positive* (*sets M*) μ' *countably_additive* (*sets M*) μ'
and *i* ∈ *I*
shows *emeasure M* (*G i*) = μ *i*
proof *cases*
assume ∗: (∀*i*∈*I*. μ *i* = *0*)
with *M* **have** *M_eq*: *M* = *measure_of* Ω (*G*'*I*) (λ_. *0*)
by (*simp add*: *extend_measure_def*)
from *measure_space_0*[*OF ms*(*1*)] *ms* ‹*i*∈*I*›
have *emeasure M* (*G i*) = *0*
by (*intro emeasure_measure_of*[*OF M_eq*]) (*auto simp add*: *M measure_space_def sets_extend_measure*)
with ‹*i*∈*I*› ∗ **show** *?thesis*
by *simp*
next
define *P* **where** *P* μ' ⟷ (∀*i*∈*I*. μ' (*G i*) = μ *i*) ∧ *measure_space* Ω (*sigma_sets* Ω (*G*'*I*)) μ' **for** μ'
assume ¬ (∀*i*∈*I*. μ *i* = *0*)
moreover
have *measure_space* (*space M*) (*sets M*) μ'
using *ms* **unfolding** *measure_space_def* **by** *auto standard*
with *ms eq* **have** ∃μ'. *P* μ'
unfolding *P_def*
by (*intro exI*[*of* _ μ']) (*auto simp add*: *M space_extend_measure sets_extend_measure*)
ultimately have *M_eq*: *M* = *measure_of* Ω (*G*'*I*) (*Eps P*)
by (*simp add*: *M extend_measure_def P_def*[*symmetric*])

from ‹∃μ'. *P* μ'› **have** *P*: *P* (*Eps P*) **by** (*rule someI_ex*)

show *emeasure M* (*G i*) = *μ i*
proof (*subst emeasure_measure_of*[*OF M_eq*])
have *sets_M*: *sets M* = *sigma_sets* Ω (*G*‘*I*)
using *M_eq ms* **by** (*auto simp*: *sets_extend_measure*)
then show *G i* ∈ *sets M* **using** ‹*i* ∈ *I*› **by** *auto*
show *positive* (*sets M*) (*Eps P*) *countably_additive* (*sets M*) (*Eps P*) *Eps P* (*G i*) = *μ i*
using *P* ‹*i*∈*I*› **by** (*auto simp add*: *sets_M measure_space_def P_def*)
qed *fact*
qed

lemma *emeasure_extend_measure_Pair*:
assumes *M*: *M* = *extend_measure* Ω {(*i*, *j*). *I i j*} (λ(*i*, *j*). *G i j*) (λ(*i*, *j*). *μ i j*)
and *eq*: ⋀*i j*. *I i j* ⟹ *μ*′ (*G i j*) = *μ i j*
and *ms*: ⋀*i j*. *I i j* ⟹ *G i j* ∈ *Pow* Ω *positive* (*sets M*) *μ*′ *countably_additive* (*sets M*) *μ*′
and *I i j*
shows *emeasure M* (*G i j*) = *μ i j*
using *emeasure_extend_measure*[*OF M _ _ ms*(*2*,*3*), *of* (*i*,*j*)] *eq ms*(*1*) ‹*I i j*›
by (*auto simp*: *subset_eq*)

6.1.3 The smallest σ-algebra regarding a function

definition *vimage_algebra* :: *'a set* ⇒ (*'a* ⇒ *'b*) ⇒ *'b measure* ⇒ *'a measure* **where**
vimage_algebra X f M = *sigma X* {*f* −‘ *A* ∩ *X* | *A*. *A* ∈ *sets M*}

lemma *space_vimage_algebra*[*simp*]: *space* (*vimage_algebra X f M*) = *X*
unfolding *vimage_algebra_def* **by** (*rule space_measure_of*) *auto*

lemma *sets_vimage_algebra*: *sets* (*vimage_algebra X f M*) = *sigma_sets X* {*f* −‘ *A* ∩ *X* | *A*. *A* ∈ *sets M*}
unfolding *vimage_algebra_def* **by** (*rule sets_measure_of*) *auto*

lemma *sets_vimage_algebra2*:
f ∈ *X* → *space M* ⟹ *sets* (*vimage_algebra X f M*) = {*f* −‘ *A* ∩ *X* | *A*. *A* ∈ *sets M*}
using *sigma_sets_vimage_commute*[*of f X space M sets M*]
unfolding *sets_vimage_algebra sets.sigma_sets_eq* **by** *simp*

lemma *sets_vimage_algebra_cong*: *sets M* = *sets N* ⟹ *sets* (*vimage_algebra X f M*) = *sets* (*vimage_algebra X f N*)
by (*simp add*: *sets_vimage_algebra*)

lemma *vimage_algebra_cong*:
assumes *X* = *Y*
assumes ⋀*x*. *x* ∈ *Y* ⟹ *f x* = *g x*
assumes *sets M* = *sets N*
shows *vimage_algebra X f M* = *vimage_algebra Y g N*

by (*auto simp*: *vimage_algebra_def assms intro*!: *arg_cong2*[**where** *f*=*sigma*])

lemma *in_vimage_algebra*: $A \in sets\ M \Longrightarrow f -` A \cap X \in sets\ (vimage\_algebra\ X\ f\ M)$
by (*auto simp*: *vimage_algebra_def*)

lemma *sets_image_in_sets*:
assumes *N*: *space N* = *X*
assumes *f*: $f \in measurable\ N\ M$
shows $sets\ (vimage\_algebra\ X\ f\ M) \subseteq sets\ N$
unfolding *sets_vimage_algebra N*[*symmetric*]
by (*rule sets.sigma_sets_subset*) (*auto intro*!: *measurable_sets f*)

lemma *measurable_vimage_algebra1*: $f \in X \to space\ M \Longrightarrow f \in measurable\ (vimage\_algebra\ X\ f\ M)\ M$
unfolding *measurable_def* **by** (*auto intro*: *in_vimage_algebra*)

lemma *measurable_vimage_algebra2*:
assumes *g*: $g \in space\ N \to X$ **and** *f*: $(\lambda x.\ f\ (g\ x)) \in measurable\ N\ M$
shows $g \in measurable\ N\ (vimage\_algebra\ X\ f\ M)$
unfolding *vimage_algebra_def*
proof (*rule measurable_measure_of*)
fix *A* **assume** $A \in \{f -` A \cap X \mid A.\ A \in sets\ M\}$
then obtain *Y* **where** *Y*: $Y \in sets\ M$ **and** *A*: $A = f -` Y \cap X$
by *auto*
then have $g -` A \cap space\ N = (\lambda x.\ f\ (g\ x)) -` Y \cap space\ N$
using *g* **by** *auto*
also have $\ldots \in sets\ N$
using *f Y* **by** (*rule measurable_sets*)
finally show $g -` A \cap space\ N \in sets\ N$ **.**
qed (*insert g*, *auto*)

lemma *vimage_algebra_sigma*:
assumes *X*: $X \subseteq Pow\ \Omega'$ **and** *f*: $f \in \Omega \to \Omega'$
shows $vimage\_algebra\ \Omega\ f\ (sigma\ \Omega'\ X) = sigma\ \Omega\ \{f -` A \cap \Omega \mid A.\ A \in X\ \}$
(**is** *?V* = *?S*)
proof (*rule measure_eqI*)
have Ω: $\{f -` A \cap \Omega \mid A.\ A \in X\} \subseteq Pow\ \Omega$ **by** *auto*
show *sets ?V* = *sets ?S*
using *sigma_sets_vimage_commute*[*OF f*, *of X*]
by (*simp add*: *space_measure_of_conv f sets_vimage_algebra2* Ω *X*)
qed (*simp add*: *vimage_algebra_def emeasure_sigma*)

lemma *vimage_algebra_vimage_algebra_eq*:
assumes ∗: $f \in X \to Y\ g \in Y \to space\ M$
shows $vimage\_algebra\ X\ f\ (vimage\_algebra\ Y\ g\ M) = vimage\_algebra\ X\ (\lambda x.\ g\ (f\ x))\ M$
(**is** *?VV* = *?V*)
proof (*rule measure_eqI*)

have $(\lambda x.\ g\ (f\ x)) \in X \to \mathit{space}\ M\ \bigwedge A.\ A \cap f\ -`\ Y \cap X = A \cap X$
using $*$ **by** *auto*
with $*$ **show** *sets ?VV = sets ?V*
by (*simp add*: *sets_vimage_algebra2 vimage_comp comp_def flip*: *ex_simps*)
qed (*simp add*: *vimage_algebra_def emeasure_sigma*)

Restricted Space Sigma Algebra

definition *restrict_space* :: *'a measure* ⇒ *'a set* ⇒ *'a measure* **where**
restrict_space M Ω = measure_of (*Ω ∩ space M*) (((∩) *Ω*) ` *sets M*) (*emeasure M*)

lemma *space_restrict_space*: *space* (*restrict_space M Ω*) = *Ω ∩ space M*
using *sets.sets_into_space* **unfolding** *restrict_space_def* **by** (*subst space_measure_of*) *auto*

lemma *space_restrict_space2* [*simp*]: *Ω ∈ sets M* ⟹ *space* (*restrict_space M Ω*) = *Ω*
by (*simp add*: *space_restrict_space sets.sets_into_space*)

lemma *sets_restrict_space*: *sets* (*restrict_space M Ω*) = ((∩) *Ω*) ` *sets M*
unfolding *restrict_space_def*
proof (*subst sets_measure_of*)
show (∩) *Ω* ` *sets M* ⊆ *Pow* (*Ω ∩ space M*)
by (*auto dest*: *sets.sets_into_space*)
have *sigma_sets* (*Ω ∩ space M*) {((λ*x*. *x*) −` *X*) ∩ (*Ω ∩ space M*) | *X*. *X ∈ sets M*} =
(λ*X*. *X ∩* (*Ω ∩ space M*)) ` *sets M*
by (*subst sigma_sets_vimage_commute*[*symmetric*, **where** *Ω′ = space M*])
(*auto simp add*: *sets.sigma_sets_eq*)
moreover have {((λ*x*. *x*) −` *X*) ∩ (*Ω ∩ space M*) | *X*. *X ∈ sets M*} = (λ*X*. *X ∩* (*Ω ∩ space M*)) ` *sets M*
by *auto*
moreover have (λ*X*. *X ∩* (*Ω ∩ space M*)) ` *sets M* = ((∩) *Ω*) ` *sets M*
by (*intro image_cong*) (*auto dest*: *sets.sets_into_space*)
ultimately show *sigma_sets* (*Ω ∩ space M*) ((∩) *Ω* ` *sets M*) = (∩) *Ω* ` *sets M*
by *simp*
qed

lemma *restrict_space_sets_cong*:
A = B ⟹ *sets M = sets N* ⟹ *sets* (*restrict_space M A*) = *sets* (*restrict_space N B*)
by (*auto simp*: *sets_restrict_space*)

lemma *sets_restrict_space_count_space* :
sets (*restrict_space* (*count_space A*) *B*) = *sets* (*count_space* (*A ∩ B*))
by(*auto simp add*: *sets_restrict_space*)

lemma *sets_restrict_UNIV*[*simp*]: *sets* (*restrict_space M UNIV*) = *sets M*

by (*auto simp add*: *sets_restrict_space*)

lemma *sets_restrict_restrict_space*:
sets (*restrict_space* (*restrict_space M A*) *B*) = *sets* (*restrict_space M* (*A* ∩ *B*))
unfolding *sets_restrict_space image_comp* **by** (*intro image_cong*) *auto*

lemma *sets_restrict_space_iff*:
Ω ∩ *space M* ∈ *sets M* ⟹ *A* ∈ *sets* (*restrict_space M* Ω) ⟷ (*A* ⊆ Ω ∧ *A* ∈ *sets M*)
proof (*subst sets_restrict_space*, *safe*)
fix *A* **assume** Ω ∩ *space M* ∈ *sets M* **and** *A*: *A* ∈ *sets M*
then have (Ω ∩ *space M*) ∩ *A* ∈ *sets M*
by *rule*
also have (Ω ∩ *space M*) ∩ *A* = Ω ∩ *A*
using *sets.sets_into_space*[*OF A*] **by** *auto*
finally show Ω ∩ *A* ∈ *sets M*
by *auto*
qed *auto*

lemma *sets_restrict_space_cong*: *sets M* = *sets N* ⟹ *sets* (*restrict_space M* Ω) = *sets* (*restrict_space N* Ω)
by (*simp add*: *sets_restrict_space*)

lemma *restrict_space_eq_vimage_algebra*:
Ω ⊆ *space M* ⟹ *sets* (*restrict_space M* Ω) = *sets* (*vimage_algebra* Ω (λ*x*. *x*) *M*)
unfolding *restrict_space_def*
apply (*subst sets_measure_of*)
apply (*auto simp add*: *image_subset_iff dest*: *sets.sets_into_space*) []
apply (*auto simp add*: *sets_vimage_algebra intro*!: *arg_cong2*[**where** *f*=*sigma_sets*])
done

lemma *sets_Collect_restrict_space_iff*:
assumes *S* ∈ *sets M*
shows {*x*∈*space* (*restrict_space M S*). *P x*} ∈ *sets* (*restrict_space M S*) ⟷ {*x*∈*space M*. *x* ∈ *S* ∧ *P x*} ∈ *sets M*
proof −
have {*x*∈*S*. *P x*} = {*x*∈*space M*. *x* ∈ *S* ∧ *P x*}
using *sets.sets_into_space*[*OF assms*] **by** *auto*
then show *?thesis*
by (*subst sets_restrict_space_iff*) (*auto simp add*: *space_restrict_space assms*)
qed

lemma *measurable_restrict_space1*:
assumes *f*: *f* ∈ *measurable M N*
shows *f* ∈ *measurable* (*restrict_space M* Ω) *N*
unfolding *measurable_def*
proof (*intro CollectI conjI ballI*)
show *sp*: *f* ∈ *space* (*restrict_space M* Ω) → *space N*

using *measurable_space*[*OF f*] **by** (*auto simp*: *space_restrict_space*)

fix *A* **assume** *A* ∈ *sets N*
have *f* −‘ *A* ∩ *space* (*restrict_space M* Ω) = (*f* −‘ *A* ∩ *space M*) ∩ (Ω ∩ *space M*)
by (*auto simp*: *space_restrict_space*)
also have ... ∈ *sets* (*restrict_space M* Ω)
unfolding *sets_restrict_space*
using *measurable_sets*[*OF f* ‹*A* ∈ *sets N*›] **by** *blast*
finally show *f* −‘ *A* ∩ *space* (*restrict_space M* Ω) ∈ *sets* (*restrict_space M* Ω)
.
qed

lemma *measurable_restrict_space2_iff*:
f ∈ *measurable M* (*restrict_space N* Ω) ⟷ (*f* ∈ *measurable M N* ∧ *f* ∈ *space M* → Ω)
proof −
have ⋀*A*. *f* ∈ *space M* → Ω ⟹ *f* −‘ Ω ∩ *f* −‘ *A* ∩ *space M* = *f* −‘ *A* ∩ *space M*
by *auto*
then show *?thesis*
by (*auto simp*: *measurable_def space_restrict_space Pi_Int*[*symmetric*] *sets_restrict_space*)
qed

lemma *measurable_restrict_space2*:
f ∈ *space M* → Ω ⟹ *f* ∈ *measurable M N* ⟹ *f* ∈ *measurable M* (*restrict_space N* Ω)
by (*simp add*: *measurable_restrict_space2_iff*)

lemma *measurable_piecewise_restrict*:
assumes *I*: *countable C*
and *X*: ⋀Ω. Ω ∈ *C* ⟹ Ω ∩ *space M* ∈ *sets M space M* ⊆ ⋃ *C*
and *f*: ⋀Ω. Ω ∈ *C* ⟹ *f* ∈ *measurable* (*restrict_space M* Ω) *N*
shows *f* ∈ *measurable M N*
proof (*rule measurableI*)
fix *x* **assume** *x* ∈ *space M*
with *X* **obtain** Ω **where** Ω ∈ *C x* ∈ Ω *x* ∈ *space M* **by** *auto*
then show *f x* ∈ *space N*
by (*auto simp*: *space_restrict_space intro*: *f measurable_space*)
next
fix *A* **assume** *A*: *A* ∈ *sets N*
have *f* −‘ *A* ∩ *space M* = (⋃Ω∈*C*. (*f* −‘ *A* ∩ (Ω ∩ *space M*)))
using *X* **by** (*auto simp*: *subset_eq*)
also have ... ∈ *sets M*
using *measurable_sets*[*OF f A*] *X I*
by (*intro sets.countable_UN′*) (*auto simp*: *sets_restrict_space_iff space_restrict_space*)
finally show *f* −‘ *A* ∩ *space M* ∈ *sets M* .
qed

lemma *measurable_piecewise_restrict_iff*:
countable $C \Longrightarrow (\bigwedge\Omega.\ \Omega \in C \Longrightarrow \Omega \cap \mathit{space}\ M \in \mathit{sets}\ M) \Longrightarrow \mathit{space}\ M \subseteq (\bigcup C)$
$\Longrightarrow$
$f \in \mathit{measurable}\ M\ N \longleftrightarrow (\forall\,\Omega{\in}C.\ f \in \mathit{measurable}\ (\mathit{restrict\_space}\ M\ \Omega)\ N)$
by (*auto intro*: *measurable_piecewise_restrict measurable_restrict_space1*)

lemma *measurable_If_restrict_space_iff*:
$\{x{\in}\mathit{space}\ M.\ P\ x\} \in \mathit{sets}\ M \Longrightarrow$
$(\lambda x.\ \mathit{if}\ P\ x\ \mathit{then}\ f\ x\ \mathit{else}\ g\ x) \in \mathit{measurable}\ M\ N \longleftrightarrow$
$(f \in \mathit{measurable}\ (\mathit{restrict\_space}\ M\ \{x.\ P\ x\})\ N \wedge g \in \mathit{measurable}\ (\mathit{restrict\_space}\ M\ \{x.\ \neg\ P\ x\})\ N)$
by (*subst measurable_piecewise_restrict_iff*[**where** $C{=}\{\{x.\ P\ x\}, \{x.\ \neg\ P\ x\}\}$])
(*auto simp*: *Int_def sets.sets_Collect_neg space_restrict_space conj_commute*[*of* _ $x \in \mathit{space}\ M$ **for** x]
cong: *measurable_cong'*)

lemma *measurable_If*:
$f \in \mathit{measurable}\ M\ M' \Longrightarrow g \in \mathit{measurable}\ M\ M' \Longrightarrow \{x{\in}\mathit{space}\ M.\ P\ x\} \in \mathit{sets}\ M \Longrightarrow$
$(\lambda x.\ \mathit{if}\ P\ x\ \mathit{then}\ f\ x\ \mathit{else}\ g\ x) \in \mathit{measurable}\ M\ M'$
unfolding *measurable_If_restrict_space_iff* **by** (*auto intro*: *measurable_restrict_space1*)

lemma *measurable_If_set*:
assumes *measure*: $f \in \mathit{measurable}\ M\ M'\ g \in \mathit{measurable}\ M\ M'$
assumes P: $A \cap \mathit{space}\ M \in \mathit{sets}\ M$
shows $(\lambda x.\ \mathit{if}\ x \in A\ \mathit{then}\ f\ x\ \mathit{else}\ g\ x) \in \mathit{measurable}\ M\ M'$
proof (*rule measurable_If*[*OF measure*])
have $\{x \in \mathit{space}\ M.\ x \in A\} = A \cap \mathit{space}\ M$ **by** *auto*
thus $\{x \in \mathit{space}\ M.\ x \in A\} \in \mathit{sets}\ M$ **using** ‹$A \cap \mathit{space}\ M \in \mathit{sets}\ M$› **by** *auto*
qed

lemma *measurable_restrict_space_iff*:
$\Omega \cap \mathit{space}\ M \in \mathit{sets}\ M \Longrightarrow c \in \mathit{space}\ N \Longrightarrow$
$f \in \mathit{measurable}\ (\mathit{restrict\_space}\ M\ \Omega)\ N \longleftrightarrow (\lambda x.\ \mathit{if}\ x \in \Omega\ \mathit{then}\ f\ x\ \mathit{else}\ c) \in \mathit{measurable}\ M\ N$
by (*subst measurable_If_restrict_space_iff*)
(*simp_all add*: *Int_def conj_commute measurable_const*)

lemma *restrict_space_singleton*: $\{x\} \in \mathit{sets}\ M \Longrightarrow \mathit{sets}\ (\mathit{restrict\_space}\ M\ \{x\}) = \mathit{sets}\ (\mathit{count\_space}\ \{x\})$
using *sets_restrict_space_iff*[*of* $\{x\}$ M]
by (*auto simp add*: *sets_restrict_space_iff dest*!: *subset_singletonD*)

lemma *measurable_restrict_countable*:
assumes X[*intro*]: *countable* X
assumes *sets*[*simp*]: $\bigwedge x.\ x \in X \Longrightarrow \{x\} \in \mathit{sets}\ M$
assumes *space*[*simp*]: $\bigwedge x.\ x \in X \Longrightarrow f\ x \in \mathit{space}\ N$
assumes f: $f \in \mathit{measurable}\ (\mathit{restrict\_space}\ M\ (-\ X))\ N$
shows $f \in \mathit{measurable}\ M\ N$

using *f sets.countable*[*OF sets X*]
by (*intro measurable_piecewise_restrict*[**where** *M*=*M* **and** *C*={− *X*} ∪ ((λ*x*. {*x*}) ‘ *X*)])
(*auto simp*: *Diff_Int_distrib2 Compl_eq_Diff_UNIV Int_insert_left sets.Diff restrict_space_singleton*
simp del: *sets_count_space cong*: *measurable_cong_sets*)

lemma *measurable_discrete_difference*:
assumes *f*: *f* ∈ *measurable M N*
assumes *X*: *countable X* ⋀*x*. *x* ∈ *X* ⟹ {*x*} ∈ *sets M* ⋀*x*. *x* ∈ *X* ⟹ *g x* ∈ *space N*
assumes *eq*: ⋀*x*. *x* ∈ *space M* ⟹ *x* ∉ *X* ⟹ *f x* = *g x*
shows *g* ∈ *measurable M N*
by (*rule measurable_restrict_countable*[*OF X*])
(*auto simp*: *eq*[*symmetric*] *space_restrict_space cong*: *measurable_cong′ intro*: *f measurable_restrict_space1*)

lemma *measurable_count_space_extend*: *A* ⊆ *B* ⟹ *f* ∈ *space M* → *A* ⟹ *f* ∈ *M* →$_M$ *count_space B* ⟹ *f* ∈ *M* →$_M$ *count_space A*
by (*auto simp*: *measurable_def*)

end

6.2 Measurability Prover

theory *Measurable*
imports
Sigma_Algebra
HOL−Library.Order_Continuity
begin

lemma (**in** *algebra*) *sets_Collect_finite_All*:
assumes ⋀*i*. *i* ∈ *S* ⟹ {*x*∈Ω. *P i x*} ∈ *M finite S*
shows {*x*∈Ω. ∀ *i*∈*S*. *P i x*} ∈ *M*
proof −
have {*x*∈Ω. ∀ *i*∈*S*. *P i x*} = (*if S* = {} *then* Ω *else* ⋂ *i*∈*S*. {*x*∈Ω. *P i x*})
by *auto*
with *assms* **show** *?thesis* **by** (*auto intro*!: *sets_Collect_finite_All′*)
qed

abbreviation *pred M P* ≡ *P* ∈ *measurable M* (*count_space* (*UNIV*::*bool set*))

lemma *pred_def*: *pred M P* ⟷ {*x*∈*space M*. *P x*} ∈ *sets M*
proof
assume *pred M P*
then have *P* −‘ {*True*} ∩ *space M* ∈ *sets M*
by (*auto simp*: *measurable_count_space_eq2*)
also have *P* −‘ {*True*} ∩ *space M* = {*x*∈*space M*. *P x*} **by** *auto*

finally show $\{x{\in}space\ M.\ P\ x\} \in sets\ M$ **.**
next
assume *P*: $\{x{\in}space\ M.\ P\ x\} \in sets\ M$
moreover
{ fix *X*
have $X \in Pow\ (UNIV :: bool\ set)$ **by** *simp*
then have $P -` X \cap space\ M = \{x{\in}space\ M.\ ((X = \{True\} \longrightarrow P\ x) \wedge (X = \{False\} \longrightarrow \neg\ P\ x) \wedge X \neq \{\})\}$
unfolding *UNIV_bool Pow_insert Pow_empty* **by** *auto*
then have $P -` X \cap space\ M \in sets\ M$
by (*auto intro*!: *sets.sets_Collect_neg sets.sets_Collect_imp sets.sets_Collect_conj sets.sets_Collect_const P*) **}**
then show *pred M P*
by (*auto simp*: *measurable_def*)
qed

lemma *pred_sets1*: $\{x{\in}space\ M.\ P\ x\} \in sets\ M \Longrightarrow f \in measurable\ N\ M \Longrightarrow pred\ N\ (\lambda x.\ P\ (f\ x))$
by (*rule measurable_compose*[**where** *f*=*f* **and** *N*=*M*]) (*auto simp*: *pred_def*)

lemma *pred_sets2*: $A \in sets\ N \Longrightarrow f \in measurable\ M\ N \Longrightarrow pred\ M\ (\lambda x.\ f\ x \in A)$
by (*rule measurable_compose*[**where** *f*=*f* **and** *N*=*N*]) (*auto simp*: *pred_def Int_def*[*symmetric*])

ML_file ‹*measurable.ML*›

attribute_setup *measurable* = ‹
Scan.lift (
(*Args.add* >> *K true* || *Args.del* >> *K false* || *Scan.succeed true*) --
Scan.optional (*Args.parens* (
Scan.optional (*Args.$$$ raw* >> *K true*) *false* --
Scan.optional (*Args.$$$ generic* >> *K Measurable.Generic*) *Measurable.Concrete*))
(*false*, *Measurable.Concrete*) >>
Measurable.measurable_thm_attr)
› *declaration of measurability theorems*

attribute_setup *measurable_dest* = *Measurable.dest_thm_attr*
add dest rule to measurability prover

attribute_setup *measurable_cong* = *Measurable.cong_thm_attr*
add congurence rules to measurability prover

method_setup *measurable* = ‹ *Scan.lift* (*Scan.succeed* (*METHOD o Measurable.measurable_tac*)) ›
measurability prover

simproc_setup *measurable* $(A \in sets\ M \mid f \in measurable\ M\ N)$ = ‹*K Measurable.simproc*›

setup ‹
Global_Theory.add_thms_dynamic (***binding*** ‹*measurable*›, *Measurable.get_all*)
›

declare
pred_sets1[*measurable_dest*]
pred_sets2[*measurable_dest*]
sets.sets_into_space[*measurable_dest*]

declare
sets.top[*measurable*]
sets.empty_sets[*measurable* (*raw*)]
sets.Un[*measurable* (*raw*)]
sets.Diff[*measurable* (*raw*)]

declare
measurable_count_space[*measurable* (*raw*)]
measurable_ident[*measurable* (*raw*)]
measurable_id[*measurable* (*raw*)]
measurable_const[*measurable* (*raw*)]
measurable_If[*measurable* (*raw*)]
measurable_comp[*measurable* (*raw*)]
measurable_sets[*measurable* (*raw*)]

declare *measurable_cong_sets*[*measurable_cong*]
declare *sets_restrict_space_cong*[*measurable_cong*]
declare *sets_restrict_UNIV*[*measurable_cong*]

lemma *predE*[*measurable* (*raw*)]:
pred M P ⟹ {*x*∈*space M*. *P x*} ∈ *sets M*
unfolding *pred_def* **.**

lemma *pred_intros_imp'*[*measurable* (*raw*)]:
(*K* ⟹ *pred M* (λ*x*. *P x*)) ⟹ *pred M* (λ*x*. *K* ⟶ *P x*)
by (*cases K*) *auto*

lemma *pred_intros_conj1'*[*measurable* (*raw*)]:
(*K* ⟹ *pred M* (λ*x*. *P x*)) ⟹ *pred M* (λ*x*. *K* ∧ *P x*)
by (*cases K*) *auto*

lemma *pred_intros_conj2'*[*measurable* (*raw*)]:
(*K* ⟹ *pred M* (λ*x*. *P x*)) ⟹ *pred M* (λ*x*. *P x* ∧ *K*)
by (*cases K*) *auto*

lemma *pred_intros_disj1'*[*measurable* (*raw*)]:
(¬ *K* ⟹ *pred M* (λ*x*. *P x*)) ⟹ *pred M* (λ*x*. *K* ∨ *P x*)
by (*cases K*) *auto*

lemma *pred_intros_disj2'*[*measurable* (*raw*)]:
(¬ *K* ⟹ *pred M* (λ*x*. *P x*)) ⟹ *pred M* (λ*x*. *P x* ∨ *K*)
by (*cases K*) *auto*

lemma *pred_intros_logic*[*measurable* (*raw*)]:
pred M (λ*x*. *x* ∈ *space M*)
pred M (λ*x*. *P x*) ⟹ *pred M* (λ*x*. ¬ *P x*)
pred M (λ*x*. *Q x*) ⟹ *pred M* (λ*x*. *P x*) ⟹ *pred M* (λ*x*. *Q x* ∧ *P x*)
pred M (λ*x*. *Q x*) ⟹ *pred M* (λ*x*. *P x*) ⟹ *pred M* (λ*x*. *Q x* ⟶ *P x*)
pred M (λ*x*. *Q x*) ⟹ *pred M* (λ*x*. *P x*) ⟹ *pred M* (λ*x*. *Q x* ∨ *P x*)
pred M (λ*x*. *Q x*) ⟹ *pred M* (λ*x*. *P x*) ⟹ *pred M* (λ*x*. *Q x* = *P x*)
pred M (λ*x*. *f x* ∈ *UNIV*)
pred M (λ*x*. *f x* ∈ {})
pred M (λ*x*. *P'* (*f x*) *x*) ⟹ *pred M* (λ*x*. *f x* ∈ {*y*. *P'* *y x*})
pred M (λ*x*. *f x* ∈ (*B x*)) ⟹ *pred M* (λ*x*. *f x* ∈ − (*B x*))
pred M (λ*x*. *f x* ∈ (*A x*)) ⟹ *pred M* (λ*x*. *f x* ∈ (*B x*)) ⟹ *pred M* (λ*x*. *f x* ∈ (*A x*) − (*B x*))
pred M (λ*x*. *f x* ∈ (*A x*)) ⟹ *pred M* (λ*x*. *f x* ∈ (*B x*)) ⟹ *pred M* (λ*x*. *f x* ∈ (*A x*) ∩ (*B x*))
pred M (λ*x*. *f x* ∈ (*A x*)) ⟹ *pred M* (λ*x*. *f x* ∈ (*B x*)) ⟹ *pred M* (λ*x*. *f x* ∈ (*A x*) ∪ (*B x*))
pred M (λ*x*. *g x* (*f x*) ∈ (*X x*)) ⟹ *pred M* (λ*x*. *f x* ∈ (*g x*) −‘ (*X x*))
by (*auto simp*: *iff_conv_conj_imp pred_def*)

lemma *pred_intros_countable*[*measurable* (*raw*)]:
fixes *P* :: *'a* ⇒ *'i* :: *countable* ⇒ *bool*
shows
(⋀*i*. *pred M* (λ*x*. *P x i*)) ⟹ *pred M* (λ*x*. ∀ *i*. *P x i*)
(⋀*i*. *pred M* (λ*x*. *P x i*)) ⟹ *pred M* (λ*x*. ∃ *i*. *P x i*)
by (*auto intro*!: *sets.sets_Collect_countable_All sets.sets_Collect_countable_Ex simp*: *pred_def*)

lemma *pred_intros_countable_bounded*[*measurable* (*raw*)]:
fixes *X* :: *'i* :: *countable set*
shows
(⋀*i*. *i* ∈ *X* ⟹ *pred M* (λ*x*. *x* ∈ *N x i*)) ⟹ *pred M* (λ*x*. *x* ∈ (⋂ *i*∈*X*. *N x i*))
(⋀*i*. *i* ∈ *X* ⟹ *pred M* (λ*x*. *x* ∈ *N x i*)) ⟹ *pred M* (λ*x*. *x* ∈ (⋃ *i*∈*X*. *N x i*))
(⋀*i*. *i* ∈ *X* ⟹ *pred M* (λ*x*. *P x i*)) ⟹ *pred M* (λ*x*. ∀ *i*∈*X*. *P x i*)
(⋀*i*. *i* ∈ *X* ⟹ *pred M* (λ*x*. *P x i*)) ⟹ *pred M* (λ*x*. ∃ *i*∈*X*. *P x i*)
by *simp_all* (*auto simp*: *Bex_def Ball_def*)

lemma *pred_intros_finite*[*measurable* (*raw*)]:
finite I ⟹ (⋀*i*. *i* ∈ *I* ⟹ *pred M* (λ*x*. *x* ∈ *N x i*)) ⟹ *pred M* (λ*x*. *x* ∈ (⋂ *i*∈*I*. *N x i*))
finite I ⟹ (⋀*i*. *i* ∈ *I* ⟹ *pred M* (λ*x*. *x* ∈ *N x i*)) ⟹ *pred M* (λ*x*. *x* ∈ (⋃ *i*∈*I*. *N x i*))
finite I ⟹ (⋀*i*. *i* ∈ *I* ⟹ *pred M* (λ*x*. *P x i*)) ⟹ *pred M* (λ*x*. ∀ *i*∈*I*. *P x i*)
finite I ⟹ (⋀*i*. *i* ∈ *I* ⟹ *pred M* (λ*x*. *P x i*)) ⟹ *pred M* (λ*x*. ∃ *i*∈*I*. *P x i*)
by (*auto intro*!: *sets.sets_Collect_finite_Ex sets.sets_Collect_finite_All simp*:

iff_conv_conj_imp pred_def)

lemma *countable_Un_Int*[*measurable* (*raw*)]:
(⋀*i* :: *'i* :: *countable*. *i* ∈ *I* ⟹ *N i* ∈ *sets M*) ⟹ (⋃ *i*∈*I*. *N i*) ∈ *sets M*
I ≠ {} ⟹ (⋀*i* :: *'i* :: *countable*. *i* ∈ *I* ⟹ *N i* ∈ *sets M*) ⟹ (⋂ *i*∈*I*. *N i*) ∈ *sets M*
by *auto*

declare
finite_UN[*measurable* (*raw*)]
finite_INT[*measurable* (*raw*)]

lemma *sets_Int_pred*[*measurable* (*raw*)]:
assumes *space*: *A* ∩ *B* ⊆ *space M* **and** [*measurable*]: *pred M* (λ*x*. *x* ∈ *A*) *pred M* (λ*x*. *x* ∈ *B*)
shows *A* ∩ *B* ∈ *sets M*
proof −
have {*x*∈*space M*. *x* ∈ *A* ∩ *B*} ∈ *sets M* **by** *auto*
also have {*x*∈*space M*. *x* ∈ *A* ∩ *B*} = *A* ∩ *B*
using *space* **by** *auto*
finally show *?thesis* **.**
qed

lemma [*measurable* (*raw generic*)]:
assumes *f*: *f* ∈ *measurable M N* **and** *c*: *c* ∈ *space N* ⟹ {*c*} ∈ *sets N*
shows *pred_eq_const1*: *pred M* (λ*x*. *f x* = *c*)
and *pred_eq_const2*: *pred M* (λ*x*. *c* = *f x*)
proof −
show *pred M* (λ*x*. *f x* = *c*)
proof *cases*
assume *c* ∈ *space N*
with *measurable_sets*[*OF f c*] **show** *?thesis*
by (*auto simp*: *Int_def conj_commute pred_def*)
next
assume *c* ∉ *space N*
with *f*[*THEN measurable_space*] **have** {*x* ∈ *space M*. *f x* = *c*} = {} **by** *auto*
then show *?thesis* **by** (*auto simp*: *pred_def cong*: *conj_cong*)
qed
then show *pred M* (λ*x*. *c* = *f x*)
by (*simp add*: *eq_commute*)
qed

lemma *pred_count_space_const1*[*measurable* (*raw*)]:
f ∈ *measurable M* (*count_space UNIV*) ⟹ *Measurable.pred M* (λ*x*. *f x* = *c*)
by (*intro pred_eq_const1*[**where** *N*=*count_space UNIV*]) (*auto*)

lemma *pred_count_space_const2*[*measurable* (*raw*)]:
f ∈ *measurable M* (*count_space UNIV*) ⟹ *Measurable.pred M* (λ*x*. *c* = *f x*)
by (*intro pred_eq_const2*[**where** *N*=*count_space UNIV*]) (*auto*)

lemma *pred_le_const*[*measurable* (*raw generic*)]:
assumes *f*: $f \in$ *measurable M N* **and** *c*: $\{.. c\} \in$ *sets N* **shows** *pred M* ($\lambda x.$ $f\ x \leq c$)
using *measurable_sets*[*OF f c*]
by (*auto simp*: *Int_def conj_commute eq_commute pred_def*)

lemma *pred_const_le*[*measurable* (*raw generic*)]:
assumes *f*: $f \in$ *measurable M N* **and** *c*: $\{c ..\} \in$ *sets N* **shows** *pred M* ($\lambda x.$ $c \leq f\ x$)
using *measurable_sets*[*OF f c*]
by (*auto simp*: *Int_def conj_commute eq_commute pred_def*)

lemma *pred_less_const*[*measurable* (*raw generic*)]:
assumes *f*: $f \in$ *measurable M N* **and** *c*: $\{..< c\} \in$ *sets N* **shows** *pred M* ($\lambda x.$ $f\ x < c$)
using *measurable_sets*[*OF f c*]
by (*auto simp*: *Int_def conj_commute eq_commute pred_def*)

lemma *pred_const_less*[*measurable* (*raw generic*)]:
assumes *f*: $f \in$ *measurable M N* **and** *c*: $\{c <..\} \in$ *sets N* **shows** *pred M* ($\lambda x.$ $c < f\ x$)
using *measurable_sets*[*OF f c*]
by (*auto simp*: *Int_def conj_commute eq_commute pred_def*)

declare
sets.Int[*measurable* (*raw*)]

lemma *pred_in_If*[*measurable* (*raw*)]:
($P \Longrightarrow$ *pred M* ($\lambda x.$ $x \in A\ x$)) $\Longrightarrow$ ($\neg\ P \Longrightarrow$ *pred M* ($\lambda x.$ $x \in B\ x$)) $\Longrightarrow$
pred M ($\lambda x.$ $x \in$ (*if P then A x else B x*))
by *auto*

lemma *sets_range*[*measurable_dest*]:
A ‘ $I \subseteq$ *sets M* $\Longrightarrow$ $i \in I \Longrightarrow$ $A\ i \in$ *sets M*
by *auto*

lemma *pred_sets_range*[*measurable_dest*]:
A ‘ $I \subseteq$ *sets N* $\Longrightarrow$ $i \in I \Longrightarrow$ $f \in$ *measurable M N* $\Longrightarrow$ *pred M* ($\lambda x.$ $f\ x \in A\ i$)
using *pred_sets2*[*OF sets_range*] **by** *auto*

lemma *sets_All*[*measurable_dest*]:
$\forall i.$ $A\ i \in$ *sets* ($M\ i$) $\Longrightarrow$ $A\ i \in$ *sets* ($M\ i$)
by *auto*

lemma *pred_sets_All*[*measurable_dest*]:
$\forall i.$ $A\ i \in$ *sets* ($N\ i$) $\Longrightarrow$ $f \in$ *measurable M* ($N\ i$) $\Longrightarrow$ *pred M* ($\lambda x.$ $f\ x \in A\ i$)
using *pred_sets2*[*OF sets_All*, *of A N f*] **by** *auto*

lemma *sets_Ball*[*measurable_dest*]:
 $\forall i{\in}I.\ A\ i \in sets\ (M\ i) \Longrightarrow i{\in}I \Longrightarrow A\ i \in sets\ (M\ i)$
 by *auto*

lemma *pred_sets_Ball*[*measurable_dest*]:
 $\forall i{\in}I.\ A\ i \in sets\ (N\ i) \Longrightarrow i{\in}I \Longrightarrow f \in measurable\ M\ (N\ i) \Longrightarrow pred\ M\ (\lambda x.\ f\ x \in A\ i)$
 using *pred_sets2*[*OF sets_Ball, of _ _ _ f*] **by** *auto*

lemma *measurable_finite*[*measurable* (*raw*)]:
 fixes $S :: 'a \Rightarrow nat\ set$
 assumes [*measurable*]: $\bigwedge i.\ \{x{\in}space\ M.\ i \in S\ x\} \in sets\ M$
 shows *pred M* ($\lambda x.$ *finite* ($S\ x$))
 unfolding *finite_nat_set_iff_bounded* **by** (*simp add*: *Ball_def*)

lemma *measurable_Least*[*measurable*]:
  assumes [*measurable*]: ($\bigwedge i{::}nat.$ ($\lambda x.$ $P\ i\ x$) $\in$ *measurable M* (*count_space UNIV*))
 shows ($\lambda x.$ *LEAST i. P i x*) $\in$ *measurable M* (*count_space UNIV*)
 unfolding *measurable_def* **by** (*safe intro*!: *sets_Least*) *simp_all*

lemma *measurable_Max_nat*[*measurable* (*raw*)]:
 fixes $P :: nat \Rightarrow 'a \Rightarrow bool$
 assumes [*measurable*]: $\bigwedge i.$ *Measurable.pred M* ($P\ i$)
 shows ($\lambda x.$ *Max* $\{i.\ P\ i\ x\}$) $\in$ *measurable M* (*count_space UNIV*)
 unfolding *measurable_count_space_eq2_countable*
proof *safe*
 fix *n*

 { fix *x* **assume** $\forall i.\ \exists n{\geq}i.\ P\ n\ x$
  then have *infinite* $\{i.\ P\ i\ x\}$
   unfolding *infinite_nat_iff_unbounded_le* **by** *auto*
  then have *Max* $\{i.\ P\ i\ x\}$ = *the None*
   by (*rule Max.infinite*) **}**
 note *1* = *this*

 { fix *x i j* **assume** *P i x* $\forall n{\geq}j.\ \neg\ P\ n\ x$
  then have *finite* $\{i.\ P\ i\ x\}$
   by (*auto simp*: *subset_eq not_le*[*symmetric*] *finite_nat_iff_bounded*)
  with ‹*P i x*› **have** *P* (*Max* $\{i.\ P\ i\ x\}$) *x i* $\leq$ *Max* $\{i.\ P\ i\ x\}$ *finite* $\{i.\ P\ i\ x\}$
   using *Max_in*[*of* $\{i.\ P\ i\ x\}$] **by** *auto* **}**
 note *2* = *this*

 have ($\lambda x.$ *Max* $\{i.\ P\ i\ x\}$) $-$ ‘ $\{n\}$ $\cap$ *space M* = $\{x{\in}space\ M.\ Max\ \{i.\ P\ i\ x\} = n\}$
  by *auto*
 also have ... =
  $\{x{\in}space\ M.$ *if* ($\forall i.\ \exists n{\geq}i.\ P\ n\ x$) *then the None* = *n else*
   if ($\exists i.\ P\ i\ x$) *then* $P\ n\ x \wedge (\forall i{>}n.\ \neg\ P\ i\ x)$

```
      else Max {} = n}
    by (intro arg_cong[where f=Collect] ext conj_cong)
       (auto simp add: 1 2 not_le[symmetric] intro!: Max_eqI)
  also have ... ∈ sets M
    by measurable
  finally show (λx. Max {i. P i x}) -` {n} ∩ space M ∈ sets M .
qed simp

lemma measurable_Min_nat[measurable (raw)]:
  fixes P :: nat ⇒ 'a ⇒ bool
  assumes [measurable]: ⋀i. Measurable.pred M (P i)
  shows (λx. Min {i. P i x}) ∈ measurable M (count_space UNIV)
  unfolding measurable_count_space_eq2_countable
proof safe
  fix n

  { fix x assume ∀ i. ∃ n≥i. P n x
    then have infinite {i. P i x}
      unfolding infinite_nat_iff_unbounded_le by auto
    then have Min {i. P i x} = the None
      by (rule Min.infinite) }
  note 1 = this

  { fix x i j assume P i x ∀ n≥j. ¬ P n x
    then have finite {i. P i x}
      by (auto simp: subset_eq not_le[symmetric] finite_nat_iff_bounded)
    with ‹P i x› have P (Min {i. P i x}) x Min {i. P i x} ≤ i finite {i. P i x}
      using Min_in[of {i. P i x}] by auto }
  note 2 = this

  have (λx. Min {i. P i x}) -` {n} ∩ space M = {x∈space M. Min {i. P i x} =
n}
    by auto
  also have ... =
    {x∈space M. if (∀ i. ∃ n≥i. P n x) then the None = n else
      if (∃ i. P i x) then P n x ∧ (∀ i<n. ¬ P i x)
      else Min {} = n}
    by (intro arg_cong[where f=Collect] ext conj_cong)
       (auto simp add: 1 2 not_le[symmetric] intro!: Min_eqI)
  also have ... ∈ sets M
    by measurable
  finally show (λx. Min {i. P i x}) -` {n} ∩ space M ∈ sets M .
qed simp

lemma measurable_count_space_insert[measurable (raw)]:
  s ∈ S ⟹ A ∈ sets (count_space S) ⟹ insert s A ∈ sets (count_space S)
  by simp

lemma sets_UNIV [measurable (raw)]: A ∈ sets (count_space UNIV)
```

by *simp*

lemma *measurable_card*[*measurable*]:
fixes $S :: 'a \Rightarrow nat\ set$
assumes [*measurable*]: $\bigwedge i.\ \{x \in space\ M.\ i \in S\ x\} \in sets\ M$
shows $(\lambda x.\ card\ (S\ x)) \in measurable\ M\ (count\_space\ UNIV)$
unfolding *measurable_count_space_eq2_countable*
proof *safe*
fix *n* **show** $(\lambda x.\ card\ (S\ x)) -` \{n\} \cap space\ M \in sets\ M$
proof (*cases n*)
case *0*
then have $(\lambda x.\ card\ (S\ x)) -` \{n\} \cap space\ M = \{x \in space\ M.\ infinite\ (S\ x) \vee (\forall i.\ i \notin S\ x)\}$
by *auto*
also have $\ldots \in sets\ M$
by *measurable*
finally show *?thesis* **.**
next
case (*Suc i*)
then have $(\lambda x.\ card\ (S\ x)) -` \{n\} \cap space\ M =$
$(\bigcup F \in \{A \in \{A.\ finite\ A\}.\ card\ A = n\}.\ \{x \in space\ M.\ (\forall i.\ i \in S\ x \longleftrightarrow i \in F)\})$
unfolding *set_eq_iff*[*symmetric*] *Collect_bex_eq*[*symmetric*] **by** (*auto intro*: *card_ge_0_finite*)
also have $\ldots \in sets\ M$
by (*intro sets.countable_UN' countable_Collect countable_Collect_finite*) *auto*
finally show *?thesis* **.**
qed
qed *rule*

lemma *measurable_pred_countable*[*measurable* (*raw*)]:
assumes *countable X*
shows
$(\bigwedge i.\ i \in X \Longrightarrow Measurable.pred\ M\ (\lambda x.\ P\ x\ i)) \Longrightarrow Measurable.pred\ M\ (\lambda x.\ \forall i \in X.\ P\ x\ i)$
$(\bigwedge i.\ i \in X \Longrightarrow Measurable.pred\ M\ (\lambda x.\ P\ x\ i)) \Longrightarrow Measurable.pred\ M\ (\lambda x.\ \exists i \in X.\ P\ x\ i)$
unfolding *pred_def*
by (*auto intro*!: *sets.sets_Collect_countable_All' sets.sets_Collect_countable_Ex' assms*)

6.2.1 Measurability for (co)inductive predicates

lemma *measurable_bot*[*measurable*]: $bot \in measurable\ M\ (count\_space\ UNIV)$
by (*simp add*: *bot_fun_def*)

lemma *measurable_top*[*measurable*]: $top \in measurable\ M\ (count\_space\ UNIV)$
by (*simp add*: *top_fun_def*)

lemma *measurable_SUP*[*measurable*]:
fixes $F :: 'i \Rightarrow 'a \Rightarrow 'b$::{*complete_lattice*, *countable*}
assumes [*simp*]: *countable I*
assumes [*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow F\ i \in$ *measurable M* (*count_space UNIV*)
shows $(\lambda x.\ SUP\ i{\in}I.\ F\ i\ x) \in$ *measurable M* (*count_space UNIV*)
unfolding *measurable_count_space_eq2_countable*
proof (*safe intro*!: *UNIV_I*)
fix *a*
have $(\lambda x.\ SUP\ i{\in}I.\ F\ i\ x) -` \{a\} \cap$ *space M* =
$\{x{\in}$*space M*. $(\forall\, i{\in}I.\ F\ i\ x \le a) \wedge (\forall\, b.\ (\forall\, i{\in}I.\ F\ i\ x \le b) \longrightarrow a \le b)\}$
unfolding *SUP_le_iff*[*symmetric*] **by** *auto*
also have $\ldots \in$ *sets M*
by *measurable*
finally show $(\lambda x.\ SUP\ i{\in}I.\ F\ i\ x) -` \{a\} \cap$ *space M* $\in$ *sets M* **.**
qed

lemma *measurable_INF*[*measurable*]:
fixes $F :: 'i \Rightarrow 'a \Rightarrow 'b$::{*complete_lattice*, *countable*}
assumes [*simp*]: *countable I*
assumes [*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow F\ i \in$ *measurable M* (*count_space UNIV*)
shows $(\lambda x.\ INF\ i{\in}I.\ F\ i\ x) \in$ *measurable M* (*count_space UNIV*)
unfolding *measurable_count_space_eq2_countable*
proof (*safe intro*!: *UNIV_I*)
fix *a*
have $(\lambda x.\ INF\ i{\in}I.\ F\ i\ x) -` \{a\} \cap$ *space M* =
$\{x{\in}$*space M*. $(\forall\, i{\in}I.\ a \le F\ i\ x) \wedge (\forall\, b.\ (\forall\, i{\in}I.\ b \le F\ i\ x) \longrightarrow b \le a)\}$
unfolding *le_INF_iff*[*symmetric*] **by** *auto*
also have $\ldots \in$ *sets M*
by *measurable*
finally show $(\lambda x.\ INF\ i{\in}I.\ F\ i\ x) -` \{a\} \cap$ *space M* $\in$ *sets M* **.**
qed

lemma *measurable_lfp_coinduct*[*consumes 1*, *case_names continuity step*]:
fixes $F :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b$::{*complete_lattice*, *countable*}$)$
assumes *P M*
assumes *F*: *sup_continuous F*
assumes $*$: $\bigwedge M\ A.\ P\ M \Longrightarrow (\bigwedge N.\ P\ N \Longrightarrow A \in$ *measurable N* (*count_space UNIV*)$) \Longrightarrow F\ A \in$ *measurable M* (*count_space UNIV*)
shows *lfp F* $\in$ *measurable M* (*count_space UNIV*)
proof −
{ fix *i* **from** ‹*P M*› **have** $((F \frown\frown i)\ bot) \in$ *measurable M* (*count_space UNIV*)
by (*induct i arbitrary*: *M*) (*auto intro*!: $*$) **}**
then have $(\lambda x.\ SUP\ i.\ (F \frown\frown i)\ bot\ x) \in$ *measurable M* (*count_space UNIV*)
by *measurable*
also have $(\lambda x.\ SUP\ i.\ (F \frown\frown i)\ bot\ x) =$ *lfp F*
by (*subst sup_continuous_lfp*) (*auto intro*: *F simp*: *image_comp*)
finally show *?thesis* **.**
qed

lemma *measurable_lfp*:
fixes $F :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b::\{complete\_lattice, countable\})$
assumes *F*: *sup_continuous F*
assumes $*$: $\bigwedge A.\ A \in measurable\ M\ (count\_space\ UNIV) \Longrightarrow F\ A \in measurable\ M\ (count\_space\ UNIV)$
shows *lfp F* $\in$ *measurable M* (*count_space UNIV*)
by (*coinduction rule*: *measurable_lfp_coinduct*[*OF* _ *F*]) (*blast intro*: $*$)

lemma *measurable_gfp_coinduct*[*consumes 1*, *case_names continuity step*]:
fixes $F :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b::\{complete\_lattice, countable\})$
assumes *P M*
assumes *F*: *inf_continuous F*
assumes $*$: $\bigwedge M\ A.\ P\ M \Longrightarrow (\bigwedge N.\ P\ N \Longrightarrow A \in measurable\ N\ (count\_space\ UNIV)) \Longrightarrow F\ A \in measurable\ M\ (count\_space\ UNIV)$
shows *gfp F* $\in$ *measurable M* (*count_space UNIV*)
proof −
{ fix *i* **from** ‹*P M*› **have** $((F \frown i)\ top) \in measurable\ M\ (count\_space\ UNIV)$
by (*induct i arbitrary*: *M*) (*auto intro*!: $*$) **}**
then have $(\lambda x.\ INF\ i.\ (F \frown i)\ top\ x) \in measurable\ M\ (count\_space\ UNIV)$
by *measurable*
also have $(\lambda x.\ INF\ i.\ (F \frown i)\ top\ x) = gfp\ F$
by (*subst inf_continuous_gfp*) (*auto intro*: *F simp*: *image_comp*)
finally show *?thesis* **.**
qed

lemma *measurable_gfp*:
fixes $F :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b::\{complete\_lattice, countable\})$
assumes *F*: *inf_continuous F*
assumes $*$: $\bigwedge A.\ A \in measurable\ M\ (count\_space\ UNIV) \Longrightarrow F\ A \in measurable\ M\ (count\_space\ UNIV)$
shows *gfp F* $\in$ *measurable M* (*count_space UNIV*)
by (*coinduction rule*: *measurable_gfp_coinduct*[*OF* _ *F*]) (*blast intro*: $*$)

lemma *measurable_lfp2_coinduct*[*consumes 1*, *case_names continuity step*]:
fixes $F :: ('a \Rightarrow 'c \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'c \Rightarrow 'b::\{complete\_lattice, countable\})$
assumes *P M s*
assumes *F*: *sup_continuous F*
assumes $*$: $\bigwedge M\ A\ s.\ P\ M\ s \Longrightarrow (\bigwedge N\ t.\ P\ N\ t \Longrightarrow A\ t \in measurable\ N\ (count\_space\ UNIV)) \Longrightarrow F\ A\ s \in measurable\ M\ (count\_space\ UNIV)$
shows *lfp F s* $\in$ *measurable M* (*count_space UNIV*)
proof −
{ fix *i* **from** ‹*P M s*› **have** $(\lambda x.\ (F \frown i)\ bot\ s\ x) \in measurable\ M\ (count\_space\ UNIV)$
by (*induct i arbitrary*: *M s*) (*auto intro*!: $*$) **}**
then have $(\lambda x.\ SUP\ i.\ (F \frown i)\ bot\ s\ x) \in measurable\ M\ (count\_space\ UNIV)$
by *measurable*
also have $(\lambda x.\ SUP\ i.\ (F \frown i)\ bot\ s\ x) = lfp\ F\ s$
by (*subst sup_continuous_lfp*) (*auto simp*: *F simp*: *image_comp*)

finally show *?thesis* **.**
qed

lemma *measurable_gfp2_coinduct*[*consumes 1*, *case_names continuity step*]:
fixes $F :: ('a \Rightarrow 'c \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'c \Rightarrow 'b::\{complete\_lattice, countable\})$
assumes $P\ M\ s$
assumes F: *inf_continuous F*
assumes $*$: $\bigwedge M\ A\ s.\ P\ M\ s \Longrightarrow (\bigwedge N\ t.\ P\ N\ t \Longrightarrow A\ t \in measurable\ N\ (count\_space\ UNIV)) \Longrightarrow F\ A\ s \in measurable\ M\ (count\_space\ UNIV)$
shows *gfp F s* $\in$ *measurable M* (*count_space UNIV*)
proof −
{ fix *i* **from** ‹*P M s*› **have** $(\lambda x.\ (F\ \widehat{\ }\widehat{\ }\ i)\ top\ s\ x) \in measurable\ M\ (count\_space\ UNIV)$
by (*induct i arbitrary*: *M s*) (*auto intro*!: $*$) **}**
then have $(\lambda x.\ INF\ i.\ (F\ \widehat{\ }\widehat{\ }\ i)\ top\ s\ x) \in measurable\ M\ (count\_space\ UNIV)$
by *measurable*
also have $(\lambda x.\ INF\ i.\ (F\ \widehat{\ }\widehat{\ }\ i)\ top\ s\ x) = gfp\ F\ s$
by (*subst inf_continuous_gfp*) (*auto simp*: *F simp*: *image_comp*)
finally show *?thesis* **.**
qed

lemma *measurable_enat_coinduct*:
fixes $f :: 'a \Rightarrow enat$
assumes $R\ f$
assumes $*$: $\bigwedge f.\ R\ f \Longrightarrow \exists g\ h\ i\ P.\ R\ g \wedge f = (\lambda x.\ if\ P\ x\ then\ h\ x\ else\ eSuc\ (g\ (i\ x))) \wedge$
Measurable.pred M P $\wedge$
$i \in measurable\ M\ M \wedge$
$h \in measurable\ M\ (count\_space\ UNIV)$
shows $f \in measurable\ M\ (count\_space\ UNIV)$
proof (*simp add*: *measurable_count_space_eq2_countable*, *rule*)
fix $a :: enat$
have $f -` \{a\} \cap space\ M = \{x \in space\ M.\ f\ x = a\}$
by *auto*
{ fix $i :: nat$
from ‹*R f*› **have** *Measurable.pred M* $(\lambda x.\ f\ x = enat\ i)$
proof (*induction i arbitrary*: *f*)
case *0*
from $*$[*OF this*] **obtain** *g h i P*
where *f*: $f = (\lambda x.\ if\ P\ x\ then\ h\ x\ else\ eSuc\ (g\ (i\ x)))$ **and**
[*measurable*]: *Measurable.pred M P i* $\in$ *measurable M M h* $\in$ *measurable M* (*count_space UNIV*)
by *auto*
have *Measurable.pred M* $(\lambda x.\ P\ x \wedge h\ x = 0)$
by *measurable*
also have $(\lambda x.\ P\ x \wedge h\ x = 0) = (\lambda x.\ f\ x = enat\ 0)$
by (*auto simp*: *f zero_enat_def*[*symmetric*])
finally show *?case* **.**
next

```
    case (Suc n)
    from *[OF Suc.prems] obtain g h i P
      where f: f = (λx. if P x then h x else eSuc (g (i x))) and R g and
        M[measurable]: Measurable.pred M P i ∈ measurable M M h ∈ measurable
M (count_space UNIV)
      by auto
    have (λx. f x = enat (Suc n)) =
      (λx. (P x ⟶ h x = enat (Suc n)) ∧ (¬ P x ⟶ g (i x) = enat n))
      by (auto simp: f zero_enat_def[symmetric] eSuc_enat[symmetric])
    also have Measurable.pred M ...
      by (intro pred_intros_logic measurable_compose[OF M(2)] Suc ‹R g›)
measurable
    finally show ?case .
  qed
  then have f -` {enat i} ∩ space M ∈ sets M
    by (simp add: pred_def Int_def conj_commute) }
 note fin = this
 show f -` {a} ∩ space M ∈ sets M
 proof (cases a)
  case infinity
  then have f -` {a} ∩ space M = space M - (⋃n. f -` {enat n} ∩ space M)
    by auto
  also have ... ∈ sets M
    by (intro sets.Diff sets.top sets.Un sets.countable_UN) (auto intro!: fin)
  finally show ?thesis .
 qed (simp add: fin)
qed

lemma measurable_THE:
 fixes P :: 'a ⇒ 'b ⇒ bool
 assumes [measurable]: ⋀i. Measurable.pred M (P i)
 assumes I[simp]: countable I ⋀i x. x ∈ space M ⟹ P i x ⟹ i ∈ I
 assumes unique: ⋀x i j. x ∈ space M ⟹ P i x ⟹ P j x ⟹ i = j
 shows (λx. THE i. P i x) ∈ measurable M (count_space UNIV)
 unfolding measurable_def
proof safe
 fix X
 define f where f x = (THE i. P i x) for x
 define undef where undef = (THE i::'a. False)
 { fix i x assume x ∈ space M P i x then have f x = i
    unfolding f_def using unique by auto }
 note f_eq = this
 { fix x assume x ∈ space M ∀i∈I. ¬ P i x
  then have ⋀i. ¬ P i x
    using I(2)[of x] by auto
  then have f x = undef
    by (auto simp: undef_def f_def) }
 then have f -` X ∩ space M = (⋃i∈I ∩ X. {x∈space M. P i x}) ∪
   (if undef ∈ X then space M - (⋃i∈I. {x∈space M. P i x}) else {})
```

by (*auto dest*: *f_eq*)
also have ... ∈ *sets M*
by (*auto intro*!: *sets.Diff sets.countable_UN'*)
finally show *f* −‘ *X* ∩ *space M* ∈ *sets M* **.**
qed *simp*

lemma *measurable_Ex1*[*measurable* (*raw*)]:
assumes [*simp*]: *countable I* **and** [*measurable*]: ⋀*i*. *i* ∈ *I* ⟹ *Measurable.pred M* (*P i*)
shows *Measurable.pred M* (λ*x*. ∃!*i*∈*I*. *P i x*)
unfolding *bex1_def* **by** *measurable*

lemma *measurable_Sup_nat*[*measurable* (*raw*)]:
fixes *F* :: ′*a* ⇒ *nat set*
assumes [*measurable*]: ⋀*i*. *Measurable.pred M* (λ*x*. *i* ∈ *F x*)
shows (λ*x*. *Sup* (*F x*)) ∈ *M* →$_M$ *count_space UNIV*
proof (*clarsimp simp add*: *measurable_count_space_eq2_countable*)
fix *a*
have *F_empty_iff*: *F x* = {} ⟷ (∀ *i*. *i* ∉ *F x*) **for** *x*
by *auto*
have *Measurable.pred M* (λ*x*. *if finite* (*F x*) *then if F x* = {} *then a* = *0*
else a ∈ *F x* ∧ (∀ *j*. *j* ∈ *F x* ⟶ *j* ≤ *a*) *else a* = *the None*)
unfolding *finite_nat_set_iff_bounded Ball_def F_empty_iff* **by** *measurable*
moreover have (λ*x*. *Sup* (*F x*)) −‘ {*a*} ∩ *space M* =
{*x*∈*space M*. *if finite* (*F x*) *then if F x* = {} *then a* = *0*
else a ∈ *F x* ∧ (∀ *j*. *j* ∈ *F x* ⟶ *j* ≤ *a*) *else a* = *the None*}
by (*intro set_eqI*)
(*auto simp*: *Sup_nat_def Max.infinite intro*!: *Max_in Max_eqI*)
ultimately show (λ*x*. *Sup* (*F x*)) −‘ {*a*} ∩ *space M* ∈ *sets M*
by *auto*
qed

lemma *measurable_if_split*[*measurable* (*raw*)]:
(*c* ⟹ *Measurable.pred M f*) ⟹ (¬ *c* ⟹ *Measurable.pred M g*) ⟹
Measurable.pred M (*if c then f else g*)
by *simp*

lemma *pred_restrict_space*:
assumes *S* ∈ *sets M*
shows *Measurable.pred* (*restrict_space M S*) *P* ⟷ *Measurable.pred M* (λ*x*. *x* ∈ *S* ∧ *P x*)
unfolding *pred_def sets_Collect_restrict_space_iff*[*OF assms*] **..**

lemma *measurable_predpow*[*measurable*]:
assumes *Measurable.pred M T*
assumes ⋀*Q*. *Measurable.pred M Q* ⟹ *Measurable.pred M* (*R Q*)
shows *Measurable.pred M* ((*R* ^^ *n*) *T*)
by (*induct n*) (*auto intro*: *assms*)

lemma *measurable_compose_countable_restrict*:
assumes *P*: *countable* {*i*. *P i*}
and *f*: *f* ∈ *M* →$_M$ *count_space UNIV*
and *Q*: ⋀*i*. *P i* ⟹ *pred M* (*Q i*)
shows *pred M* (λ*x*. *P* (*f x*) ∧ *Q* (*f x*) *x*)
proof −
have *P_f*: {*x* ∈ *space M*. *P* (*f x*)} ∈ *sets M*
unfolding *pred_def*[*symmetric*] **by** (*rule measurable_compose*[*OF f*]) *simp*
have *pred* (*restrict_space M* {*x*∈*space M*. *P* (*f x*)}) (λ*x*. *Q* (*f x*) *x*)
proof (*rule measurable_compose_countable'*[**where** *g*=*f*, *OF _ _ P*])
show *f* ∈ *restrict_space M* {*x*∈*space M*. *P* (*f x*)} →$_M$ *count_space* {*i*. *P i*}
by (*rule measurable_count_space_extend*[*OF subset_UNIV*])
(*auto simp*: *space_restrict_space intro*!: *measurable_restrict_space1 f*)
qed (*auto intro*!: *measurable_restrict_space1 Q*)
then show *?thesis*
unfolding *pred_restrict_space*[*OF P_f*] **by** (*simp cong*: *measurable_cong*)
qed

lemma *measurable_limsup* [*measurable* (*raw*)]:
assumes [*measurable*]: ⋀*n*. *A n* ∈ *sets M*
shows *limsup A* ∈ *sets M*
by (*subst limsup_INF_SUP*, *auto*)

lemma *measurable_liminf* [*measurable* (*raw*)]:
assumes [*measurable*]: ⋀*n*. *A n* ∈ *sets M*
shows *liminf A* ∈ *sets M*
by (*subst liminf_SUP_INF*, *auto*)

lemma *measurable_case_enat*[*measurable* (*raw*)]:
assumes *f*: *f* ∈ *M* →$_M$ *count_space UNIV* **and** *g*: ⋀*i*. *g i* ∈ *M* →$_M$ *N* **and** *h*: *h* ∈ *M* →$_M$ *N*
shows (λ*x*. *case f x of enat i* ⇒ *g i x* | ∞ ⇒ *h x*) ∈ *M* →$_M$ *N*
apply (*rule measurable_compose_countable*[*OF _ f*])
subgoal for *i*
by (*cases i*) (*auto intro*: *g h*)
done

hide_const (**open**) *pred*

end

6.3 Measure Spaces

theory *Measure_Space*
imports
Measurable HOL−Library.Extended_Nonnegative_Real
begin

6.3.1 Relate extended reals and the indicator function

lemma *suminf_cmult_indicator*:
fixes $f :: nat \Rightarrow ennreal$
assumes *disjoint_family* A $x \in A$ i
shows $(\sum n.\ f\ n * indicator\ (A\ n)\ x) = f\ i$
proof −
have $**$: $\bigwedge n.\ f\ n * indicator\ (A\ n)\ x = (if\ n = i\ then\ f\ n\ else\ 0 :: ennreal)$
using ‹$x \in A\ i$› *assms* **unfolding** *disjoint_family_on_def indicator_def* **by** *auto*
then have $\bigwedge n.\ (\sum j{<}n.\ f\ j * indicator\ (A\ j)\ x) = (if\ i < n\ then\ f\ i\ else\ 0 :: ennreal)$
by (*auto simp*: *sum.If_cases*)
moreover have $(SUP\ n.\ if\ i < n\ then\ f\ i\ else\ 0) = (f\ i :: ennreal)$
proof (*rule SUP_eqI*)
fix $y :: ennreal$ **assume** $\bigwedge n.\ n \in UNIV \Longrightarrow (if\ i < n\ then\ f\ i\ else\ 0) \leq y$
from *this*[*of Suc i*] **show** $f\ i \leq y$ **by** *auto*
qed (*use assms* **in** *simp*)
ultimately show *?thesis* **using** *assms*
by (*simp add*: *suminf_eq_SUP*)
qed

lemma *suminf_indicator*:
assumes *disjoint_family* A
shows $(\sum n.\ indicator\ (A\ n)\ x :: ennreal) = indicator\ (\bigcup i.\ A\ i)\ x$
proof *cases*
assume $*$: $x \in (\bigcup i.\ A\ i)$
then obtain i **where** $x \in A\ i$ **by** *auto*
from *suminf_cmult_indicator*[*OF assms*(*1*), *OF* ‹$x \in A\ i$›, *of* $\lambda k.\ 1$]
show *?thesis* **using** $*$ **by** *simp*
qed *simp*

lemma *sum_indicator_disjoint_family*:
fixes $f :: 'd \Rightarrow 'e{::}semiring\_1$
assumes d: *disjoint_family_on* A P **and** $x \in A\ j$ **and** *finite* P **and** $j \in P$
shows $(\sum i{\in}P.\ f\ i * indicator\ (A\ i)\ x) = f\ j$
proof −
have $P \cap \{i.\ x \in A\ i\} = \{j\}$
using d ‹$x \in A\ j$› ‹$j \in P$› **unfolding** *disjoint_family_on_def*
by *auto*
with ‹*finite* P› **show** *?thesis*
by (*simp add*: *indicator_def*)
qed

The type for emeasure spaces is already defined in *HOL−Analysis.Sigma_Algebra*, as it is also used to represent sigma algebras (with an arbitrary emeasure).

6.3.2 Extend binary sets

lemma *LIMSEQ_binaryset*:

assumes f: $f\ \{\} = 0$
shows $(\lambda n.\ \sum i{<}n.\ f\ (\mathit{binaryset}\ A\ B\ i)) \longrightarrow f\ A + f\ B$
proof −
have $(\lambda n.\ \sum i < \mathit{Suc}\ (\mathit{Suc}\ n).\ f\ (\mathit{binaryset}\ A\ B\ i)) = (\lambda n.\ f\ A + f\ B)$
proof
fix n
show $(\sum i < \mathit{Suc}\ (\mathit{Suc}\ n).\ f\ (\mathit{binaryset}\ A\ B\ i)) = f\ A + f\ B$
by (*induct n*) (*auto simp add*: *binaryset_def f*)
qed
moreover
have $\ldots \longrightarrow f\ A + f\ B$ **by** (*rule tendsto_const*)
ultimately have $(\lambda n.\ \sum i{<}\ n{+}2.\ f\ (\mathit{binaryset}\ A\ B\ i)) \longrightarrow f\ A + f\ B$
by *simp*
thus *?thesis* **by** (*rule LIMSEQ_offset* [**where** $k{=}2$])
qed

lemma *binaryset_sums*:
assumes f: $f\ \{\} = 0$
shows $(\lambda n.\ f\ (\mathit{binaryset}\ A\ B\ n))\ \mathit{sums}\ (f\ A + f\ B)$
using *LIMSEQ_binaryset f sums_def* **by** *blast*

lemma *suminf_binaryset_eq*:
fixes $f :: 'a\ \mathit{set} \Rightarrow 'b{::}\{\mathit{comm\_monoid\_add}, \mathit{t2\_space}\}$
shows $f\ \{\} = 0 \Longrightarrow (\sum n.\ f\ (\mathit{binaryset}\ A\ B\ n)) = f\ A + f\ B$
by (*metis binaryset_sums sums_unique*)

6.3.3 Properties of a premeasure μ

The definitions for *positive* and *countably_additive* should be here, by they are necessary to define *'a measure* in *HOL−Analysis.Sigma_Algebra*.

definition *subadditive* **where**
$\mathit{subadditive}\ M\ f \longleftrightarrow (\forall x{\in}M.\ \forall y{\in}M.\ x \cap y = \{\} \longrightarrow f\ (x \cup y) \le f\ x + f\ y)$

lemma *subadditiveD*: $\mathit{subadditive}\ M\ f \Longrightarrow x \cap y = \{\} \Longrightarrow x \in M \Longrightarrow y \in M \Longrightarrow f\ (x \cup y) \le f\ x + f\ y$
by (*auto simp add*: *subadditive_def*)

definition *countably_subadditive* **where**
$\mathit{countably\_subadditive}\ M\ f \longleftrightarrow$
$(\forall A.\ \mathit{range}\ A \subseteq M \longrightarrow \mathit{disjoint\_family}\ A \longrightarrow (\bigcup i.\ A\ i) \in M \longrightarrow (f\ (\bigcup i.\ A\ i) \le (\sum i.\ f\ (A\ i))))$

lemma (**in** *ring_of_sets*) *countably_subadditive_subadditive*:
fixes $f :: 'a\ \mathit{set} \Rightarrow \mathit{ennreal}$
assumes f: *positive M f* **and** *cs*: *countably_subadditive M f*
shows *subadditive M f*
proof (*auto simp add*: *subadditive_def*)
fix $x\ y$
assume x: $x \in M$ **and** y: $y \in M$ **and** $x \cap y = \{\}$

hence *disjoint_family* (*binaryset x y*)
by (*auto simp add*: *disjoint_family_on_def binaryset_def*)
hence *range* (*binaryset x y*) ⊆ *M* ⟶
(⋃ *i*. *binaryset x y i*) ∈ *M* ⟶
f (⋃ *i*. *binaryset x y i*) ≤ (∑ *n*. *f* (*binaryset x y n*))
using *cs* **by** (*auto simp add*: *countably_subadditive_def*)
hence {*x*,*y*,{}} ⊆ *M* ⟶ *x* ∪ *y* ∈ *M* ⟶
f (*x* ∪ *y*) ≤ (∑ *n*. *f* (*binaryset x y n*))
by (*simp add*: *range_binaryset_eq UN_binaryset_eq*)
thus *f* (*x* ∪ *y*) ≤ *f x* + *f y* **using** *f x y*
by (*auto simp add*: *Un o_def suminf_binaryset_eq positive_def*)
qed

definition *additive* **where**
additive M μ ⟷ (∀ *x*∈*M*. ∀ *y*∈*M*. *x* ∩ *y* = {} ⟶ μ (*x* ∪ *y*) = μ *x* + μ *y*)

definition *increasing* **where**
increasing M μ ⟷ (∀ *x*∈*M*. ∀ *y*∈*M*. *x* ⊆ *y* ⟶ μ *x* ≤ μ *y*)

lemma *positiveD1*: *positive M f* ⟹ *f* {} = *0* **by** (*auto simp*: *positive_def*)

lemma *positiveD_empty*:
positive M f ⟹ *f* {} = *0*
by (*auto simp add*: *positive_def*)

lemma *additiveD*:
additive M f ⟹ *x* ∩ *y* = {} ⟹ *x* ∈ *M* ⟹ *y* ∈ *M* ⟹ *f* (*x* ∪ *y*) = *f x* + *f y*
by (*auto simp add*: *additive_def*)

lemma *increasingD*:
increasing M f ⟹ *x* ⊆ *y* ⟹ *x*∈*M* ⟹ *y*∈*M* ⟹ *f x* ≤ *f y*
by (*auto simp add*: *increasing_def*)

lemma *countably_additiveI*[*case_names countably*]:
(⋀*A*. ⟦*range A* ⊆ *M*; *disjoint_family A*; (⋃ *i*. *A i*) ∈ *M*⟧ ⟹ (∑ *i*. *f*(*A i*)) = *f*(⋃ *i*. *A i*))
⟹ *countably_additive M f*
by (*simp add*: *countably_additive_def*)

lemma (**in** *ring_of_sets*) *disjointed_additive*:
assumes *f*: *positive M f additive M f* **and** *A*: *range A* ⊆ *M incseq A*
shows (∑ *i*≤*n*. *f* (*disjointed A i*)) = *f* (*A n*)
proof (*induct n*)
case (*Suc n*)
then have (∑ *i*≤*Suc n*. *f* (*disjointed A i*)) = *f* (*A n*) + *f* (*disjointed A* (*Suc n*))
by *simp*
also have ... = *f* (*A n* ∪ *disjointed A* (*Suc n*))
using *A* **by** (*subst f*(*2*)[*THEN additiveD*]) (*auto simp*: *disjointed_mono*)
also have *A n* ∪ *disjointed A* (*Suc n*) = *A* (*Suc n*)

using ‹*incseq A*› **by** (*auto dest*: *incseq_SucD simp*: *disjointed_mono*)
finally show *?case* **.**
qed *simp*

lemma (**in** *ring_of_sets*) *additive_sum*:
fixes *A*:: *'i* ⇒ *'a set*
assumes *f*: *positive M f* **and** *ad*: *additive M f* **and** *finite S*
and *A*: *A'S* ⊆ *M*
and *disj*: *disjoint_family_on A S*
shows (∑ *i*∈*S*. *f* (*A i*)) = *f* (⋃ *i*∈*S*. *A i*)
using ‹*finite S*› *disj A*
proof *induct*
case *empty* **show** *?case* **using** *f* **by** (*simp add*: *positive_def*)
next
case (*insert s S*)
then have *A s* ∩ (⋃ *i*∈*S*. *A i*) = {}
by (*auto simp add*: *disjoint_family_on_def neq_iff*)
moreover
have *A s* ∈ *M* **using** *insert* **by** *blast*
moreover have (⋃ *i*∈*S*. *A i*) ∈ *M*
using *insert* ‹*finite S*› **by** *auto*
ultimately have *f* (*A s* ∪ (⋃ *i*∈*S*. *A i*)) = *f* (*A s*) + *f*(⋃ *i*∈*S*. *A i*)
using *ad UNION_in_sets A* **by** (*auto simp add*: *additive_def*)
with *insert* **show** *?case* **using** *ad disjoint_family_on_mono*[*of S insert s S A*]
by (*auto simp add*: *additive_def subset_insertI*)
qed

lemma (**in** *ring_of_sets*) *additive_increasing*:
fixes *f* :: *'a set* ⇒ *ennreal*
assumes *posf*: *positive M f* **and** *addf*: *additive M f*
shows *increasing M f*
proof (*auto simp add*: *increasing_def*)
fix *x y*
assume *xy*: *x* ∈ *M y* ∈ *M x* ⊆ *y*
then have *y* − *x* ∈ *M* **by** *auto*
then have *f x* + *0* ≤ *f x* + *f* (*y*−*x*) **by** (*intro add_left_mono zero_le*)
also have ... = *f* (*x* ∪ (*y*−*x*))
by (*metis addf Diff_disjoint* ‹*y* − *x* ∈ *M*› *additiveD xy*(*1*))
also have ... = *f y*
by (*metis Un_Diff_cancel Un_absorb1 xy*(*3*))
finally show *f x* ≤ *f y* **by** *simp*
qed

lemma (**in** *ring_of_sets*) *subadditive*:
fixes *f* :: *'a set* ⇒ *ennreal*
assumes *f*: *positive M f additive M f* **and** *A*: *A'S* ⊆ *M* **and** *S*: *finite S*
shows *f* (⋃ *i*∈*S*. *A i*) ≤ (∑ *i*∈*S*. *f* (*A i*))
using *S A*
proof (*induct S*)

case *empty* **thus** *?case* **using** *f* **by** (*auto simp*: *positive_def*)
next
case (*insert x F*)
hence *in_M*: $A\ x \in M$ $(\bigcup i{\in}F.\ A\ i) \in M$ $(\bigcup i{\in}F.\ A\ i) - A\ x \in M$ **using** *A*
by *force+*
have *subs*: $(\bigcup i{\in}F.\ A\ i) - A\ x \subseteq (\bigcup i{\in}F.\ A\ i)$ **by** *auto*
have $(\bigcup i{\in}(\mathit{insert}\ x\ F).\ A\ i) = A\ x \cup ((\bigcup i{\in}F.\ A\ i) - A\ x)$ **by** *auto*
hence $f\ (\bigcup i{\in}(\mathit{insert}\ x\ F).\ A\ i) = f\ (A\ x \cup ((\bigcup i{\in}F.\ A\ i) - A\ x))$
by *simp*
also have $\ldots = f\ (A\ x) + f\ ((\bigcup i{\in}F.\ A\ i) - A\ x)$
using *f*(*2*) **by** (*rule additiveD*) (*insert in_M*, *auto*)
also have $\ldots \leq f\ (A\ x) + f\ (\bigcup i{\in}F.\ A\ i)$
using *additive_increasing*[*OF f*] *in_M subs*
by (*simp add*: *increasingD*)
also have $\ldots \leq f\ (A\ x) + (\sum i{\in}F.\ f\ (A\ i))$
using *insert* **by** (*auto intro*: *add_left_mono*)
finally show $f\ (\bigcup i{\in}(\mathit{insert}\ x\ F).\ A\ i) \leq (\sum i{\in}(\mathit{insert}\ x\ F).\ f\ (A\ i))$
by (*simp add*: *insert*)
qed

lemma (**in** *ring_of_sets*) *countably_additive_additive*:
fixes *f* :: *'a set* ⇒ *ennreal*
assumes *posf*: *positive M f* **and** *ca*: *countably_additive M f*
shows *additive M f*
proof (*auto simp add*: *additive_def*)
fix *x y*
assume *x*: $x \in M$ **and** *y*: $y \in M$ **and** $x \cap y = \{\}$
hence *disjoint_family* (*binaryset x y*)
by (*auto simp add*: *disjoint_family_on_def binaryset_def*)
hence *range* (*binaryset x y*) $\subseteq M \longrightarrow$
$(\bigcup i.\ \mathit{binaryset}\ x\ y\ i) \in M \longrightarrow$
$f\ (\bigcup i.\ \mathit{binaryset}\ x\ y\ i) = (\sum\ n.\ f\ (\mathit{binaryset}\ x\ y\ n))$
using *ca* **by** (*simp add*: *countably_additive_def*)
hence $\{x,y,\{\}\} \subseteq M \longrightarrow x \cup y \in M \longrightarrow f\ (x \cup y) = (\sum n.\ f\ (\mathit{binaryset}\ x\ y\ n))$
by (*simp add*: *range_binaryset_eq UN_binaryset_eq*)
thus $f\ (x \cup y) = f\ x + f\ y$ **using** *posf x y*
by (*auto simp add*: *Un suminf_binaryset_eq positive_def*)
qed

lemma (**in** *algebra*) *increasing_additive_bound*:
fixes *A*:: *nat* ⇒ *'a set* **and** *f* :: *'a set* ⇒ *ennreal*
assumes *f*: *positive M f* **and** *ad*: *additive M f*
and *inc*: *increasing M f*
and *A*: *range A* $\subseteq M$
and *disj*: *disjoint_family A*
shows $(\sum i.\ f\ (A\ i)) \leq f\ \Omega$
proof (*safe intro*!: *suminf_le_const*)
fix *N*
note *disj_N* = *disjoint_family_on_mono*[*OF* _ *disj*, *of* {..<*N*}]

```
  have (∑ i<N. f (A i)) = f (⋃ i∈{..<N}. A i)
    using A by (intro additive_sum [OF f ad]) (auto simp: disj_N)
  also have ... ≤ f Ω using space_closed A
    by (intro increasingD[OF inc] finite_UN) auto
  finally show (∑ i<N. f (A i)) ≤ f Ω by simp
qed (insert f A, auto simp: positive_def)

lemma (in ring_of_sets) countably_additiveI_finite:
  fixes μ :: 'a set ⇒ ennreal
  assumes finite Ω positive M μ additive M μ
  shows countably_additive M μ
proof (rule countably_additiveI)
  fix F :: nat ⇒ 'a set assume F: range F ⊆ M (⋃ i. F i) ∈ M and disj:
disjoint_family F

  have ∀ i. F i ≠ {} ⟶ (∃ x. x ∈ F i) by auto
  then obtain f where f: ⋀i. F i ≠ {} ⟹ f i ∈ F i by metis

  have finU: finite (⋃ i. F i)
    by (metis F(2) assms(1) infinite_super sets_into_space)

  have F_subset: {i. μ (F i) ≠ 0} ⊆ {i. F i ≠ {}}
    by (auto simp: positiveD_empty[OF ‹positive M μ›])
  moreover have fin_not_empty: finite {i. F i ≠ {}}
  proof (rule finite_imageD)
    from f have f`{i. F i ≠ {}} ⊆ (⋃ i. F i) by auto
    then show finite (f`{i. F i ≠ {}})
      by (simp add: finU finite_subset)
    show inj_f: inj_on f {i. F i ≠ {}}
      using f disj
      by (simp add: inj_on_def disjoint_family_on_def disjoint_iff) metis
  qed
  ultimately have fin_not_0: finite {i. μ (F i) ≠ 0}
    by (rule finite_subset)

  have disj_not_empty: disjoint_family_on F {i. F i ≠ {}}
    using disj by (auto simp: disjoint_family_on_def)

  from fin_not_0 have (∑ i. μ (F i)) = (∑ i | μ (F i) ≠ 0. μ (F i))
    by (rule suminf_finite) auto
  also have ... = (∑ i | F i ≠ {}. μ (F i))
    using fin_not_empty F_subset by (rule sum.mono_neutral_left) auto
  also have ... = μ (⋃ i∈{i. F i ≠ {}}. F i)
    using ‹positive M μ› ‹additive M μ› fin_not_empty disj_not_empty F by
(intro additive_sum) auto
  also have ... = μ (⋃ i. F i)
    by (rule arg_cong[where f=μ]) auto
  finally show (∑ i. μ (F i)) = μ (⋃ i. F i) .
qed
```

lemma (**in** *ring_of_sets*) *countably_additive_iff_continuous_from_below*:
fixes $f :: 'a\ set \Rightarrow ennreal$
assumes f: *positive M f additive M f*
shows *countably_additive M f* $\longleftrightarrow$
$(\forall A.\ range\ A \subseteq M \longrightarrow incseq\ A \longrightarrow (\bigcup i.\ A\ i) \in M \longrightarrow (\lambda i.\ f\ (A\ i)) \longrightarrow f\ (\bigcup i.\ A\ i))$
unfolding *countably_additive_def*
proof *safe*
assume *count_sum*: $\forall A.\ range\ A \subseteq M \longrightarrow disjoint\_family\ A \longrightarrow \bigcup (A\ `\ UNIV) \in M \longrightarrow (\sum i.\ f\ (A\ i)) = f\ (\bigcup (A\ `\ UNIV))$
fix $A :: nat \Rightarrow 'a\ set$ **assume** A: *range A* $\subseteq$ *M incseq A* $(\bigcup i.\ A\ i) \in M$
then have dA: *range* (*disjointed A*) $\subseteq$ *M* **by** (*auto simp*: *range_disjointed_sets*)
with *count_sum*[*THEN spec*, *of disjointed A*] *A*(*3*)
have *f_UN*: $(\sum i.\ f\ (disjointed\ A\ i)) = f\ (\bigcup i.\ A\ i)$
by (*auto simp*: *UN_disjointed_eq disjoint_family_disjointed*)
moreover have $(\lambda n.\ (\sum i{<}n.\ f\ (disjointed\ A\ i))) \longrightarrow (\sum i.\ f\ (disjointed\ A\ i))$
by (*simp add*: *summable_LIMSEQ*)
from *LIMSEQ_Suc*[*OF this*]
have $(\lambda n.\ (\sum i{\leq}n.\ f\ (disjointed\ A\ i))) \longrightarrow (\sum i.\ f\ (disjointed\ A\ i))$
unfolding *lessThan_Suc_atMost* **.**
moreover have $\bigwedge n.\ (\sum i{\leq}n.\ f\ (disjointed\ A\ i)) = f\ (A\ n)$
using *disjointed_additive*[*OF f A*(*1*,*2*)] **.**
ultimately show $(\lambda i.\ f\ (A\ i)) \longrightarrow f\ (\bigcup i.\ A\ i)$ **by** *simp*
next
assume *cont*[*rule_format*]: $\forall A.\ range\ A \subseteq M \longrightarrow incseq\ A \longrightarrow (\bigcup i.\ A\ i) \in M \longrightarrow (\lambda i.\ f\ (A\ i)) \longrightarrow f\ (\bigcup i.\ A\ i)$
fix $A :: nat \Rightarrow 'a\ set$ **assume** A: *range A* $\subseteq$ *M disjoint_family A* $(\bigcup i.\ A\ i) \in M$
have $*$: $(\bigcup n.\ (\bigcup i{<}n.\ A\ i)) = (\bigcup i.\ A\ i)$ **by** *auto*
have *range* $(\lambda i.\ \bigcup i{<}i.\ A\ i) \subseteq M$ $(\bigcup i.\ \bigcup i{<}i.\ A\ i) \in M$
using A $*$ **by** *auto*
then have $(\lambda n.\ f\ (\bigcup i{<}n.\ A\ i)) \longrightarrow f\ (\bigcup i.\ A\ i)$
unfolding $*$[*symmetric*] **by** (*force intro*!: *cont incseq_SucI*)+
moreover have $\bigwedge n.\ f\ (\bigcup i{<}n.\ A\ i) = (\sum i{<}n.\ f\ (A\ i))$
using A
by (*intro additive_sum*[*OF f*, *symmetric*]) (*auto intro*: *disjoint_family_on_mono*)
ultimately
have $(\lambda i.\ f\ (A\ i))\ sums\ f\ (\bigcup i.\ A\ i)$
unfolding *sums_def* **by** *simp*
then show $(\sum i.\ f\ (A\ i)) = f\ (\bigcup i.\ A\ i)$
by (*metis sums_unique*)
qed

lemma (**in** *ring_of_sets*) *continuous_from_above_iff_empty_continuous*:
fixes $f :: 'a\ set \Rightarrow ennreal$
assumes f: *positive M f additive M f*
shows $(\forall A.\ range\ A \subseteq M \longrightarrow decseq\ A \longrightarrow (\bigcap i.\ A\ i) \in M \longrightarrow (\forall i.\ f\ (A\ i) \neq \infty) \longrightarrow (\lambda i.\ f\ (A\ i)) \longrightarrow f\ (\bigcap i.\ A\ i))$

$\longleftrightarrow$ ($\forall$ *A. range A* $\subseteq$ *M* $\longrightarrow$ *decseq A* $\longrightarrow$ ($\bigcap$ *i. A i*) = {} $\longrightarrow$ ($\forall$ *i. f* (*A i*) $\neq$ ∞) $\longrightarrow$ (λ*i. f* (*A i*)) $\longrightarrow$ *0*)
proof *safe*
assume *cont*[*rule_format*]: ($\forall$ *A. range A* $\subseteq$ *M* $\longrightarrow$ *decseq A* $\longrightarrow$ ($\bigcap$ *i. A i*) $\in$ *M* $\longrightarrow$ ($\forall$ *i. f* (*A i*) $\neq$ ∞) $\longrightarrow$ (λ*i. f* (*A i*)) $\longrightarrow$ *f* ($\bigcap$ *i. A i*))
fix *A* :: *nat* $\Rightarrow$ *'a set*
assume *A*: *range A* $\subseteq$ *M decseq A* ($\bigcap$ *i. A i*) = {} $\forall$ *i. f* (*A i*) $\neq$ ∞
with *cont*[*of A*] **show** (λ*i. f* (*A i*)) $\longrightarrow$ *0*
using ‹*positive M f*›[*unfolded positive_def*] **by** *auto*
next
assume *cont*[*rule_format*]: $\forall$ *A. range A* $\subseteq$ *M* $\longrightarrow$ *decseq A* $\longrightarrow$ ($\bigcap$ *i. A i*) = {} $\longrightarrow$ ($\forall$ *i. f* (*A i*) $\neq$ ∞) $\longrightarrow$ (λ*i. f* (*A i*)) $\longrightarrow$ *0*
fix *A* :: *nat* $\Rightarrow$ *'a set*
assume *A*: *range A* $\subseteq$ *M decseq A* ($\bigcap$ *i. A i*) $\in$ *M* $\forall$ *i. f* (*A i*) $\neq$ ∞

have *f_mono*: $\bigwedge$*a b. a* $\in$ *M* $\Longrightarrow$ *b* $\in$ *M* $\Longrightarrow$ *a* $\subseteq$ *b* $\Longrightarrow$ *f a* $\leq$ *f b*
using *additive_increasing*[*OF f*] **unfolding** *increasing_def* **by** *simp*

have *decseq_fA*: *decseq* (λ*i. f* (*A i*))
using *A* **by** (*auto simp*: *decseq_def intro*!: *f_mono*)
have *decseq*: *decseq* (λ*i. A i* $-$ ($\bigcap$ *i. A i*))
using *A* **by** (*auto simp*: *decseq_def*)
then have *decseq_f*: *decseq* (λ*i. f* (*A i* $-$ ($\bigcap$ *i. A i*)))
using *A* **unfolding** *decseq_def* **by** (*auto intro*!: *f_mono Diff*)
have *f* ($\bigcap$ *x. A x*) $\leq$ *f* (*A 0*)
using *A* **by** (*auto intro*!: *f_mono*)
then have *f_Int_fin*: *f* ($\bigcap$ *x. A x*) $\neq$ ∞
using *A* **by** (*auto simp*: *top_unique*)
have *f_fin*: *f* (*A i* $-$ ($\bigcap$ *i. A i*)) $\neq$ ∞ **for** *i*
using *A* **by** (*metis Diff Diff_subset f_mono infinity_ennreal_def range_subsetD top_unique*)
have (λ*i. f* (*A i* $-$ ($\bigcap$ *i. A i*))) $\longrightarrow$ *0*
proof (*intro cont*[*OF* _ *decseq* _ *f_fin*])
show *range* (λ*i. A i* $-$ ($\bigcap$ *i. A i*)) $\subseteq$ *M* ($\bigcap$ *i. A i* $-$ ($\bigcap$ *i. A i*)) = {}
using *A* **by** *auto*
qed
with *INF_Lim decseq_f* **have** (*INF n. f* (*A n* $-$ ($\bigcap$ *i. A i*))) = *0* **by** *metis*
moreover have (*INF n. f* ($\bigcap$ *i. A i*)) = *f* ($\bigcap$ *i. A i*)
by *auto*
ultimately have (*INF n. f* (*A n* $-$ ($\bigcap$ *i. A i*)) + *f* ($\bigcap$ *i. A i*)) = *0* + *f* ($\bigcap$ *i. A i*)
using *A*(*4*) *f_fin f_Int_fin*
using *INF_ennreal_add_const* **by** *presburger*
moreover {
fix *n*
have *f* (*A n* $-$ ($\bigcap$ *i. A i*)) + *f* ($\bigcap$ *i. A i*) = *f* ((*A n* $-$ ($\bigcap$ *i. A i*)) $\cup$ ($\bigcap$ *i. A i*))
using *A* **by** (*subst f*(*2*)[*THEN additiveD*]) *auto*
also have (*A n* $-$ ($\bigcap$ *i. A i*)) $\cup$ ($\bigcap$ *i. A i*) = *A n*
by *auto*

finally have $f\ (A\ n - (\bigcap i.\ A\ i)) + f\ (\bigcap i.\ A\ i) = f\ (A\ n)$ **. }**
ultimately have $(INF\ n.\ f\ (A\ n)) = f\ (\bigcap i.\ A\ i)$
by *simp*
with *LIMSEQ_INF*[*OF decseq_fA*]
show $(\lambda i.\ f\ (A\ i)) \longrightarrow f\ (\bigcap i.\ A\ i)$ **by** *simp*
qed

lemma (**in** *ring_of_sets*) *empty_continuous_imp_continuous_from_below*:
fixes $f :: 'a\ set \Rightarrow ennreal$
assumes *f*: *positive M f additive M f* $\forall A{\in}M.\ f\ A \neq \infty$
assumes *cont*: $\forall A.\ range\ A \subseteq M \longrightarrow decseq\ A \longrightarrow (\bigcap i.\ A\ i) = \{\} \longrightarrow (\lambda i.\ f\ (A\ i)) \longrightarrow 0$
assumes *A*: *range A* $\subseteq$ *M incseq A* $(\bigcup i.\ A\ i) \in M$
shows $(\lambda i.\ f\ (A\ i)) \longrightarrow f\ (\bigcup i.\ A\ i)$
proof −
from *A* **have** $(\lambda i.\ f\ ((\bigcup i.\ A\ i) - A\ i)) \longrightarrow 0$
by (*intro cont*[*rule_format*]) (*auto simp*: *decseq_def incseq_def*)
moreover
{ fix *i*
have $f\ ((\bigcup i.\ A\ i) - A\ i \cup A\ i) = f\ ((\bigcup i.\ A\ i) - A\ i) + f\ (A\ i)$
using *A* **by** (*intro f*(*2*)[*THEN additiveD*]) *auto*
also have $((\bigcup i.\ A\ i) - A\ i) \cup A\ i = (\bigcup i.\ A\ i)$
by *auto*
finally have $f\ ((\bigcup i.\ A\ i) - A\ i) = f\ (\bigcup i.\ A\ i) - f\ (A\ i)$
using *assms f* **by** *fastforce*
}
moreover have $\forall_F\ i\ in\ sequentially.\ f\ (A\ i) \leq f\ (\bigcup i.\ A\ i)$
using *increasingD*[*OF additive_increasing*[*OF f*(*1*, *2*)], *of A _* $\bigcup i.\ A\ i$] *A*
by (*auto intro*!: *always_eventually simp*: *subset_eq*)
ultimately show $(\lambda i.\ f\ (A\ i)) \longrightarrow f\ (\bigcup i.\ A\ i)$
by (*auto intro*: *ennreal_tendsto_const_minus*)
qed

lemma (**in** *ring_of_sets*) *empty_continuous_imp_countably_additive*:
fixes $f :: 'a\ set \Rightarrow ennreal$
assumes *f*: *positive M f* **and** *fin*: $\forall A{\in}M.\ f\ A \neq \infty$
assumes *cont*: $\bigwedge A.\ range\ A \subseteq M \Longrightarrow decseq\ A \Longrightarrow (\bigcap i.\ A\ i) = \{\} \Longrightarrow (\lambda i.\ f\ (A\ i)) \longrightarrow 0$
shows *countably_additive M f*
using *countably_additive_iff_continuous_from_below*[*OF f*]
using *empty_continuous_imp_continuous_from_below*[*OF f fin*] *cont*
by *blast*

6.3.4 Properties of *emeasure*

lemma *emeasure_positive*: *positive* (*sets M*) (*emeasure M*)
by (*cases M*) (*auto simp*: *sets_def emeasure_def Abs_measure_inverse measure_space_def*)

lemma *emeasure_empty*[*simp*, *intro*]: *emeasure M* {} = *0*
using *emeasure_positive*[*of M*] **by** (*simp add*: *positive_def*)

lemma *emeasure_single_in_space*: *emeasure M* {*x*} ≠ *0* ⟹ *x* ∈ *space M*
using *emeasure_notin_sets*[*of* {*x*} *M*] **by** (*auto dest*: *sets.sets_into_space zero_less_iff_neq_zero*[*TH*
iffD2])

lemma *emeasure_countably_additive*: *countably_additive* (*sets M*) (*emeasure M*)
by (*cases M*) (*auto simp*: *sets_def emeasure_def Abs_measure_inverse measure_space_def*)

lemma *suminf_emeasure*:
range A ⊆ *sets M* ⟹ *disjoint_family A* ⟹ (∑ *i*. *emeasure M* (*A i*)) = *emeasure M* (⋃ *i*. *A i*)
using *sets.countable_UN*[*of A UNIV M*] *emeasure_countably_additive*[*of M*]
by (*simp add*: *countably_additive_def*)

lemma *sums_emeasure*:
disjoint_family F ⟹ (⋀*i*. *F i* ∈ *sets M*) ⟹ (λ*i*. *emeasure M* (*F i*)) *sums* *emeasure M* (⋃ *i*. *F i*)
unfolding *sums_iff* **by** (*intro conjI suminf_emeasure*) *auto*

lemma *emeasure_additive*: *additive* (*sets M*) (*emeasure M*)
by (*metis sets.countably_additive_additive emeasure_positive emeasure_countably_additive*)

lemma *plus_emeasure*:
a ∈ *sets M* ⟹ *b* ∈ *sets M* ⟹ *a* ∩ *b* = {} ⟹ *emeasure M a* + *emeasure M b* = *emeasure M* (*a* ∪ *b*)
using *additiveD*[*OF emeasure_additive*] **..**

lemma *emeasure_Un*:
A ∈ *sets M* ⟹ *B* ∈ *sets M* ⟹ *emeasure M* (*A* ∪ *B*) = *emeasure M A* + *emeasure M* (*B* − *A*)
using *plus_emeasure*[*of A M B* − *A*] **by** *auto*

lemma *emeasure_Un_Int*:
assumes *A* ∈ *sets M B* ∈ *sets M*
shows *emeasure M A* + *emeasure M B* = *emeasure M* (*A* ∪ *B*) + *emeasure M* (*A* ∩ *B*)
proof −
have *A* = (*A*−*B*) ∪ (*A* ∩ *B*) **by** *auto*
then have *emeasure M A* = *emeasure M* (*A*−*B*) + *emeasure M* (*A* ∩ *B*)
by (*metis Diff_Diff_Int Diff_disjoint assms plus_emeasure sets.Diff*)
moreover have *A* ∪ *B* = (*A*−*B*) ∪ *B* **by** *auto*
then have *emeasure M* (*A* ∪ *B*) = *emeasure M* (*A*−*B*) + *emeasure M B*
by (*metis Diff_disjoint Int_commute assms plus_emeasure sets.Diff*)
ultimately show *?thesis* **by** (*metis add.assoc add.commute*)
qed

lemma *sum_emeasure*:
F'I ⊆ *sets M* ⟹ *disjoint_family_on F I* ⟹ *finite I* ⟹
(∑ *i*∈*I*. *emeasure M* (*F i*)) = *emeasure M* (⋃ *i*∈*I*. *F i*)
by (*metis sets.additive_sum emeasure_positive emeasure_additive*)

lemma *emeasure_mono*:
a ⊆ *b* ⟹ *b* ∈ *sets M* ⟹ *emeasure M a* ≤ *emeasure M b*
by (*metis zero_le sets.additive_increasing emeasure_additive emeasure_notin_sets emeasure_positive increasingD*)

lemma *emeasure_space*:
emeasure M A ≤ *emeasure M* (*space M*)
by (*metis emeasure_mono emeasure_notin_sets sets.sets_into_space sets.top zero_le*)

lemma *emeasure_Diff*:
assumes *emeasure M B* ≠ ∞
and *A* ∈ *sets M B* ∈ *sets M* **and** *B* ⊆ *A*
shows *emeasure M* (*A* − *B*) = *emeasure M A* − *emeasure M B*
by (*smt* (*verit*, *best*) *add_diff_self_ennreal assms emeasure_Un emeasure_mono*

ennreal_add_left_cancel le_iff_sup)

lemma *emeasure_compl*:
s ∈ *sets M* ⟹ *emeasure M s* ≠ ∞ ⟹ *emeasure M* (*space M* − *s*) = *emeasure M* (*space M*) − *emeasure M s*
by (*rule emeasure_Diff*) (*auto dest*: *sets.sets_into_space*)

lemma *Lim_emeasure_incseq*:
range A ⊆ *sets M* ⟹ *incseq A* ⟹ (λ*i*. (*emeasure M* (*A i*))) ⟶ *emeasure M* (⋃ *i*. *A i*)
using *emeasure_countably_additive*
by (*auto simp add*: *sets.countably_additive_iff_continuous_from_below emeasure_positive*
emeasure_additive)

lemma *incseq_emeasure*:
assumes *range B* ⊆ *sets M incseq B*
shows *incseq* (λ*i*. *emeasure M* (*B i*))
using *assms* **by** (*auto simp*: *incseq_def intro*!: *emeasure_mono*)

lemma *SUP_emeasure_incseq*:
assumes *A*: *range A* ⊆ *sets M incseq A*
shows (*SUP n*. *emeasure M* (*A n*)) = *emeasure M* (⋃ *i*. *A i*)
using *LIMSEQ_SUP*[*OF incseq_emeasure*, *OF A*] *Lim_emeasure_incseq*[*OF A*]
by (*simp add*: *LIMSEQ_unique*)

lemma *decseq_emeasure*:
assumes *range B* ⊆ *sets M decseq B*

shows *decseq* (λ*i*. *emeasure M* (*B i*))
using *assms* **by** (*auto simp*: *decseq_def intro*!: *emeasure_mono*)

lemma *INF_emeasure_decseq*:
assumes *A*: *range A* ⊆ *sets M* **and** *decseq A*
and *finite*: ⋀*i*. *emeasure M* (*A i*) ≠ ∞
shows (*INF n*. *emeasure M* (*A n*)) = *emeasure M* (⋂ *i*. *A i*)
proof −
have *le_MI*: *emeasure M* (⋂ *i*. *A i*) ≤ *emeasure M* (*A 0*)
using *A* **by** (*auto intro*!: *emeasure_mono*)
hence ∗: *emeasure M* (⋂ *i*. *A i*) ≠ ∞ **using** *finite*[*of 0*] **by** (*auto simp*: *top_unique*)

have *emeasure M* (*A 0*) − (*INF n*. *emeasure M* (*A n*)) = (*SUP n*. *emeasure M* (*A 0*) − *emeasure M* (*A n*))
by (*simp add*: *ennreal_INF_const_minus*)
also have ... = (*SUP n*. *emeasure M* (*A 0* − *A n*))
using *A finite* ‹*decseq A*›[*unfolded decseq_def*] **by** (*subst emeasure_Diff*) *auto*
also have ... = *emeasure M* (⋃ *i*. *A 0* − *A i*)
proof (*rule SUP_emeasure_incseq*)
show *range* (λ*n*. *A 0* − *A n*) ⊆ *sets M*
using *A* **by** *auto*
show *incseq* (λ*n*. *A 0* − *A n*)
using ‹*decseq A*› **by** (*auto simp add*: *incseq_def decseq_def*)
qed
also have ... = *emeasure M* (*A 0*) − *emeasure M* (⋂ *i*. *A i*)
using *A finite* ∗ **by** (*simp*, *subst emeasure_Diff*) *auto*
finally show *?thesis*
by (*smt* (*verit*, *best*) *Inf_lower diff_diff_ennreal le_MI finite range_eqI*)
qed

lemma *INF_emeasure_decseq'*:
assumes *A*: ⋀*i*. *A i* ∈ *sets M* **and** *decseq A*
and *finite*: ∃ *i*. *emeasure M* (*A i*) ≠ ∞
shows (*INF n*. *emeasure M* (*A n*)) = *emeasure M* (⋂ *i*. *A i*)
proof −
from *finite* **obtain** *i* **where** *i*: *emeasure M* (*A i*) < ∞
by (*auto simp*: *less_top*)
have *fin*: *i* ≤ *j* ⟹ *emeasure M* (*A j*) < ∞ **for** *j*
by (*rule le_less_trans*[*OF emeasure_mono i*]) (*auto intro*!: *decseqD*[*OF* ‹*decseq A*›] *A*)

have (*INF n*. *emeasure M* (*A n*)) = (*INF n*. *emeasure M* (*A* (*n* + *i*)))
proof (*rule INF_eq*)
show ∃ *j*∈*UNIV*. *emeasure M* (*A* (*j* + *i*)) ≤ *emeasure M* (*A i'*) **for** *i'*
by (*meson A* ‹*decseq A*› *decseq_def emeasure_mono iso_tuple_UNIV_I nat_le_iff_add*)
qed *auto*
also have ... = *emeasure M* (*INF n*. (*A* (*n* + *i*)))
using *A* ‹*decseq A*› *fin* **by** (*intro INF_emeasure_decseq*) (*auto simp*: *decseq_def*

less_top)
also have (*INF n.* (*A* (*n* + *i*))) = (*INF n. A n*)
by (*meson INF_eq UNIV_I assms*(*2*) *decseqD le_add1*)
finally show *?thesis* **.**
qed

lemma *emeasure_INT_decseq_subset*:
fixes *F* :: *nat* ⇒ *'a set*
assumes *I*: *I* ≠ {} **and** *F*: ⋀*i j*. *i* ∈ *I* ⟹ *j* ∈ *I* ⟹ *i* ≤ *j* ⟹ *F j* ⊆ *F i*
assumes *F_sets*[*measurable*]: ⋀*i*. *i* ∈ *I* ⟹ *F i* ∈ *sets M*
and *fin*: ⋀*i*. *i* ∈ *I* ⟹ *emeasure M* (*F i*) ≠ ∞
shows *emeasure M* (⋂*i*∈*I*. *F i*) = (*INF i*∈*I*. *emeasure M* (*F i*))
proof *cases*
assume *finite I*
have (⋂*i*∈*I*. *F i*) = *F* (*Max I*)
using *I* ‹*finite I*› **by** (*intro antisym INF_lower INF_greatest F*) *auto*
moreover have (*INF i*∈*I*. *emeasure M* (*F i*)) = *emeasure M* (*F* (*Max I*))
using *I* ‹*finite I*› **by** (*intro antisym INF_lower INF_greatest F emeasure_mono*)
auto
ultimately show *?thesis*
by *simp*
next
assume *infinite I*
define *L* **where** *L n* = (*LEAST i*. *i* ∈ *I* ∧ *i* ≥ *n*) **for** *n*
have *L*: *L n* ∈ *I* ∧ *n* ≤ *L n* **for** *n*
unfolding *L_def*
proof (*rule LeastI_ex*)
show ∃*x*. *x* ∈ *I* ∧ *n* ≤ *x*
using ‹*infinite I*› *finite_subset*[*of I* {..< *n*}]
by (*rule_tac ccontr*) (*auto simp*: *not_le*)
qed
have *L_eq*[*simp*]: *i* ∈ *I* ⟹ *L i* = *i* **for** *i*
unfolding *L_def* **by** (*intro Least_equality*) *auto*
have *L_mono*: *i* ≤ *j* ⟹ *L i* ≤ *L j* **for** *i j*
using *L*[*of j*] **unfolding** *L_def* **by** (*intro Least_le*) (*auto simp*: *L_def*)

have *emeasure M* (⋂*i*. *F* (*L i*)) = (*INF i*. *emeasure M* (*F* (*L i*)))
proof (*intro INF_emeasure_decseq*[*symmetric*])
show *decseq* (λ*i*. *F* (*L i*))
using *L* **by** (*intro antimonoI F L_mono*) *auto*
qed (*insert L fin*, *auto*)
also have ... = (*INF i*∈*I*. *emeasure M* (*F i*))
proof (*intro antisym INF_greatest*)
show *i* ∈ *I* ⟹ (*INF i*. *emeasure M* (*F* (*L i*))) ≤ *emeasure M* (*F i*) **for** *i*
by (*intro INF_lower2*[*of i*]) *auto*
qed (*insert L*, *auto intro*: *INF_lower*)
also have (⋂*i*. *F* (*L i*)) = (⋂*i*∈*I*. *F i*)
proof (*intro antisym INF_greatest*)
show *i* ∈ *I* ⟹ (⋂*i*. *F* (*L i*)) ⊆ *F i* **for** *i*

by (*intro INF_lower2*[*of i*]) *auto*
qed (*insert L*, *auto*)
finally show *?thesis* **.**
qed

lemma *Lim_emeasure_decseq*:
assumes *A*: *range A* ⊆ *sets M decseq A* **and** *fin*: ⋀*i*. *emeasure M* (*A i*) ≠ ∞
shows (λ*i*. *emeasure M* (*A i*)) ⟶ *emeasure M* (⋂ *i*. *A i*)
using *LIMSEQ_INF*[*OF decseq_emeasure*, *OF A*]
using *INF_emeasure_decseq*[*OF A fin*] **by** *simp*

lemma *emeasure_lfp'*[*consumes 1*, *case_names cont measurable*]:
assumes *P M*
assumes *cont*: *sup_continuous F*
assumes ∗: ⋀*M A*. *P M* ⟹ (⋀*N*. *P N* ⟹ *Measurable.pred N A*) ⟹ *Measurable.pred M* (*F A*)
shows *emeasure M* {*x*∈*space M*. *lfp F x*} = (*SUP i*. *emeasure M* {*x*∈*space M*. (*F* ^^ *i*) (λ*x*. *False*) *x*})
proof −
have *emeasure M* {*x*∈*space M*. *lfp F x*} = *emeasure M* (⋃ *i*. {*x*∈*space M*. (*F* ^^ *i*) (λ*x*. *False*) *x*})
using *sup_continuous_lfp*[*OF cont*] **by** (*auto simp add*: *bot_fun_def intro*!: *arg_cong2*[**where** *f*=*emeasure*])
moreover { fix *i* **from** ‹*P M*› **have** {*x*∈*space M*. (*F* ^^ *i*) (λ*x*. *False*) *x*} ∈ *sets M*
by (*induct i arbitrary*: *M*) (*auto simp add*: *pred_def*[*symmetric*] *intro*: ∗) **}**
moreover have *incseq* (λ*i*. {*x*∈*space M*. (*F* ^^ *i*) (λ*x*. *False*) *x*})
proof (*rule incseq_SucI*)
fix *i*
have (*F* ^^ *i*) (λ*x*. *False*) ≤ (*F* ^^ (*Suc i*)) (λ*x*. *False*)
proof (*induct i*)
case *0* **show** *?case* **by** (*simp add*: *le_fun_def*)
next
case *Suc* **thus** *?case* **using** *monoD*[*OF sup_continuous_mono*[*OF cont*] *Suc*] **by** *auto*
qed
then show {*x* ∈ *space M*. (*F* ^^ *i*) (λ*x*. *False*) *x*} ⊆ {*x* ∈ *space M*. (*F* ^^ *Suc i*) (λ*x*. *False*) *x*}
by *auto*
qed
ultimately show *?thesis*
by (*subst SUP_emeasure_incseq*) *auto*
qed

lemma *emeasure_lfp*:
assumes [*simp*]: ⋀*s*. *sets* (*M s*) = *sets N*
assumes *cont*: *sup_continuous F sup_continuous f*
assumes *meas*: ⋀*P*. *Measurable.pred N P* ⟹ *Measurable.pred N* (*F P*)
assumes *iter*: ⋀*P s*. *Measurable.pred N P* ⟹ *P* ≤ *lfp F* ⟹ *emeasure* (*M s*)

$\{x{\in}space\ N.\ F\ P\ x\} = f\ (\lambda s.\ emeasure\ (M\ s)\ \{x{\in}space\ N.\ P\ x\})\ s$
shows *emeasure* $(M\ s)\ \{x{\in}space\ N.\ lfp\ F\ x\} = lfp\ f\ s$
proof (*subst lfp_transfer_bounded*[**where** $\alpha{=}\lambda F\ s.\ emeasure\ (M\ s)\ \{x{\in}space\ N.\ F\ x\}$ **and** $f{=}F$, *symmetric*])
fix C **assume** *incseq* C $\bigwedge i.$ *Measurable.pred* $N\ (C\ i)$
then show $(\lambda s.\ emeasure\ (M\ s)\ \{x \in space\ N.\ (SUP\ i.\ C\ i)\ x\}) = (SUP\ i.\ (\lambda s.\ emeasure\ (M\ s)\ \{x \in space\ N.\ C\ i\ x\}))$
unfolding *SUP_apply*
by (*subst SUP_emeasure_incseq*) (*auto simp*: *mono_def fun_eq_iff intro*!: *arg_cong2*[**where** $f{=}emeasure$])
qed (*auto simp add*: *iter le_fun_def SUP_apply intro*!: *meas cont*)

lemma *emeasure_subadditive_finite*:
finite $I \Longrightarrow A\ `\ I \subseteq sets\ M \Longrightarrow emeasure\ M\ (\bigcup i{\in}I.\ A\ i) \leq (\sum i{\in}I.\ emeasure\ M\ (A\ i))$
by (*rule sets.subadditive*[*OF emeasure_positive emeasure_additive*]) *auto*

lemma *emeasure_subadditive*:
$A \in sets\ M \Longrightarrow B \in sets\ M \Longrightarrow emeasure\ M\ (A \cup B) \leq emeasure\ M\ A + emeasure\ M\ B$
using *emeasure_subadditive_finite*[*of* $\{True, False\}\ \lambda True \Rightarrow A \mid False \Rightarrow B\ M$]
by *simp*

lemma *emeasure_subadditive_countably*:
assumes *range* $f \subseteq sets\ M$
shows *emeasure* $M\ (\bigcup i.\ f\ i) \leq (\sum i.\ emeasure\ M\ (f\ i))$
proof −
have *emeasure* $M\ (\bigcup i.\ f\ i) = emeasure\ M\ (\bigcup i.\ disjointed\ f\ i)$
unfolding *UN_disjointed_eq* **..**
also have $\ldots = (\sum i.\ emeasure\ M\ (disjointed\ f\ i))$
using *sets.range_disjointed_sets*[*OF assms*] *suminf_emeasure*[*of disjointed f*]
by (*simp add*: *disjoint_family_disjointed comp_def*)
also have $\ldots \leq (\sum i.\ emeasure\ M\ (f\ i))$
using *sets.range_disjointed_sets*[*OF assms*] *assms*
by (*auto intro*!: *suminf_le emeasure_mono disjointed_subset*)
finally show *?thesis* **.**
qed

lemma *emeasure_insert*:
assumes *sets*: $\{x\} \in sets\ M\ A \in sets\ M$ **and** $x \notin A$
shows *emeasure* $M\ (insert\ x\ A) = emeasure\ M\ \{x\} + emeasure\ M\ A$
proof −
have $\{x\} \cap A = \{\}$ **using** ‹$x \notin A$› **by** *auto*
from *plus_emeasure*[*OF sets this*] **show** *?thesis* **by** *simp*
qed

lemma *emeasure_insert_ne*:
$A \neq \{\} \Longrightarrow \{x\} \in sets\ M \Longrightarrow A \in sets\ M \Longrightarrow x \notin A \Longrightarrow emeasure\ M\ (insert\ x\ A) = emeasure\ M\ \{x\} + emeasure\ M\ A$

by (*rule emeasure_insert*)

lemma *emeasure_eq_sum_singleton*:
assumes *finite S* $\bigwedge x.\ x \in S \Longrightarrow \{x\} \in sets\ M$
shows *emeasure M S* $= (\sum x \in S.\ emeasure\ M\ \{x\})$
using *sum_emeasure*[*of* $\lambda x.\ \{x\}$ *S M*] *assms*
by (*auto simp*: *disjoint_family_on_def subset_eq*)

lemma *sum_emeasure_cover*:
assumes *finite S* **and** $A \in sets\ M$ **and** *br_in_M*: $B\ `\ S \subseteq sets\ M$
assumes *A*: $A \subseteq (\bigcup i \in S.\ B\ i)$
assumes *disj*: *disjoint_family_on B S*
shows *emeasure M A* $= (\sum i \in S.\ emeasure\ M\ (A \cap (B\ i)))$
proof −
have $(\sum i \in S.\ emeasure\ M\ (A \cap (B\ i))) = emeasure\ M\ (\bigcup i \in S.\ A \cap (B\ i))$
proof (*rule sum_emeasure*)
show *disjoint_family_on* $(\lambda i.\ A \cap B\ i)\ S$
using ‹*disjoint_family_on B S*›
unfolding *disjoint_family_on_def* **by** *auto*
qed (*insert assms*, *auto*)
also have $(\bigcup i \in S.\ A \cap (B\ i)) = A$
using *A* **by** *auto*
finally show *?thesis* **by** *simp*
qed

lemma *emeasure_eq_0*:
$N \in sets\ M \Longrightarrow emeasure\ M\ N = 0 \Longrightarrow K \subseteq N \Longrightarrow emeasure\ M\ K = 0$
by (*metis emeasure_mono order_eq_iff zero_le*)

lemma *emeasure_UN_eq_0*:
assumes $\bigwedge i :: nat.\ emeasure\ M\ (N\ i) = 0$ **and** $range\ N \subseteq sets\ M$
shows *emeasure M* $(\bigcup i.\ N\ i) = 0$
proof −
have *emeasure M* $(\bigcup i.\ N\ i) \leq 0$
using *emeasure_subadditive_countably*[*OF assms*(*2*)] *assms*(*1*) **by** *simp*
then show *?thesis*
by (*auto intro*: *antisym zero_le*)
qed

lemma *measure_eqI_finite*:
assumes [*simp*]: *sets M* $=$ *Pow A sets N* $=$ *Pow A* **and** *finite A*
assumes *eq*: $\bigwedge a.\ a \in A \Longrightarrow emeasure\ M\ \{a\} = emeasure\ N\ \{a\}$
shows $M = N$
proof (*rule measure_eqI*)
fix *X* **assume** $X \in sets\ M$
then have *X*: $X \subseteq A$ **by** *auto*
then have *emeasure M X* $= (\sum a \in X.\ emeasure\ M\ \{a\})$
using ‹*finite A*› **by** (*subst emeasure_eq_sum_singleton*) (*auto dest*: *finite_subset*)
also have $\ldots = (\sum a \in X.\ emeasure\ N\ \{a\})$

using *X eq* **by** (*auto intro*!: *sum.cong*)
also have ... = *emeasure N X*
using *X* ‹*finite A*› **by** (*subst emeasure_eq_sum_singleton*) (*auto dest*: *finite_subset*)
finally show *emeasure M X* = *emeasure N X* **.**
qed *simp*

lemma *measure_eqI_generator_eq*:
fixes *M N* :: *'a measure* **and** *E* :: *'a set set* **and** *A* :: *nat* ⇒ *'a set*
assumes *Int_stable E E* ⊆ *Pow* Ω
and *eq*: ⋀*X*. *X* ∈ *E* ⟹ *emeasure M X* = *emeasure N X*
and *M*: *sets M* = *sigma_sets* Ω *E*
and *N*: *sets N* = *sigma_sets* Ω *E*
and *A*: *range A* ⊆ *E* (⋃*i*. *A i*) = Ω ⋀*i*. *emeasure M* (*A i*) ≠ ∞
shows *M* = *N*
proof −
let *?μ* = *emeasure M* **and** *?ν* = *emeasure N*
interpret *S*: *sigma_algebra* Ω *sigma_sets* Ω *E* **by** (*rule sigma_algebra_sigma_sets*) *fact*
have *space M* = Ω
using *sets.top*[*of M*] *sets.space_closed*[*of M*] *S.top S.space_closed* ‹*sets M* = *sigma_sets* Ω *E*›
by *blast*

{ fix *F D* **assume** *F* ∈ *E* **and** *?μ F* ≠ ∞
then have [*intro*]: *F* ∈ *sigma_sets* Ω *E* **by** *auto*
have *?ν F* ≠ ∞ **using** ‹*?μ F* ≠ ∞› ‹*F* ∈ *E*› *eq* **by** *simp*
assume *D* ∈ *sets M*
with ‹*Int_stable E*› ‹*E* ⊆ *Pow* Ω› **have** *emeasure M* (*F* ∩ *D*) = *emeasure N* (*F* ∩ *D*)
unfolding *M*
proof (*induct rule*: *sigma_sets_induct_disjoint*)
case (*basic A*)
then have *F* ∩ *A* ∈ *E* **using** ‹*Int_stable E*› ‹*F* ∈ *E*› **by** (*auto simp*: *Int_stable_def*)
then show *?case* **using** *eq* **by** *auto*
next
case *empty* **then show** *?case* **by** *simp*
next
case (*compl A*)
then have ∗∗: *F* ∩ (Ω − *A*) = *F* − (*F* ∩ *A*)
and [*intro*]: *F* ∩ *A* ∈ *sigma_sets* Ω *E*
using ‹*F* ∈ *E*› *S.sets_into_space* **by** (*auto simp*: *M*)
have *?ν* (*F* ∩ *A*) ≤ *?ν F* **by** (*auto intro*!: *emeasure_mono simp*: *M N*)
then have *?ν* (*F* ∩ *A*) ≠ ∞ **using** ‹*?ν F* ≠ ∞› **by** (*auto simp*: *top_unique*)
have *?μ* (*F* ∩ *A*) ≤ *?μ F* **by** (*auto intro*!: *emeasure_mono simp*: *M N*)
then have *?μ* (*F* ∩ *A*) ≠ ∞ **using** ‹*?μ F* ≠ ∞› **by** (*auto simp*: *top_unique*)
then have *?μ* (*F* ∩ (Ω − *A*)) = *?μ F* − *?μ* (*F* ∩ *A*) **unfolding** ∗∗
using ‹*F* ∩ *A* ∈ *sigma_sets* Ω *E*› **by** (*auto intro*!: *emeasure_Diff simp*: *M*

```
N)
      also have ... = ?ν F − ?ν (F ∩ A) using eq ‹F ∈ E› compl by simp
      also have ... = ?ν (F ∩ (Ω − A)) unfolding **
        using ‹F ∩ A ∈ sigma_sets Ω E› ‹?ν (F ∩ A) ≠ ∞›
        by (auto intro!: emeasure_Diff[symmetric] simp: M N)
      finally show ?case
        using ‹space M = Ω› by auto
    next
      case (union A)
      then have ?μ (⋃ x. F ∩ A x) = ?ν (⋃ x. F ∩ A x)
      by (subst (1 2) suminf_emeasure[symmetric]) (auto simp: disjoint_family_on_def
subset_eq M N)
      with A show ?case
        by auto
    qed }
  note * = this
  show M = N
  proof (rule measure_eqI)
    show sets M = sets N
      using M N by simp
    have [simp, intro]: ⋀i. A i ∈ sets M
      using A(1) by (auto simp: subset_eq M)
    fix F assume F ∈ sets M
    let ?D = disjointed (λi. F ∩ A i)
    from ‹space M = Ω› have F_eq: F = (⋃ i. ?D i)
      using ‹F ∈ sets M›[THEN sets.sets_into_space] A(2)[symmetric] by (auto
simp: UN_disjointed_eq)
    have [simp, intro]: ⋀i. ?D i ∈ sets M
      using sets.range_disjointed_sets[of λi. F ∩ A i M] ‹F ∈ sets M›
      by (auto simp: subset_eq)
    have disjoint_family ?D
      by (auto simp: disjoint_family_disjointed)
    moreover
    have (∑ i. emeasure M (?D i)) = (∑ i. emeasure N (?D i))
    proof (intro arg_cong[where f=suminf] ext)
      fix i
      have A i ∩ ?D i = ?D i
        by (auto simp: disjointed_def)
      then show emeasure M (?D i) = emeasure N (?D i)
        using *[of A i ?D i, OF _ A(3)] A(1) by auto
    qed
    ultimately show emeasure M F = emeasure N F
    by (simp add: image_subset_iff ‹sets M = sets N›[symmetric] F_eq[symmetric]
suminf_emeasure)
  qed
qed

lemma space_empty: space M = {} ⟹ M = count_space {}
  by (rule measure_eqI) (simp_all add: space_empty_iff)
```

lemma *measure_eqI_generator_eq_countable*:
fixes $M\ N :: 'a\ measure$ **and** $E :: 'a\ set\ set$ **and** $A :: 'a\ set\ set$
assumes *E*: *Int_stable E* $E \subseteq Pow\ \Omega$ $\bigwedge X.\ X \in E \Longrightarrow emeasure\ M\ X = emeasure\ N\ X$
and *sets*: *sets M* = *sigma_sets* Ω *E sets N* = *sigma_sets* Ω *E*
and *A*: $A \subseteq E$ $(\bigcup A) = \Omega$ *countable A* $\bigwedge a.\ a \in A \Longrightarrow emeasure\ M\ a \neq \infty$
shows *M* = *N*
proof *cases*
assume $\Omega = \{\}$
have $*$: *sigma_sets* Ω *E* = *sets* (*sigma* Ω *E*)
using *E*(*2*) **by** *simp*
have *space M* = Ω *space N* = Ω
using *sets E*(*2*) **unfolding** $*$ **by** (*auto dest*: *sets_eq_imp_space_eq simp del*: *sets_measure_of*)
then show *M* = *N*
unfolding ‹$\Omega = \{\}$› **by** (*auto dest*: *space_empty*)
next
assume $\Omega \neq \{\}$ **with** ‹$\bigcup A = \Omega$› **have** $A \neq \{\}$ **by** *auto*
from *this* ‹*countable A*› **have** *rng*: *range* (*from_nat_into A*) = *A*
by (*rule range_from_nat_into*)
show *M* = *N*
proof (*rule measure_eqI_generator_eq*[*OF E sets*])
show *range* (*from_nat_into A*) $\subseteq$ *E*
unfolding *rng* **using** ‹$A \subseteq E$› **.**
show $(\bigcup i.$ *from_nat_into A i*$) = \Omega$
unfolding *rng* **using** ‹$\bigcup A = \Omega$› **.**
show *emeasure M* (*from_nat_into A i*) $\neq \infty$ **for** *i*
using *rng* **by** (*intro A*) *auto*
qed
qed

lemma *measure_of_of_measure*: *measure_of* (*space M*) (*sets M*) (*emeasure M*) = *M*
proof (*intro measure_eqI emeasure_measure_of_sigma*)
show *sigma_algebra* (*space M*) (*sets M*) **..**
show *positive* (*sets M*) (*emeasure M*)
by (*simp add*: *positive_def*)
show *countably_additive* (*sets M*) (*emeasure M*)
by (*simp add*: *emeasure_countably_additive*)
qed *simp_all*

6.3.5 μ-null sets

definition *null_sets* :: $'a\ measure \Rightarrow 'a\ set\ set$ **where**
null_sets M = $\{N \in sets\ M.\ emeasure\ M\ N = 0\}$

lemma *null_setsD1*[*dest*]: $A \in null\_sets\ M \Longrightarrow emeasure\ M\ A = 0$
by (*simp add*: *null_sets_def*)

lemma *null_setsD2*[*dest*]: $A \in$ *null_sets* $M \Longrightarrow A \in$ *sets* M
unfolding *null_sets_def* **by** *simp*

lemma *null_setsI*[*intro*]: *emeasure* M $A = 0 \Longrightarrow A \in$ *sets* $M \Longrightarrow A \in$ *null_sets* M
unfolding *null_sets_def* **by** *simp*

interpretation *null_sets*: *ring_of_sets space* M *null_sets* M **for** M
proof (*rule ring_of_setsI*)
show *null_sets* $M \subseteq$ *Pow* (*space* M)
using *sets.sets_into_space* **by** *auto*
show $\{\} \in$ *null_sets* M
by *auto*
fix A B **assume** *null_sets*: $A \in$ *null_sets* M $B \in$ *null_sets* M
then have *sets*: $A \in$ *sets* M $B \in$ *sets* M
by *auto*
then have $*$: *emeasure* M $(A \cup B) \leq$ *emeasure* M $A +$ *emeasure* M B
emeasure M $(A - B) \leq$ *emeasure* M A
by (*auto intro*!: *emeasure_subadditive emeasure_mono*)
then have *emeasure* M $B = 0$ *emeasure* M $A = 0$
using *null_sets* **by** *auto*
with *sets* $*$ **show** $A - B \in$ *null_sets* M $A \cup B \in$ *null_sets* M
by (*auto intro*!: *antisym zero_le*)
qed

lemma *UN_from_nat_into*:
assumes I: *countable* I $I \neq \{\}$
shows $(\bigcup i \in I.\ N\ i) = (\bigcup i.\ N\ (\mathit{from\_nat\_into}\ I\ i))$
proof −
have $(\bigcup i \in I.\ N\ i) = \bigcup (N\ `\ \mathit{range}\ (\mathit{from\_nat\_into}\ I))$
using I **by** *simp*
also have $\ldots = (\bigcup i.\ (N \circ \mathit{from\_nat\_into}\ I)\ i)$
by *simp*
finally show *?thesis* **by** *simp*
qed

lemma *null_sets_UN′*:
assumes *countable* I
assumes $\bigwedge i.\ i \in I \Longrightarrow N\ i \in$ *null_sets* M
shows $(\bigcup i \in I.\ N\ i) \in$ *null_sets* M
proof *cases*
assume $I = \{\}$ **then show** *?thesis* **by** *simp*
next
assume $I \neq \{\}$
show *?thesis*
proof (*intro conjI CollectI null_setsI*)
show $(\bigcup i \in I.\ N\ i) \in$ *sets* M
using *assms* **by** (*intro sets.countable_UN′*) *auto*

```
    have emeasure M (⋃ i∈I. N i) ≤ (∑ n. emeasure M (N (from_nat_into I n)))
      unfolding UN_from_nat_into[OF ‹countable I› ‹I ≠ {}›]
     using assms ‹I ≠ {}› by (intro emeasure_subadditive_countably) (auto intro: from_nat_into)
    also have (λn. emeasure M (N (from_nat_into I n))) = (λ_. 0)
      using assms ‹I ≠ {}› by (auto intro: from_nat_into)
    finally show emeasure M (⋃ i∈I. N i) = 0
      by (intro antisym zero_le) simp
  qed
qed

lemma null_sets_UN[intro]:
  (⋀i::'i::countable. N i ∈ null_sets M) ⟹ (⋃ i. N i) ∈ null_sets M
  by (rule null_sets_UN') auto

lemma null_set_Int1:
  assumes B ∈ null_sets M A ∈ sets M shows A ∩ B ∈ null_sets M
proof (intro CollectI conjI null_setsI)
  show emeasure M (A ∩ B) = 0 using assms
    by (intro emeasure_eq_0[of B _ A ∩ B]) auto
qed (insert assms, auto)

lemma null_set_Int2:
  assumes B ∈ null_sets M A ∈ sets M shows B ∩ A ∈ null_sets M
  using assms by (subst Int_commute) (rule null_set_Int1)

lemma emeasure_Diff_null_set:
  assumes B ∈ null_sets M A ∈ sets M
  shows emeasure M (A − B) = emeasure M A
proof −
  have ∗: A − B = (A − (A ∩ B)) by auto
  have A ∩ B ∈ null_sets M using assms by (rule null_set_Int1)
  then show ?thesis
    unfolding ∗ using assms
    by (subst emeasure_Diff) auto
qed

lemma null_set_Diff:
  assumes B ∈ null_sets M A ∈ sets M shows B − A ∈ null_sets M
proof (intro CollectI conjI null_setsI)
  show emeasure M (B − A) = 0 using assms by (intro emeasure_eq_0[of B _
B − A]) auto
qed (insert assms, auto)

lemma emeasure_Un_null_set:
  assumes A ∈ sets M B ∈ null_sets M
  shows emeasure M (A ∪ B) = emeasure M A
proof −
  have ∗: A ∪ B = A ∪ (B − A) by auto
```

have $B - A \in$ *null_sets M* **using** *assms*(*2*,*1*) **by** (*rule null_set_Diff*)
then show *?thesis*
unfolding $*$ **using** *assms*
by (*subst plus_emeasure*[*symmetric*]) *auto*
qed

lemma *emeasure_Un'*:
assumes $A \in$ *sets M* $B \in$ *sets M* $A \cap B \in$ *null_sets M*
shows *emeasure M* $(A \cup B) =$ *emeasure M A* $+$ *emeasure M B*
proof $-$
have $A \cup B = A \cup (B - A \cap B)$ **by** *blast*
also have *emeasure M* $\ldots =$ *emeasure M A* $+$ *emeasure M* $(B - A \cap B)$
using *assms* **by** (*subst plus_emeasure*) *auto*
also have *emeasure M* $(B - A \cap B) =$ *emeasure M B*
using *assms* **by** (*intro emeasure_Diff_null_set*) *auto*
finally show *?thesis* **.**
qed

6.3.6 The almost everywhere filter (i.e. quantifier)

definition *ae_filter* :: *'a measure* $\Rightarrow$ *'a filter* **where**
ae_filter M $=$ (*INF* $N \in$*null_sets M*. *principal* (*space M* $-$ *N*))

abbreviation *almost_everywhere* :: *'a measure* $\Rightarrow$ (*'a* $\Rightarrow$ *bool*) $\Rightarrow$ *bool* **where**
almost_everywhere M P $\equiv$ *eventually P* (*ae_filter M*)

syntax
_almost_everywhere :: *pttrn* $\Rightarrow$ *'a* $\Rightarrow$ *bool* $\Rightarrow$ *bool* (*AE _ in _. _* [*0*,*0*,*10*] *10*)

translations
AE x in M. P $\rightleftharpoons$ *CONST almost_everywhere M* $(\lambda x.\ P)$

abbreviation
set_almost_everywhere A M P $\equiv$ *AE x in M*. $x \in A \longrightarrow P\ x$

syntax
_set_almost_everywhere :: *pttrn* $\Rightarrow$ *'a set* $\Rightarrow$ *'a* $\Rightarrow$ *bool* $\Rightarrow$ *bool*
(*AE _*$\in$*_ in _./ _* [*0*,*0*,*0*,*10*] *10*)

translations
AE x$\in$*A in M. P* $\rightleftharpoons$ *CONST set_almost_everywhere A M* $(\lambda x.\ P)$

lemma *eventually_ae_filter*: *eventually P* (*ae_filter M*) $\longleftrightarrow$ ($\exists$ *N*$\in$*null_sets M*. $\{x \in$ *space M*. $\neg$ *P x*$\} \subseteq$ *N*)
unfolding *ae_filter_def* **by** (*subst eventually_INF_base*) (*auto simp*: *eventually_principal subset_eq*)

lemma *AE_I'*:
N $\in$ *null_sets M* $\Longrightarrow$ $\{x \in$*space M*. $\neg$ *P x*$\} \subseteq$ *N* $\Longrightarrow$ (*AE x in M*. *P x*)

unfolding *eventually_ae_filter* **by** *auto*

lemma *AE_iff_null*:
assumes $\{x \in space\ M.\ \neg\ P\ x\} \in sets\ M$ (**is** $?P \in sets\ M$)
shows $(AE\ x\ in\ M.\ P\ x) \longleftrightarrow \{x \in space\ M.\ \neg\ P\ x\} \in null\_sets\ M$
proof
assume $AE\ x\ in\ M.\ P\ x$ **then obtain** N **where** N: $N \in sets\ M\ ?P \subseteq N\ emeasure\ M\ N = 0$
unfolding *eventually_ae_filter* **by** *auto*
have $emeasure\ M\ ?P \leq emeasure\ M\ N$
using *assms* $N(1,2)$ **by** (*auto intro*: *emeasure_mono*)
then have $emeasure\ M\ ?P = 0$
unfolding ‹$emeasure\ M\ N = 0$› **by** *auto*
then show $?P \in null\_sets\ M$ **using** *assms* **by** *auto*
next
assume $?P \in null\_sets\ M$ **with** *assms* **show** $AE\ x\ in\ M.\ P\ x$ **by** (*auto intro*: *AE_I'*)
qed

lemma *AE_iff_null_sets*:
$N \in sets\ M \Longrightarrow N \in null\_sets\ M \longleftrightarrow (AE\ x\ in\ M.\ x \notin N)$
using *Int_absorb1*[*OF sets.sets_into_space*, *of N M*]
by (*subst AE_iff_null*) (*auto simp*: *Int_def*[*symmetric*])

lemma *ae_filter_eq_bot_iff*: $ae\_filter\ M = bot \longleftrightarrow emeasure\ M\ (space\ M) = 0$
proof −
have $ae\_filter\ M = bot \longleftrightarrow (AE\ x\ in\ M.\ False)$
using *trivial_limit_def* **by** *blast*
also have $\ldots \longleftrightarrow space\ M \in null\_sets\ M$
by (*simp add*: *AE_iff_null_sets eventually_ae_filter*)
also have $\ldots \longleftrightarrow emeasure\ M\ (space\ M) = 0$
by *auto*
finally show *?thesis* **.**
qed

lemma *AE_not_in*:
$N \in null\_sets\ M \Longrightarrow AE\ x\ in\ M.\ x \notin N$
by (*metis AE_iff_null_sets null_setsD2*)

lemma *AE_iff_measurable*:
$N \in sets\ M \Longrightarrow \{x \in space\ M.\ \neg\ P\ x\} = N \Longrightarrow (AE\ x\ in\ M.\ P\ x) \longleftrightarrow emeasure\ M\ N = 0$
using *AE_iff_null*[*of* _ *P*] **by** *auto*

lemma *AE_E*[*consumes 1*]:
assumes $AE\ x\ in\ M.\ P\ x$
obtains N **where** $\{x \in space\ M.\ \neg\ P\ x\} \subseteq N\ emeasure\ M\ N = 0\ N \in sets\ M$
using *assms* **unfolding** *eventually_ae_filter* **by** *auto*

lemma *AE_E2*:
assumes *AE x in M. P x*
shows *emeasure M {x∈space M. ¬ P x} = 0*
by (*metis* (*mono_tags*, *lifting*) *AE_iff_null assms emeasure_notin_sets null_setsD1*)

lemma *AE_E3*:
assumes *AE x in M. P x*
obtains *N* **where** ⋀*x*. *x ∈ space M − N ⟹ P x N ∈ null_sets M*
using *assms* **unfolding** *eventually_ae_filter* **by** *auto*

lemma *AE_I*:
assumes *{x ∈ space M. ¬ P x} ⊆ N emeasure M N = 0 N ∈ sets M*
shows *AE x in M. P x*
using *assms* **unfolding** *eventually_ae_filter* **by** *auto*

lemma *AE_mp*[*elim*!]:
assumes *AE_P*: *AE x in M. P x* **and** *AE_imp*: *AE x in M. P x ⟶ Q x*
shows *AE x in M. Q x*
using *assms* **by** (*fact eventually_rev_mp*)

The next lemma is convenient to combine with a lemma whose conclusion is of the form *AE x in M. P x = Q x*: for such a lemma, there is no [*symmetric*] variant, but using *AE_symmetric*[*OF* . . .] will replace it.

lemma
shows *AE_iffI*: *AE x in M. P x ⟹ AE x in M. P x ⟷ Q x ⟹ AE x in M. Q x*
and *AE_disjI1*: *AE x in M. P x ⟹ AE x in M. P x ∨ Q x*
and *AE_disjI2*: *AE x in M. Q x ⟹ AE x in M. P x ∨ Q x*
and *AE_conjI*: *AE x in M. P x ⟹ AE x in M. Q x ⟹ AE x in M. P x ∧ Q x*
and *AE_conj_iff*[*simp*]: *(AE x in M. P x ∧ Q x) ⟷ (AE x in M. P x) ∧ (AE x in M. Q x)*
by *auto*

lemma *AE_symmetric*:
assumes *AE x in M. P x = Q x*
shows *AE x in M. Q x = P x*
using *assms* **by** *auto*

lemma *AE_impI*:
(P ⟹ AE x in M. Q x) ⟹ AE x in M. P ⟶ Q x
by *fastforce*

lemma *AE_measure*:
assumes *AE*: *AE x in M. P x* **and** *sets*: *{x∈space M. P x} ∈ sets M* (**is** *?P ∈ sets M*)
shows *emeasure M {x∈space M. P x} = emeasure M (space M)*
proof −
from *AE_E*[*OF AE*] **obtain** *N*

where N: $\{x \in space\ M.\ \neg\ P\ x\} \subseteq N\ emeasure\ M\ N = 0\ N \in sets\ M$
by *auto*
with *sets* **have** $emeasure\ M\ (space\ M) \leq emeasure\ M\ (?P \cup N)$
by (*intro emeasure_mono*) *auto*
also have $\ldots \leq emeasure\ M\ ?P + emeasure\ M\ N$
using *sets* N **by** (*intro emeasure_subadditive*) *auto*
also have $\ldots = emeasure\ M\ ?P$ **using** N **by** *simp*
finally show $emeasure\ M\ ?P = emeasure\ M\ (space\ M)$
using *emeasure_space*[*of M ?P*] **by** *auto*
qed

lemma *AE_space*: $AE\ x\ in\ M.\ x \in space\ M$
by (*rule AE_I*[**where** N={}]) *auto*

lemma *AE_I2*[*simp*, *intro*]:
$(\bigwedge x.\ x \in space\ M \Longrightarrow P\ x) \Longrightarrow AE\ x\ in\ M.\ P\ x$
using *AE_space* **by** *force*

lemma *AE_Ball_mp*:
$\forall\, x{\in}space\ M.\ P\ x \Longrightarrow AE\ x\ in\ M.\ P\ x \longrightarrow Q\ x \Longrightarrow AE\ x\ in\ M.\ Q\ x$
by *auto*

lemma *AE_cong*[*cong*]:
$(\bigwedge x.\ x \in space\ M \Longrightarrow P\ x \longleftrightarrow Q\ x) \Longrightarrow (AE\ x\ in\ M.\ P\ x) \longleftrightarrow (AE\ x\ in\ M.\ Q\ x)$
by *auto*

lemma *AE_cong_simp*: $M = N \Longrightarrow (\bigwedge x.\ x \in space\ N =simp=> P\ x = Q\ x) \Longrightarrow (AE\ x\ in\ M.\ P\ x) \longleftrightarrow (AE\ x\ in\ N.\ Q\ x)$
by (*auto simp*: *simp_implies_def*)

lemma *AE_all_countable*:
$(AE\ x\ in\ M.\ \forall\, i.\ P\ i\ x) \longleftrightarrow (\forall\, i{::}'i{::}countable.\ AE\ x\ in\ M.\ P\ i\ x)$
proof
assume $\forall\, i.\ AE\ x\ in\ M.\ P\ i\ x$
from *this*[*unfolded eventually_ae_filter Bex_def*, *THEN choice*]
obtain N **where** N: $\bigwedge i.\ N\ i \in null\_sets\ M\ \bigwedge i.\ \{x{\in}space\ M.\ \neg\ P\ i\ x\} \subseteq N\ i$
by *auto*
have $\{x{\in}space\ M.\ \neg\ (\forall\, i.\ P\ i\ x)\} \subseteq (\bigcup i.\ \{x{\in}space\ M.\ \neg\ P\ i\ x\})$ **by** *auto*
also have $\ldots \subseteq (\bigcup i.\ N\ i)$ **using** N **by** *auto*
finally have $\{x{\in}space\ M.\ \neg\ (\forall\, i.\ P\ i\ x)\} \subseteq (\bigcup i.\ N\ i)$ **.**
moreover from N **have** $(\bigcup i.\ N\ i) \in null\_sets\ M$
by (*intro null_sets_UN*) *auto*
ultimately show $AE\ x\ in\ M.\ \forall\, i.\ P\ i\ x$
unfolding *eventually_ae_filter* **by** *auto*
qed *auto*

lemma *AE_ball_countable*:
assumes [*intro*]: *countable* X

shows (*AE x in M.* ∀ *y*∈*X. P x y*) ⟷ (∀ *y*∈*X. AE x in M. P x y*)
proof
assume ∀ *y*∈*X. AE x in M. P x y*
from *this*[*unfolded eventually_ae_filter Bex_def*, *THEN bchoice*]
obtain *N* **where** *N*: ⋀*y*. *y* ∈ *X* ⟹ *N y* ∈ *null_sets M* ⋀*y*. *y* ∈ *X* ⟹ {*x*∈*space M*. ¬ *P x y*} ⊆ *N y*
by *auto*
have {*x*∈*space M*. ¬ (∀ *y*∈*X. P x y*)} ⊆ (⋃ *y*∈*X*. {*x*∈*space M*. ¬ *P x y*})
by *auto*
also have ... ⊆ (⋃ *y*∈*X*. *N y*)
using *N* **by** *auto*
finally have {*x*∈*space M*. ¬ (∀ *y*∈*X. P x y*)} ⊆ (⋃ *y*∈*X*. *N y*) **.**
moreover from *N* **have** (⋃ *y*∈*X*. *N y*) ∈ *null_sets M*
by (*intro null_sets_UN′*) *auto*
ultimately show *AE x in M.* ∀ *y*∈*X. P x y*
unfolding *eventually_ae_filter* **by** *auto*
qed *auto*

lemma *AE_ball_countable′*:
(⋀*N*. *N* ∈ *I* ⟹ *AE x in M. P N x*) ⟹ *countable I* ⟹ *AE x in M.* ∀ *N* ∈ *I*. *P N x*
unfolding *AE_ball_countable* **by** *simp*

lemma *AE_pairwise*: *countable F* ⟹ *pairwise* (λ*A B*. *AE x in M. R x A B*) *F* ⟷ (*AE x in M. pairwise* (*R x*) *F*)
unfolding *pairwise_alt* **by** (*simp add*: *AE_ball_countable*)

lemma *AE_discrete_difference*:
assumes *X*: *countable X*
assumes *null*: ⋀*x*. *x* ∈ *X* ⟹ *emeasure M* {*x*} = *0*
assumes *sets*: ⋀*x*. *x* ∈ *X* ⟹ {*x*} ∈ *sets M*
shows *AE x in M. x* ∉ *X*
proof −
have (⋃ *x*∈*X*. {*x*}) ∈ *null_sets M*
using *assms* **by** (*intro null_sets_UN′*) *auto*
from *AE_not_in*[*OF this*] **show** *AE x in M. x* ∉ *X*
by *auto*
qed

lemma *AE_finite_all*:
assumes *f*: *finite S* **shows** (*AE x in M.* ∀ *i*∈*S. P i x*) ⟷ (∀ *i*∈*S. AE x in M. P i x*)
using *f* **by** *induct auto*

lemma *AE_finite_allI*:
assumes *finite S*
shows (⋀*s*. *s* ∈ *S* ⟹ *AE x in M. Q s x*) ⟹ *AE x in M.* ∀ *s*∈*S. Q s x*
using *AE_finite_all*[*OF* ‹*finite S*›] **by** *auto*

lemma *emeasure_mono_AE*:
assumes *imp*: *AE x in M*. $x \in A \longrightarrow x \in B$
and *B*: $B \in$ *sets M*
shows *emeasure M A* $\leq$ *emeasure M B*
proof *cases*
assume *A*: $A \in$ *sets M*
from *imp* **obtain** *N* **where** *N*: $\{x \in$*space M*. $\neg\ (x \in A \longrightarrow x \in B)\} \subseteq N\ N \in$ *null_sets M*
by (*auto simp*: *eventually_ae_filter*)
have *emeasure M A* = *emeasure M* $(A - N)$
using *N A* **by** (*subst emeasure_Diff_null_set*) *auto*
also have *emeasure M* $(A - N) \leq$ *emeasure M* $(B - N)$
using *N A B sets.sets_into_space* **by** (*auto intro*!: *emeasure_mono*)
also have *emeasure M* $(B - N)$ = *emeasure M B*
using *N B* **by** (*subst emeasure_Diff_null_set*) *auto*
finally show *?thesis* **.**
qed (*simp add*: *emeasure_notin_sets*)

lemma *emeasure_eq_AE*:
assumes *iff*: *AE x in M*. $x \in A \longleftrightarrow x \in B$
assumes *A*: $A \in$ *sets M* **and** *B*: $B \in$ *sets M*
shows *emeasure M A* = *emeasure M B*
using *assms* **by** (*safe intro*!: *antisym emeasure_mono_AE*) *auto*

lemma *emeasure_Collect_eq_AE*:
AE x in M. *P x* $\longleftrightarrow$ *Q x* $\Longrightarrow$ *Measurable.pred M Q* $\Longrightarrow$ *Measurable.pred M P* $\Longrightarrow$
emeasure M $\{x \in$*space M*. *P x*$\}$ = *emeasure M* $\{x \in$*space M*. *Q x*$\}$
by (*intro emeasure_eq_AE*) *auto*

lemma *emeasure_eq_0_AE*: *AE x in M*. $\neg$ *P x* $\Longrightarrow$ *emeasure M* $\{x \in$*space M*. *P x*$\}$ = *0*
using *AE_iff_measurable*[*OF _ refl*, *of M* $\lambda x.\ \neg$ *P x*]
by (*cases* $\{x \in$*space M*. *P x*$\} \in$ *sets M*) (*simp_all add*: *emeasure_notin_sets*)

lemma *emeasure_0_AE*:
assumes *emeasure M* (*space M*) = *0*
shows *AE x in M*. *P x*
using *eventually_ae_filter assms* **by** *blast*

lemma *emeasure_add_AE*:
assumes [*measurable*]: $A \in$ *sets M* $B \in$ *sets M* $C \in$ *sets M*
assumes *1*: *AE x in M*. $x \in C \longleftrightarrow x \in A \vee x \in B$
assumes *2*: *AE x in M*. $\neg\ (x \in A \wedge x \in B)$
shows *emeasure M C* = *emeasure M A* + *emeasure M B*
proof −
have *emeasure M C* = *emeasure M* $(A \cup B)$
by (*rule emeasure_eq_AE*) (*insert 1*, *auto*)
also have ... = *emeasure M A* + *emeasure M* $(B - A)$

by (*subst plus_emeasure*) *auto*
also have *emeasure M* (*B* − *A*) = *emeasure M B*
by (*rule emeasure_eq_AE*) (*insert 2*, *auto*)
finally show *?thesis* **.**
qed

6.3.7 σ-finite Measures

locale *sigma_finite_measure* =
fixes *M* :: *'a measure*
assumes *sigma_finite_countable*:
∃ *A*::*'a set set*. *countable A* ∧ *A* ⊆ *sets M* ∧ (⋃ *A*) = *space M* ∧ (∀ *a*∈*A*. *emeasure M a* ≠ ∞)

lemma (**in** *sigma_finite_measure*) *sigma_finite*:
obtains *A* :: *nat* ⇒ *'a set*
where *range A* ⊆ *sets M* (⋃ *i*. *A i*) = *space M* ⋀*i*. *emeasure M* (*A i*) ≠ ∞
proof −
obtain *A* :: *'a set set* **where**
[*simp*]: *countable A* **and**
A: *A* ⊆ *sets M* (⋃ *A*) = *space M* ⋀*a*. *a* ∈ *A* ⟹ *emeasure M a* ≠ ∞
using *sigma_finite_countable* **by** *metis*
show *thesis*
proof *cases*
assume *A* = {} **with** ‹(⋃ *A*) = *space M*› **show** *thesis*
by (*intro that*[*of* λ_. {}]) *auto*
next
assume *A* ≠ {}
show *thesis*
proof
show *range* (*from_nat_into A*) ⊆ *sets M*
using ‹*A* ≠ {}› *A* **by** *auto*
have (⋃ *i*. *from_nat_into A i*) = ⋃ *A*
using *range_from_nat_into*[*OF* ‹*A* ≠ {}› ‹*countable A*›] **by** *auto*
with *A* **show** (⋃ *i*. *from_nat_into A i*) = *space M*
by *auto*
qed (*intro A from_nat_into* ‹*A* ≠ {}›)
qed
qed

lemma (**in** *sigma_finite_measure*) *sigma_finite_disjoint*:
obtains *A* :: *nat* ⇒ *'a set*
where *range A* ⊆ *sets M* (⋃ *i*. *A i*) = *space M* ⋀*i*. *emeasure M* (*A i*) ≠ ∞
disjoint_family A
proof −
obtain *A* :: *nat* ⇒ *'a set* **where**
range: *range A* ⊆ *sets M* **and**
space: (⋃ *i*. *A i*) = *space M* **and**
measure: ⋀*i*. *emeasure M* (*A i*) ≠ ∞

```
    using sigma_finite by blast
  show thesis
  proof (rule that[of disjointed A])
    show range (disjointed A) ⊆ sets M
      by (rule sets.range_disjointed_sets[OF range])
    show (⋃i. disjointed A i) = space M
      and disjoint_family (disjointed A)
      using disjoint_family_disjointed UN_disjointed_eq[of A] space range
      by auto
    show emeasure M (disjointed A i) ≠ ∞ for i
    proof −
      have emeasure M (disjointed A i) ≤ emeasure M (A i)
        using range disjointed_subset[of A i] by (auto intro!: emeasure_mono)
      then show ?thesis using measure[of i] by (auto simp: top_unique)
    qed
  qed
qed

lemma (in sigma_finite_measure) sigma_finite_incseq:
  obtains A :: nat ⇒ 'a set
  where range A ⊆ sets M (⋃i. A i) = space M ⋀i. emeasure M (A i) ≠ ∞
incseq A
proof −
  obtain F :: nat ⇒ 'a set where
    F: range F ⊆ sets M (⋃i. F i) = space M ⋀i. emeasure M (F i) ≠ ∞
    using sigma_finite by blast
  show thesis
  proof (rule that[of λn. ⋃i≤n. F i])
    show range (λn. ⋃i≤n. F i) ⊆ sets M
      using F by (force simp: incseq_def)
    show (⋃n. ⋃i≤n. F i) = space M
    proof −
      from F have ⋀x. x ∈ space M ⟹ ∃i. x ∈ F i by auto
      with F show ?thesis by fastforce
    qed
    show emeasure M (⋃i≤n. F i) ≠ ∞ for n
    proof −
      have emeasure M (⋃i≤n. F i) ≤ (∑i≤n. emeasure M (F i))
        using F by (auto intro!: emeasure_subadditive_finite)
      also have ... < ∞
        using F by (auto simp: sum_Pinfty less_top)
      finally show ?thesis by simp
    qed
    show incseq (λn. ⋃i≤n. F i)
      by (force simp: incseq_def)
  qed
qed

lemma (in sigma_finite_measure) approx_PInf_emeasure_with_finite:
```

fixes C::*real*
assumes *W_meas*: $W \in$ *sets* M
and *W_inf*: *emeasure* M $W = \infty$
obtains Z **where** $Z \in$ *sets* M $Z \subseteq W$ *emeasure* M $Z < \infty$ *emeasure* M $Z > C$
proof −
obtain $A :: nat \Rightarrow 'a$ *set*
where *A*: *range* $A \subseteq$ *sets* M $(\bigcup i.\ A\ i) =$ *space* M $\bigwedge i.$ *emeasure* M $(A\ i) \neq \infty$ *incseq* A
using *sigma_finite_incseq* **by** *blast*
define B **where** $B = (\lambda i.\ W \cap A\ i)$
have *B_meas*: $\bigwedge i.\ B\ i \in$ *sets* M **using** *W_meas* ‹*range* $A \subseteq$ *sets* M› *B_def* **by** *blast*
have *b*: $\bigwedge i.\ B\ i \subseteq W$ **using** *B_def* **by** *blast*

{ fix i
have *emeasure* M $(B\ i) \leq$ *emeasure* M $(A\ i)$
using A **by** (*intro emeasure_mono*) (*auto simp*: *B_def*)
also have *emeasure* M $(A\ i) < \infty$
using ‹$\bigwedge i.$ *emeasure* M $(A\ i) \neq \infty$› **by** (*simp add*: *less_top*)
finally have *emeasure* M $(B\ i) < \infty$ **. }**
note $c =$ *this*

have $W = (\bigcup i.\ B\ i)$ **using** *B_def* ‹$(\bigcup i.\ A\ i) =$ *space* M› *W_meas* **by** *auto*
moreover have *incseq* B **using** *B_def* ‹*incseq* A› **by** (*simp add*: *incseq_def subset_eq*)
ultimately have $(\lambda i.$ *emeasure* M $(B\ i)) \longrightarrow$ *emeasure* M W **using** *W_meas B_meas*
by (*simp add*: *B_meas Lim_emeasure_incseq image_subset_iff*)
then have $(\lambda i.$ *emeasure* M $(B\ i)) \longrightarrow \infty$ **using** *W_inf* **by** *simp*
from *order_tendstoD(1)*[*OF this*, *of C*]
obtain i **where** *d*: *emeasure* M $(B\ i) > C$
by (*auto simp*: *eventually_sequentially*)
have $B\ i \in$ *sets* M $B\ i \subseteq W$ *emeasure* M $(B\ i) < \infty$ *emeasure* M $(B\ i) > C$
using *B_meas b c d* **by** *auto*
then show *?thesis* **using** *that* **by** *blast*
qed

6.3.8 Measure space induced by distribution of $(\rightarrow_M)$-functions

definition *distr* :: $'a$ *measure* $\Rightarrow$ $'b$ *measure* $\Rightarrow$ $('a \Rightarrow 'b) \Rightarrow$ $'b$ *measure* **where**
distr M N $f =$
measure_of (*space* N) (*sets* N) $(\lambda A.$ *emeasure* M $(f -` A \cap$ *space* $M))$

lemma
shows *sets_distr*[*simp*, *measurable_cong*]: *sets* (*distr* M N f) = *sets* N
and *space_distr*[*simp*]: *space* (*distr* M N f) = *space* N
by (*auto simp*: *distr_def*)

lemma

shows *measurable_distr_eq1*[*simp*]: *measurable* (*distr Mf Nf f*) *Mf'* = *measurable Nf Mf'*
and *measurable_distr_eq2*[*simp*]: *measurable Mg'* (*distr Mg Ng g*) = *measurable Mg' Ng*
by (*auto simp*: *measurable_def*)

lemma *distr_cong*:
M = *K* ⟹ *sets N* = *sets L* ⟹ (⋀*x*. *x* ∈ *space M* ⟹ *f x* = *g x*) ⟹ *distr M N f* = *distr K L g*
using *sets_eq_imp_space_eq*[*of N L*] **by** (*simp add*: *distr_def Int_def cong*: *rev_conj_cong*)

lemma *emeasure_distr*:
fixes *f* :: *'a* ⇒ *'b*
assumes *f*: *f* ∈ *measurable M N* **and** *A*: *A* ∈ *sets N*
shows *emeasure* (*distr M N f*) *A* = *emeasure M* (*f* −‘ *A* ∩ *space M*) (**is** _ = *?μ A*)
unfolding *distr_def*
proof (*rule emeasure_measure_of_sigma*)
show *positive* (*sets N*) *?μ*
by (*auto simp*: *positive_def*)

show *countably_additive* (*sets N*) *?μ*
proof (*intro countably_additiveI*)
fix *A* :: *nat* ⇒ *'b set* **assume** *range A* ⊆ *sets N disjoint_family A*
then have *A*: ⋀*i*. *A i* ∈ *sets N* (⋃ *i*. *A i*) ∈ *sets N* **by** *auto*
then have ∗: *range* (λ*i*. *f* −‘ (*A i*) ∩ *space M*) ⊆ *sets M*
using *f* **by** (*auto simp*: *measurable_def*)
moreover have (⋃ *i*. *f* −‘ *A i* ∩ *space M*) ∈ *sets M*
using ∗ **by** *blast*
moreover have ∗∗: *disjoint_family* (λ*i*. *f* −‘ *A i* ∩ *space M*)
using ‹*disjoint_family A*› **by** (*auto simp*: *disjoint_family_on_def*)
ultimately show (∑ *i*. *?μ* (*A i*)) = *?μ* (⋃ *i*. *A i*)
using *suminf_emeasure*[*OF* _ ∗∗] *A f*
by (*auto simp*: *comp_def vimage_UN*)
qed
show *sigma_algebra* (*space N*) (*sets N*) **..**
qed *fact*

lemma *emeasure_Collect_distr*:
assumes *X*[*measurable*]: *X* ∈ *measurable M N Measurable.pred N P*
shows *emeasure* (*distr M N X*) {*x*∈*space N*. *P x*} = *emeasure M* {*x*∈*space M*. *P* (*X x*)}
by (*subst emeasure_distr*)
(*auto intro*!: *arg_cong2*[**where** *f*=*emeasure*] *X*(*1*)[*THEN measurable_space*])

lemma *emeasure_lfp2*[*consumes 1*, *case_names cont f measurable*]:
assumes *P M*
assumes *cont*: *sup_continuous F*

assumes f: $\bigwedge M.\ P\ M \Longrightarrow f \in \mathit{measurable}\ M'\ M$
assumes $*$: $\bigwedge M\ A.\ P\ M \Longrightarrow (\bigwedge N.\ P\ N \Longrightarrow \mathit{Measurable.pred}\ N\ A) \Longrightarrow \mathit{Measurable.pred}\ M\ (F\ A)$
shows $\mathit{emeasure}\ M'\ \{x \in \mathit{space}\ M'.\ \mathit{lfp}\ F\ (f\ x)\} = (\mathit{SUP}\ i.\ \mathit{emeasure}\ M'\ \{x \in \mathit{space}\ M'.\ (F\ \widehat{\ }\widehat{\ }\ i)\ (\lambda x.\ \mathit{False})\ (f\ x)\})$
proof ($\mathit{subst}\ (1\ 2)\ \mathit{emeasure\_Collect\_distr}[\mathit{symmetric}$, **where** $X=f]$)
show $f \in \mathit{measurable}\ M'\ M\ \ f \in \mathit{measurable}\ M'\ M$
using $f[\mathit{OF}\ \langle P\ M\rangle]$ **by** *auto*
{ fix i **show** $\mathit{Measurable.pred}\ M\ ((F\ \widehat{\ }\widehat{\ }\ i)\ (\lambda x.\ \mathit{False}))$
using $\langle P\ M\rangle$ **by** ($\mathit{induction}\ i\ \mathit{arbitrary}$: M) ($\mathit{auto}\ \mathit{intro}!$: $*$) **}**
show $\mathit{Measurable.pred}\ M\ (\mathit{lfp}\ F)$
using $\langle P\ M\rangle\ \mathit{cont}\ *$ **by** ($\mathit{rule}\ \mathit{measurable\_lfp\_coinduct}[\mathit{of}\ P]$)

have $\mathit{emeasure}\ (\mathit{distr}\ M'\ M\ f)\ \{x \in \mathit{space}\ (\mathit{distr}\ M'\ M\ f).\ \mathit{lfp}\ F\ x\} =$
$(\mathit{SUP}\ i.\ \mathit{emeasure}\ (\mathit{distr}\ M'\ M\ f)\ \{x \in \mathit{space}\ (\mathit{distr}\ M'\ M\ f).\ (F\ \widehat{\ }\widehat{\ }\ i)\ (\lambda x.\ \mathit{False})\ x\})$
using $\langle P\ M\rangle$
proof ($\mathit{coinduction}\ \mathit{arbitrary}$: M rule: $\mathit{emeasure\_lfp}'$)
case ($\mathit{measurable}\ A\ N$) **then have** $\bigwedge N.\ P\ N \Longrightarrow \mathit{Measurable.pred}\ (\mathit{distr}\ M'\ N\ f)\ A$
by *metis*
then have $\bigwedge N.\ P\ N \Longrightarrow \mathit{Measurable.pred}\ N\ A$
by *simp*
with $\langle P\ N\rangle[\mathit{THEN}\ *]$ **show** *?case*
by *auto*
qed *fact*
then show $\mathit{emeasure}\ (\mathit{distr}\ M'\ M\ f)\ \{x \in \mathit{space}\ M.\ \mathit{lfp}\ F\ x\} =$
$(\mathit{SUP}\ i.\ \mathit{emeasure}\ (\mathit{distr}\ M'\ M\ f)\ \{x \in \mathit{space}\ M.\ (F\ \widehat{\ }\widehat{\ }\ i)\ (\lambda x.\ \mathit{False})\ x\})$
by *simp*
qed

lemma $\mathit{distr\_id}[\mathit{simp}]$: $\mathit{distr}\ N\ N\ (\lambda x.\ x) = N$
by ($\mathit{rule}\ \mathit{measure\_eqI}$) ($\mathit{auto}\ \mathit{simp}$: $\mathit{emeasure\_distr}$)

lemma $\mathit{distr\_id2}$: $\mathit{sets}\ M = \mathit{sets}\ N \Longrightarrow \mathit{distr}\ N\ M\ (\lambda x.\ x) = N$
by ($\mathit{rule}\ \mathit{measure\_eqI}$) ($\mathit{auto}\ \mathit{simp}$: $\mathit{emeasure\_distr}$)

lemma $\mathit{measure\_distr}$:
$f \in \mathit{measurable}\ M\ N \Longrightarrow S \in \mathit{sets}\ N \Longrightarrow \mathit{measure}\ (\mathit{distr}\ M\ N\ f)\ S = \mathit{measure}\ M\ (f\ -`\ S \cap \mathit{space}\ M)$
by ($\mathit{simp}\ \mathit{add}$: $\mathit{emeasure\_distr}\ \mathit{measure\_def}$)

lemma $\mathit{distr\_cong\_AE}$:
assumes 1: $M = K\ \mathit{sets}\ N = \mathit{sets}\ L$ **and**
2: ($\mathit{AE}\ x\ \mathit{in}\ M.\ f\ x = g\ x$) **and** $f \in \mathit{measurable}\ M\ N$ **and** $g \in \mathit{measurable}\ K\ L$
shows $\mathit{distr}\ M\ N\ f = \mathit{distr}\ K\ L\ g$
proof ($\mathit{rule}\ \mathit{measure\_eqI}$)
fix A **assume** $A \in \mathit{sets}\ (\mathit{distr}\ M\ N\ f)$
with assms **show** $\mathit{emeasure}\ (\mathit{distr}\ M\ N\ f)\ A = \mathit{emeasure}\ (\mathit{distr}\ K\ L\ g)\ A$

by (*auto simp add*: *emeasure_distr intro*!: *emeasure_eq_AE measurable_sets*)
qed (*insert 1*, *simp*)

lemma *AE_distrD*:
assumes *f*: $f \in$ *measurable M M'*
and *AE*: *AE x in distr M M' f*. *P x*
shows *AE x in M*. *P* (*f x*)
proof −
from *AE*[*THEN AE_E*] **obtain** *N*
where {*x* ∈ *space* (*distr M M' f*). ¬ *P x*} ⊆ *N*
emeasure (*distr M M' f*) *N* = *0*
N ∈ *sets* (*distr M M' f*)
by *auto*
with *f* **show** *?thesis*
by (*simp add*: *eventually_ae_filter*, *intro bexI*[*of* _ *f* −‘ *N* ∩ *space M*])
(*auto simp*: *emeasure_distr measurable_def*)
qed

lemma *AE_distr_iff*:
assumes *f*[*measurable*]: $f \in$ *measurable M N* **and** *P*[*measurable*]: {*x* ∈ *space N*. *P x*} ∈ *sets N*
shows (*AE x in distr M N f*. *P x*) ⟷ (*AE x in M*. *P* (*f x*))
proof (*subst* (*1 2*) *AE_iff_measurable*[*OF* _ *refl*])
have *f* −‘ {*x*∈*space N*. ¬ *P x*} ∩ *space M* = {*x* ∈ *space M*. ¬ *P* (*f x*)}
using *f*[*THEN measurable_space*] **by** *auto*
then show (*emeasure* (*distr M N f*) {*x* ∈ *space* (*distr M N f*). ¬ *P x*} = *0*) =
(*emeasure M* {*x* ∈ *space M*. ¬ *P* (*f x*)} = *0*)
by (*simp add*: *emeasure_distr*)
qed *auto*

lemma *null_sets_distr_iff*:
f ∈ *measurable M N* ⟹ *A* ∈ *null_sets* (*distr M N f*) ⟷ *f* −‘ *A* ∩ *space M* ∈ *null_sets M* ∧ *A* ∈ *sets N*
by (*auto simp add*: *null_sets_def emeasure_distr*)

proposition *distr_distr*:
g ∈ *measurable N L* ⟹ *f* ∈ *measurable M N* ⟹ *distr* (*distr M N f*) *L g* = *distr M L* (*g* ∘ *f*)
by (*auto simp add*: *emeasure_distr measurable_space*
intro!: *arg_cong*[**where** *f*=*emeasure M*] *measure_eqI*)

6.3.9 Real measure values

lemma *ring_of_finite_sets*: *ring_of_sets* (*space M*) {*A*∈*sets M*. *emeasure M A* ≠ *top*}
proof (*rule ring_of_setsI*)
show *a* ∈ {*A* ∈ *sets M*. *emeasure M A* ≠ *top*} ⟹ *b* ∈ {*A* ∈ *sets M*. *emeasure M A* ≠ *top*} ⟹
a ∪ *b* ∈ {*A* ∈ *sets M*. *emeasure M A* ≠ *top*} **for** *a b*

using *emeasure_subadditive*[*of a M b*] **by** (*auto simp*: *top_unique*)

show $a \in \{A \in sets\ M.\ emeasure\ M\ A \neq top\} \Longrightarrow b \in \{A \in sets\ M.\ emeasure\ M\ A \neq top\} \Longrightarrow$
$a - b \in \{A \in sets\ M.\ emeasure\ M\ A \neq top\}$ **for** *a b*
using *emeasure_mono*[*of a − b a M*] **by** (*auto simp*: *top_unique*)
qed (*auto dest*: *sets.sets_into_space*)

lemma *measure_nonneg*[*simp*]: $0 \leq measure\ M\ A$
unfolding *measure_def* **by** *auto*

lemma *measure_nonneg'* [*simp*]: $\neg\ measure\ M\ A < 0$
using *measure_nonneg not_le* **by** *blast*

lemma *zero_less_measure_iff*: $0 < measure\ M\ A \longleftrightarrow measure\ M\ A \neq 0$
using *measure_nonneg*[*of M A*] **by** (*auto simp add*: *le_less*)

lemma *measure_le_0_iff*: $measure\ M\ X \leq 0 \longleftrightarrow measure\ M\ X = 0$
using *measure_nonneg*[*of M X*] **by** *linarith*

lemma *measure_empty*[*simp*]: $measure\ M\ \{\} = 0$
unfolding *measure_def* **by** (*simp add*: *zero_ennreal.rep_eq*)

lemma *emeasure_eq_ennreal_measure*:
$emeasure\ M\ A \neq top \Longrightarrow emeasure\ M\ A = ennreal\ (measure\ M\ A)$
by (*cases emeasure M A rule*: *ennreal_cases*) (*auto simp*: *measure_def*)

lemma *measure_zero_top*: $emeasure\ M\ A = top \Longrightarrow measure\ M\ A = 0$
by (*simp add*: *measure_def*)

lemma *measure_eq_emeasure_eq_ennreal*: $0 \leq x \Longrightarrow emeasure\ M\ A = ennreal\ x \Longrightarrow measure\ M\ A = x$
using *emeasure_eq_ennreal_measure*[*of M A*]
by (*cases* $A \in M$) (*auto simp*: *measure_notin_sets emeasure_notin_sets*)

lemma *enn2real_plus*:$a < top \Longrightarrow b < top \Longrightarrow enn2real\ (a + b) = enn2real\ a + enn2real\ b$
by (*simp add*: *enn2real_def plus_ennreal.rep_eq real_of_ereal_add less_top*
del: *real_of_ereal_enn2ereal*)

lemma *enn2real_sum*:$(\bigwedge i.\ i \in I \Longrightarrow f\ i < top) \Longrightarrow enn2real\ (sum\ f\ I) = sum\ (enn2real \circ f)\ I$
by (*induction I rule*: *infinite_finite_induct*) (*auto simp*: *enn2real_plus*)

lemma *measure_eq_AE*:
assumes *iff*: $AE\ x\ in\ M.\ x \in A \longleftrightarrow x \in B$
assumes *A*: $A \in sets\ M$ **and** *B*: $B \in sets\ M$
shows $measure\ M\ A = measure\ M\ B$
using *assms emeasure_eq_AE*[*OF assms*] **by** (*simp add*: *measure_def*)

lemma *measure_Union*:
emeasure M A $\neq \infty \Longrightarrow$ *emeasure M B* $\neq \infty \Longrightarrow$ *A* $\in$ *sets M* $\Longrightarrow$ *B* $\in$ *sets M* $\Longrightarrow$ *A* $\cap$ *B* = {} $\Longrightarrow$
measure M (*A* $\cup$ *B*) = *measure M A* + *measure M B*
by (*simp add*: *measure_def plus_emeasure*[*symmetric*] *enn2real_plus less_top*)

lemma *measure_finite_Union*:
finite S $\Longrightarrow$ *A*‘*S* $\subseteq$ *sets M* $\Longrightarrow$ *disjoint_family_on A S* $\Longrightarrow$ ($\bigwedge$*i*. *i* $\in$ *S* $\Longrightarrow$ *emeasure M* (*A i*) $\neq \infty$) $\Longrightarrow$
measure M ($\bigcup$*i*$\in$*S*. *A i*) = ($\sum$*i*$\in$*S*. *measure M* (*A i*))
by (*induction S rule*: *finite_induct*)
(*auto simp*: *disjoint_family_on_insert measure_Union sum_emeasure*[*symmetric*] *sets.countable_UN'*[*OF countable_finite*])

lemma *measure_Diff*:
assumes *finite*: *emeasure M A* $\neq \infty$
and *measurable*: *A* $\in$ *sets M B* $\in$ *sets M B* $\subseteq$ *A*
shows *measure M* (*A* − *B*) = *measure M A* − *measure M B*
proof −
have *emeasure M* (*A* − *B*) $\leq$ *emeasure M A emeasure M B* $\leq$ *emeasure M A*
using *measurable* **by** (*auto intro*!: *emeasure_mono*)
hence *measure M* ((*A* − *B*) $\cup$ *B*) = *measure M* (*A* − *B*) + *measure M B*
using *measurable finite* **by** (*rule_tac measure_Union*) (*auto simp*: *top_unique*)
thus *?thesis* **using** ‹*B* $\subseteq$ *A*› **by** (*auto simp*: *Un_absorb2*)
qed

lemma *measure_UNION*:
assumes *measurable*: *range A* $\subseteq$ *sets M disjoint_family A*
assumes *finite*: *emeasure M* ($\bigcup$*i*. *A i*) $\neq \infty$
shows (λ*i*. *measure M* (*A i*)) *sums* (*measure M* ($\bigcup$*i*. *A i*))
proof −
have (λ*i*. *emeasure M* (*A i*)) *sums* (*emeasure M* ($\bigcup$*i*. *A i*))
unfolding *suminf_emeasure*[*OF measurable*, *symmetric*] **by** (*simp add*: *summable_sums*)
moreover
{ fix *i*
have *emeasure M* (*A i*) $\leq$ *emeasure M* ($\bigcup$*i*. *A i*)
using *measurable* **by** (*auto intro*!: *emeasure_mono*)
then have *emeasure M* (*A i*) = *ennreal* ((*measure M* (*A i*)))
using *finite* **by** (*intro emeasure_eq_ennreal_measure*) (*auto simp*: *top_unique*)
}
ultimately show *?thesis* **using** *finite*
by (*subst* (*asm*) (*2*) *emeasure_eq_ennreal_measure*) *simp_all*
qed

lemma *measure_subadditive*:
assumes *measurable*: *A* $\in$ *sets M B* $\in$ *sets M*
and *fin*: *emeasure M A* $\neq \infty$ *emeasure M B* $\neq \infty$
shows *measure M* (*A* $\cup$ *B*) $\leq$ *measure M A* + *measure M B*

```
proof −
  have emeasure M (A ∪ B) ≠ ∞
    using emeasure_subadditive[OF measurable] fin by (auto simp: top_unique)
  then show (measure M (A ∪ B)) ≤ (measure M A) + (measure M B)
    unfolding measure_def
   by (metis emeasure_subadditive[OF measurable] fin   enn2real_mono enn2real_plus

       ennreal_add_less_top infinity_ennreal_def less_top)
qed

lemma measure_subadditive_finite:
  assumes A: finite I A‘I ⊆ sets M and fin: ⋀i. i ∈ I ⟹ emeasure M (A i) ≠
∞
  shows measure M (⋃i∈I. A i) ≤ (∑ i∈I. measure M (A i))
proof −
  { have emeasure M (⋃i∈I. A i) ≤ (∑ i∈I. emeasure M (A i))
      using emeasure_subadditive_finite[OF A] .
    also have ... < ∞
      using fin by (simp add: less_top A)
    finally have emeasure M (⋃i∈I. A i) ≠ top by simp }
  note ∗ = this
  show ?thesis
    using emeasure_subadditive_finite[OF A] fin
    unfolding emeasure_eq_ennreal_measure[OF ∗]
    by (simp_all add: sum_nonneg emeasure_eq_ennreal_measure)
qed

lemma measure_subadditive_countably:
  assumes A: range A ⊆ sets M and fin: (∑ i. emeasure M (A i)) ≠ ∞
  shows measure M (⋃ i. A i) ≤ (∑ i. measure M (A i))
proof −
  have ∗∗: ⋀i. emeasure M (A i) ≠ top
     using fin ennreal_suminf_lessD[of λi. emeasure M (A i)] by (simp add:
less_top)
  have ge0: (∑ i. Sigma_Algebra.measure M (A i)) ≥ 0
    using fin emeasure_eq_ennreal_measure[OF ∗∗]
    by (metis infinity_ennreal_def measure_nonneg suminf_cong suminf_nonneg
summable_suminf_not_top)
  have emeasure M (⋃ i. A i) ≠ top
   by (metis A emeasure_subadditive_countably fin infinity_ennreal_def neq_top_trans)
  then have ennreal (measure M (⋃ i. A i)) = emeasure M (⋃ i. A i)
    by (rule emeasure_eq_ennreal_measure[symmetric])
  also have ... ≤ (∑ i. emeasure M (A i))
    using emeasure_subadditive_countably[OF A] .
  also have ... = ennreal (∑ i. measure M (A i))
    using fin unfolding emeasure_eq_ennreal_measure[OF ∗∗]
    by (subst suminf_ennreal) (auto simp: ∗∗)
  finally show ?thesis
    using ge0 ennreal_le_iff by blast
```

qed

lemma *measure_Un_null_set*: $A \in$ *sets* $M \Longrightarrow B \in$ *null_sets* $M \Longrightarrow$ *measure* M $(A \cup B) =$ *measure* M A
by (*simp add*: *measure_def emeasure_Un_null_set*)

lemma *measure_Diff_null_set*: $A \in$ *sets* $M \Longrightarrow B \in$ *null_sets* $M \Longrightarrow$ *measure* M $(A - B) =$ *measure* M A
by (*simp add*: *measure_def emeasure_Diff_null_set*)

lemma *measure_eq_sum_singleton*:
finite $S \Longrightarrow (\bigwedge x.\ x \in S \Longrightarrow \{x\} \in$ *sets* $M) \Longrightarrow (\bigwedge x.\ x \in S \Longrightarrow$ *emeasure* M $\{x\} \neq \infty) \Longrightarrow$
measure M $S = (\sum x \in S.$ *measure* M $\{x\})$
using *emeasure_eq_sum_singleton*[*of* S M]
by (*intro measure_eq_emeasure_eq_ennreal*) (*auto simp*: *sum_nonneg emeasure_eq_ennreal_measure*)

lemma *Lim_measure_incseq*:
assumes A: *range* $A \subseteq$ *sets* M *incseq* A **and** *fin*: *emeasure* M $(\bigcup i.\ A\ i) \neq \infty$
shows $(\lambda i.$ *measure* M $(A\ i)) \longrightarrow$ *measure* M $(\bigcup i.\ A\ i)$
proof (*rule tendsto_ennrealD*)
have *ennreal* (*measure* M $(\bigcup i.\ A\ i)) =$ *emeasure* M $(\bigcup i.\ A\ i)$
using *fin* **by** (*auto simp*: *emeasure_eq_ennreal_measure*)
moreover have *ennreal* (*measure* M $(A\ i)) =$ *emeasure* M $(A\ i)$ **for** i
using *assms emeasure_mono*[*of* A _ $\bigcup i.\ A\ i$ M]
by (*intro emeasure_eq_ennreal_measure*[*symmetric*]) (*auto simp*: *less_top UN_upper intro*: *le_less_trans*)
ultimately show $(\lambda x.$ *ennreal* (*measure* M $(A\ x))) \longrightarrow$ *ennreal* (*measure* M $(\bigcup i.\ A\ i))$
using A **by** (*auto intro*!: *Lim_emeasure_incseq*)
qed *auto*

lemma *Lim_measure_decseq*:
assumes A: *range* $A \subseteq$ *sets* M *decseq* A **and** *fin*: $\bigwedge i.$ *emeasure* M $(A\ i) \neq \infty$
shows $(\lambda n.$ *measure* M $(A\ n)) \longrightarrow$ *measure* M $(\bigcap i.\ A\ i)$
proof (*rule tendsto_ennrealD*)
have *ennreal* (*measure* M $(\bigcap i.\ A\ i)) =$ *emeasure* M $(\bigcap i.\ A\ i)$
using *fin*[*of* 0] A *emeasure_mono*[*of* $\bigcap i.\ A\ i$ A 0 M]
by (*auto intro*!: *emeasure_eq_ennreal_measure*[*symmetric*] *simp*: *INT_lower less_top intro*: *le_less_trans*)
moreover have *ennreal* (*measure* M $(A\ i)) =$ *emeasure* M $(A\ i)$ **for** i
using A *fin*[*of* i] **by** (*intro emeasure_eq_ennreal_measure*[*symmetric*]) *auto*
ultimately show $(\lambda x.$ *ennreal* (*measure* M $(A\ x))) \longrightarrow$ *ennreal* (*measure* M $(\bigcap i.\ A\ i))$
using *fin* A **by** (*auto intro*!: *Lim_emeasure_decseq*)
qed *auto*

6.3.10 Set of measurable sets with finite measure

definition *fmeasurable* :: *'a measure* ⇒ *'a set set* **where**
fmeasurable M = {*A*∈*sets M*. *emeasure M A* < ∞}

lemma *fmeasurableD*[*dest*, *measurable_dest*]: *A* ∈ *fmeasurable M* ⟹ *A* ∈ *sets M*
by (*auto simp*: *fmeasurable_def*)

lemma *fmeasurableD2*: *A* ∈ *fmeasurable M* ⟹ *emeasure M A* ≠ *top*
by (*auto simp*: *fmeasurable_def*)

lemma *fmeasurableI*: *A* ∈ *sets M* ⟹ *emeasure M A* < ∞ ⟹ *A* ∈ *fmeasurable M*
by (*auto simp*: *fmeasurable_def*)

lemma *fmeasurableI_null_sets*: *A* ∈ *null_sets M* ⟹ *A* ∈ *fmeasurable M*
by (*auto simp*: *fmeasurable_def*)

lemma *fmeasurableI2*: *A* ∈ *fmeasurable M* ⟹ *B* ⊆ *A* ⟹ *B* ∈ *sets M* ⟹ *B* ∈ *fmeasurable M*
using *emeasure_mono*[*of B A M*] **by** (*auto simp*: *fmeasurable_def*)

lemma *measure_mono_fmeasurable*:
A ⊆ *B* ⟹ *A* ∈ *sets M* ⟹ *B* ∈ *fmeasurable M* ⟹ *measure M A* ≤ *measure M B*
by (*auto simp*: *measure_def fmeasurable_def intro*!: *emeasure_mono enn2real_mono*)

lemma *emeasure_eq_measure2*: *A* ∈ *fmeasurable M* ⟹ *emeasure M A* = *measure M A*
by (*simp add*: *emeasure_eq_ennreal_measure fmeasurable_def less_top*)

interpretation *fmeasurable*: *ring_of_sets space M fmeasurable M*
proof (*rule ring_of_setsI*)
show *fmeasurable M* ⊆ *Pow* (*space M*) {} ∈ *fmeasurable M*
by (*auto simp*: *fmeasurable_def dest*: *sets.sets_into_space*)
fix *a b* **assume** ∗: *a* ∈ *fmeasurable M b* ∈ *fmeasurable M*
then have *emeasure M* (*a* ∪ *b*) ≤ *emeasure M a* + *emeasure M b*
by (*intro emeasure_subadditive*) *auto*
also have ... < *top*
using ∗ **by** (*auto simp*: *fmeasurable_def*)
finally show *a* ∪ *b* ∈ *fmeasurable M*
using ∗ **by** (*auto intro*: *fmeasurableI*)
show *a* − *b* ∈ *fmeasurable M*
using *emeasure_mono*[*of a* − *b a M*] ∗ **by** (*auto simp*: *fmeasurable_def*)
qed

6.3.11 Measurable sets formed by unions and intersections

lemma *fmeasurable_Diff*: *A* ∈ *fmeasurable M* ⟹ *B* ∈ *sets M* ⟹ *A* − *B* ∈ *fmeasurable M*

using *fmeasurableI2*[*of A M A − B*] **by** *auto*

lemma *fmeasurable_Int_fmeasurable*:
⟦*S* ∈ *fmeasurable M*; *T* ∈ *sets M*⟧ ⟹ (*S* ∩ *T*) ∈ *fmeasurable M*
by (*meson fmeasurableD fmeasurableI2 inf_le1 sets.Int*)

lemma *fmeasurable_UN*:
assumes *countable I* ⋀*i*. *i* ∈ *I* ⟹ *F i* ⊆ *A* ⋀*i*. *i* ∈ *I* ⟹ *F i* ∈ *sets M A* ∈ *fmeasurable M*
shows (⋃*i*∈*I*. *F i*) ∈ *fmeasurable M*
proof (*rule fmeasurableI2*)
show *A* ∈ *fmeasurable M* (⋃*i*∈*I*. *F i*) ⊆ *A* **using** *assms* **by** *auto*
show (⋃*i*∈*I*. *F i*) ∈ *sets M*
using *assms* **by** (*intro sets.countable_UN′*) *auto*
qed

lemma *fmeasurable_INT*:
assumes *countable I i* ∈ *I* ⋀*i*. *i* ∈ *I* ⟹ *F i* ∈ *sets M F i* ∈ *fmeasurable M*
shows (⋂*i*∈*I*. *F i*) ∈ *fmeasurable M*
proof (*rule fmeasurableI2*)
show *F i* ∈ *fmeasurable M* (⋂*i*∈*I*. *F i*) ⊆ *F i*
using *assms* **by** *auto*
show (⋂*i*∈*I*. *F i*) ∈ *sets M*
using *assms* **by** (*intro sets.countable_INT′*) *auto*
qed

lemma *measurable_measure_Diff*:
assumes *A* ∈ *fmeasurable M B* ∈ *sets M B* ⊆ *A*
shows *measure M* (*A* − *B*) = *measure M A* − *measure M B*
by (*simp add*: *assms fmeasurableD fmeasurableD2 measure_Diff*)

lemma *measurable_Un_null_set*:
assumes *B* ∈ *null_sets M*
shows (*A* ∪ *B* ∈ *fmeasurable M* ∧ *A* ∈ *sets M*) ⟷ *A* ∈ *fmeasurable M*
using *assms* **by** (*fastforce simp add*: *fmeasurable.Un fmeasurableI_null_sets intro*: *fmeasurableI2*)

lemma *measurable_Diff_null_set*:
assumes *B* ∈ *null_sets M*
shows (*A* − *B*) ∈ *fmeasurable M* ∧ *A* ∈ *sets M* ⟷ *A* ∈ *fmeasurable M*
using *assms*
by (*metis Un_Diff_cancel2 fmeasurable.Diff fmeasurableD fmeasurableI_null_sets measurable_Un_null_set*)

lemma *fmeasurable_Diff_D*:
assumes *m*: *T* − *S* ∈ *fmeasurable M S* ∈ *fmeasurable M* **and** *sub*: *S* ⊆ *T*
shows *T* ∈ *fmeasurable M*
proof −
have *T* = *S* ∪ (*T* − *S*)

using *assms* **by** *blast*
then show *?thesis*
by (*metis m fmeasurable.Un*)
qed

lemma *measure_Un2*:
$A \in$ *fmeasurable* $M \Longrightarrow B \in$ *fmeasurable* $M \Longrightarrow$ *measure* $M\ (A \cup B) =$ *measure* $M\ A +$ *measure* $M\ (B - A)$
using *measure_Union*[*of* $M\ A\ B - A$] **by** (*auto simp*: *fmeasurableD2 fmeasurable.Diff*)

lemma *measure_Un3*:
assumes $A \in$ *fmeasurable* $M\ B \in$ *fmeasurable* M
shows *measure* $M\ (A \cup B) =$ *measure* $M\ A +$ *measure* $M\ B -$ *measure* $M\ (A \cap B)$
proof −
have *measure* $M\ (A \cup B) =$ *measure* $M\ A +$ *measure* $M\ (B - A)$
using *assms* **by** (*rule measure_Un2*)
also have $B - A = B - (A \cap B)$
by *auto*
also have *measure* $M\ (B - (A \cap B)) =$ *measure* $M\ B -$ *measure* $M\ (A \cap B)$
using *assms* **by** (*intro measure_Diff*) (*auto simp*: *fmeasurable_def*)
finally show *?thesis*
by *simp*
qed

lemma *measure_Un_AE*:
AE x in M. $x \notin A \vee x \notin B \Longrightarrow A \in$ *fmeasurable* $M \Longrightarrow B \in$ *fmeasurable* $M \Longrightarrow$
measure $M\ (A \cup B) =$ *measure* $M\ A +$ *measure* $M\ B$
by (*subst measure_Un2*) (*auto intro*!: *measure_eq_AE*)

lemma *measure_UNION_AE*:
assumes *I*: *finite I*
shows $(\bigwedge i.\ i \in I \Longrightarrow F\ i \in$ *fmeasurable* $M) \Longrightarrow$ *pairwise* $(\lambda i\ j.$ *AE x in M*. $x \notin F\ i \vee x \notin F\ j)\ I \Longrightarrow$
measure $M\ (\bigcup i{\in}I.\ F\ i) = (\sum i{\in}I.$ *measure* $M\ (F\ i))$
unfolding *AE_pairwise*[*OF countable_finite*, *OF I*]
using *I*
proof (*induction I rule*: *finite_induct*)
case (*insert x I*)
have *measure* $M\ (F\ x \cup \bigcup(F\ `\ I)) =$ *measure* $M\ (F\ x) +$ *measure* $M\ (\bigcup(F\ `\ I))$
by (*rule measure_Un_AE*) (*use insert* **in** ‹*auto simp*: *pairwise_insert*›)
with *insert* **show** *?case*
by (*simp add*: *pairwise_insert*)
qed *simp*

lemma *measure_UNION′*:
finite I $\Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow F\ i \in$ *fmeasurable* $M) \Longrightarrow$ *pairwise* $(\lambda i\ j.$ *disjnt* $(F$

i) (*F j*)) *I* ⟹
measure M (⋃*i*∈*I*. *F i*) = (∑*i*∈*I*. *measure M* (*F i*))
by (*intro measure_UNION_AE*) (*auto simp*: *disjnt_def elim*!: *pairwise_mono intro*!: *always_eventually*)

lemma *measure_Union_AE*:
finite F ⟹ (⋀*S*. *S* ∈ *F* ⟹ *S* ∈ *fmeasurable M*) ⟹ *pairwise* (λ*S T*. *AE x in M*. *x* ∉ *S* ∨ *x* ∉ *T*) *F* ⟹
measure M (⋃*F*) = (∑*S*∈*F*. *measure M S*)
using *measure_UNION_AE*[*of F* λ*x*. *x M*] **by** *simp*

lemma *measure_Union'*:
finite F ⟹ (⋀*S*. *S* ∈ *F* ⟹ *S* ∈ *fmeasurable M*) ⟹ *pairwise disjnt F* ⟹ *measure M* (⋃*F*) = (∑*S*∈*F*. *measure M S*)
using *measure_UNION'*[*of F* λ*x*. *x M*] **by** *simp*

lemma *measure_Un_le*:
assumes *A* ∈ *sets M B* ∈ *sets M* **shows** *measure M* (*A* ∪ *B*) ≤ *measure M A* + *measure M B*
proof *cases*
assume *A* ∈ *fmeasurable M* ∧ *B* ∈ *fmeasurable M*
with *measure_subadditive*[*of A M B*] *assms* **show** *?thesis*
by (*auto simp*: *fmeasurableD2*)
next
assume ¬ (*A* ∈ *fmeasurable M* ∧ *B* ∈ *fmeasurable M*)
then have *A* ∪ *B* ∉ *fmeasurable M*
using *fmeasurableI2*[*of A* ∪ *B M A*] *fmeasurableI2*[*of A* ∪ *B M B*] *assms* **by** *auto*
with *assms* **show** *?thesis*
by (*auto simp*: *fmeasurable_def measure_def less_top*[*symmetric*])
qed

lemma *measure_UNION_le*:
finite I ⟹ (⋀*i*. *i* ∈ *I* ⟹ *F i* ∈ *sets M*) ⟹ *measure M* (⋃*i*∈*I*. *F i*) ≤ (∑*i*∈*I*. *measure M* (*F i*))
proof (*induction I rule*: *finite_induct*)
case (*insert i I*)
then have *measure M* (⋃*i*∈*insert i I*. *F i*) = *measure M* (*F i* ∪ ⋃ (*F* ‘ *I*))
by *simp*
also from *insert* **have** *measure M* (*F i* ∪ ⋃ (*F* ‘ *I*)) ≤ *measure M* (*F i*) + *measure M* (⋃ (*F* ‘ *I*))
by (*intro measure_Un_le sets.finite_Union*) *auto*
also have *measure M* (⋃*i*∈*I*. *F i*) ≤ (∑*i*∈*I*. *measure M* (*F i*))
using *insert* **by** *auto*
finally show *?case*
using *insert* **by** *simp*
qed *simp*

lemma *measure_Union_le*:

finite F $\Longrightarrow$ ($\bigwedge$*S. S* $\in$ *F* $\Longrightarrow$ *S* $\in$ *sets M*) $\Longrightarrow$ *measure M* ($\bigcup$ *F*) $\leq$ ($\sum$ *S*$\in$*F. measure M S*)
using *measure_UNION_le*[*of F* λ*x. x M*] **by** *simp*

Version for indexed union over a countable set

lemma
assumes *countable I* **and** *I*: $\bigwedge$*i. i* $\in$ *I* $\Longrightarrow$ *A i* $\in$ *fmeasurable M*
and *bound*: $\bigwedge$*I'. I'* $\subseteq$ *I* $\Longrightarrow$ *finite I'* $\Longrightarrow$ *measure M* ($\bigcup$*i*$\in$*I'. A i*) $\leq$ *B*
shows *fmeasurable_UN_bound*: ($\bigcup$*i*$\in$*I. A i*) $\in$ *fmeasurable M* (**is** *?fm*)
and *measure_UN_bound*: *measure M* ($\bigcup$*i*$\in$*I. A i*) $\leq$ *B* (**is** *?m*)
proof −
have *B* $\geq$ *0*
using *bound* **by** *force*
have *?fm* $\wedge$ *?m*
proof *cases*
assume *I* = {}
with ‹*B* $\geq$ *0*› **show** *?thesis*
by *simp*
next
assume *I* $\neq$ {}
have ($\bigcup$*i*$\in$*I. A i*) = ($\bigcup$*i. ($\bigcup$n*$\leq$*i. A* (*from_nat_into I n*)))
by (*subst range_from_nat_into*[*symmetric*, *OF* ‹*I* $\neq$ {}› ‹*countable I*›]) *auto*
then have *emeasure M* ($\bigcup$*i*$\in$*I. A i*) = *emeasure M* ($\bigcup$*i. (*$\bigcup$*n*$\leq$*i. A* (*from_nat_into I n*))) **by** *simp*
also have ... = (*SUP i. emeasure M* ($\bigcup$*n*$\leq$*i. A* (*from_nat_into I n*)))
using *I* ‹*I* $\neq$ {}›[*THEN from_nat_into*] **by** (*intro SUP_emeasure_incseq*[*symmetric*]) (*fastforce simp*: *incseq_Suc_iff*)+
also have ... $\leq$ *B*
proof (*intro SUP_least*)
fix *i* :: *nat*
have *emeasure M* ($\bigcup$*n*$\leq$*i. A* (*from_nat_into I n*)) = *measure M* ($\bigcup$*n*$\leq$*i. A* (*from_nat_into I n*))
using *I* ‹*I* $\neq$ {}›[*THEN from_nat_into*] **by** (*intro emeasure_eq_measure2 fmeasurable.finite_UN*) *auto*
also have ... = *measure M* ($\bigcup$*n*$\in$*from_nat_into I* ‘ {..*i*}. *A n*)
by *simp*
also have ... $\leq$ *B*
by (*intro ennreal_leI bound*) (*auto intro*: *from_nat_into*[*OF* ‹*I* $\neq$ {}›])
finally show *emeasure M* ($\bigcup$*n*$\leq$*i. A* (*from_nat_into I n*)) $\leq$ *ennreal B* **.**
qed
finally have $*$: *emeasure M* ($\bigcup$*i*$\in$*I. A i*) $\leq$ *B* **.**
then have *?fm*
using *I* ‹*countable I*› **by** (*intro fmeasurableI conjI*) (*auto simp*: *less_top*[*symmetric*] *top_unique*)
with $*$ ‹*0*$\leq$*B*› **show** *?thesis*
by (*simp add*: *emeasure_eq_measure2*)
qed
then show *?fm ?m* **by** *auto*
qed

Version for big union of a countable set

lemma
assumes *countable* $\mathcal{D}$
and *meas*: $\bigwedge D.\ D \in \mathcal{D} \Longrightarrow D \in$ *fmeasurable* M
and *bound*: $\bigwedge \mathcal{E}.\ [\![\mathcal{E} \subseteq \mathcal{D}; \textit{finite } \mathcal{E}]\!] \Longrightarrow \textit{measure } M\ (\bigcup \mathcal{E}) \leq B$
shows *fmeasurable_Union_bound*: $\bigcup \mathcal{D} \in$ *fmeasurable* M (**is** *?fm*)
and *measure_Union_bound*: *measure* $M\ (\bigcup \mathcal{D}) \leq B$ (**is** *?m*)
proof −
have $B \geq 0$
using *bound* **by** *force*
have *?fm* $\wedge$ *?m*
proof (*cases* $\mathcal{D} = \{\}$)
case *True*
with ‹$B \geq 0$› **show** *?thesis*
by *auto*
next
case *False*
then obtain $D :: \textit{nat} \Rightarrow 'a\ \textit{set}$ **where** D: $\mathcal{D} = \textit{range } D$
using ‹*countable* $\mathcal{D}$› *uncountable_def* **by** *force*
have *1*: $\bigwedge i.\ D\ i \in$ *fmeasurable* M
by (*simp add*: D *meas*)
have *2*: $\bigwedge I'.\ \textit{finite } I' \Longrightarrow \textit{measure } M\ (\bigcup x{\in}I'.\ D\ x) \leq B$
by (*simp add*: D *bound image_subset_iff*)
show *?thesis*
unfolding D
by (*intro conjI fmeasurable_UN_bound* [*OF* _ *1 2*] *measure_UN_bound* [*OF* _ *1 2*]) *auto*
qed
then show *?fm ?m* **by** *auto*
qed

Version for indexed union over the type of naturals

lemma
fixes $S :: \textit{nat} \Rightarrow 'a\ \textit{set}$
assumes S: $\bigwedge i.\ S\ i \in$ *fmeasurable* M **and** B: $\bigwedge n.\ \textit{measure } M\ (\bigcup i{\leq}n.\ S\ i) \leq B$
shows *fmeasurable_countable_Union*: $(\bigcup i.\ S\ i) \in$ *fmeasurable* M
and *measure_countable_Union_le*: *measure* $M\ (\bigcup i.\ S\ i) \leq B$
proof −
have *mB*: *measure* $M\ (\bigcup i{\in}I.\ S\ i) \leq B$ **if** *finite* I **for** I
proof −
have $(\bigcup i{\in}I.\ S\ i) \subseteq (\bigcup i{\leq}\textit{Max } I.\ S\ i)$
using *Max_ge that* **by** *force*
then have *measure* $M\ (\bigcup i{\in}I.\ S\ i) \leq \textit{measure } M\ (\bigcup i \leq \textit{Max } I.\ S\ i)$
by (*rule measure_mono_fmeasurable*) (*use* S **in** ‹*blast*+›)
then show *?thesis*
using B *order_trans* **by** *blast*
qed
show $(\bigcup i.\ S\ i) \in$ *fmeasurable* M
by (*auto intro*: *fmeasurable_UN_bound* [*OF* _ S *mB*])

```
  show measure M (⋃ n. S n) ≤ B
    by (auto intro: measure_UN_bound [OF _ S mB])
qed

lemma measure_diff_le_measure_setdiff:
  assumes S ∈ fmeasurable M T ∈ fmeasurable M
  shows measure M S − measure M T ≤ measure M (S − T)
proof −
  have measure M S ≤ measure M ((S − T) ∪ T)
   by (simp add: assms fmeasurable.Un fmeasurableD measure_mono_fmeasurable)
  also have ... ≤ measure M (S − T) + measure M T
    using assms by (blast intro: measure_Un_le)
  finally show ?thesis
    by (simp add: algebra_simps)
qed

lemma suminf_exist_split2:
  fixes f :: nat ⇒ 'a::real_normed_vector
  assumes summable f
  shows (λn. (∑ k. f(k+n))) ⟶ 0
by (subst lim_sequentially, auto simp add: dist_norm suminf_exist_split[OF _
assms])

lemma emeasure_union_summable:
  assumes [measurable]: ⋀n. A n ∈ sets M
    and ⋀n. emeasure M (A n) < ∞ summable (λn. measure M (A n))
  shows emeasure M (⋃ n. A n) < ∞ emeasure M (⋃ n. A n) ≤ (∑ n. measure
M (A n))
proof −
  define B where B = (λN. (⋃ n∈{..<N}. A n))
  have [measurable]: B N ∈ sets M for N unfolding B_def by auto
  have (λN. emeasure M (B N)) ⟶ emeasure M (⋃ N. B N)
     apply (rule Lim_emeasure_incseq) unfolding B_def by (auto simp add:
SUP_subset_mono incseq_def)
  moreover have emeasure M (B N) ≤ ennreal (∑ n. measure M (A n)) for N
  proof −
    have ∗: (∑ n<N. measure M (A n)) ≤ (∑ n. measure M (A n))
      using assms(3) measure_nonneg sum_le_suminf by blast

    have emeasure M (B N) ≤ (∑ n<N. emeasure M (A n))
      unfolding B_def by (rule emeasure_subadditive_finite, auto)
    also have ... = (∑ n<N. ennreal(measure M (A n)))
      using assms(2) by (simp add: emeasure_eq_ennreal_measure less_top)
    also have ... = ennreal (∑ n<N. measure M (A n))
      by auto
    also have ... ≤ ennreal (∑ n. measure M (A n))
      using ∗ by (auto simp: ennreal_leI)
    finally show ?thesis by simp
  qed
```

```
  ultimately have emeasure M (⋃ N. B N) ≤ ennreal (∑ n. measure M (A n))
    by (simp add: Lim_bounded)
  then show emeasure M (⋃ n. A n) ≤ (∑ n. measure M (A n))
    unfolding B_def by (metis UN_UN_flatten UN_lessThan_UNIV)
  then show emeasure M (⋃ n. A n) < ∞
    by (auto simp: less_top[symmetric] top_unique)
qed

lemma borel_cantelli_limsup1:
  assumes [measurable]: ⋀n. A n ∈ sets M
    and ⋀n. emeasure M (A n) < ∞ summable (λn. measure M (A n))
  shows limsup A ∈ null_sets M
proof −
  have emeasure M (limsup A) ≤ 0
  proof (rule LIMSEQ_le_const)
   have (λn. (∑ k. measure M (A (k+n)))) ⟶ 0 by (rule suminf_exist_split2[OF
assms(3)])
    then show (λn. ennreal (∑ k. measure M (A (k+n)))) ⟶ 0
      unfolding ennreal_0[symmetric] by (intro tendsto_ennrealI)
    have emeasure M (limsup A) ≤ (∑ k. measure M (A (k+n))) for n
    proof −
     have I: (⋃ k∈{n..}. A k) = (⋃ k. A (k+n)) by (auto, metis le_add_diff_inverse2,
fastforce)
      have emeasure M (limsup A) ≤ emeasure M (⋃ k∈{n..}. A k)
        by (rule emeasure_mono, auto simp add: limsup_INF_SUP)
      also have ... = emeasure M (⋃ k. A (k+n))
        using I by auto
      also have ... ≤ (∑ k. measure M (A (k+n)))
        apply (rule emeasure_union_summable)
       using assms summable_ignore_initial_segment[OF assms(3), of n] by auto
      finally show ?thesis by simp
    qed
     then show ∃ N. ∀ n≥N. emeasure M (limsup A) ≤ (∑ k. measure M (A
(k+n)))
      by auto
  qed
  then show ?thesis using assms(1) measurable_limsup by auto
qed

lemma borel_cantelli_AE1:
  assumes [measurable]: ⋀n. A n ∈ sets M
    and ⋀n. emeasure M (A n) < ∞ summable (λn. measure M (A n))
  shows AE x in M. eventually (λn. x ∈ space M − A n) sequentially
proof −
  have AE x in M. x ∉ limsup A
    using borel_cantelli_limsup1[OF assms] unfolding eventually_ae_filter by
auto
  moreover have ∀ F n in sequentially. x ∉ A n if x ∉ limsup A for x
    using that  by (auto simp: limsup_INF_SUP eventually_sequentially)
```

ultimately show *?thesis* **by** *auto*
qed

6.3.12 Measure spaces with *emeasure M (space M)* $< \infty$

locale *finite_measure* = *sigma_finite_measure M* **for** *M* +
assumes *finite_emeasure_space*: *emeasure M (space M)* $\neq$ *top*

lemma *finite_measureI*[*Pure.intro*!]:
emeasure M (space M) $\neq \infty \Longrightarrow$ *finite_measure M*
proof qed (*auto intro*!: *exI*[*of* _ {*space M*}])

lemma (**in** *finite_measure*) *emeasure_finite*[*simp*, *intro*]: *emeasure M A* $\neq$ *top*
using *finite_emeasure_space emeasure_space*[*of M A*] **by** (*auto simp*: *top_unique*)

lemma (**in** *finite_measure*) *fmeasurable_eq_sets*: *fmeasurable M* = *sets M*
by (*auto simp*: *fmeasurable_def less_top*[*symmetric*])

lemma (**in** *finite_measure*) *emeasure_eq_measure*: *emeasure M A* = *ennreal* (*measure M A*)
by (*intro emeasure_eq_ennreal_measure*) *simp*

lemma (**in** *finite_measure*) *emeasure_real*: $\exists r.\ 0 \leq r \wedge$ *emeasure M A* = *ennreal r*
using *emeasure_finite*[*of A*] **by** (*cases emeasure M A rule*: *ennreal_cases*) *auto*

lemma (**in** *finite_measure*) *bounded_measure*: *measure M A* $\leq$ *measure M* (*space M*)
using *emeasure_space*[*of M A*] *emeasure_real*[*of A*] *emeasure_real*[*of space M*]
by (*auto simp*: *measure_def*)

lemma (**in** *finite_measure*) *finite_measure_Diff*:
assumes *sets*: *A* $\in$ *sets M B* $\in$ *sets M* **and** *B* $\subseteq$ *A*
shows *measure M* (*A* $-$ *B*) = *measure M A* $-$ *measure M B*
using *measure_Diff*[*OF* _ *assms*] **by** *simp*

lemma (**in** *finite_measure*) *finite_measure_Union*:
assumes *sets*: *A* $\in$ *sets M B* $\in$ *sets M* **and** *A* $\cap$ *B* = {}
shows *measure M* (*A* $\cup$ *B*) = *measure M A* + *measure M B*
using *measure_Union*[*OF* _ _ *assms*] **by** *simp*

lemma (**in** *finite_measure*) *finite_measure_finite_Union*:
assumes *measurable*: *finite S A'S* $\subseteq$ *sets M disjoint_family_on A S*
shows *measure M* ($\bigcup i \in S.\ A\ i$) = ($\sum i \in S.$ *measure M* (*A i*))
using *measure_finite_Union*[*OF assms*] **by** *simp*

lemma (**in** *finite_measure*) *finite_measure_UNION*:
assumes *A*: *range A* $\subseteq$ *sets M disjoint_family A*
shows ($\lambda i.$ *measure M* (*A i*)) *sums* (*measure M* ($\bigcup i.$ *A i*))

using *measure_UNION*[*OF A*] **by** *simp*

lemma (**in** *finite_measure*) *finite_measure_mono*:
assumes $A \subseteq B$ $B \in$ *sets M* **shows** *measure M A* $\leq$ *measure M B*
using *emeasure_mono*[*OF assms*] *emeasure_real*[*of A*] *emeasure_real*[*of B*] **by** (*auto simp*: *measure_def*)

lemma (**in** *finite_measure*) *finite_measure_subadditive*:
assumes *m*: $A \in$ *sets M* $B \in$ *sets M*
shows *measure M* $(A \cup B) \leq$ *measure M A* + *measure M B*
using *measure_subadditive*[*OF m*] **by** *simp*

lemma (**in** *finite_measure*) *finite_measure_subadditive_finite*:
assumes *finite I A*‘*I* $\subseteq$ *sets M* **shows** *measure M* $(\bigcup i \in I.\ A\ i) \leq (\sum i \in I.$ *measure M* $(A\ i))$
using *measure_subadditive_finite*[*OF assms*] **by** *simp*

lemma (**in** *finite_measure*) *finite_measure_subadditive_countably*:
range A $\subseteq$ *sets M* $\Longrightarrow$ *summable* $(\lambda i.$ *measure M* $(A\ i)) \Longrightarrow$ *measure M* $(\bigcup i.\ A\ i) \leq (\sum i.$ *measure M* $(A\ i))$
by (*rule measure_subadditive_countably*)
(*simp_all add*: *ennreal_suminf_neq_top emeasure_eq_measure*)

lemma (**in** *finite_measure*) *finite_measure_eq_sum_singleton*:
assumes *finite S* **and** $*$: $\bigwedge x.\ x \in S \Longrightarrow \{x\} \in$ *sets M*
shows *measure M S* $= (\sum x \in S.$ *measure M* $\{x\})$
using *measure_eq_sum_singleton*[*OF assms*] **by** *simp*

lemma (**in** *finite_measure*) *finite_Lim_measure_incseq*:
assumes *A*: *range A* $\subseteq$ *sets M incseq A*
shows $(\lambda i.$ *measure M* $(A\ i)) \longrightarrow$ *measure M* $(\bigcup i.\ A\ i)$
using *Lim_measure_incseq*[*OF A*] **by** *simp*

lemma (**in** *finite_measure*) *finite_Lim_measure_decseq*:
assumes *A*: *range A* $\subseteq$ *sets M decseq A*
shows $(\lambda n.$ *measure M* $(A\ n)) \longrightarrow$ *measure M* $(\bigcap i.\ A\ i)$
using *Lim_measure_decseq*[*OF A*] **by** *simp*

lemma (**in** *finite_measure*) *finite_measure_compl*:
assumes *S*: $S \in$ *sets M*
shows *measure M* (*space M* $-$ *S*) $=$ *measure M* (*space M*) $-$ *measure M S*
using *measure_Diff*[*OF _ sets.top S sets.sets_into_space*] *S* **by** *simp*

lemma (**in** *finite_measure*) *finite_measure_mono_AE*:
assumes *imp*: *AE x in M*. $x \in A \longrightarrow x \in B$ **and** *B*: $B \in$ *sets M*
shows *measure M A* $\leq$ *measure M B*
using *assms emeasure_mono_AE*[*OF imp B*]
by (*simp add*: *emeasure_eq_measure*)

lemma (**in** *finite_measure*) *finite_measure_eq_AE*:
assumes *iff*: *AE x in M*. $x \in A \longleftrightarrow x \in B$
assumes *A*: $A \in$ *sets M* **and** *B*: $B \in$ *sets M*
shows *measure M A* = *measure M B*
using *assms emeasure_eq_AE*[*OF assms*] **by** (*simp add*: *emeasure_eq_measure*)

lemma (**in** *finite_measure*) *measure_increasing*: *increasing M* (*measure M*)
by (*auto intro*!: *finite_measure_mono simp*: *increasing_def*)

lemma (**in** *finite_measure*) *measure_zero_union*:
assumes $s \in$ *sets M* $t \in$ *sets M measure M t* = *0*
shows *measure M* $(s \cup t)$ = *measure M s*
using *assms*
proof −
have *measure M* $(s \cup t) \leq$ *measure M s*
using *finite_measure_subadditive*[*of s t*] *assms* **by** *auto*
moreover have *measure M* $(s \cup t) \geq$ *measure M s*
using *assms* **by** (*blast intro*: *finite_measure_mono*)
ultimately show *?thesis* **by** *simp*
qed

lemma (**in** *finite_measure*) *measure_eq_compl*:
assumes $s \in$ *sets M* $t \in$ *sets M*
assumes *measure M* (*space M* − *s*) = *measure M* (*space M* − *t*)
shows *measure M s* = *measure M t*
using *assms finite_measure_compl* **by** *auto*

lemma (**in** *finite_measure*) *measure_eq_bigunion_image*:
assumes *range f* $\subseteq$ *sets M range g* $\subseteq$ *sets M*
assumes *disjoint_family f disjoint_family g*
assumes $\bigwedge$ *n* :: *nat*. *measure M* (*f n*) = *measure M* (*g n*)
shows *measure M* $(\bigcup i.\ f\ i)$ = *measure M* $(\bigcup i.\ g\ i)$
using *assms*
proof −
have *a*: $(\lambda\ i.$ *measure M* $(f\ i))$ *sums* (*measure M* $(\bigcup i.\ f\ i)$)
by (*rule finite_measure_UNION*[*OF assms*(*1*,*3*)])
have *b*: $(\lambda\ i.$ *measure M* $(g\ i))$ *sums* (*measure M* $(\bigcup i.\ g\ i)$)
by (*rule finite_measure_UNION*[*OF assms*(*2*,*4*)])
show *?thesis* **using** *sums_unique*[*OF b*] *sums_unique*[*OF a*] *assms* **by** *simp*
qed

lemma (**in** *finite_measure*) *measure_countably_zero*:
assumes *range c* $\subseteq$ *sets M*
assumes $\bigwedge$ *i*. *measure M* (*c i*) = *0*
shows *measure M* $(\bigcup i ::$ *nat*. *c i*$)$ = *0*
proof (*rule antisym*)
show *measure M* $(\bigcup i ::$ *nat*. *c i*$) \leq$ *0*
using *finite_measure_subadditive_countably*[*OF assms*(*1*)] **by** (*simp add*: *assms*(*2*))
qed *simp*

lemma (**in** *finite_measure*) *measure_space_inter*:
assumes *events*:$s \in$ *sets* M $t \in$ *sets* M
assumes *measure* M t = *measure* M (*space* M)
shows *measure* M ($s \cap t$) = *measure* M s
proof −
have *measure* M ((*space* M − s) ∪ (*space* M − t)) = *measure* M (*space* M − s)
using *events assms finite_measure_compl*[*of t*] **by** (*auto intro*!: *measure_zero_union*)
also have (*space* M − s) ∪ (*space* M − t) = *space* M − ($s \cap t$)
by *blast*
finally show *measure* M ($s \cap t$) = *measure* M s
using *events* **by** (*auto intro*!: *measure_eq_compl*[*of* $s \cap t$ s])
qed

lemma (**in** *finite_measure*) *measure_equiprobable_finite_unions*:
assumes s: *finite* s $\bigwedge x.\ x \in s \Longrightarrow \{x\} \in$ *sets* M
assumes $\bigwedge$ x y. $[\![x \in s; y \in s]\!] \Longrightarrow$ *measure* M $\{x\}$ = *measure* M $\{y\}$
shows *measure* M s = *real* (*card* s) ∗ *measure* M $\{SOME\ x.\ x \in s\}$
proof *cases*
assume $s \neq \{\}$
then have ∃ $x.\ x \in s$ **by** *blast*
from *someI_ex*[*OF this*] *assms*
have *prob_some*: $\bigwedge x.\ x \in s \Longrightarrow$ *measure* M $\{x\}$ = *measure* M $\{SOME\ y.\ y \in s\}$ **by** *blast*
have *measure* M s = ($\sum$ $x \in s$. *measure* M $\{x\}$)
using *finite_measure_eq_sum_singleton*[*OF s*] **by** *simp*
also have ... = ($\sum$ $x \in s$. *measure* M $\{SOME\ y.\ y \in s\}$) **using** *prob_some* **by** *auto*
also have ... = *real* (*card* s) ∗ *measure* M $\{(SOME\ x.\ x \in s)\}$
using *sum_constant assms* **by** *simp*
finally show *?thesis* **by** *simp*
qed *simp*

lemma (**in** *finite_measure*) *measure_real_sum_image_fn*:
assumes $e \in$ *sets* M
assumes $\bigwedge x.\ x \in s \Longrightarrow e \cap f\ x \in$ *sets* M
assumes *finite* s
assumes *disjoint*: $\bigwedge$ x y. $[\![x \in s\ ;\ y \in s\ ;\ x \neq y]\!] \Longrightarrow f\ x \cap f\ y = \{\}$
assumes *upper*: *space* $M \subseteq (\bigcup i \in s.\ f\ i)$
shows *measure* M e = ($\sum$ $x \in s$. *measure* M ($e \cap f\ x$))
proof −
have $e \subseteq (\bigcup i \in s.\ f\ i)$
using ‹$e \in$ *sets* M› *sets.sets_into_space upper* **by** *blast*
then have *e*: $e = (\bigcup i \in s.\ e \cap f\ i)$
by *auto*
hence *measure* M e = *measure* M $(\bigcup i \in s.\ e \cap f\ i)$ **by** *simp*
also have ... = ($\sum$ $x \in s$. *measure* M ($e \cap f\ x$))
proof (*rule finite_measure_finite_Union*)
show *finite* s **by** *fact*

```
    show (λi. e ∩ f i)`s ⊆ sets M using assms(2) by auto
    show disjoint_family_on (λi. e ∩ f i) s
      using disjoint by (auto simp: disjoint_family_on_def)
  qed
  finally show ?thesis .
qed

lemma (in finite_measure) measure_exclude:
  assumes A ∈ sets M B ∈ sets M
  assumes measure M A = measure M (space M) A ∩ B = {}
  shows measure M B = 0
  using measure_space_inter[of B A] assms by (auto simp: ac_simps)
lemma (in finite_measure) finite_measure_distr:
  assumes f: f ∈ measurable M M'
  shows finite_measure (distr M M' f)
proof (rule finite_measureI)
   have f -` space M' ∩ space M = space M using f by (auto dest: measurable_space)
  with f show emeasure (distr M M' f) (space (distr M M' f)) ≠ ∞ by (auto simp: emeasure_distr)
qed

lemma emeasure_gfp[consumes 1, case_names cont measurable]:
  assumes sets[simp]: ⋀s. sets (M s) = sets N
  assumes ⋀s. finite_measure (M s)
  assumes cont: inf_continuous F inf_continuous f
  assumes meas: ⋀P. Measurable.pred N P ⟹ Measurable.pred N (F P)
  assumes iter: ⋀P s. Measurable.pred N P ⟹ emeasure (M s) {x∈space N. F P x} = f (λs. emeasure (M s) {x∈space N. P x}) s
  assumes bound: ⋀P. f P ≤ f (λs. emeasure (M s) (space (M s)))
  shows emeasure (M s) {x∈space N. gfp F x} = gfp f s
proof (subst gfp_transfer_bounded[where α=λF s. emeasure (M s) {x∈space N. F x} and P=Measurable.pred N, symmetric])
  interpret finite_measure M s for s by fact
  fix C assume decseq C ⋀i. Measurable.pred N (C i)
  then show (λs. emeasure (M s) {x ∈ space N. (INF i. C i) x}) = (INF i. (λs. emeasure (M s) {x ∈ space N. C i x}))
    unfolding INF_apply
    by (subst INF_emeasure_decseq) (auto simp: antimono_def fun_eq_iff intro!: arg_cong2[where f=emeasure])
next
  show f x ≤ (λs. emeasure (M s) {x ∈ space N. F top x}) for x
    using bound[of x] sets_eq_imp_space_eq[OF sets] by (simp add: iter)
qed (auto simp add: iter le_fun_def INF_apply[abs_def] intro!: meas cont)
```

6.3.13 Counting space

```
lemma strict_monoI_Suc:
  assumes (⋀n. f n < f (Suc n)) shows strict_mono f
```

by (*simp add*: *assms strict_mono_Suc_iff*)

lemma *emeasure_count_space*:
assumes $X \subseteq A$ **shows** *emeasure* (*count_space A*) X = (*if finite X then of_nat* (*card X*) *else* ∞)
(**is** _ = *?M X*)
unfolding *count_space_def*
proof (*rule emeasure_measure_of_sigma*)
show $X \in$ *Pow A* **using** ‹$X \subseteq A$› **by** *auto*
show *sigma_algebra A* (*Pow A*) **by** (*rule sigma_algebra_Pow*)
show *positive*: *positive* (*Pow A*) *?M*
by (*auto simp*: *positive_def*)
have *additive*: *additive* (*Pow A*) *?M*
by (*auto simp*: *card_Un_disjoint additive_def*)

interpret *ring_of_sets A Pow A*
by (*rule ring_of_setsI*) *auto*
show *countably_additive* (*Pow A*) *?M*
unfolding *countably_additive_iff_continuous_from_below*[*OF positive additive*]
proof *safe*
fix *F* :: *nat* ⇒ *'a set* **assume** *incseq F*
show (λ*i*. *?M* (*F i*)) ⟶ *?M* (⋃ *i*. *F i*)
proof *cases*
assume ∃ *i*. ∀ *j*≥*i*. *F i* = *F j*
then obtain *i* **where** *i*: ∀ *j*≥*i*. *F i* = *F j* **..**
with ‹*incseq F*› **have** *F j* ⊆ *F i* **for** *j*
by (*cases i* ≤ *j*) (*auto simp*: *incseq_def*)
then have *eq*: (⋃ *i*. *F i*) = *F i*
by *auto*
with *i* **show** *?thesis*
by (*auto intro*!: *Lim_transform_eventually*[*OF tendsto_const*] *eventually_sequentiallyI*[**where** *c*=*i*])
next
assume ¬ (∃ *i*. ∀ *j*≥*i*. *F i* = *F j*)
then obtain *f* **where** *f*: ⋀*i*. *i* ≤ *f i* ⋀*i*. *F i* ≠ *F* (*f i*) **by** *metis*
then have ⋀*i*. *F i* ⊆ *F* (*f i*) **using** ‹*incseq F*› **by** (*auto simp*: *incseq_def*)
with *f* **have** ∗: ⋀*i*. *F i* ⊂ *F* (*f i*) **by** *auto*

have *incseq* (λ*i*. *?M* (*F i*))
using ‹*incseq F*› **unfolding** *incseq_def* **by** (*auto simp*: *card_mono dest*: *finite_subset*)
then have (λ*i*. *?M* (*F i*)) ⟶ (*SUP n*. *?M* (*F n*))
by (*rule LIMSEQ_SUP*)

moreover have (*SUP n*. *?M* (*F n*)) = *top*
proof (*rule ennreal_SUP_eq_top*)
fix *n* :: *nat* **show** ∃ *k*::*nat*∈*UNIV*. *of_nat n* ≤ *?M* (*F k*)
proof (*induct n*)

```
        case (Suc n)
        then obtain k where of_nat n ≤ ?M (F k) ..
        moreover have finite (F k) ⟹ finite (F (f k)) ⟹ card (F k) < card (F
(f k))
          using ‹F k ⊂ F (f k)› by (simp add: psubset_card_mono)
        moreover have finite (F (f k)) ⟹ finite (F k)
         using ‹k ≤ f k› ‹incseq F› by (auto simp: incseq_def dest: finite_subset)
        ultimately show ?case
          by (auto intro!: exI[of _ f k] simp del: of_nat_Suc)
      qed auto
    qed

    moreover
    have inj (λn. F ((f ^^ n) 0))
      by (intro strict_mono_imp_inj_on strict_monoI_Suc) (simp add: ∗)
    then have 1: infinite (range (λi. F ((f ^^ i) 0)))
      by (rule range_inj_infinite)
    have infinite (Pow (⋃i. F i))
      by (rule infinite_super[OF _ 1]) auto
    then have infinite (⋃i. F i)
      by auto
    ultimately show ?thesis by (simp only:) simp

  qed
 qed
qed

lemma distr_bij_count_space:
 assumes f: bij_betw f A B
 shows distr (count_space A) (count_space B) f = count_space B
proof (rule measure_eqI)
 have f': f ∈ measurable (count_space A) (count_space B)
   using f unfolding Pi_def bij_betw_def by auto
 fix X assume X ∈ sets (distr (count_space A) (count_space B) f)
 then have X: X ∈ sets (count_space B) by auto
 moreover from X have f -` X ∩ A = the_inv_into A f ` X
  using f by (auto simp: bij_betw_def subset_image_iff image_iff the_inv_into_f_f
intro: the_inv_into_f_f[symmetric])
 moreover have inj_on (the_inv_into A f) B
   using X f by (auto simp: bij_betw_def inj_on_the_inv_into)
 with X have inj_on (the_inv_into A f) X
   by (auto intro: subset_inj_on)
  ultimately show emeasure (distr (count_space A) (count_space B) f) X =
emeasure (count_space B) X
   using f unfolding emeasure_distr[OF f' X]
    by (subst (1 2) emeasure_count_space) (auto simp: card_image dest: fi-
nite_imageD)
qed simp
```

lemma *emeasure_count_space_finite*[*simp*]:
X ⊆ *A* ⟹ *finite X* ⟹ *emeasure* (*count_space A*) *X* = *of_nat* (*card X*)
using *emeasure_count_space*[*of X A*] **by** *simp*

lemma *emeasure_count_space_infinite*[*simp*]:
X ⊆ *A* ⟹ *infinite X* ⟹ *emeasure* (*count_space A*) *X* = ∞
using *emeasure_count_space*[*of X A*] **by** *simp*

lemma *measure_count_space*: *measure* (*count_space A*) *X* = (*if X* ⊆ *A then of_nat* (*card X*) *else 0*)
by (*cases finite X*) (*auto simp*: *measure_notin_sets ennreal_of_nat_eq_real_of_nat measure_zero_top measure_eq_emeasure_eq_ennreal*)

lemma *emeasure_count_space_eq_0*:
emeasure (*count_space A*) *X* = *0* ⟷ (*X* ⊆ *A* ⟶ *X* = {})
proof *cases*
assume *X*: *X* ⊆ *A*
then show *?thesis*
proof (*intro iffI impI*)
assume *emeasure* (*count_space A*) *X* = *0*
with *X* **show** *X* = {}
by (*subst* (*asm*) *emeasure_count_space*) (*auto split*: *if_split_asm*)
qed *simp*
qed (*simp add*: *emeasure_notin_sets*)

lemma *null_sets_count_space*: *null_sets* (*count_space A*) = { {} }
unfolding *null_sets_def* **by** (*auto simp add*: *emeasure_count_space_eq_0*)

lemma *AE_count_space*: (*AE x in count_space A*. *P x*) ⟷ (∀ *x*∈*A*. *P x*)
unfolding *eventually_ae_filter* **by** (*auto simp add*: *null_sets_count_space*)

lemma *sigma_finite_measure_count_space_countable*:
assumes *A*: *countable A*
shows *sigma_finite_measure* (*count_space A*)
proof qed (*insert A*, *auto intro*!: *exI*[*of* _ (λ*a*. {*a*}) ‘ *A*])

lemma *sigma_finite_measure_count_space*:
fixes *A* :: *'a*::*countable set* **shows** *sigma_finite_measure* (*count_space A*)
by (*rule sigma_finite_measure_count_space_countable*) *auto*

lemma *finite_measure_count_space*:
assumes [*simp*]: *finite A*
shows *finite_measure* (*count_space A*)
by *rule simp*

lemma *sigma_finite_measure_count_space_finite*:
assumes *A*: *finite A* **shows** *sigma_finite_measure* (*count_space A*)
proof −
interpret *finite_measure count_space A* **using** *A* **by** (*rule finite_measure_count_space*)

show *sigma_finite_measure* (*count_space A*) **..**
qed

6.3.14 Measure restricted to space

lemma *emeasure_restrict_space*:
assumes $\Omega \cap$ *space M* $\in$ *sets M A* $\subseteq \Omega$
shows *emeasure* (*restrict_space M* Ω) *A* = *emeasure M A*
proof (*cases A* $\in$ *sets M*)
case *True*
show *?thesis*
proof (*rule emeasure_measure_of*[*OF restrict_space_def*])
show ($\cap$) Ω ' *sets M* $\subseteq$ *Pow* ($\Omega \cap$ *space M*) *A* $\in$ *sets* (*restrict_space M* Ω)
using ‹*A* $\subseteq \Omega$› ‹*A* $\in$ *sets M*› *sets.space_closed* **by** (*auto simp*: *sets_restrict_space*)
show *positive* (*sets* (*restrict_space M* Ω)) (*emeasure M*)
by (*auto simp*: *positive_def*)
show *countably_additive* (*sets* (*restrict_space M* Ω)) (*emeasure M*)
proof (*rule countably_additiveI*)
fix *A* :: *nat* $\Rightarrow$ _ **assume** *range A* $\subseteq$ *sets* (*restrict_space M* Ω) *disjoint_family A*
with *assms* **have** $\bigwedge i.$ *A i* $\in$ *sets M* $\bigwedge i.$ *A i* $\subseteq$ *space M disjoint_family A*
by (*fastforce simp*: *sets_restrict_space_iff*[*OF assms*(*1*)] *image_subset_iff*
dest: *sets.sets_into_space*)+
then show ($\sum i.$ *emeasure M* (*A i*)) = *emeasure M* ($\bigcup i.$ *A i*)
by (*subst suminf_emeasure*) (*auto simp*: *disjoint_family_subset*)
qed
qed
next
case *False*
with *assms* **have** *A* $\notin$ *sets* (*restrict_space M* Ω)
by (*simp add*: *sets_restrict_space_iff*)
with *False* **show** *?thesis*
by (*simp add*: *emeasure_notin_sets*)
qed

lemma *measure_restrict_space*:
assumes $\Omega \cap$ *space M* $\in$ *sets M A* $\subseteq \Omega$
shows *measure* (*restrict_space M* Ω) *A* = *measure M A*
using *emeasure_restrict_space*[*OF assms*] **by** (*simp add*: *measure_def*)

lemma *AE_restrict_space_iff*:
assumes $\Omega \cap$ *space M* $\in$ *sets M*
shows (*AE x in restrict_space M* Ω. *P x*) $\longleftrightarrow$ (*AE x in M*. *x* $\in \Omega \longrightarrow$ *P x*)
proof −
have *ex_cong*: $\bigwedge P\ Q\ f.\ (\bigwedge x.\ P\ x \Longrightarrow Q\ x) \Longrightarrow (\bigwedge x.\ Q\ x \Longrightarrow P\ (f\ x)) \Longrightarrow (\exists\, x.\ P\ x) \longleftrightarrow (\exists\, x.\ Q\ x)$
by *auto*
{ **fix** *X* **assume** *X*: *X* $\in$ *sets M emeasure M X* = *0*
then have *emeasure M* ($\Omega \cap$ *space M* $\cap$ *X*) $\leq$ *emeasure M X*

```
    by (intro emeasure_mono) auto
  then have emeasure M (Ω ∩ space M ∩ X) = 0
    using X by (auto intro!: antisym) }
 with assms show ?thesis
  unfolding eventually_ae_filter
  by (auto simp add: space_restrict_space null_sets_def sets_restrict_space_iff
                     emeasure_restrict_space cong: conj_cong
           intro!: ex_cong[where f=λX. (Ω ∩ space M) ∩ X])
qed

lemma restrict_restrict_space:
 assumes A ∩ space M ∈ sets M B ∩ space M ∈ sets M
 shows restrict_space (restrict_space M A) B = restrict_space M (A ∩ B) (is ?l
= ?r)
proof (rule measure_eqI[symmetric])
 show sets ?r = sets ?l
  unfolding sets_restrict_space image_comp by (intro image_cong) auto
next
 fix X assume X ∈ sets (restrict_space M (A ∩ B))
 then obtain Y where Y ∈ sets M X = Y ∩ A ∩ B
  by (auto simp: sets_restrict_space)
 with assms sets.Int[OF assms] show emeasure ?r X = emeasure ?l X
  by (subst (1 2) emeasure_restrict_space)
   (auto simp: space_restrict_space sets_restrict_space_iff emeasure_restrict_space
ac_simps)
qed

lemma restrict_count_space: restrict_space (count_space B) A = count_space
(A ∩ B)
proof (rule measure_eqI)
 show sets (restrict_space (count_space B) A) = sets (count_space (A ∩ B))
  by (subst sets_restrict_space) auto
 moreover fix X assume X ∈ sets (restrict_space (count_space B) A)
 ultimately have X ⊆ A ∩ B by auto
 then show emeasure (restrict_space (count_space B) A) X = emeasure (count_space
(A ∩ B)) X
  by (cases finite X) (auto simp add: emeasure_restrict_space)
qed

lemma sigma_finite_measure_restrict_space:
 assumes sigma_finite_measure M
 and A: A ∈ sets M
 shows sigma_finite_measure (restrict_space M A)
proof −
 interpret sigma_finite_measure M by fact
 from sigma_finite_countable obtain C
  where C: countable C C ⊆ sets M (⋃ C) = space M ∀ a∈C. emeasure M a ≠
∞
  by blast
```

```
  let ?C = (∩) A ' C
  from C have countable ?C ?C ⊆ sets (restrict_space M A) (⋃ ?C) = space
(restrict_space M A)
    by(auto simp add: sets_restrict_space space_restrict_space)
  moreover {
    fix a
    assume a ∈ ?C
    then obtain a' where a = A ∩ a' a' ∈ C ..
    then have emeasure (restrict_space M A) a ≤ emeasure M a'
     using A C by(auto simp add: emeasure_restrict_space intro: emeasure_mono)
     also have ... < ∞ using C(4)[rule_format, of a'] ‹a' ∈ C› by (simp add:
less_top)
    finally have emeasure (restrict_space M A) a ≠ ∞ by simp }
  ultimately show ?thesis
    by unfold_locales (rule exI conjI|assumption|blast)+
qed

lemma finite_measure_restrict_space:
  assumes finite_measure M
  and A: A ∈ sets M
  shows finite_measure (restrict_space M A)
proof −
  interpret finite_measure M by fact
  show ?thesis
   by(rule finite_measureI)(simp add: emeasure_restrict_space A space_restrict_space)
qed

lemma restrict_distr:
  assumes [measurable]: f ∈ measurable M N
  assumes [simp]: Ω ∩ space N ∈ sets N and restrict: f ∈ space M → Ω
  shows restrict_space (distr M N f) Ω = distr M (restrict_space N Ω) f
  (is ?l = ?r)
proof (rule measure_eqI)
  fix A assume A ∈ sets (restrict_space (distr M N f) Ω)
  with restrict show emeasure ?l A = emeasure ?r A
    by (subst emeasure_distr)
       (auto simp: sets_restrict_space_iff emeasure_restrict_space emeasure_distr
             intro!: measurable_restrict_space2)
qed (simp add: sets_restrict_space)

lemma measure_eqI_restrict_generator:
 assumes E: Int_stable E E ⊆ Pow Ω ⋀X. X ∈ E ⟹ emeasure M X = emeasure
N X
  assumes sets_eq: sets M = sets N and Ω: Ω ∈ sets M
  assumes sets (restrict_space M Ω) = sigma_sets Ω E
  assumes sets (restrict_space N Ω) = sigma_sets Ω E
  assumes ae: AE x in M. x ∈ Ω AE x in N. x ∈ Ω
  assumes A: countable A A ≠ {} A ⊆ E ⋃A = Ω ⋀a. a ∈ A ⟹ emeasure M
a ≠ ∞
```

shows $M = N$
proof (*rule measure_eqI*)
fix X **assume** X: $X \in$ *sets* M
then have *emeasure* M X = *emeasure* (*restrict_space* M Ω) ($X \cap \Omega$)
using *ae* Ω **by** (*auto simp add*: *emeasure_restrict_space intro*!: *emeasure_eq_AE*)
also have *restrict_space* M Ω = *restrict_space* N Ω
proof (*rule measure_eqI_generator_eq*)
fix X **assume** $X \in E$
then show *emeasure* (*restrict_space* M Ω) X = *emeasure* (*restrict_space* N Ω) X
using E Ω **by** (*subst* (*1 2*) *emeasure_restrict_space*) (*auto simp*: *sets_eq sets_eq*[*THEN sets_eq_imp_space_eq*])
next
show *range* (*from_nat_into* A) $\subseteq$ E ($\bigcup i$. *from_nat_into* A i) = Ω
using A **by** (*auto cong del*: *SUP_cong_simp*)
next
fix i
have *emeasure* (*restrict_space* M Ω) (*from_nat_into* A i) = *emeasure* M (*from_nat_into* A i)
using A Ω **by** (*subst emeasure_restrict_space*)
(*auto simp*: *sets_eq sets_eq*[*THEN sets_eq_imp_space_eq*] *intro*: *from_nat_into*)
with A **show** *emeasure* (*restrict_space* M Ω) (*from_nat_into* A i) $\neq \infty$
by (*auto intro*: *from_nat_into*)
qed *fact*+
also have *emeasure* (*restrict_space* N Ω) ($X \cap \Omega$) = *emeasure* N X
using X *ae* Ω **by** (*auto simp add*: *emeasure_restrict_space sets_eq intro*!: *emeasure_eq_AE*)
finally show *emeasure* M X = *emeasure* N X **.**
qed *fact*

6.3.15 Null measure

definition *null_measure* :: *'a measure* ⇒ *'a measure* **where**
null_measure M = *sigma* (*space* M) (*sets* M)

lemma *space_null_measure*[*simp*]: *space* (*null_measure* M) = *space* M
by (*simp add*: *null_measure_def*)

lemma *sets_null_measure*[*simp*, *measurable_cong*]: *sets* (*null_measure* M) = *sets* M
by (*simp add*: *null_measure_def*)

lemma *emeasure_null_measure*[*simp*]: *emeasure* (*null_measure* M) X = *0*
by (*cases* $X \in$ *sets* M, *rule emeasure_measure_of*)
(*auto simp*: *positive_def countably_additive_def emeasure_notin_sets null_measure_def*
dest: *sets.sets_into_space*)

lemma *measure_null_measure*[*simp*]: *measure* (*null_measure* M) X = *0*

by (*intro measure_eq_emeasure_eq_ennreal*) *auto*

lemma *null_measure_idem* [*simp*]: *null_measure* (*null_measure M*) = *null_measure M*
by(*rule measure_eqI*) *simp_all*

6.3.16 Scaling a measure

definition *scale_measure* :: *ennreal* ⇒ *'a measure* ⇒ *'a measure* **where**
scale_measure r M = *measure_of* (*space M*) (*sets M*) (λ*A*. *r* ∗ *emeasure M A*)

lemma *space_scale_measure*: *space* (*scale_measure r M*) = *space M*
by (*simp add*: *scale_measure_def*)

lemma *sets_scale_measure* [*simp*, *measurable_cong*]: *sets* (*scale_measure r M*) = *sets M*
by (*simp add*: *scale_measure_def*)

lemma *emeasure_scale_measure* [*simp*]:
emeasure (*scale_measure r M*) *A* = *r* ∗ *emeasure M A*
(**is** _ = *?μ A*)
proof(*cases A* ∈ *sets M*)
case *True*
show *?thesis* **unfolding** *scale_measure_def*
proof(*rule emeasure_measure_of_sigma*)
show *sigma_algebra* (*space M*) (*sets M*) **..**
show *positive* (*sets M*) *?μ* **by** (*simp add*: *positive_def*)
show *countably_additive* (*sets M*) *?μ*
proof (*rule countably_additiveI*)
fix *A* :: *nat* ⇒ _ **assume** ∗: *range A* ⊆ *sets M disjoint_family A*
have (∑ *i*. *?μ* (*A i*)) = *r* ∗ (∑ *i*. *emeasure M* (*A i*))
by *simp*
also have ... = *?μ* (⋃ *i*. *A i*) **using** ∗ **by**(*simp add*: *suminf_emeasure*)
finally show (∑ *i*. *?μ* (*A i*)) = *?μ* (⋃ *i*. *A i*) **.**
qed
qed(*fact True*)
qed(*simp add*: *emeasure_notin_sets*)

lemma *scale_measure_1* [*simp*]: *scale_measure 1 M* = *M*
by(*rule measure_eqI*) *simp_all*

lemma *scale_measure_0*[*simp*]: *scale_measure 0 M* = *null_measure M*
by(*rule measure_eqI*) *simp_all*

lemma *measure_scale_measure* [*simp*]: *0* ≤ *r* ⟹ *measure* (*scale_measure r M*) *A* = *r* ∗ *measure M A*
using *emeasure_scale_measure*[*of r M A*]
emeasure_eq_ennreal_measure[*of M A*]
measure_eq_emeasure_eq_ennreal[*of* _ *scale_measure r M A*]

by (*cases emeasure* (*scale_measure r M*) *A* = *top*)
(*auto simp del*: *emeasure_scale_measure*
simp: *ennreal_top_eq_mult_iff ennreal_mult_eq_top_iff measure_zero_top ennreal_mult*[*symmetric*])

lemma *scale_scale_measure* [*simp*]:
scale_measure r (*scale_measure r′ M*) = *scale_measure* (*r* ∗ *r′*) *M*
by (*rule measure_eqI*) (*simp_all add*: *max_def mult.assoc*)

lemma *scale_null_measure* [*simp*]: *scale_measure r* (*null_measure M*) = *null_measure M*
by (*rule measure_eqI*) *simp_all*

6.3.17 Complete lattice structure on measures

lemma (**in** *finite_measure*) *finite_measure_Diff′*:
A ∈ *sets M* ⟹ *B* ∈ *sets M* ⟹ *measure M* (*A* − *B*) = *measure M A* − *measure M* (*A* ∩ *B*)
using *finite_measure_Diff*[*of A A* ∩ *B*] **by** (*auto simp*: *Diff_Int*)

lemma (**in** *finite_measure*) *finite_measure_Union′*:
A ∈ *sets M* ⟹ *B* ∈ *sets M* ⟹ *measure M* (*A* ∪ *B*) = *measure M A* + *measure M* (*B* − *A*)
using *finite_measure_Union*[*of A B* − *A*] **by** *auto*

lemma *finite_unsigned_Hahn_decomposition*:
assumes *finite_measure M finite_measure N* **and** [*simp*]: *sets N* = *sets M*
shows ∃ *Y*∈*sets M*. (∀ *X*∈*sets M*. *X* ⊆ *Y* ⟶ *N X* ≤ *M X*) ∧ (∀ *X*∈*sets M*. *X* ∩ *Y* = {} ⟶ *M X* ≤ *N X*)
proof −
interpret *M*: *finite_measure M* **by** *fact*
interpret *N*: *finite_measure N* **by** *fact*

define *d* **where** *d X* = *measure M X* − *measure N X* **for** *X*

have [*intro*]: *bdd_above* (*d*‘*sets M*)
using *sets.sets_into_space*[*of _ M*]
by (*intro bdd_aboveI*[**where** *M*=*measure M* (*space M*)])
(*auto simp*: *d_def field_simps subset_eq intro*!: *add_increasing M.finite_measure_mono*)

define γ **where** γ = (*SUP X*∈*sets M*. *d X*)
have *le_γ*[*intro*]: *X* ∈ *sets M* ⟹ *d X* ≤ γ **for** *X*
by (*auto simp*: γ*_def intro*!: *cSUP_upper*)

have ∃*f*. ∀ *n*. *f n* ∈ *sets M* ∧ *d* (*f n*) > γ − *1* / *2*^*n*
proof (*intro choice_iff*[*THEN iffD1*] *allI*)
fix *n*
have ∃ *X*∈*sets M*. γ − *1* / *2*^*n* < *d X*
unfolding γ*_def* **by** (*intro less_cSUP_iff*[*THEN iffD1*]) *auto*

```
    then show ∃y. y ∈ sets M ∧ γ − 1 / 2 ^ n < d y
      by auto
  qed
  then obtain E where [measurable]: E n ∈ sets M and E: d (E n) > γ − 1 /
2^n for n
    by auto

  define F where F m n = (if m ≤ n then ⋂i∈{m..n}. E i else space M) for m
n

  have [measurable]: m ≤ n ⟹ F m n ∈ sets M for m n
    by (auto simp: F_def)

  have 1: γ − 2 / 2 ^ m + 1 / 2 ^ n ≤ d (F m n) if m ≤ n for m n
    using that
  proof (induct rule: dec_induct)
    case base with E[of m] show ?case
      by (simp add: F_def field_simps)
  next
    case (step i)
    have F_Suc: F m (Suc i) = F m i ∩ E (Suc i)
      using ‹m ≤ i› by (auto simp: F_def le_Suc_eq)

    have γ + (γ − 2 / 2^m + 1 / 2 ^ Suc i) ≤ (γ − 1 / 2^Suc i) + (γ − 2 /
2^m + 1 / 2^i)
      by (simp add: field_simps)
    also have ... ≤ d (E (Suc i)) + d (F m i)
      using E[of Suc i] by (intro add_mono step) auto
    also have ... = d (E (Suc i)) + d (F m i − E (Suc i)) + d (F m (Suc i))
    using ‹m ≤ i› by (simp add: d_def field_simps F_Suc M.finite_measure_Diff'
N.finite_measure_Diff')
    also have ... = d (E (Suc i) ∪ F m i) + d (F m (Suc i))
      using ‹m ≤ i› by (simp add: d_def field_simps M.finite_measure_Union'
N.finite_measure_Union')
    also have ... ≤ γ + d (F m (Suc i))
      using ‹m ≤ i› by auto
    finally show ?case
      by auto
  qed

  define F' where F' m = (⋂i∈{m..}. E i) for m
  have F'_eq: F' m = (⋂i. F m (i + m)) for m
    by (fastforce simp: le_iff_add[of m] F'_def F_def)

  have [measurable]: F' m ∈ sets M for m
    by (auto simp: F'_def)

  have γ_le: γ − 0 ≤ d (⋃m. F' m)
  proof (rule LIMSEQ_le)
```

```
    show (λn. γ − 2 / 2 ^ n) ⟶ γ − 0
      by (intro tendsto_intros LIMSEQ_divide_realpow_zero) auto
    have incseq F'
      by (auto simp: incseq_def F'_def)
    then show (λm. d (F' m)) ⟶ d (⋃ m. F' m)
      unfolding d_def
    by (intro tendsto_diff M.finite_Lim_measure_incseq N.finite_Lim_measure_incseq)
auto

    have γ − 2 / 2 ^ m + 0 ≤ d (F' m) for m
    proof (rule LIMSEQ_le)
      have ∗: decseq (λn. F m (n + m))
        by (auto simp: decseq_def F_def)
      show (λn. d (F m n)) ⟶ d (F' m)
        unfolding d_def F'_eq
        by (rule LIMSEQ_offset[where k=m])
        (auto intro!: tendsto_diff M.finite_Lim_measure_decseq N.finite_Lim_measure_decseq
∗)
      show (λn. γ − 2 / 2 ^ m + 1 / 2 ^ n) ⟶ γ − 2 / 2 ^ m + 0
        by (intro tendsto_add LIMSEQ_divide_realpow_zero tendsto_const) auto
      show ∃ N. ∀ n≥N. γ − 2 / 2 ^ m + 1 / 2 ^ n ≤ d (F m n)
        using 1[of m] by (intro exI[of _ m]) auto
    qed
    then show ∃ N. ∀ n≥N. γ − 2 / 2 ^ n ≤ d (F' n)
      by auto
  qed

  show ?thesis
  proof (safe intro!: bexI[of _ ⋃ m. F' m])
    fix X assume [measurable]: X ∈ sets M and X: X ⊆ (⋃ m. F' m)
    have d (⋃ m. F' m) − d X = d ((⋃ m. F' m) − X)
     using X by (auto simp: d_def M.finite_measure_Diff N.finite_measure_Diff)
    also have ... ≤ γ
      by auto
    finally have 0 ≤ d X
      using γ_le by auto
    then show emeasure N X ≤ emeasure M X
      by (auto simp: d_def M.emeasure_eq_measure N.emeasure_eq_measure)
  next
    fix X assume [measurable]: X ∈ sets M and X: X ∩ (⋃ m. F' m) = {}
    then have d (⋃ m. F' m) + d X = d (X ∪ (⋃ m. F' m))
      by (auto simp: d_def M.finite_measure_Union N.finite_measure_Union)
    also have ... ≤ γ
      by auto
    finally have d X ≤ 0
      using γ_le by auto
    then show emeasure M X ≤ emeasure N X
      by (auto simp: d_def M.emeasure_eq_measure N.emeasure_eq_measure)
  qed auto
```

qed

proposition *unsigned_Hahn_decomposition*:
assumes [*simp*]: *sets N = sets M* **and** [*measurable*]: *A ∈ sets M*
and [*simp*]: *emeasure M A ≠ top emeasure N A ≠ top*
shows ∃ *Y*∈*sets M*. *Y ⊆ A* ∧ (∀ *X*∈*sets M*. *X ⊆ Y ⟶ N X ≤ M X*) ∧ (∀ *X*∈*sets M*. *X ⊆ A ⟶ X ∩ Y = {} ⟶ M X ≤ N X*)
proof −
have ∃ *Y*∈*sets* (*restrict_space M A*).
(∀ *X*∈*sets* (*restrict_space M A*). *X ⊆ Y ⟶* (*restrict_space N A*) *X ≤* (*restrict_space M A*) *X*) ∧
(∀ *X*∈*sets* (*restrict_space M A*). *X ∩ Y = {} ⟶* (*restrict_space M A*) *X ≤* (*restrict_space N A*) *X*)
proof (*rule finite_unsigned_Hahn_decomposition*)
show *finite_measure* (*restrict_space M A*) *finite_measure* (*restrict_space N A*)
by (*auto simp*: *space_restrict_space emeasure_restrict_space less_top intro*!: *finite_measureI*)
qed (*simp add*: *sets_restrict_space*)
with *assms* **show** *?thesis*
by (*metis Int_subset_iff emeasure_restrict_space sets.Int_space_eq2 sets_restrict_space_iff space_restrict_space*)
qed

Define a lexicographical order on *measure*, in the order space, sets and measure. The parts of the lexicographical order are point-wise ordered.

instantiation *measure* :: (*type*) *order_bot*
begin

inductive *less_eq_measure* :: *'a measure ⇒ 'a measure ⇒ bool* **where**
space M ⊂ space N ⟹ less_eq_measure M N
| *space M = space N ⟹ sets M ⊂ sets N ⟹ less_eq_measure M N*
| *space M = space N ⟹ sets M = sets N ⟹ emeasure M ≤ emeasure N ⟹ less_eq_measure M N*

lemma *le_measure_iff*:
M ≤ N ⟷ (*if space M = space N then*
if sets M = sets N then emeasure M ≤ emeasure N else sets M ⊆ sets N else space M ⊆ space N)
by (*auto elim*: *less_eq_measure.cases intro*: *less_eq_measure.intros*)

definition *less_measure* :: *'a measure ⇒ 'a measure ⇒ bool* **where**
less_measure M N ⟷ (*M ≤ N* ∧ ¬ *N ≤ M*)

definition *bot_measure* :: *'a measure* **where**
bot_measure = sigma {} {}

lemma
shows *space_bot*[*simp*]: *space bot = {}*
and *sets_bot*[*simp*]: *sets bot = {{}}*

and *emeasure_bot*[*simp*]: *emeasure bot X = 0*
by (*auto simp*: *bot_measure_def sigma_sets_empty_eq emeasure_sigma*)

instance
proof *standard*
show *bot* ≤ *a* **for** *a* :: *'a measure*
by (*simp add*: *le_measure_iff bot_measure_def sigma_sets_empty_eq emeasure_sigma le_fun_def*)
qed (*auto simp*: *le_measure_iff less_measure_def split*: *if_split_asm intro*: *measure_eqI*)

end

proposition *le_measure*: *sets M = sets N* ⟹ *M* ≤ *N* ⟷ (∀ *A*∈*sets M*. *emeasure M A* ≤ *emeasure N A*)
by (*metis emeasure_neq_0_sets le_fun_def le_measure_iff order_class.order_eq_iff sets_eq_imp_space_eq*)

definition *sup_measure'* :: *'a measure* ⇒ *'a measure* ⇒ *'a measure* **where**
sup_measure' A B =
measure_of (*space A*) (*sets A*)
(λ*X*. *SUP Y*∈*sets A*. *emeasure A* (*X* ∩ *Y*) + *emeasure B* (*X* ∩ − *Y*))

lemma assumes [*simp*]: *sets B = sets A*
shows *space_sup_measure'*[*simp*]: *space* (*sup_measure' A B*) = *space A*
and *sets_sup_measure'*[*simp*]: *sets* (*sup_measure' A B*) = *sets A*
using *sets_eq_imp_space_eq*[*OF assms*] **by** (*simp_all add*: *sup_measure'_def*)

lemma *emeasure_sup_measure'*:
assumes *sets_eq*[*simp*]: *sets B = sets A* **and** [*simp*, *intro*]: *X* ∈ *sets A*
shows *emeasure* (*sup_measure' A B*) *X* = (*SUP Y*∈*sets A*. *emeasure A* (*X* ∩ *Y*) + *emeasure B* (*X* ∩ − *Y*))
(**is** _ = *?S X*)
proof −
note *sets_eq_imp_space_eq*[*OF sets_eq*, *simp*]
show *?thesis*
using *sup_measure'_def*
proof (*rule emeasure_measure_of*)
let *?d* = λ*X Y*. *emeasure A* (*X* ∩ *Y*) + *emeasure B* (*X* ∩ − *Y*)
show *countably_additive* (*sets* (*sup_measure' A B*)) (λ*X*. *SUP Y* ∈ *sets A*. *emeasure A* (*X* ∩ *Y*) + *emeasure B* (*X* ∩ − *Y*))
proof (*rule countably_additiveI*, *goal_cases*)
case (*1 X*)
then have [*measurable*]: ⋀*i*. *X i* ∈ *sets A* **and** *disjoint_family X*
by *auto*
have *disjoint*: *disjoint_family* (λ*i*. *X i* ∩ *Y*) *disjoint_family* (λ*i*. *X i* − *Y*)
for *Y*
using *1*(*2*) *disjoint_family_subset* **by** *fastforce*+
have (∑ *i*. *?S* (*X i*)) = (*SUP Y*∈*sets A*. ∑ *i*. *?d* (*X i*) *Y*)

proof (*rule ennreal_suminf_SUP_eq_directed*)
fix J :: *nat set* **and** a b **assume** *finite* J **and** [*measurable*]: $a \in$ *sets* A $b \in$ *sets* A
have $\exists\, c{\in}$*sets* A. $c \subseteq X\ i \wedge (\forall\, a{\in}$*sets* A. *?d* $(X\ i)\ a \leq$ *?d* $(X\ i)\ c)$ **for** i
proof *cases*
assume *emeasure* $A\ (X\ i) =$ *top* $\vee$ *emeasure* $B\ (X\ i) =$ *top*
then show *?thesis*
by *force*
next
assume *finite*: $\neg$ (*emeasure* $A\ (X\ i) =$ *top* $\vee$ *emeasure* $B\ (X\ i) =$ *top*)
then have $\exists\, Y{\in}$*sets* A. $Y \subseteq X\ i \wedge (\forall\, C{\in}$*sets* A. $C \subseteq Y \longrightarrow B\ C \leq A\ C) \wedge (\forall\, C{\in}$*sets* A. $C \subseteq X\ i \longrightarrow C \cap Y = \{\} \longrightarrow A\ C \leq B\ C)$
using *unsigned_Hahn_decomposition*[*of* B A X i] **by** *simp*
then obtain Y **where** [*measurable*]: $Y \in$ *sets* A **and** [*simp*]: $Y \subseteq X\ i$
and *B_le_A*: $\bigwedge C.\ C \in$ *sets* $A \Longrightarrow C \subseteq Y \Longrightarrow B\ C \leq A\ C$
and *A_le_B*: $\bigwedge C.\ C \in$ *sets* $A \Longrightarrow C \subseteq X\ i \Longrightarrow C \cap Y = \{\} \Longrightarrow A\ C \leq B\ C$
by *auto*

show *?thesis*
proof (*intro bexI ballI conjI*)
fix a **assume** [*measurable*]: $a \in$ *sets* A
have $*$: $(X\ i \cap a \cap Y \cup (X\ i \cap a - Y)) = X\ i \cap a$ $(X\ i - a) \cap Y \cup (X\ i - a - Y) = X\ i \cap - a$
for a Y **by** *auto*
then have *?d* $(X\ i)\ a =$
$(A\ (X\ i \cap a \cap Y) + A\ (X\ i \cap a \cap - Y)) + (B\ (X\ i \cap - a \cap Y) + B\ (X\ i \cap - a \cap - Y))$
by (*subst* (*1 2*) *plus_emeasure*) (*auto simp*: *Diff_eq*[*symmetric*])
also have $\ldots \leq (A\ (X\ i \cap a \cap Y) + B\ (X\ i \cap a \cap - Y)) + (A\ (X\ i \cap - a \cap Y) + B\ (X\ i \cap - a \cap - Y))$
by (*intro add_mono order_refl B_le_A A_le_B*) (*auto simp*: *Diff_eq*[*symmetric*])
also have $\ldots \leq (A\ (X\ i \cap Y \cap a) + A\ (X\ i \cap Y \cap - a)) + (B\ (X\ i \cap - Y \cap a) + B\ (X\ i \cap - Y \cap - a))$
by (*simp add*: *ac_simps*)
also have $\ldots \leq A\ (X\ i \cap Y) + B\ (X\ i \cap - Y)$
by (*subst* (*1 2*) *plus_emeasure*) (*auto simp*: *Diff_eq*[*symmetric*] $*$)
finally show *?d* $(X\ i)\ a \leq$ *?d* $(X\ i)\ Y$ **.**
qed *auto*
qed
then obtain C **where** [*measurable*]: $C\ i \in$ *sets* A **and** $C\ i \subseteq X\ i$
and C: $\bigwedge a.\ a \in$ *sets* $A \Longrightarrow$ *?d* $(X\ i)\ a \leq$ *?d* $(X\ i)\ (C\ i)$ **for** i
by *metis*
have $*$: $X\ i \cap (\bigcup i.\ C\ i) = X\ i \cap C\ i$ **for** i
using ‹*disjoint_family X*› ‹$\bigwedge i.\ C\ i \subseteq X\ i$›
by (*simp add*: *disjoint_family_on_def disjoint_iff_not_equal set_eq_iff*) (*metis subsetD*)
then have $**$: $X\ i \cap - (\bigcup i.\ C\ i) = X\ i \cap - C\ i$ **for** i **by** *blast*

moreover have $(\bigcup i.\ C\ i) \in sets\ A$
by *fastforce*
ultimately show $\exists c{\in}sets\ A.\ \forall i{\in}J.\ ?d\ (X\ i)\ a \leq ?d\ (X\ i)\ c \wedge ?d\ (X\ i)\ b \leq ?d\ (X\ i)\ c$
by (*metis* $*$ *C* ‹$a \in sets\ A$› ‹$b \in sets\ A$›)
qed
also have $\ldots = ?S\ (\bigcup i.\ X\ i)$
proof −
have $\bigwedge Y.\ Y \in sets\ A \Longrightarrow (\sum i.\ emeasure\ A\ (X\ i \cap Y) + emeasure\ B\ (X\ i \cap -Y))$
$= emeasure\ A\ (\bigcup i.\ X\ i \cap Y) + emeasure\ B\ (\bigcup i.\ X\ i \cap -Y)$
using *disjoint*
by (*auto simp flip*: *suminf_add Diff_eq simp add*: *image_subset_iff suminf_emeasure*)
then show *?thesis* **by** *force*
qed
finally show $(\sum i.\ ?S\ (X\ i)) = ?S\ (\bigcup i.\ X\ i)$ **.**
qed
qed (*auto dest*: *sets.sets_into_space simp*: *positive_def intro*!: *SUP_const*)
qed

lemma *le_emeasure_sup_measure'1*:
assumes *sets B* = *sets A X* ∈ *sets A* **shows** *emeasure A X* ≤ *emeasure* (*sup_measure' A B*) *X*
by (*subst emeasure_sup_measure'*[*OF assms*]) (*auto intro*!: *SUP_upper2*[*of X*] *assms*)

lemma *le_emeasure_sup_measure'2*:
assumes *sets B* = *sets A X* ∈ *sets A* **shows** *emeasure B X* ≤ *emeasure* (*sup_measure' A B*) *X*
by (*subst emeasure_sup_measure'*[*OF assms*]) (*auto intro*!: *SUP_upper2*[*of* {}] *assms*)

lemma *emeasure_sup_measure'_le2*:
assumes [*simp*]: *sets B* = *sets C sets A* = *sets C* **and** [*measurable*]: *X* ∈ *sets C*
assumes *A*: $\bigwedge Y.\ Y \subseteq X \Longrightarrow Y \in sets\ A \Longrightarrow emeasure\ A\ Y \leq emeasure\ C\ Y$
assumes *B*: $\bigwedge Y.\ Y \subseteq X \Longrightarrow Y \in sets\ A \Longrightarrow emeasure\ B\ Y \leq emeasure\ C\ Y$
shows *emeasure* (*sup_measure' A B*) *X* ≤ *emeasure C X*
proof (*subst emeasure_sup_measure'*)
show $(SUP\ Y{\in}sets\ A.\ emeasure\ A\ (X \cap Y) + emeasure\ B\ (X \cap -\ Y)) \leq emeasure\ C\ X$
unfolding ‹*sets A* = *sets C*›
proof (*intro SUP_least*)
fix *Y* **assume** [*measurable*]: *Y* ∈ *sets C*
have [*simp*]: $X \cap Y \cup (X - Y) = X$
by *auto*
have *emeasure A* (*X* ∩ *Y*) + *emeasure B* (*X* ∩ − *Y*) ≤ *emeasure C* (*X* ∩ *Y*) + *emeasure C* (*X* ∩ − *Y*)

```
    by (intro add_mono A B) (auto simp: Diff_eq[symmetric])
  also have ... = emeasure C X
    by (subst plus_emeasure) (auto simp: Diff_eq[symmetric])
  finally show emeasure A (X ∩ Y) + emeasure B (X ∩ - Y) ≤ emeasure C
X .
 qed
qed simp_all

definition sup_lexord :: 'a ⇒ 'a ⇒ ('a ⇒ 'b::order) ⇒ 'a ⇒ 'a ⇒ 'a where
sup_lexord A B k s c =
 (if k A = k B then c else
  if ¬ k A ≤ k B ∧ ¬ k B ≤ k A then s else
  if k B ≤ k A then A else B)

lemma sup_lexord:
 (k A < k B ⟹ P B) ⟹ (k B < k A ⟹ P A) ⟹ (k A = k B ⟹ P c) ⟹
  (¬ k B ≤ k A ⟹ ¬ k A ≤ k B ⟹ P s) ⟹ P (sup_lexord A B k s c)
 by (auto simp: sup_lexord_def)

lemmas le_sup_lexord = sup_lexord[where P=λa. c ≤ a for c]

lemma sup_lexord1: k A = k B ⟹ sup_lexord A B k s c = c
 by (simp add: sup_lexord_def)

lemma sup_lexord_commute: sup_lexord A B k s c = sup_lexord B A k s c
 by (auto simp: sup_lexord_def)

lemma sigma_sets_le_sets_iff: (sigma_sets (space x) A ⊆ sets x) = (A ⊆ sets
x)
 using sets.sigma_sets_subset[of A x] by auto

lemma sigma_le_iff: A ⊆ Pow Ω ⟹ sigma Ω A ≤ x ⟷ (Ω ⊆ space x ∧ (space
x = Ω ⟶ A ⊆ sets x))
 by (cases Ω = space x)
     (simp_all add: eq_commute[of _ sets x] le_measure_iff emeasure_sigma
le_fun_def
                sigma_sets_superset_generator sigma_sets_le_sets_iff)

instantiation measure :: (type) semilattice_sup
begin

definition sup_measure :: 'a measure ⇒ 'a measure ⇒ 'a measure where
 sup_measure A B =
   sup_lexord A B space (sigma (space A ∪ space B) {})
     (sup_lexord A B sets (sigma (space A) (sets A ∪ sets B)) (sup_measure' A
B))

instance
proof
```

fix $x\ y\ z :: 'a\ measure$
show $x \leq sup\ x\ y$
unfolding *sup_measure_def*
proof (*intro le_sup_lexord*)
assume $space\ x = space\ y$
then have $*$: $sets\ x \cup sets\ y \subseteq Pow\ (space\ x)$
using *sets.space_closed* **by** *auto*
assume $\neg\ sets\ y \subseteq sets\ x \neg\ sets\ x \subseteq sets\ y$
then have $sets\ x \subset sets\ x \cup sets\ y$
by *auto*
also have $\ldots \leq sigma\ (space\ x)\ (sets\ x \cup sets\ y)$
by (*subst sets_measure_of*[*OF* $*$]) (*rule sigma_sets_superset_generator*)
finally show $x \leq sigma\ (space\ x)\ (sets\ x \cup sets\ y)$
by (*simp add*: *space_measure_of*[*OF* $*$] *less_eq_measure.intros*(*2*))
next
assume $\neg\ space\ y \subseteq space\ x \neg\ space\ x \subseteq space\ y$
then show $x \leq sigma\ (space\ x \cup space\ y)\ \{\}$
by (*intro less_eq_measure.intros*) *auto*
next
assume $sets\ x = sets\ y$ **then show** $x \leq sup\_measure'\ x\ y$
by (*simp add*: *le_measure le_emeasure_sup_measure'1*)
qed (*auto intro*: *less_eq_measure.intros*)
show $y \leq sup\ x\ y$
unfolding *sup_measure_def*
proof (*intro le_sup_lexord*)
assume $**$: $space\ x = space\ y$
then have $*$: $sets\ x \cup sets\ y \subseteq Pow\ (space\ y)$
using *sets.space_closed* **by** *auto*
assume $\neg\ sets\ y \subseteq sets\ x \neg\ sets\ x \subseteq sets\ y$
then have $sets\ y \subset sets\ x \cup sets\ y$
by *auto*
also have $\ldots \leq sigma\ (space\ y)\ (sets\ x \cup sets\ y)$
by (*subst sets_measure_of*[*OF* $*$]) (*rule sigma_sets_superset_generator*)
finally show $y \leq sigma\ (space\ x)\ (sets\ x \cup sets\ y)$
by (*simp add*: $**$ *space_measure_of*[*OF* $*$] *less_eq_measure.intros*(*2*))
next
assume $\neg\ space\ y \subseteq space\ x \neg\ space\ x \subseteq space\ y$
then show $y \leq sigma\ (space\ x \cup space\ y)\ \{\}$
by (*intro less_eq_measure.intros*) *auto*
next
assume $sets\ x = sets\ y$ **then show** $y \leq sup\_measure'\ x\ y$
by (*simp add*: *le_measure le_emeasure_sup_measure'2*)
qed (*auto intro*: *less_eq_measure.intros*)
show $x \leq y \Longrightarrow z \leq y \Longrightarrow sup\ x\ z \leq y$
unfolding *sup_measure_def*
proof (*intro sup_lexord*[**where** $P{=}\lambda x.\ x \leq y$])
assume $x \leq y\ z \leq y$ **and** [*simp*]: $space\ x = space\ z\ sets\ x = sets\ z$
from $\langle x \leq y \rangle$ **show** $sup\_measure'\ x\ z \leq y$
proof *cases*

```
    case 1 then show ?thesis
      by (intro less_eq_measure.intros(1)) simp
  next
    case 2 then show ?thesis
      by (intro less_eq_measure.intros(2)) simp_all
  next
    case 3 with ‹z ≤ y› ‹x ≤ y› show ?thesis
      by (auto simp add: le_measure intro!: emeasure_sup_measure'_le2)
  qed
 next
  assume **: x ≤ y z ≤ y space x = space z ¬ sets z ⊆ sets x ¬ sets x ⊆ sets z
  then have *: sets x ∪ sets z ⊆ Pow (space x)
    using sets.space_closed by auto
  show sigma (space x) (sets x ∪ sets z) ≤ y
   unfolding sigma_le_iff[OF *] using ** by (auto simp: le_measure_iff split:
if_split_asm)
 next
  assume x ≤ y z ≤ y ¬ space z ⊆ space x ¬ space x ⊆ space z
  then have space x ⊆ space y space z ⊆ space y
    by (auto simp: le_measure_iff split: if_split_asm)
  then show sigma (space x ∪ space z) {} ≤ y
    by (simp add: sigma_le_iff)
 qed
qed

end

lemma space_empty_eq_bot: space a = {} ⟷ a = bot
 using space_empty[of a] by (auto intro!: measure_eqI)

lemma sets_eq_iff_bounded: A ≤ B ⟹ B ≤ C ⟹ sets A = sets C ⟹ sets B
= sets A
 by (auto dest: sets_eq_imp_space_eq simp add: le_measure_iff split: if_split_asm)

lemma sets_sup: sets A = sets M ⟹ sets B = sets M ⟹ sets (sup A B) = sets
M
 by (auto simp add: sup_measure_def sup_lexord_def dest: sets_eq_imp_space_eq)

lemma le_measureD1: A ≤ B ⟹ space A ≤ space B
 by (auto simp: le_measure_iff split: if_split_asm)

lemma le_measureD2: A ≤ B ⟹ space A = space B ⟹ sets A ≤ sets B
 by (auto simp: le_measure_iff split: if_split_asm)

lemma le_measureD3: A ≤ B ⟹ sets A = sets B ⟹ emeasure A X ≤ emeasure
B X
  by (auto simp: le_measure_iff le_fun_def dest: sets_eq_imp_space_eq split:
if_split_asm)
```

lemma *UN_space_closed*: $\bigcup(sets\ `\ S) \subseteq Pow\ (\bigcup(space\ `\ S))$
using *sets.space_closed* **by** *auto*

definition
$Sup\_lexord :: ('a \Rightarrow 'b{::}complete\_lattice) \Rightarrow ('a\ set \Rightarrow 'a) \Rightarrow ('a\ set \Rightarrow 'a) \Rightarrow 'a\ set \Rightarrow 'a$
where
$Sup\_lexord\ k\ c\ s\ A =$
$(let\ U = (SUP\ a{\in}A.\ k\ a)$
$in\ if\ \exists\, a{\in}A.\ k\ a = U\ then\ c\ \{a{\in}A.\ k\ a = U\}\ else\ s\ A)$

lemma *Sup_lexord*:
$(\bigwedge a\ S.\ a \in A \Longrightarrow k\ a = (SUP\ a{\in}A.\ k\ a) \Longrightarrow S = \{a'{\in}A.\ k\ a' = k\ a\} \Longrightarrow P\ (c\ S)) \Longrightarrow ((\bigwedge a.\ a \in A \Longrightarrow k\ a \neq (SUP\ a{\in}A.\ k\ a)) \Longrightarrow P\ (s\ A)) \Longrightarrow$
$P\ (Sup\_lexord\ k\ c\ s\ A)$
by (*auto simp*: *Sup_lexord_def Let_def*)

lemma *Sup_lexord1*:
assumes *A*: $A \neq \{\}$ $(\bigwedge a.\ a \in A \Longrightarrow k\ a = (\bigcup a{\in}A.\ k\ a))\ P\ (c\ A)$
shows $P\ (Sup\_lexord\ k\ c\ s\ A)$
unfolding *Sup_lexord_def Let_def*
proof (*clarsimp*, *safe*)
show $\forall\, a{\in}A.\ k\ a \neq (\bigcup x{\in}A.\ k\ x) \Longrightarrow P\ (s\ A)$
by (*metis assms*(*1*,*2*) *ex_in_conv*)
next
fix *a* **assume** $a \in A\ k\ a = (\bigcup x{\in}A.\ k\ x)$
then have $\{a \in A.\ k\ a = (\bigcup x{\in}A.\ k\ x)\} = \{a \in A.\ k\ a = k\ a\}$
by (*metis A*(*2*)[*symmetric*])
then show $P\ (c\ \{a \in A.\ k\ a = (\bigcup x{\in}A.\ k\ x)\})$
by (*simp add*: *A*(*3*))
qed

instantiation *measure* :: (*type*) *complete_lattice*
begin

interpretation *sup_measure*: *comm_monoid_set sup bot* :: *'a measure*
by *standard* (*auto intro*!: *antisym*)

lemma *sup_measure_F_mono'*:
finite J $\Longrightarrow$ *finite I* $\Longrightarrow$ *sup_measure.F id I* $\leq$ *sup_measure.F id* $(I \cup J)$
proof (*induction J rule*: *finite_induct*)
case *empty* **then show** *?case*
by *simp*
next
case (*insert i J*)
show *?case*
proof *cases*
assume $i \in I$ **with** *insert* **show** *?thesis*
by (*auto simp*: *insert_absorb*)

next
assume $i \notin I$
have *sup_measure.F id I* $\leq$ *sup_measure.F id* $(I \cup J)$
by (*intro insert*)
also have $\ldots \leq$ *sup_measure.F id* (*insert i* $(I \cup J)$)
using *insert* ‹$i \notin I$› **by** (*subst sup_measure.insert*) *auto*
finally show *?thesis*
by *auto*
qed
qed

lemma *sup_measure_F_mono*: *finite* $I \Longrightarrow J \subseteq I \Longrightarrow$ *sup_measure.F id* $J \leq$ *sup_measure.F id I*
using *sup_measure_F_mono'*[*of I J*] **by** (*auto simp*: *finite_subset Un_absorb1*)

lemma *sets_sup_measure_F*:
finite $I \Longrightarrow I \neq \{\} \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow$ *sets i* $=$ *sets M*$) \Longrightarrow$ *sets* (*sup_measure.F id I*) $=$ *sets M*
by (*induction I rule*: *finite_ne_induct*) (*simp_all add*: *sets_sup*)

definition *Sup_measure'* :: *'a measure set* $\Rightarrow$ *'a measure* **where**
Sup_measure' M $=$
measure_of $(\bigcup a{\in}M.$ *space a*$)$ $(\bigcup a{\in}M.$ *sets a*$)$
$(\lambda X.$ $(SUP\ P{\in}\{P.$ *finite* $P \wedge P \subseteq M\ \}.$ *sup_measure.F id P X*$))$

lemma *space_Sup_measure'2*: *space* (*Sup_measure' M*) $= (\bigcup m{\in}M.$ *space m*$)$
unfolding *Sup_measure'_def* **by** (*intro space_measure_of*[*OF UN_space_closed*])

lemma *sets_Sup_measure'2*: *sets* (*Sup_measure' M*) $=$ *sigma_sets* $(\bigcup m{\in}M.$ *space m*$)$ $(\bigcup m{\in}M.$ *sets m*$)$
unfolding *Sup_measure'_def* **by** (*intro sets_measure_of*[*OF UN_space_closed*])

lemma *sets_Sup_measure'*:
assumes *sets_eq*[*simp*]: $\bigwedge m.\ m \in M \Longrightarrow$ *sets m* $=$ *sets A* **and** $M \neq \{\}$
shows *sets* (*Sup_measure' M*) $=$ *sets A*
using *sets_eq*[*THEN sets_eq_imp_space_eq*, *simp*] ‹$M \neq \{\}$› **by** (*simp add*: *Sup_measure'_def*)

lemma *space_Sup_measure'*:
assumes *sets_eq*[*simp*]: $\bigwedge m.\ m \in M \Longrightarrow$ *sets m* $=$ *sets A* **and** $M \neq \{\}$
shows *space* (*Sup_measure' M*) $=$ *space A*
using *sets_eq*[*THEN sets_eq_imp_space_eq*, *simp*] ‹$M \neq \{\}$›
by (*simp add*: *Sup_measure'_def*)

lemma *emeasure_Sup_measure'*:
assumes *sets_eq*[*simp*]: $\bigwedge m.\ m \in M \Longrightarrow$ *sets m* $=$ *sets A* **and** $X \in$ *sets A M* $\neq \{\}$
shows *emeasure* (*Sup_measure' M*) *X* $= (SUP\ P{\in}\{P.$ *finite* $P \wedge P \subseteq M\}.$ *sup_measure.F id P X*$)$

```
    (is _ = ?S X)
  using Sup_measure'_def
proof (rule emeasure_measure_of)
  note sets_eq[THEN sets_eq_imp_space_eq, simp]
  have ∗: sets (Sup_measure' M) = sets A space (Sup_measure' M) = space A
    using ‹M ≠ {}› by (simp_all add: Sup_measure'_def)
  let ?μ = sup_measure.F id
  show countably_additive (sets (Sup_measure' M)) ?S
  proof (rule countably_additiveI, goal_cases)
    case (1 F)
    then have ∗∗: range F ⊆ sets A
      by (auto simp: ∗)
    show (∑ i. ?S (F i)) = ?S (⋃ i. F i)
    proof (subst ennreal_suminf_SUP_eq_directed)
      fix i j and N :: nat set assume ij: i ∈ {P. finite P ∧ P ⊆ M} j ∈ {P. finite
P ∧ P ⊆ M}
      have (i ≠ {} ⟶ sets (?μ i) = sets A) ∧ (j ≠ {} ⟶ sets (?μ j) = sets A) ∧
        (i ≠ {} ∨ j ≠ {} ⟶ sets (?μ (i ∪ j)) = sets A)
        using ij by (intro impI sets_sup_measure_F conjI) auto
      then have ?μ j (F n) ≤ ?μ (i ∪ j) (F n) ∧ ?μ i (F n) ≤ ?μ (i ∪ j) (F n)
for n
        using ij
        by (cases i = {}; cases j = {})
        (auto intro!: le_measureD3 sup_measure_F_mono simp: sets_sup_measure_F
              simp del: id_apply)
      with ij show ∃ k∈{P. finite P ∧ P ⊆ M}. ∀ n∈N. ?μ i (F n) ≤ ?μ k (F n)
∧ ?μ j (F n) ≤ ?μ k (F n)
        by (safe intro!: bexI[of _ i ∪ j]) auto
    next
      show (SUP P ∈ {P. finite P ∧ P ⊆ M}. ∑ n. ?μ P (F n)) = (SUP P ∈ {P.
finite P ∧ P ⊆ M}. ?μ P (⋃ (F ‘ UNIV)))
      proof (intro arg_cong [of _ _ Sup] image_cong refl)
        fix i assume i: i ∈ {P. finite P ∧ P ⊆ M}
        show (∑ n. ?μ i (F n)) = ?μ i (⋃ (F ‘ UNIV))
        proof cases
          assume i ≠ {} with i ∗∗ show ?thesis
          by (smt (verit, best) 1(2) Measure_Space.sets_sup_measure_F assms(1)
mem_Collect_eq subset_eq suminf_cong suminf_emeasure)
        qed simp
      qed
    qed
  qed
  show positive (sets (Sup_measure' M)) ?S
    by (auto simp: positive_def bot_ennreal[symmetric])
  show X ∈ sets (Sup_measure' M)
    using assms ∗ by auto
qed (rule UN_space_closed)

definition Sup_measure :: 'a measure set ⇒ 'a measure where
```

```
Sup_measure =
  Sup_lexord space
    (Sup_lexord sets Sup_measure'
      (λU. sigma (⋃u∈U. space u) (⋃u∈U. sets u)))
    (λU. sigma (⋃u∈U. space u) {})

definition Inf_measure :: 'a measure set ⇒ 'a measure where
  Inf_measure A = Sup {x. ∀a∈A. x ≤ a}

definition inf_measure :: 'a measure ⇒ 'a measure ⇒ 'a measure where
  inf_measure a b = Inf {a, b}

definition top_measure :: 'a measure where
  top_measure = Inf {}

instance
proof
  note UN_space_closed [simp]
  show upper: x ≤ Sup A if x: x ∈ A for x :: 'a measure and A
    unfolding Sup_measure_def
  proof (intro Sup_lexord[where P=λy. x ≤ y])
    assume ⋀a. a ∈ A ⟹ space a ≠ (⋃a∈A. space a)
    from this[OF ‹x ∈ A›] ‹x ∈ A› show x ≤ sigma (⋃a∈A. space a) {}
      by (intro less_eq_measure.intros) auto
  next
    fix a S assume a ∈ A and a: space a = (⋃a∈A. space a) and S: S = {a' ∈
A. space a' = space a}
      and neq: ⋀aa. aa ∈ S ⟹ sets aa ≠ (⋃a∈S. sets a)
    have sp_a: space a = (⋃(space ‘ S))
      using ‹a∈A› by (auto simp: S)
    show x ≤ sigma (⋃(space ‘ S)) (⋃(sets ‘ S))
    proof cases
      assume [simp]: space x = space a
      have sets x ⊂ (⋃a∈S. sets a)
        using ‹x∈A› neq[of x] by (auto simp: S)
      also have ... ⊆ sigma_sets (⋃x∈S. space x) (⋃x∈S. sets x)
        by (rule sigma_sets_superset_generator)
      finally show ?thesis
        by (intro less_eq_measure.intros(2)) (simp_all add: sp_a)
    next
      assume space x ≠ space a
      moreover have space x ≤ space a
        unfolding a using ‹x∈A› by auto
      ultimately show ?thesis
        by (intro less_eq_measure.intros) (simp add: less_le sp_a)
    qed
  next
    fix a b S S' assume a ∈ A and a: space a = (⋃a∈A. space a) and S: S =
{a' ∈ A. space a' = space a}
```

```
      and b ∈ S and b: sets b = (⋃a∈S. sets a) and S': S' = {a' ∈ S. sets a' =
sets b}
    then have S' ≠ {} space b = space a
      by auto
    have sets_eq: ⋀x. x ∈ S' ⟹ sets x = sets b
      by (auto simp: S')
    note sets_eq[THEN sets_eq_imp_space_eq, simp]
    have *: sets (Sup_measure' S') = sets b space (Sup_measure' S') = space b
      using ‹S' ≠ {}› by (simp_all add: Sup_measure'_def sets_eq)
    show x ≤ Sup_measure' S'
    proof cases
      assume x ∈ S
      with ‹b ∈ S› have space x = space b
        by (simp add: S)
      show ?thesis
      proof cases
        assume x ∈ S'
        show x ≤ Sup_measure' S'
        proof (intro le_measure[THEN iffD2] ballI)
          show sets x = sets (Sup_measure' S')
            using ‹x∈S'› * by (simp add: S')
          fix X assume X ∈ sets x
          show emeasure x X ≤ emeasure (Sup_measure' S') X
          proof (subst emeasure_Sup_measure'[OF _ ‹X ∈ sets x›])
            show emeasure x X ≤ (SUP P ∈ {P. finite P ∧ P ⊆ S'}. emeasure
(sup_measure.F id P) X)
              using ‹x∈S'› by (intro SUP_upper2[where i={x}]) auto
          qed (insert ‹x∈S'› S', auto)
        qed
      next
        assume x ∉ S'
        then have sets x ≠ sets b
          using ‹x∈S› by (auto simp: S')
        moreover have sets x ≤ sets b
          using ‹x∈S› unfolding b by auto
        ultimately show ?thesis
          using * ‹x ∈ S›
          by (intro less_eq_measure.intros(2))
            (simp_all add: * ‹space x = space b› less_le)
      qed
    next
      assume x ∉ S
      with ‹x∈A› ‹x ∉ S› ‹space b = space a› show ?thesis
        by (intro less_eq_measure.intros)
          (simp_all add: * less_le a SUP_upper S)
    qed
  qed
  show least: Sup A ≤ x if x: ⋀z. z ∈ A ⟹ z ≤ x for x :: 'a measure and A
    unfolding Sup_measure_def
```

```
  proof (intro Sup_lexord[where P=λy. y ≤ x])
    assume ⋀a. a ∈ A ⟹ space a ≠ (⋃a∈A. space a)
    show sigma (⋃(space ` A)) {} ≤ x
      using x[THEN le_measureD1] by (subst sigma_le_iff) auto
  next
    fix a S assume a ∈ A space a = (⋃a∈A. space a) and S: S = {a' ∈ A. space
a' = space a}
      ⋀a. a ∈ S ⟹ sets a ≠ (⋃a∈S. sets a)
    have ⋃(space ` S) ⊆ space x
      using S le_measureD1[OF x] by auto
    moreover
    have ⋃(space ` S) = space a
      using ‹a∈A› S by auto
    then have space x = ⋃(space ` S) ⟹ ⋃(sets ` S) ⊆ sets x
      using ‹a ∈ A› le_measureD2[OF x] by (auto simp: S)
    ultimately show sigma (⋃(space ` S)) (⋃(sets ` S)) ≤ x
      by (subst sigma_le_iff) simp_all
  next
    fix a b S S' assume a ∈ A and a: space a = (⋃a∈A. space a) and S: S =
{a' ∈ A. space a' = space a}
      and b ∈ S and b: sets b = (⋃a∈S. sets a) and S': S' = {a' ∈ S. sets a' =
sets b}
    then have S' ≠ {} space b = space a
      by auto
    have sets_eq: ⋀x. x ∈ S' ⟹ sets x = sets b
      by (auto simp: S')
    note sets_eq[THEN sets_eq_imp_space_eq, simp]
    have *: sets (Sup_measure' S') = sets b space (Sup_measure' S') = space b
      using ‹S' ≠ {}› by (simp_all add: Sup_measure'_def sets_eq)
    show Sup_measure' S' ≤ x
    proof cases
      assume space x = space a
      show ?thesis
      proof cases
        assume **: sets x = sets b
        show ?thesis
        proof (intro le_measure[THEN iffD2] ballI)
          show ***: sets (Sup_measure' S') = sets x
            by (simp add: * **)
          fix X assume X ∈ sets (Sup_measure' S')
          show emeasure (Sup_measure' S') X ≤ emeasure x X
            unfolding ***
            proof (subst emeasure_Sup_measure'[OF _ ‹X ∈ sets (Sup_measure'
S')›])
              show (SUP P ∈ {P. finite P ∧ P ⊆ S'}. emeasure (sup_measure.F id
P) X) ≤ emeasure x X
              proof (safe intro!: SUP_least)
                fix P assume P: finite P P ⊆ S'
                show emeasure (sup_measure.F id P) X ≤ emeasure x X
```

```
            proof cases
              assume P = {} then show ?thesis
                by auto
            next
              assume P ≠ {}
              from P have finite P P ⊆ A
                unfolding S' S by (simp_all add: subset_eq)
              then have sup_measure.F id P ≤ x
                by (induction P) (auto simp: x)
              moreover have sets (sup_measure.F id P) = sets x
                using ‹finite P› ‹P ≠ {}› ‹P ⊆ S'› ‹sets x = sets b›
                by (intro sets_sup_measure_F) (auto simp: S')
             ultimately show emeasure (sup_measure.F id P) X ≤ emeasure x X
                by (rule le_measureD3)
            qed
          qed
          show m ∈ S' ⟹ sets m = sets (Sup_measure' S') for m
            unfolding * by (simp add: S')
        qed fact
      qed
    next
      assume sets x ≠ sets b
      moreover have sets b ≤ sets x
       unfolding b S using x[THEN le_measureD2] ‹space x = space a› by auto
      ultimately show Sup_measure' S' ≤ x
        using ‹space x = space a› ‹b ∈ S›
        by (intro less_eq_measure.intros(2)) (simp_all add: * S)
    qed
  next
    assume space x ≠ space a
    then have space a < space x
      using le_measureD1[OF x[OF ‹a∈A›]] by auto
    then show Sup_measure' S' ≤ x
      by (intro less_eq_measure.intros) (simp add: * ‹space b = space a›)
  qed
qed
show Sup {} = (bot::'a measure) Inf {} = (top::'a measure)
  by (auto intro!: antisym least simp: top_measure_def)
show lower: x ∈ A ⟹ Inf A ≤ x for x :: 'a measure and A
  unfolding Inf_measure_def by (intro least) auto
show greatest: (⋀z. z ∈ A ⟹ x ≤ z) ⟹ x ≤ Inf A for x :: 'a measure and A
  unfolding Inf_measure_def by (intro upper) auto
show inf x y ≤ x inf x y ≤ y x ≤ y ⟹ x ≤ z ⟹ x ≤ inf y z for x y z :: 'a
measure
  by (auto simp: inf_measure_def intro!: lower greatest)
qed

end
```

lemma *sets_SUP*:
assumes $\bigwedge x.\ x \in I \Longrightarrow sets\ (M\ x) = sets\ N$
shows $I \neq \{\} \Longrightarrow sets\ (SUP\ i{\in}I.\ M\ i) = sets\ N$
unfolding *Sup_measure_def*
using *assms assms*[*THEN sets_eq_imp_space_eq*]
sets_Sup_measure'[**where** *A*=*N* **and** *M*=*M*‘*I*]
by (*intro Sup_lexord1*[**where** $P{=}\lambda x.\ sets\ x = sets\ N$]) *auto*

lemma *emeasure_SUP*:
assumes *sets*: $\bigwedge i.\ i \in I \Longrightarrow sets\ (M\ i) = sets\ N\ X \in sets\ N\ I \neq \{\}$
shows *emeasure* (*SUP i*∈*I*. *M i*) *X* = (*SUP J*∈{*J*. *J* ≠ {} ∧ *finite J* ∧ *J* ⊆ *I*}. *emeasure* (*SUP i*∈*J*. *M i*) *X*)
proof −
interpret *sup_measure*: *comm_monoid_set sup bot* :: ′*b measure*
by *standard* (*auto intro*!: *antisym*)
have *eq*: *finite J* ⟹ *sup_measure.F id J* = (*SUP i*∈*J*. *i*) **for** *J* :: ′*b measure set*
by (*induction J rule*: *finite_induct*) *auto*
have *1*: *J* ≠ {} ⟹ *J* ⊆ *I* ⟹ *sets* (*SUP x*∈*J*. *M x*) = *sets N* **for** *J*
by (*intro sets_SUP sets*) (*auto*)
from ‹*I* ≠ {}› **obtain** *i* **where** *i*∈*I* **by** *auto*
have *Sup_measure'* (*M*‘*I*) *X* = (*SUP P*∈{*P*. *finite P* ∧ *P* ⊆ *M*‘*I*}. *sup_measure.F id P X*)
using *sets* **by** (*intro emeasure_Sup_measure'*) *auto*
also have *Sup_measure'* (*M*‘*I*) = (*SUP i*∈*I*. *M i*)
unfolding *Sup_measure_def* **using** ‹*I* ≠ {}› *sets sets*(*1*)[*THEN sets_eq_imp_space_eq*]
by (*intro Sup_lexord1*[**where** $P{=}\lambda x.\ \_ = x$]) *auto*
also have (*SUP P*∈{*P*. *finite P* ∧ *P* ⊆ *M*‘*I*}. *sup_measure.F id P X*) =
(*SUP J*∈{*J*. *J* ≠ {} ∧ *finite J* ∧ *J* ⊆ *I*}. (*SUP i*∈*J*. *M i*) *X*)
proof (*intro SUP_eq*)
fix *J* **assume** *J* ∈ {*P*. *finite P* ∧ *P* ⊆ *M*‘*I*}
then obtain *J′* **where** *J′*: *J′* ⊆ *I finite J′* **and** *J*: *J* = *M*‘*J′* **and** *finite J*
using *finite_subset_image*[*of J M I*] **by** *auto*
show ∃*j*∈{*J*. *J* ≠ {} ∧ *finite J* ∧ *J* ⊆ *I*}. *sup_measure.F id J X* ≤ (*SUP i*∈*j*. *M i*) *X*
proof *cases*
assume *J′* = {} **with** ‹*i* ∈ *I*› **show** *?thesis*
by (*auto simp add*: *J*)
next
assume *J′* ≠ {} **with** *J J′* **show** *?thesis*
by (*intro bexI*[*of* _ *J′*]) (*auto simp add*: *eq simp del*: *id_apply*)
qed
next
fix *J* **assume** *J*: *J* ∈ {*P*. *P* ≠ {} ∧ *finite P* ∧ *P* ⊆ *I*}
show ∃*J′*∈{*J*. *finite J* ∧ *J* ⊆ *M*‘*I*}. (*SUP i*∈*J*. *M i*) *X* ≤ *sup_measure.F id J′ X*
using *J* **by** (*intro bexI*[*of* _ *M*‘*J*]) (*auto simp add*: *eq simp del*: *id_apply*)
qed
finally show *?thesis* **.**

qed

lemma *emeasure_SUP_chain*:
assumes *sets*: $\bigwedge i.\ i \in A \Longrightarrow sets\ (M\ i) = sets\ N\ X \in sets\ N$
assumes *ch*: *Complete_Partial_Order.chain* $(\leq)$ $(M\ `\ A)$ **and** $A \neq \{\}$
shows *emeasure* $(SUP\ i{\in}A.\ M\ i)\ X = (SUP\ i{\in}A.\ emeasure\ (M\ i)\ X)$
proof (*subst emeasure_SUP*[*OF sets* ‹$A \neq \{\}$›])
show $(SUP\ J{\in}\{J.\ J \neq \{\} \wedge finite\ J \wedge J \subseteq A\}.\ emeasure\ (Sup\ (M\ `\ J))\ X) = (SUP\ i{\in}A.\ emeasure\ (M\ i)\ X)$
proof (*rule SUP_eq*)
fix J **assume** $J \in \{J.\ J \neq \{\} \wedge finite\ J \wedge J \subseteq A\}$
then have J: *Complete_Partial_Order.chain* $(\leq)$ $(M\ `\ J)$ *finite* J $J \neq \{\}$ **and** $J \subseteq A$
using *ch*[*THEN chain_subset*, *of M*‘*J*] **by** *auto*
with *in_chain_finite*[*OF J*(*1*)] **obtain** j **where** $j \in J$ $(SUP\ j{\in}J.\ M\ j) = M\ j$
by *auto*
with ‹$J \subseteq A$› **show** $\exists j{\in}A.\ emeasure\ (Sup\ (M\ `\ J))\ X \leq emeasure\ (M\ j)\ X$
by *auto*
next
fix j **assume** $j{\in}A$ **then show** $\exists i{\in}\{J.\ J \neq \{\} \wedge finite\ J \wedge J \subseteq A\}.\ emeasure\ (M\ j)\ X \leq emeasure\ (Sup\ (M\ `\ i))\ X$
by (*intro bexI*[*of* _ $\{j\}$]) *auto*
qed
qed

Supremum of a set of σ-algebras

lemma *space_Sup_eq_UN*: $space\ (Sup\ M) = (\bigcup x{\in}M.\ space\ x)$ (**is** *?L*=*?R*)
proof
show *?L* $\subseteq$ *?R*
using *Sup_lexord*[**where** $P{=}\lambda x.\ space\ x = \_$]
apply (*clarsimp simp*: *Sup_measure_def*)
by (*smt* (*verit*) *Sup_lexord_def UN_E mem_Collect_eq space_Sup_measure'2 space_measure_of_conv*)
qed (*use Sup_upper le_measureD1* **in** *fastforce*)

lemma *sets_Sup_eq*:
assumes $*$: $\bigwedge m.\ m \in M \Longrightarrow space\ m = X$ **and** $M \neq \{\}$
shows $sets\ (Sup\ M) = sigma\_sets\ X\ (\bigcup x{\in}M.\ sets\ x)$
unfolding *Sup_measure_def*
proof (*rule Sup_lexord1* [*OF* ‹$M \neq \{\}$›])
show $sets\ (Sup\_lexord\ sets\ Sup\_measure'\ (\lambda U.\ sigma\ (\bigcup\ (space\ `\ U))\ (\bigcup\ (sets\ `\ U)))\ M)$
$= sigma\_sets\ X\ (\bigcup\ (sets\ `\ M))$
apply (*rule Sup_lexord*)
apply (*metis* (*mono_tags*, *lifting*) $*$ *empty_iff mem_Collect_eq sets.sigma_sets_eq sets_Sup_measure'*)
by (*metis* $*$ *SUP_eq_const UN_space_closed assms*(*2*) *sets_measure_of*)

qed (*use* ∗ **in** *blast*)

lemma *in_sets_Sup*: (⋀*m*. *m* ∈ *M* ⟹ *space m* = *X*) ⟹ *m* ∈ *M* ⟹ *A* ∈ *sets m* ⟹ *A* ∈ *sets* (*Sup M*)
by (*subst sets_Sup_eq*[**where** *X*=*X*]) *auto*

lemma *Sup_lexord_rel*:
assumes ⋀*i*. *i* ∈ *I* ⟹ *k* (*A i*) = *k* (*B i*)
R (*c* (*A* ‘ {*a* ∈ *I*. *k* (*B a*) = (*SUP x*∈*I*. *k* (*B x*))})) (*c* (*B* ‘ {*a* ∈ *I*. *k* (*B a*) = (*SUP x*∈*I*. *k* (*B x*))}))
R (*s* (*A*‘*I*)) (*s* (*B*‘*I*))
shows *R* (*Sup_lexord k c s* (*A*‘*I*)) (*Sup_lexord k c s* (*B*‘*I*))
proof −
have *A* ‘ {*a* ∈ *I*. *k* (*B a*) = (*SUP x*∈*I*. *k* (*B x*))} = {*a* ∈ *A* ‘ *I*. *k a* = (*SUP x*∈*I*. *k* (*B x*))}
using *assms*(*1*) **by** *auto*
moreover have *B* ‘ {*a* ∈ *I*. *k* (*B a*) = (*SUP x*∈*I*. *k* (*B x*))} = {*a* ∈ *B* ‘ *I*. *k a* = (*SUP x*∈*I*. *k* (*B x*))}
by *auto*
ultimately show *?thesis*
using *assms* **by** (*auto simp*: *Sup_lexord_def Let_def image_comp*)
qed

lemma *sets_SUP_cong*:
assumes *eq*: ⋀*i*. *i* ∈ *I* ⟹ *sets* (*M i*) = *sets* (*N i*)
shows *sets* (*SUP i*∈*I*. *M i*) = *sets* (*SUP i*∈*I*. *N i*)
unfolding *Sup_measure_def*
using *eq eq*[*THEN sets_eq_imp_space_eq*]
by (*intro Sup_lexord_rel*[**where** *R*=λ*x y*. *sets x* = *sets y*], *simp_all add*: *sets_Sup_measure'2*)

lemma *sets_Sup_in_sets*:
assumes *M* ≠ {}
assumes ⋀*m*. *m* ∈ *M* ⟹ *space m* = *space N*
assumes ⋀*m*. *m* ∈ *M* ⟹ *sets m* ⊆ *sets N*
shows *sets* (*Sup M*) ⊆ *sets N*
proof −
have ∗: ⋃(*space* ‘ *M*) = *space N*
using *assms* **by** *auto*
show *?thesis*
unfolding ∗ **using** *assms* **by** (*subst sets_Sup_eq*[*of M space N*]) (*auto intro*!: *sets.sigma_sets_subset*)
qed

lemma *measurable_Sup1*:
assumes *m*: *m* ∈ *M* **and** *f*: *f* ∈ *measurable m N*
and *const_space*: ⋀*m n*. *m* ∈ *M* ⟹ *n* ∈ *M* ⟹ *space m* = *space n*
shows *f* ∈ *measurable* (*Sup M*) *N*
proof −

have *space* (*Sup M*) = *space m*
using *m* **by** (*auto simp add*: *space_Sup_eq_UN dest*: *const_space*)
then show *?thesis*
using *m f* **unfolding** *measurable_def* **by** (*auto intro*: *in_sets_Sup*[*OF const_space*])
qed

lemma *measurable_Sup2*:
assumes *M*: *M* ≠ {}
assumes *f*: ⋀*m*. *m* ∈ *M* ⟹ *f* ∈ *measurable N m*
and *const_space*: ⋀*m n*. *m* ∈ *M* ⟹ *n* ∈ *M* ⟹ *space m* = *space n*
shows *f* ∈ *measurable N* (*Sup M*)
proof −
from *M* **obtain** *m* **where** *m* ∈ *M* **by** *auto*
have *space_eq*: ⋀*n*. *n* ∈ *M* ⟹ *space n* = *space m*
by (*intro const_space* ‹*m* ∈ *M*›)
have *eq*: *sets* (*sigma* (⋃ (*space* ‘ *M*)) (⋃ (*sets* ‘ *M*))) = *sets* (*Sup M*)
by (*metis M SUP_eq_const UN_space_closed sets_Sup_eq sets_measure_of space_eq*)
have *f* ∈ *measurable N* (*sigma* (⋃*m*∈*M*. *space m*) (⋃*m*∈*M*. *sets m*))
proof (*rule measurable_measure_of*)
show *f* ∈ *space N* → ⋃(*space* ‘ *M*)
using *measurable_space*[*OF f*] *M* **by** *auto*
qed (*auto intro*: *measurable_sets f dest*: *sets.sets_into_space*)
also have *measurable N* (*sigma* (⋃*m*∈*M*. *space m*) (⋃*m*∈*M*. *sets m*)) = *measurable N* (*Sup M*)
using *eq measurable_cong_sets* **by** *blast*
finally show *?thesis* **.**
qed

lemma *measurable_SUP2*:
I ≠ {} ⟹ (⋀*i*. *i* ∈ *I* ⟹ *f* ∈ *measurable N* (*M i*)) ⟹
(⋀*i j*. *i* ∈ *I* ⟹ *j* ∈ *I* ⟹ *space* (*M i*) = *space* (*M j*)) ⟹ *f* ∈ *measurable N* (*SUP i*∈*I*. *M i*)
by (*auto intro*!: *measurable_Sup2*)

lemma *sets_Sup_sigma*:
assumes [*simp*]: *M* ≠ {} **and** *M*: ⋀*m*. *m* ∈ *M* ⟹ *m* ⊆ *Pow* Ω
shows *sets* (*SUP m*∈*M*. *sigma* Ω *m*) = *sets* (*sigma* Ω (⋃ *M*))
proof −
{ **fix** *a m* **assume** *a* ∈ *sigma_sets* Ω *m m* ∈ *M*
then have *a* ∈ *sigma_sets* Ω (⋃ *M*)
by *induction* (*auto intro*: *sigma_sets.intros*(*2*−)) }
then have *sigma_sets* Ω (⋃ (*sigma_sets* Ω ‘ *M*)) = *sigma_sets* Ω (⋃ *M*)
by (*smt* (*verit*, *best*) *UN_iff Union_iff sigma_sets.Basic sigma_sets_eqI*)
then show *sets* (*SUP m*∈*M*. *sigma* Ω *m*) = *sets* (*sigma* Ω (⋃ *M*))
by (*subst sets_Sup_eq*) (*fastforce simp add*: *M Union_least*)+
qed

lemma *Sup_sigma*:

```
  assumes [simp]: M ≠ {} and M: ⋀m. m ∈ M ⟹ m ⊆ Pow Ω
  shows (SUP m∈M. sigma Ω m) = (sigma Ω (⋃ M))
proof (intro antisym SUP_least)
  have ∗: ⋃ M ⊆ Pow Ω
    using M by auto
  show sigma Ω (⋃ M) ≤ (SUP m∈M. sigma Ω m)
  proof (intro less_eq_measure.intros(3))
    show space (sigma Ω (⋃ M)) = space (SUP m∈M. sigma Ω m)
        sets (sigma Ω (⋃ M)) = sets (SUP m∈M. sigma Ω m)
     by (auto simp add: M sets_Sup_sigma sets_eq_imp_space_eq space_measure_of_conv)
  qed (simp add: emeasure_sigma le_fun_def)
  fix m assume m ∈ M then show sigma Ω m ≤ sigma Ω (⋃ M)
    by (subst sigma_le_iff) (auto simp add: M ∗)
qed

lemma SUP_sigma_sigma:
  M ≠ {} ⟹ (⋀m. m ∈ M ⟹ f m ⊆ Pow Ω) ⟹ (SUP m∈M. sigma Ω (f m))
= sigma Ω (⋃ m∈M. f m)
  using Sup_sigma[of f‘M Ω] by (auto simp: image_comp)

lemma sets_vimage_Sup_eq:
  assumes ∗: M ≠ {} f ∈ X → Y ⋀m. m ∈ M ⟹ space m = Y
  shows sets (vimage_algebra X f (Sup M)) = sets (SUP m ∈ M. vimage_algebra
X f m)
  (is ?L = ?R)
proof
  have ⋀m. m ∈ M ⟹ f ∈ Sup (vimage_algebra X f ‘ M) →M m
    using assms
    by (smt (verit, del_insts) Pi_iff imageE image_eqI measurable_Sup1
          measurable_vimage_algebra1 space_vimage_algebra)
  then show ?L ⊆ ?R
    by (intro sets_image_in_sets measurable_Sup2) (simp_all add: space_Sup_eq_UN
∗)
  show ?R ⊆ ?L
    apply (intro sets_Sup_in_sets)
     apply (force simp add: ∗ space_Sup_eq_UN sets_vimage_algebra2 intro:
in_sets_Sup)+
    done
qed

lemma restrict_space_eq_vimage_algebra':
  sets (restrict_space M Ω) = sets (vimage_algebra (Ω ∩ space M) (λx. x) M)
proof −
  have ∗: {A ∩ (Ω ∩ space M) |A. A ∈ sets M} = {A ∩ Ω |A. A ∈ sets M}
    using sets.sets_into_space[of _ M] by blast

  show ?thesis
    unfolding restrict_space_def
    by (subst sets_measure_of)
```

(*auto simp add*: *image_subset_iff sets_vimage_algebra* ∗ *dest*: *sets.sets_into_space intro*!: *arg_cong2*[**where** *f*=*sigma_sets*])
qed

lemma *sigma_le_sets*:
assumes [*simp*]: $A \subseteq Pow\ X$ **shows** $sets\ (sigma\ X\ A) \subseteq sets\ N \longleftrightarrow X \in sets\ N \wedge A \subseteq sets\ N$
proof
have $X \in sigma\_sets\ X\ A\ A \subseteq sigma\_sets\ X\ A$
by (*auto intro*: *sigma_sets_top*)
moreover assume $sets\ (sigma\ X\ A) \subseteq sets\ N$
ultimately show $X \in sets\ N \wedge A \subseteq sets\ N$
by *auto*
next
assume ∗: $X \in sets\ N \wedge A \subseteq sets\ N$
{ fix *Y* **assume** $Y \in sigma\_sets\ X\ A$ **from** *this* ∗ **have** $Y \in sets\ N$
by *induction auto* **}**
then show $sets\ (sigma\ X\ A) \subseteq sets\ N$
by *auto*
qed

lemma *measurable_iff_sets*:
$f \in measurable\ M\ N \longleftrightarrow (f \in space\ M \to space\ N \wedge sets\ (vimage\_algebra\ (space\ M)\ f\ N) \subseteq sets\ M)$
unfolding *measurable_def*
by (*smt* (*verit*, *ccfv_threshold*) *mem_Collect_eq sets_vimage_algebra sigma_sets_le_sets_iff subset_eq*)

lemma *sets_vimage_algebra_space*: $X \in sets\ (vimage\_algebra\ X\ f\ M)$
using *sets.top*[*of vimage_algebra X f M*] **by** *simp*

lemma *measurable_mono*:
assumes *N*: $sets\ N' \leq sets\ N\ space\ N = space\ N'$
assumes *M*: $sets\ M \leq sets\ M'\ space\ M = space\ M'$
shows $measurable\ M\ N \subseteq measurable\ M'\ N'$
unfolding *measurable_def*
proof *safe*
fix *f A* **assume** $f \in space\ M \to space\ N\ A \in sets\ N'$
moreover assume $\forall y{\in}sets\ N.\ f\ -`\ y \cap space\ M \in sets\ M$ **note** *this*[*THEN bspec*, *of A*]
ultimately show $f\ -`\ A \cap space\ M' \in sets\ M'$
using *assms* **by** *auto*
qed (*use N M* **in** *auto*)

lemma *measurable_Sup_measurable*:
assumes *f*: $f \in space\ N \to A$
shows $f \in measurable\ N\ (Sup\ \{M.\ space\ M = A \wedge f \in measurable\ N\ M\})$
proof (*rule measurable_Sup2*)
show $\{M.\ space\ M = A \wedge f \in measurable\ N\ M\} \neq \{\}$

using *f* **unfolding** *ex_in_conv*[*symmetric*]
by (*intro exI*[*of* _ *sigma A* {}]) (*auto intro*!: *measurable_measure_of*)
qed *auto*

lemma (**in** *sigma_algebra*) *sigma_sets_subset'*:
assumes *a*: $a \subseteq M$ $\Omega' \in M$
shows *sigma_sets* Ω' *a* $\subseteq M$
proof
show $x \in M$ **if** *x*: $x \in$ *sigma_sets* Ω' *a* **for** *x*
using *x* **by** (*induct rule*: *sigma_sets.induct*) (*use a* **in** *auto*)
qed

lemma *in_sets_SUP*: $i \in I \Longrightarrow (\bigwedge i.\ i \in I \Longrightarrow$ *space* $(M\ i) = Y) \Longrightarrow X \in$ *sets* $(M\ i) \Longrightarrow X \in$ *sets* (*SUP* $i{\in}I.\ M\ i$)
by (*intro in_sets_Sup*[**where** $X{=}Y$]) *auto*

lemma *measurable_SUP1*:
$i \in I \Longrightarrow f \in$ *measurable* $(M\ i)\ N \Longrightarrow (\bigwedge m\ n.\ m \in I \Longrightarrow n \in I \Longrightarrow$ *space* $(M\ m) =$ *space* $(M\ n)) \Longrightarrow$
$f \in$ *measurable* (*SUP* $i{\in}I.\ M\ i$) *N*
by (*auto intro*: *measurable_Sup1*)

lemma *sets_image_in_sets'*:
assumes *X*: $X \in$ *sets N*
assumes *f*: $\bigwedge A.\ A \in$ *sets* $M \Longrightarrow f -` A \cap X \in$ *sets N*
shows *sets* (*vimage_algebra X f M*) $\subseteq$ *sets N*
unfolding *sets_vimage_algebra*
by (*rule sets.sigma_sets_subset'*) (*auto intro*!: *measurable_sets X f*)

lemma *mono_vimage_algebra*:
sets M $\leq$ *sets N* $\Longrightarrow$ *sets* (*vimage_algebra X f M*) $\subseteq$ *sets* (*vimage_algebra X f N*)
using *sets.top*[*of sigma X* $\{f -` A \cap X \mid A.\ A \in$ *sets N*$\}$]
unfolding *vimage_algebra_def*
by (*smt* (*verit*, *del_insts*) *space_measure_of sigma_le_sets Pow_iff inf_le2 mem_Collect_eq subset_eq*)

lemma *mono_restrict_space*: *sets M* $\leq$ *sets N* $\Longrightarrow$ *sets* (*restrict_space M X*) $\subseteq$ *sets* (*restrict_space N X*)
unfolding *sets_restrict_space* **by** (*rule image_mono*)

lemma *sets_eq_bot*: *sets M* $= \{\{\}\} \longleftrightarrow M =$ *bot*
by (*metis measure_eqI emeasure_empty sets_bot singletonD*)

lemma *sets_eq_bot2*: $\{\{\}\} =$ *sets M* $\longleftrightarrow M =$ *bot*
using *sets_eq_bot*[*of M*] **by** *blast*

lemma (**in** *finite_measure*) *countable_support*:

countable {*x*. *measure M* {*x*} ≠ *0*}
proof *cases*
assume *measure M* (*space M*) = *0*
then show *?thesis*
by (*metis* (*mono_tags*, *lifting*) *bounded_measure measure_le_0_iff Collect_empty_eq countable_empty*)
next
let *?M* = *measure M* (*space M*) **and** *?m* = λ*x*. *measure M* {*x*}
assume *?M* ≠ *0*
then have ∗: {*x*. *?m x* ≠ *0*} = (⋃ *n*. {*x*. *?M* / *Suc n* < *?m x*})
using *reals_Archimedean*[*of ?m x* / *?M* **for** *x*]
by (*auto simp*: *field_simps not_le*[*symmetric*] *divide_le_0_iff measure_le_0_iff*)
have ∗∗: ⋀*n*. *finite* {*x*. *?M* / *Suc n* < *?m x*}
proof (*rule ccontr*)
fix *n* **assume** *infinite* {*x*. *?M* / *Suc n* < *?m x*} (**is** *infinite ?X*)
then obtain *X* **where** *finite X card X* = *Suc* (*Suc n*) *X* ⊆ *?X*
by (*metis infinite_arbitrarily_large*)
then have ∗: ⋀*x*. *x* ∈ *X* ⟹ *?M* / *Suc n* ≤ *?m x*
by *auto*
{ fix *x* **assume** *x* ∈ *X*
from ‹*?M* ≠ *0*› ∗[*OF this*] **have** *?m x* ≠ *0* **by** (*auto simp*: *field_simps measure_le_0_iff*)
then have {*x*} ∈ *sets M* **by** (*auto dest*: *measure_notin_sets*) **}**
note *singleton_sets* = *this*
have *?M* < (∑ *x*∈*X*. *?M* / *Suc n*)
using ‹*?M* ≠ *0*›
by (*simp add*: ‹*card X* = *Suc* (*Suc n*)› *field_simps less_le*)
also have ... ≤ (∑ *x*∈*X*. *?m x*)
by (*rule sum_mono*) *fact*
also have ... = *measure M* (⋃ *x*∈*X*. {*x*})
using *singleton_sets* ‹*finite X*›
by (*intro finite_measure_finite_Union*[*symmetric*]) (*auto simp*: *disjoint_family_on_def*)
finally have *?M* < *measure M* (⋃ *x*∈*X*. {*x*}) **.**
moreover have *measure M* (⋃ *x*∈*X*. {*x*}) ≤ *?M*
using *singleton_sets*[*THEN sets.sets_into_space*] **by** (*intro finite_measure_mono*) *auto*
ultimately show *False* **by** *simp*
qed
show *?thesis*
unfolding ∗ **by** (*intro countable_UN countableI_type countable_finite*[*OF* ∗∗])
qed

end

6.4 Borel Space

theory *Borel_Space*
imports
Measurable Derivative Ordered_Euclidean_Space Extended_Real_Limits

begin

lemma *is_interval_real_ereal_oo*: *is_interval* (*real_of_ereal* ‘ {*N*<..<*M*::*ereal*})
by (*auto simp*: *real_atLeastGreaterThan_eq*)

lemma *sets_Collect_eventually_sequentially*[*measurable*]:
(⋀*i*. {*x*∈*space M*. *P x i*} ∈ *sets M*) ⟹ {*x*∈*space M*. *eventually* (*P x*) *sequentially*} ∈ *sets M*
unfolding *eventually_sequentially* **by** *simp*

lemma *topological_basis_trivial*: *topological_basis* {*A*. *open A*}
by (*auto simp*: *topological_basis_def*)

proposition *open_prod_generated*: *open* = *generate_topology* {*A* × *B* | *A B*. *open A* ∧ *open B*}
proof −
have {*A* × *B* :: (′*a* × ′*b*) *set* | *A B*. *open A* ∧ *open B*} = ((λ(*a*, *b*). *a* × *b*) ‘ ({*A*. *open A*} × {*A*. *open A*}))
by *auto*
then show *?thesis*
by (*auto intro*: *topological_basis_prod topological_basis_trivial topological_basis_imp_subbasis*)
qed

proposition *mono_on_imp_deriv_nonneg*:
assumes *mono*: *mono_on A f* **and** *deriv*: (*f has_real_derivative D*) (*at x*)
assumes *x* ∈ *interior A*
shows *D* ≥ *0*
proof (*rule tendsto_lowerbound*)
let *?A′* = (λ*y*. *y* − *x*) ‘ *interior A*
from *deriv* **show** ((λ*h*. (*f* (*x* + *h*) − *f x*) / *h*) ⟶ *D*) (*at 0*)
by (*simp add*: *field_has_derivative_at has_field_derivative_def*)
from *mono* **have** *mono′*: *mono_on* (*interior A*) *f* **by** (*rule mono_on_subset*) (*rule interior_subset*)

show *eventually* (λ*h*. (*f* (*x* + *h*) − *f x*) / *h* ≥ *0*) (*at 0*)
proof (*subst eventually_at_topological*, *intro exI conjI ballI impI*)
have *open* (*interior A*) **by** *simp*
hence *open* ((+) (−*x*) ‘ *interior A*) **by** (*rule open_translation*)
also have ((+) (−*x*) ‘ *interior A*) = *?A′* **by** *auto*
finally show *open ?A′* **.**
next
from ‹*x* ∈ *interior A*› **show** *0* ∈ *?A′* **by** *auto*
next
fix *h* **assume** *h* ∈ *?A′*
hence *x* + *h* ∈ *interior A* **by** *auto*
with *mono′* **and** ‹*x* ∈ *interior A*› **show** (*f* (*x* + *h*) − *f x*) / *h* ≥ *0*
by (*cases h rule*: *linorder_cases*[*of* _ *0*])
(*simp_all add*: *divide_nonpos_neg divide_nonneg_pos mono_onD field_simps*)
qed

qed *simp*

proposition *mono_on_ctble_discont*:
fixes $f :: real \Rightarrow real$
fixes $A :: real\ set$
assumes *mono_on A f*
shows *countable* $\{a{\in}A.\ \neg\ continuous\ (at\ a\ within\ A)\ f\}$
proof −
have *mono*: $\bigwedge x\ y.\ x \in A \Longrightarrow y \in A \Longrightarrow x \leq y \Longrightarrow f\ x \leq f\ y$
using ‹*mono_on A f*› **by** (*simp add*: *mono_on_def*)
have $\forall\, a \in \{a{\in}A.\ \neg\ continuous\ (at\ a\ within\ A)\ f\}.\ \exists\, q :: nat \times rat.$
(*fst q* = *0* ∧ *of_rat* (*snd q*) < *f a* ∧ (∀ *x* ∈ *A*. *x* < *a* ⟶ *f x* < *of_rat* (*snd q*))) ∨
(*fst q* = *1* ∧ *of_rat* (*snd q*) > *f a* ∧ (∀ *x* ∈ *A*. *x* > *a* ⟶ *f x* > *of_rat* (*snd q*)))
proof (*clarsimp simp del*: *One_nat_def*)
fix *a* **assume** *a* ∈ *A* **assume** ¬ *continuous* (*at a within A*) *f*
thus ∃ *q1 q2*.
q1 = *0* ∧ *real_of_rat q2* < *f a* ∧ (∀ *x*∈*A*. *x* < *a* ⟶ *f x* < *real_of_rat q2*) ∨
q1 = *1* ∧ *f a* < *real_of_rat q2* ∧ (∀ *x*∈*A*. *a* < *x* ⟶ *real_of_rat q2* < *f x*)
proof (*auto simp add*: *continuous_within order_tendsto_iff eventually_at*)
fix *l* **assume** *l* < *f a*
then obtain *q2* **where** *q2*: *l* < *of_rat q2 of_rat q2* < *f a*
using *of_rat_dense* **by** *blast*
assume ∗ [*rule_format*]: ∀ *d*>*0*. ∃ *x*∈*A*. *x* ≠ *a* ∧ *dist x a* < *d* ∧ ¬ *l* < *f x*
from *q2* **have** *real_of_rat q2* < *f a* ∧ (∀ *x*∈*A*. *x* < *a* ⟶ *f x* < *real_of_rat q2*)
proof *auto*
fix *x* **assume** *x* ∈ *A x* < *a*
with *q2* ∗[*of a* − *x*] **show** *f x* < *real_of_rat q2*
apply (*auto simp add*: *dist_real_def not_less*)
apply (*subgoal_tac f x* ≤ *f xa*)
by (*auto intro*: *mono*)
qed
thus *?thesis* **by** *auto*
next
fix *u* **assume** *u* > *f a*
then obtain *q2* **where** *q2*: *f a* < *of_rat q2 of_rat q2* < *u*
using *of_rat_dense* **by** *blast*
assume ∗[*rule_format*]: ∀ *d*>*0*. ∃ *x*∈*A*. *x* ≠ *a* ∧ *dist x a* < *d* ∧ ¬ *u* > *f x*
from *q2* **have** *real_of_rat q2* > *f a* ∧ (∀ *x*∈*A*. *x* > *a* ⟶ *f x* > *real_of_rat q2*)
proof *auto*
fix *x* **assume** *x* ∈ *A x* > *a*
with *q2* ∗[*of x* − *a*] **show** *f x* > *real_of_rat q2*
apply (*auto simp add*: *dist_real_def*)
apply (*subgoal_tac f x* ≥ *f xa*)

```
        by (auto intro: mono)
      qed
      thus ?thesis by auto
    qed
  qed
  then obtain g :: real ⇒ nat × rat where ∀ a ∈ {a∈A. ¬ continuous (at a within
A) f}.
      (fst (g a) = 0 ∧ of_rat (snd (g a)) < f a ∧ (∀ x ∈ A. x < a ⟶ f x < of_rat
(snd (g a)))) |
      (fst (g a) = 1 ∧ of_rat (snd (g a)) > f a ∧ (∀ x ∈ A. x > a ⟶ f x > of_rat
(snd (g a))))
    by (rule bchoice [THEN exE]) blast
  hence g: ⋀a x. a ∈ A ⟹ ¬ continuous (at a within A) f ⟹ x ∈ A ⟹
      (fst (g a) = 0 ∧ of_rat (snd (g a)) < f a ∧ (x < a ⟶ f x < of_rat (snd (g
a)))) |
      (fst (g a) = 1 ∧ of_rat (snd (g a)) > f a ∧ (x > a ⟶ f x > of_rat (snd (g
a))))
    by auto
  have inj_on g {a∈A. ¬ continuous (at a within A) f}
  proof (auto simp add: inj_on_def)
    fix w z
    assume 1: w ∈ A and 2: ¬ continuous (at w within A) f and
          3: z ∈ A and 4: ¬ continuous (at z within A) f and
          5: g w = g z
    from g [OF 1 2 3] g [OF 3 4 1] 5
    show w = z by auto
  qed
  thus ?thesis
    by (rule countableI')
qed

lemma mono_on_ctble_discont_open:
  fixes f :: real ⇒ real
  fixes A :: real set
  assumes open A mono_on A f
  shows countable {a∈A. ¬isCont f a}
proof −
  have {a∈A. ¬isCont f a} = {a∈A. ¬(continuous (at a within A) f)}
    by (auto simp add: continuous_within_open [OF _ ‹open A›])
  thus ?thesis
    apply (elim ssubst)
    by (rule mono_on_ctble_discont, rule assms)
qed

lemma mono_ctble_discont:
  fixes f :: real ⇒ real
  assumes mono f
  shows countable {a. ¬ isCont f a}
   using assms mono_on_ctble_discont [of UNIV f] unfolding mono_on_def
```

mono_def **by** *auto*

lemma *has_real_derivative_imp_continuous_on*:
assumes $\bigwedge x.\ x \in A \Longrightarrow (f \text{ has\_real\_derivative } f'\ x)\ (at\ x)$
shows *continuous_on A f*
apply (*intro differentiable_imp_continuous_on*, *unfold differentiable_on_def*)
using *assms differentiable_at_withinI real_differentiable_def* **by** *blast*

lemma *continuous_interval_vimage_Int*:
assumes *continuous_on* $\{a::real..b\}$ *g* **and** *mono*: $\bigwedge x\ y.\ a \le x \Longrightarrow x \le y \Longrightarrow y \le b \Longrightarrow g\ x \le g\ y$
assumes $a \le b$ $(c::real) \le d$ $\{c..d\} \subseteq \{g\ a..g\ b\}$
obtains $c'\ d'$ **where** $\{a..b\} \cap g -` \{c..d\} = \{c'..d'\}$ $c' \le d'$ $g\ c' = c$ $g\ d' = d$
proof−
let *?A* $= \{a..b\} \cap g -` \{c..d\}$
from *IVT'*[*of g a c b*, *OF* _ _ ‹$a \le b$› *assms*(*1*)] *assms*(*4*,*5*)
obtain c'' **where** c'': $c'' \in$ *?A g* $c'' = c$ **by** *auto*
from *IVT'*[*of g a d b*, *OF* _ _ ‹$a \le b$› *assms*(*1*)] *assms*(*4*,*5*)
obtain d'' **where** d'': $d'' \in$ *?A g* $d'' = d$ **by** *auto*
hence [*simp*]: *?A* $\neq \{\}$ **by** *blast*

define c' **where** $c' =$ *Inf ?A*
define d' **where** $d' =$ *Sup ?A*
have *?A* $\subseteq \{c'..d'\}$ **unfolding** c'*_def* d'*_def*
by (*intro subsetI*) (*auto intro*: *cInf_lower cSup_upper*)
moreover from *assms* **have** *closed ?A*
using *continuous_on_closed_vimage*[*of* $\{a..b\}$ *g*] **by** (*subst Int_commute*) *simp*
hence $c'd'$*_in_set*: $c' \in$ *?A* $d' \in$ *?A* **unfolding** c'*_def* d'*_def*
by ((*intro closed_contains_Inf closed_contains_Sup*, *simp_all*)[])+
hence $\{c'..d'\} \subseteq$ *?A* **using** *assms*
by (*intro subsetI*)
(*auto intro*!: *order_trans*[*of c g c' g x* **for** *x*] *order_trans*[*of g x g d' d* **for** *x*]
intro!: *mono*)
moreover have $c' \le d'$ **using** $c'd'$*_in_set*(*2*) **unfolding** c'*_def* **by** (*intro cInf_lower*) *auto*
moreover have $g\ c' \le c$ $g\ d' \ge d$
apply (*insert* c'' d'' $c'd'$*_in_set*)
apply (*subst* c''(*2*)[*symmetric*])
apply (*auto simp*: c'*_def intro*!: *mono cInf_lower* c'') []
apply (*subst* d''(*2*)[*symmetric*])
apply (*auto simp*: d'*_def intro*!: *mono cSup_upper* d'') []
done
with $c'd'$*_in_set* **have** $g\ c' = c$ $g\ d' = d$ **by** *auto*
ultimately show *?thesis* **using** *that* **by** *blast*
qed

6.4.1 Generic Borel spaces

definition (**in** *topological_space*) *borel* :: *'a measure* **where**

borel = *sigma UNIV* {*S*. *open S*}

abbreviation *borel_measurable M* ≡ *measurable M borel*

lemma *in_borel_measurable*:
f ∈ *borel_measurable M* ⟷
(∀ *S* ∈ *sigma_sets UNIV* {*S*. *open S*}. *f* −‘ *S* ∩ *space M* ∈ *sets M*)
by (*auto simp add*: *measurable_def borel_def*)

lemma *in_borel_measurable_borel*:
f ∈ *borel_measurable M* ⟷
(∀ *S* ∈ *sets borel*.
f −‘ *S* ∩ *space M* ∈ *sets M*)
by (*auto simp add*: *measurable_def borel_def*)

lemma *space_borel*[*simp*]: *space borel* = *UNIV*
unfolding *borel_def* **by** *auto*

lemma *space_in_borel*[*measurable*]: *UNIV* ∈ *sets borel*
unfolding *borel_def* **by** *auto*

lemma *sets_borel*: *sets borel* = *sigma_sets UNIV* {*S*. *open S*}
unfolding *borel_def* **by** (*rule sets_measure_of*) *simp*

lemma *measurable_sets_borel*:
⟦*f* ∈ *measurable borel M*; *A* ∈ *sets M*⟧ ⟹ *f* −‘ *A* ∈ *sets borel*
by (*drule* (*1*) *measurable_sets*) *simp*

lemma *pred_Collect_borel*[*measurable* (*raw*)]: *Measurable.pred borel P* ⟹ {*x*. *P x*} ∈ *sets borel*
unfolding *borel_def pred_def* **by** *auto*

lemma *borel_open*[*measurable* (*raw generic*)]:
assumes *open A* **shows** *A* ∈ *sets borel*
proof −
have *A* ∈ {*S*. *open S*} **unfolding** *mem_Collect_eq* **using** *assms* **.**
thus *?thesis* **unfolding** *borel_def* **by** *auto*
qed

lemma *borel_closed*[*measurable* (*raw generic*)]:
assumes *closed A* **shows** *A* ∈ *sets borel*
proof −
have *space borel* − (− *A*) ∈ *sets borel*
using *assms* **unfolding** *closed_def* **by** (*blast intro*: *borel_open*)
thus *?thesis* **by** *simp*
qed

lemma *borel_singleton*[*measurable*]:
A ∈ *sets borel* ⟹ *insert x A* ∈ *sets* (*borel* :: *'a*::*t1_space measure*)

unfolding *insert_def* **by** (*rule sets.Un*) *auto*

lemma *sets_borel_eq_count_space*: *sets* (*borel* :: *'a*::{*countable*, *t2_space*} *measure*) = *count_space UNIV*
proof −
have ($\bigcup a{\in}A.\ \{a\}$) ∈ *sets borel* **for** *A* :: *'a set*
by (*intro sets.countable_UN'*) *auto*
then show *?thesis*
by *auto*
qed

lemma *borel_comp*[*measurable*]: *A* ∈ *sets borel* ⟹ − *A* ∈ *sets borel*
unfolding *Compl_eq_Diff_UNIV* **by** *simp*

lemma *borel_measurable_vimage*:
fixes *f* :: *'a* ⇒ *'x*::*t2_space*
assumes *borel*[*measurable*]: *f* ∈ *borel_measurable M*
shows *f* −‘ {*x*} ∩ *space M* ∈ *sets M*
by *simp*

lemma *borel_measurableI*:
fixes *f* :: *'a* ⇒ *'x*::*topological_space*
assumes ⋀*S*. *open S* ⟹ *f* −‘ *S* ∩ *space M* ∈ *sets M*
shows *f* ∈ *borel_measurable M*
unfolding *borel_def*
proof (*rule measurable_measure_of*, *simp_all*)
fix *S* :: *'x set* **assume** *open S* **thus** *f* −‘ *S* ∩ *space M* ∈ *sets M*
using *assms*[*of S*] **by** *simp*
qed

lemma *borel_measurable_const*:
(λ*x*. *c*) ∈ *borel_measurable M*
by *auto*

lemma *borel_measurable_indicator*:
assumes *A*: *A* ∈ *sets M*
shows *indicator A* ∈ *borel_measurable M*
unfolding *indicator_def* [*abs_def*] **using** *A*
by (*auto intro*!: *measurable_If_set*)

lemma *borel_measurable_count_space*[*measurable* (*raw*)]:
f ∈ *borel_measurable* (*count_space S*)
unfolding *measurable_def* **by** *auto*

lemma *borel_measurable_indicator'*[*measurable* (*raw*)]:
assumes [*measurable*]: {*x*∈*space M*. *f x* ∈ *A x*} ∈ *sets M*
shows (λ*x*. *indicator* (*A x*) (*f x*)) ∈ *borel_measurable M*
unfolding *indicator_def*[*abs_def*]
by (*auto intro*!: *measurable_If*)

lemma *borel_measurable_indicator_iff*:
(*indicator A* :: *'a* ⇒ *'x*::{*t1_space*, *zero_neq_one*}) ∈ *borel_measurable M* ⟷ *A* ∩ *space M* ∈ *sets M*
(**is** *?I* ∈ *borel_measurable M* ⟷ _)
proof
assume *?I* ∈ *borel_measurable M*
then have *?I* −‘ {*1*} ∩ *space M* ∈ *sets M*
unfolding *measurable_def* **by** *auto*
also have *?I* −‘ {*1*} ∩ *space M* = *A* ∩ *space M*
unfolding *indicator_def* [*abs_def*] **by** *auto*
finally show *A* ∩ *space M* ∈ *sets M* **.**
next
assume *A* ∩ *space M* ∈ *sets M*
moreover have *?I* ∈ *borel_measurable M* ⟷
(*indicator* (*A* ∩ *space M*) :: *'a* ⇒ *'x*) ∈ *borel_measurable M*
by (*intro measurable_cong*) (*auto simp*: *indicator_def*)
ultimately show *?I* ∈ *borel_measurable M* **by** *auto*
qed

lemma *borel_measurable_subalgebra*:
assumes *sets N* ⊆ *sets M space N* = *space M f* ∈ *borel_measurable N*
shows *f* ∈ *borel_measurable M*
using *assms* **unfolding** *measurable_def* **by** *auto*

lemma *borel_measurable_restrict_space_iff_ereal*:
fixes *f* :: *'a* ⇒ *ereal*
assumes Ω[*measurable*, *simp*]: Ω ∩ *space M* ∈ *sets M*
shows *f* ∈ *borel_measurable* (*restrict_space M* Ω) ⟷
(λ*x*. *f x* ∗ *indicator* Ω *x*) ∈ *borel_measurable M*
by (*subst measurable_restrict_space_iff*)
(*auto simp*: *indicator_def of_bool_def if_distrib*[**where** *f*=λ*x*. *a* ∗ *x* **for** *a*] *cong del*: *if_weak_cong*)

lemma *borel_measurable_restrict_space_iff_ennreal*:
fixes *f* :: *'a* ⇒ *ennreal*
assumes Ω[*measurable*, *simp*]: Ω ∩ *space M* ∈ *sets M*
shows *f* ∈ *borel_measurable* (*restrict_space M* Ω) ⟷
(λ*x*. *f x* ∗ *indicator* Ω *x*) ∈ *borel_measurable M*
by (*subst measurable_restrict_space_iff*)
(*auto simp*: *indicator_def of_bool_def if_distrib*[**where** *f*=λ*x*. *a* ∗ *x* **for** *a*] *cong del*: *if_weak_cong*)

lemma *borel_measurable_restrict_space_iff*:
fixes *f* :: *'a* ⇒ *'b*::*real_normed_vector*
assumes Ω[*measurable*, *simp*]: Ω ∩ *space M* ∈ *sets M*
shows *f* ∈ *borel_measurable* (*restrict_space M* Ω) ⟷
(λ*x*. *indicator* Ω *x* ∗$_R$ *f x*) ∈ *borel_measurable M*
by (*subst measurable_restrict_space_iff*)

(*auto simp*: *indicator_def of_bool_def if_distrib*[**where** $f{=}\lambda x.\ x *_R a$ **for** *a*] *ac_simps*
cong del: *if_weak_cong*)

lemma *cbox_borel*[*measurable*]: *cbox a b* ∈ *sets borel*
by (*auto intro*: *borel_closed*)

lemma *box_borel*[*measurable*]: *box a b* ∈ *sets borel*
by (*auto intro*: *borel_open*)

lemma *borel_compact*: *compact* (*A*::'*a*::*t2_space set*) ⟹ *A* ∈ *sets borel*
by (*auto intro*: *borel_closed dest*!: *compact_imp_closed*)

lemma *borel_sigma_sets_subset*:
A ⊆ *sets borel* ⟹ *sigma_sets UNIV A* ⊆ *sets borel*
using *sets.sigma_sets_subset*[*of A borel*] **by** *simp*

lemma *borel_eq_sigmaI1*:
fixes *F* :: '*i* ⇒ '*a*::*topological_space set* **and** *X* :: '*a*::*topological_space set set*
assumes *borel_eq*: *borel* = *sigma UNIV X*
assumes *X*: ⋀*x*. *x* ∈ *X* ⟹ *x* ∈ *sets* (*sigma UNIV* (*F* ‘ *A*))
assumes *F*: ⋀*i*. *i* ∈ *A* ⟹ *F i* ∈ *sets borel*
shows *borel* = *sigma UNIV* (*F* ‘ *A*)
unfolding *borel_def*
proof (*intro sigma_eqI antisym*)
have *borel_rev_eq*: *sigma_sets UNIV* {*S*::'*a set*. *open S*} = *sets borel*
unfolding *borel_def* **by** *simp*
also have ... = *sigma_sets UNIV X*
unfolding *borel_eq* **by** *simp*
also have ... ⊆ *sigma_sets UNIV* (*F*‘*A*)
using *X* **by** (*intro sigma_algebra.sigma_sets_subset*[*OF sigma_algebra_sigma_sets*]) *auto*
finally show *sigma_sets UNIV* {*S*. *open S*} ⊆ *sigma_sets UNIV* (*F*‘*A*) .
show *sigma_sets UNIV* (*F*‘*A*) ⊆ *sigma_sets UNIV* {*S*. *open S*}
unfolding *borel_rev_eq* **using** *F* **by** (*intro borel_sigma_sets_subset*) *auto*
qed *auto*

lemma *borel_eq_sigmaI2*:
fixes *F* :: '*i* ⇒ '*j* ⇒ '*a*::*topological_space set*
and *G* :: '*l* ⇒ '*k* ⇒ '*a*::*topological_space set*
assumes *borel_eq*: *borel* = *sigma UNIV* ((λ(*i*, *j*). *G i j*)‘*B*)
assumes *X*: ⋀*i j*. (*i*, *j*) ∈ *B* ⟹ *G i j* ∈ *sets* (*sigma UNIV* ((λ(*i*, *j*). *F i j*) ‘ *A*))
assumes *F*: ⋀*i j*. (*i*, *j*) ∈ *A* ⟹ *F i j* ∈ *sets borel*
shows *borel* = *sigma UNIV* ((λ(*i*, *j*). *F i j*) ‘ *A*)
using *assms*
by (*intro borel_eq_sigmaI1*[**where** $X{=}(\lambda(i, j).\ G\ i\ j)$ ‘ *B* **and** $F{=}(\lambda(i, j).\ F\ i\ j)$]) *auto*

lemma *borel_eq_sigmaI3*:

fixes $F :: 'i \Rightarrow 'j \Rightarrow 'a$::*topological_space set* **and** $X :: 'a$::*topological_space set set*
assumes *borel_eq*: $borel = sigma\ UNIV\ X$
assumes X: $\bigwedge x.\ x \in X \Longrightarrow x \in sets\ (sigma\ UNIV\ ((\lambda(i, j).\ F\ i\ j)\ `\ A))$
assumes F: $\bigwedge i\ j.\ (i, j) \in A \Longrightarrow F\ i\ j \in sets\ borel$
shows $borel = sigma\ UNIV\ ((\lambda(i, j).\ F\ i\ j)\ `\ A)$
using *assms* **by** (*intro borel_eq_sigmaI1*[**where** $X=X$ **and** $F=(\lambda(i, j).\ F\ i\ j)$]) *auto*

lemma *borel_eq_sigmaI4*:
fixes $F :: 'i \Rightarrow 'a$::*topological_space set*
and $G :: 'l \Rightarrow 'k \Rightarrow 'a$::*topological_space set*
assumes *borel_eq*: $borel = sigma\ UNIV\ ((\lambda(i, j).\ G\ i\ j)`A)$
assumes X: $\bigwedge i\ j.\ (i, j) \in A \Longrightarrow G\ i\ j \in sets\ (sigma\ UNIV\ (range\ F))$
assumes F: $\bigwedge i.\ F\ i \in sets\ borel$
shows $borel = sigma\ UNIV\ (range\ F)$
using *assms* **by** (*intro borel_eq_sigmaI1*[**where** $X=(\lambda(i, j).\ G\ i\ j)\ `\ A$ **and** $F=F$]) *auto*

lemma *borel_eq_sigmaI5*:
fixes $F :: 'i \Rightarrow 'j \Rightarrow 'a$::*topological_space set* **and** $G :: 'l \Rightarrow 'a$::*topological_space set*
assumes *borel_eq*: $borel = sigma\ UNIV\ (range\ G)$
assumes X: $\bigwedge i.\ G\ i \in sets\ (sigma\ UNIV\ (range\ (\lambda(i, j).\ F\ i\ j)))$
assumes F: $\bigwedge i\ j.\ F\ i\ j \in sets\ borel$
shows $borel = sigma\ UNIV\ (range\ (\lambda(i, j).\ F\ i\ j))$
using *assms* **by** (*intro borel_eq_sigmaI1*[**where** $X=range\ G$ **and** $F=(\lambda(i, j).\ F\ i\ j)$]) *auto*

theorem *second_countable_borel_measurable*:
fixes $X :: 'a$::*second_countable_topology set set*
assumes *eq*: $open = generate\_topology\ X$
shows $borel = sigma\ UNIV\ X$
unfolding *borel_def*
proof (*intro sigma_eqI sigma_sets_eqI*)
interpret X: *sigma_algebra UNIV sigma_sets UNIV X*
by (*rule sigma_algebra_sigma_sets*) *simp*

fix $S :: 'a\ set$ **assume** $S \in Collect\ open$
then have *generate_topology X S*
by (*auto simp*: *eq*)
then show $S \in sigma\_sets\ UNIV\ X$
proof *induction*
case (*UN K*)
then have K: $\bigwedge k.\ k \in K \Longrightarrow open\ k$
unfolding *eq* **by** *auto*
from *ex_countable_basis* **obtain** $B :: 'a\ set\ set$ **where**
B: $\bigwedge b.\ b \in B \Longrightarrow open\ b\ \bigwedge X.\ open\ X \Longrightarrow \exists\, b \subseteq B.\ (\bigcup b) = X$ **and** *countable B*

by (*auto simp*: *topological_basis_def*)
from *B*(*2*)[*OF K*] **obtain** *m* **where** *m*: $\bigwedge k.\ k \in K \Longrightarrow m\ k \subseteq B\ \bigwedge k.\ k \in K \Longrightarrow \bigcup (m\ k) = k$
by *metis*
define *U* **where** $U = (\bigcup k{\in}K.\ m\ k)$
with *m* **have** *countable U*
by (*intro countable_subset*[*OF _* ‹*countable B*›]) *auto*
have $\bigcup U = (\bigcup A{\in}U.\ A)$ **by** *simp*
also have $\ldots = \bigcup K$
unfolding *U_def UN_simps* **by** (*simp add*: *m*)
finally have $\bigcup U = \bigcup K$ **.**

have $\forall\, b{\in}U.\ \exists\, k{\in}K.\ b \subseteq k$
using *m* **by** (*auto simp*: *U_def*)
then obtain *u* **where** *u*: $\bigwedge b.\ b \in U \Longrightarrow u\ b \in K$ **and** $\bigwedge b.\ b \in U \Longrightarrow b \subseteq u\ b$
by *metis*
then have $(\bigcup b{\in}U.\ u\ b) \subseteq \bigcup K\ \bigcup U \subseteq (\bigcup b{\in}U.\ u\ b)$
by *auto*
then have $\bigcup K = (\bigcup b{\in}U.\ u\ b)$
unfolding ‹$\bigcup U = \bigcup K$› **by** *auto*
also have $\ldots \in$ *sigma_sets UNIV X*
using *u UN* **by** (*intro X.countable_UN′* ‹*countable U*›) *auto*
finally show $\bigcup K \in$ *sigma_sets UNIV X* **.**
qed *auto*
qed (*auto simp*: *eq intro*: *generate_topology.Basis*)

lemma *borel_eq_closed*: *borel* = *sigma UNIV* (*Collect closed*)
unfolding *borel_def*
proof (*intro sigma_eqI sigma_sets_eqI*, *safe*)
fix *x* :: *′a set* **assume** *open x*
hence *x* = *UNIV* − (*UNIV* − *x*) **by** *auto*
also have $\ldots \in$ *sigma_sets UNIV* (*Collect closed*)
by (*force intro*: *sigma_sets.Compl simp*: ‹*open x*›)
finally show *x* ∈ *sigma_sets UNIV* (*Collect closed*) **by** *simp*
next
fix *x* :: *′a set* **assume** *closed x*
hence *x* = *UNIV* − (*UNIV* − *x*) **by** *auto*
also have $\ldots \in$ *sigma_sets UNIV* (*Collect open*)
by (*force intro*: *sigma_sets.Compl simp*: ‹*closed x*›)
finally show *x* ∈ *sigma_sets UNIV* (*Collect open*) **by** *simp*
qed *simp_all*

proposition *borel_eq_countable_basis*:
fixes *B*::*′a*::*topological_space set set*
assumes *countable B*
assumes *topological_basis B*
shows *borel* = *sigma UNIV B*
unfolding *borel_def*
proof (*intro sigma_eqI sigma_sets_eqI*, *safe*)

```
  interpret countable_basis open B using assms by (rule countable_basis_openI)
  fix X::'a set assume open X
  from open_countable_basisE[OF this] obtain B' where B': B' ⊆ B X = ⋃ B'
.
  then show X ∈ sigma_sets UNIV B
    by (blast intro: sigma_sets_UNION ‹countable B› countable_subset)
next
  fix b assume b ∈ B
  hence open b by (rule topological_basis_open[OF assms(2)])
  thus b ∈ sigma_sets UNIV (Collect open) by auto
qed simp_all

lemma borel_measurable_continuous_on_restrict:
  fixes f :: 'a::topological_space ⇒ 'b::topological_space
  assumes f: continuous_on A f
  shows f ∈ borel_measurable (restrict_space borel A)
proof (rule borel_measurableI)
  fix S :: 'b set assume open S
  with f obtain T where f -` S ∩ A = T ∩ A open T
    by (metis continuous_on_open_invariant)
  then show f -` S ∩ space (restrict_space borel A) ∈ sets (restrict_space borel
A)
    by (force simp add: sets_restrict_space space_restrict_space)
qed

lemma borel_measurable_continuous_onI: continuous_on UNIV f ⟹ f ∈ borel_measurable
borel
  by (drule borel_measurable_continuous_on_restrict) simp

lemma borel_measurable_continuous_on_if:
  A ∈ sets borel ⟹ continuous_on A f ⟹ continuous_on (- A) g ⟹
    (λx. if x ∈ A then f x else g x) ∈ borel_measurable borel
  by (auto simp add: measurable_If_restrict_space_iff Collect_neg_eq
        intro!: borel_measurable_continuous_on_restrict)

lemma borel_measurable_continuous_countable_exceptions:
  fixes f :: 'a::t1_space ⇒ 'b::topological_space
  assumes X: countable X
  assumes continuous_on (- X) f
  shows f ∈ borel_measurable borel
proof (rule measurable_discrete_difference[OF _ X])
  have X ∈ sets borel
    by (rule sets.countable[OF _ X]) auto
  then show (λx. if x ∈ X then undefined else f x) ∈ borel_measurable borel
    by (intro borel_measurable_continuous_on_if assms continuous_intros)
qed auto

lemma borel_measurable_continuous_on:
  assumes f: continuous_on UNIV f and g: g ∈ borel_measurable M
```

shows $(\lambda x.\ f\ (g\ x)) \in$ *borel_measurable M*
using *measurable_comp*[*OF g borel_measurable_continuous_onI*[*OF f*]] **by** (*simp add*: *comp_def*)

lemma *borel_measurable_continuous_on_indicator*:
fixes *f g* :: *'a*::*topological_space* ⇒ *'b*::*real_normed_vector*
shows *A* ∈ *sets borel* ⟹ *continuous_on A f* ⟹ $(\lambda x.\ \mathit{indicator}\ A\ x *_R f\ x) \in$ *borel_measurable borel*
by (*subst borel_measurable_restrict_space_iff*[*symmetric*])
(*auto intro*: *borel_measurable_continuous_on_restrict*)

lemma *borel_measurable_Pair*[*measurable* (*raw*)]:
fixes *f* :: *'a* ⇒ *'b*::*second_countable_topology* **and** *g* :: *'a* ⇒ *'c*::*second_countable_topology*
assumes *f*[*measurable*]: *f* ∈ *borel_measurable M*
assumes *g*[*measurable*]: *g* ∈ *borel_measurable M*
shows $(\lambda x.\ (f\ x,\ g\ x)) \in$ *borel_measurable M*
proof (*subst borel_eq_countable_basis*)
let *?B* = *SOME B*::*'b set set*. *countable B* ∧ *topological_basis B*
let *?C* = *SOME B*::*'c set set*. *countable B* ∧ *topological_basis B*
let *?P* = $(\lambda(b,\ c).\ b \times c)$ ‘ (*?B* × *?C*)
show *countable ?P topological_basis ?P*
by (*auto intro*!: *countable_basis topological_basis_prod is_basis*)

show $(\lambda x.\ (f\ x,\ g\ x)) \in$ *measurable M* (*sigma UNIV ?P*)
proof (*rule measurable_measure_of*)
fix *S* **assume** *S* ∈ *?P*
then obtain *b c* **where** *b* ∈ *?B c* ∈ *?C* **and** *S*: *S* = *b* × *c* **by** *auto*
then have *borel*: *open b open c*
by (*auto intro*: *is_basis topological_basis_open*)
have $(\lambda x.\ (f\ x,\ g\ x)) -‘\ S \cap \mathit{space}\ M = (f -‘\ b \cap \mathit{space}\ M) \cap (g -‘\ c \cap \mathit{space}\ M)$
unfolding *S* **by** *auto*
also have ... ∈ *sets M*
using *borel* **by** *simp*
finally show $(\lambda x.\ (f\ x,\ g\ x)) -‘\ S \cap \mathit{space}\ M \in \mathit{sets}\ M$ **.**
qed *auto*
qed

lemma *borel_measurable_continuous_Pair*:
fixes *f* :: *'a* ⇒ *'b*::*second_countable_topology* **and** *g* :: *'a* ⇒ *'c*::*second_countable_topology*
assumes [*measurable*]: *f* ∈ *borel_measurable M*
assumes [*measurable*]: *g* ∈ *borel_measurable M*
assumes *H*: *continuous_on UNIV* $(\lambda x.\ H\ (\mathit{fst}\ x)\ (\mathit{snd}\ x))$
shows $(\lambda x.\ H\ (f\ x)\ (g\ x)) \in$ *borel_measurable M*
proof −
have *eq*: $(\lambda x.\ H\ (f\ x)\ (g\ x)) = (\lambda x.\ (\lambda x.\ H\ (\mathit{fst}\ x)\ (\mathit{snd}\ x))\ (f\ x,\ g\ x))$ **by** *auto*
show *?thesis*
unfolding *eq* **by** (*rule borel_measurable_continuous_on*[*OF H*]) *auto*
qed

6.4.2 Borel spaces on order topologies

lemma [*measurable*]:
fixes *a b* :: *'a*::*linorder_topology*
shows *lessThan_borel*: {..< *a*} ∈ *sets borel*
and *greaterThan_borel*: {*a* <..} ∈ *sets borel*
and *greaterThanLessThan_borel*: {*a*<..<*b*} ∈ *sets borel*
and *atMost_borel*: {..*a*} ∈ *sets borel*
and *atLeast_borel*: {*a*..} ∈ *sets borel*
and *atLeastAtMost_borel*: {*a*..*b*} ∈ *sets borel*
and *greaterThanAtMost_borel*: {*a*<..*b*} ∈ *sets borel*
and *atLeastLessThan_borel*: {*a*..<*b*} ∈ *sets borel*
unfolding *greaterThanAtMost_def atLeastLessThan_def*
by (*blast intro*: *borel_open borel_closed open_lessThan open_greaterThan open_greaterThanLessThan closed_atMost closed_atLeast closed_atLeastAtMost*)+

lemma *borel_Iio*:
borel = *sigma UNIV* (*range lessThan* :: *'a*::{*linorder_topology*, *second_countable_topology*} *set set*)
unfolding *second_countable_borel_measurable*[*OF open_generated_order*]
proof (*intro sigma_eqI sigma_sets_eqI*)
obtain *D* :: *'a set* **where** *D*: *countable D* ⋀*X*. *open X* ⟹ *X* ≠ {} ⟹ ∃ *d*∈*D*. *d* ∈ *X*
by (*rule countable_dense_setE*) *blast*

interpret *L*: *sigma_algebra UNIV sigma_sets UNIV* (*range lessThan*)
by (*rule sigma_algebra_sigma_sets*) *simp*

fix *A* :: *'a set* **assume** *A* ∈ *range lessThan* ∪ *range greaterThan*
then obtain *y* **where** *A* = {*y* <..} ∨ *A* = {..< *y*}
by *blast*
then show *A* ∈ *sigma_sets UNIV* (*range lessThan*)
proof
assume *A*: *A* = {*y* <..}
show *?thesis*
proof *cases*
assume ∀ *x*>*y*. ∃ *d*. *y* < *d* ∧ *d* < *x*
with *D*(*2*)[*of* {*y* <..< *x*} **for** *x*] **have** ∀ *x*>*y*. ∃ *d*∈*D*. *y* < *d* ∧ *d* < *x*
by (*auto simp*: *set_eq_iff*)
then have *A* = *UNIV* − (⋂ *d*∈{*d*∈*D*. *y* < *d*}. {..< *d*})
by (*auto simp*: *A*) (*metis less_asym*)
also have ... ∈ *sigma_sets UNIV* (*range lessThan*)
using *D*(*1*) **by** (*intro L.Diff L.top L.countable_INT''*) *auto*
finally show *?thesis* **.**
next
assume ¬ (∀ *x*>*y*. ∃ *d*. *y* < *d* ∧ *d* < *x*)
then obtain *x* **where** *y* < *x* ⋀*d*. *y* < *d* ⟹ ¬ *d* < *x*
by *auto*
then have *A* = *UNIV* − {..< *x*}
unfolding *A* **by** (*auto simp*: *not_less*[*symmetric*])

```
    also have ... ∈ sigma_sets UNIV (range lessThan)
      by auto
    finally show ?thesis .
  qed
 qed auto
qed auto

lemma borel_Ioi:
 borel = sigma UNIV (range greaterThan :: 'a::{linorder_topology, second_countable_topology}
set set)
 unfolding second_countable_borel_measurable[OF open_generated_order]
proof (intro sigma_eqI sigma_sets_eqI)
 obtain D :: 'a set where D: countable D ⋀X. open X ⟹ X ≠ {} ⟹ ∃ d∈D.
d ∈ X
   by (rule countable_dense_setE) blast

 interpret L: sigma_algebra UNIV sigma_sets UNIV (range greaterThan)
   by (rule sigma_algebra_sigma_sets) simp

 fix A :: 'a set assume A ∈ range lessThan ∪ range greaterThan
 then obtain y where A = {y <..} ∨ A = {..< y}
   by blast
 then show A ∈ sigma_sets UNIV (range greaterThan)
 proof
   assume A: A = {..< y}
   show ?thesis
   proof cases
     assume ∀ x<y. ∃ d. x < d ∧ d < y
     with D(2)[of {x <..< y} for x] have ∀ x<y. ∃ d∈D. x < d ∧ d < y
       by (auto simp: set_eq_iff)
     then have A = UNIV − (⋂ d∈{d∈D. d < y}. {d <..})
       by (auto simp: A) (metis less_asym)
     also have ... ∈ sigma_sets UNIV (range greaterThan)
       using D(1) by (intro L.Diff L.top L.countable_INT'') auto
     finally show ?thesis .
   next
     assume ¬ (∀ x<y. ∃ d. x < d ∧ d < y)
     then obtain x where x < y ⋀d. y > d ⟹ x ≥ d
       by (auto simp: not_less[symmetric])
     then have A = UNIV − {x <..}
       unfolding A Compl_eq_Diff_UNIV[symmetric] by auto
     also have ... ∈ sigma_sets UNIV (range greaterThan)
       by auto
     finally show ?thesis .
   qed
 qed auto
qed auto

lemma borel_measurableI_less:
```

fixes $f :: 'a \Rightarrow 'b::\{linorder\_topology, second\_countable\_topology\}$
shows $(\bigwedge y.\ \{x \in space\ M.\ f\ x < y\} \in sets\ M) \Longrightarrow f \in borel\_measurable\ M$
unfolding *borel_Iio*
by (*rule measurable_measure_of*) (*auto simp*: *Int_def conj_commute*)

lemma *borel_measurableI_greater*:
fixes $f :: 'a \Rightarrow 'b::\{linorder\_topology, second\_countable\_topology\}$
shows $(\bigwedge y.\ \{x \in space\ M.\ y < f\ x\} \in sets\ M) \Longrightarrow f \in borel\_measurable\ M$
unfolding *borel_Ioi*
by (*rule measurable_measure_of*) (*auto simp*: *Int_def conj_commute*)

lemma *borel_measurableI_le*:
fixes $f :: 'a \Rightarrow 'b::\{linorder\_topology, second\_countable\_topology\}$
shows $(\bigwedge y.\ \{x \in space\ M.\ f\ x \leq y\} \in sets\ M) \Longrightarrow f \in borel\_measurable\ M$
by (*rule borel_measurableI_greater*) (*auto simp*: *not_le*[*symmetric*])

lemma *borel_measurableI_ge*:
fixes $f :: 'a \Rightarrow 'b::\{linorder\_topology, second\_countable\_topology\}$
shows $(\bigwedge y.\ \{x \in space\ M.\ y \leq f\ x\} \in sets\ M) \Longrightarrow f \in borel\_measurable\ M$
by (*rule borel_measurableI_less*) (*auto simp*: *not_le*[*symmetric*])

lemma *borel_measurable_less*[*measurable*]:
fixes $f :: 'a \Rightarrow 'b::\{second\_countable\_topology, linorder\_topology\}$
assumes $f \in borel\_measurable\ M$
assumes $g \in borel\_measurable\ M$
shows $\{w \in space\ M.\ f\ w < g\ w\} \in sets\ M$
proof −
have $\{w \in space\ M.\ f\ w < g\ w\} = (\lambda x.\ (f\ x,\ g\ x)) -` \{x.\ fst\ x < snd\ x\} \cap space\ M$
by *auto*
also have $\ldots \in sets\ M$
by (*intro measurable_sets*[*OF borel_measurable_Pair borel_open*, *OF assms open_Collect_less*]
continuous_intros)
finally show *?thesis* **.**
qed

lemma
fixes $f :: 'a \Rightarrow 'b::\{second\_countable\_topology, linorder\_topology\}$
assumes *f*[*measurable*]: $f \in borel\_measurable\ M$
assumes *g*[*measurable*]: $g \in borel\_measurable\ M$
shows *borel_measurable_le*[*measurable*]: $\{w \in space\ M.\ f\ w \leq g\ w\} \in sets\ M$
and *borel_measurable_eq*[*measurable*]: $\{w \in space\ M.\ f\ w = g\ w\} \in sets\ M$
and *borel_measurable_neq*: $\{w \in space\ M.\ f\ w \neq g\ w\} \in sets\ M$
unfolding *eq_iff not_less*[*symmetric*]
by *measurable*

lemma *borel_measurable_SUP*[*measurable* (*raw*)]:
fixes $F :: \_ \Rightarrow \_ \Rightarrow \_::\{complete\_linorder, linorder\_topology, second\_countable\_topology\}$

assumes [*simp*]: *countable I*
assumes [*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow F\ i \in$ *borel_measurable M*
shows $(\lambda x.\ SUP\ i{\in}I.\ F\ i\ x) \in$ *borel_measurable M*
by (*rule borel_measurableI_greater*) (*simp add*: *less_SUP_iff*)

lemma *borel_measurable_INF*[*measurable* (*raw*)]:
fixes $F :: \_ \Rightarrow \_ \Rightarrow \_ ::\{$*complete_linorder*, *linorder_topology*, *second_countable_topology*$\}$
assumes [*simp*]: *countable I*
assumes [*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow F\ i \in$ *borel_measurable M*
shows $(\lambda x.\ INF\ i{\in}I.\ F\ i\ x) \in$ *borel_measurable M*
by (*rule borel_measurableI_less*) (*simp add*: *INF_less_iff*)

lemma *borel_measurable_cSUP*[*measurable* (*raw*)]:
fixes $F :: \_ \Rightarrow \_ \Rightarrow {'a}::\{$*conditionally_complete_linorder*, *linorder_topology*, *second_countable_topology*$\}$
assumes [*simp*]: *countable I*
assumes [*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow F\ i \in$ *borel_measurable M*
assumes *bdd*: $\bigwedge x.\ x \in$ *space M* $\Longrightarrow$ *bdd_above* $((\lambda i.\ F\ i\ x)\ `\ I)$
shows $(\lambda x.\ SUP\ i{\in}I.\ F\ i\ x) \in$ *borel_measurable M*
proof *cases*
assume $I = \{\}$ **then show** *?thesis*
unfolding ‹$I = \{\}$› *image_empty* **by** *simp*
next
assume $I \neq \{\}$
show *?thesis*
proof (*rule borel_measurableI_le*)
fix *y*
have $\{x \in$ *space M*$.\ \forall\, i{\in}I.\ F\ i\ x \leq y\} \in$ *sets M*
by *measurable*
also have $\{x \in$ *space M*$.\ \forall\, i{\in}I.\ F\ i\ x \leq y\} = \{x \in$ *space M*$.\ (SUP\ i{\in}I.\ F\ i\ x) \leq y\}$
by (*simp add*: *cSUP_le_iff* ‹$I \neq \{\}$› *bdd cong*: *conj_cong*)
finally show $\{x \in$ *space M*$.\ (SUP\ i{\in}I.\ F\ i\ x) \leq\ y\} \in$ *sets M* **.**
qed
qed

lemma *borel_measurable_cINF*[*measurable* (*raw*)]:
fixes $F :: \_ \Rightarrow \_ \Rightarrow {'a}::\{$*conditionally_complete_linorder*, *linorder_topology*, *second_countable_topology*$\}$
assumes [*simp*]: *countable I*
assumes [*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow F\ i \in$ *borel_measurable M*
assumes *bdd*: $\bigwedge x.\ x \in$ *space M* $\Longrightarrow$ *bdd_below* $((\lambda i.\ F\ i\ x)\ `\ I)$
shows $(\lambda x.\ INF\ i{\in}I.\ F\ i\ x) \in$ *borel_measurable M*
proof *cases*
assume $I = \{\}$ **then show** *?thesis*
unfolding ‹$I = \{\}$› *image_empty* **by** *simp*
next
assume $I \neq \{\}$
show *?thesis*

proof (*rule borel_measurableI_ge*)
fix *y*
have $\{x \in space\ M.\ \forall i{\in}I.\ y \leq F\ i\ x\} \in sets\ M$
by *measurable*
also have $\{x \in space\ M.\ \forall i{\in}I.\ y \leq F\ i\ x\} = \{x \in space\ M.\ y \leq (INF\ i{\in}I.\ F\ i\ x)\}$
by (*simp add*: *le_cINF_iff* ‹$I \neq \{\}$› *bdd cong*: *conj_cong*)
finally show $\{x \in space\ M.\ y \leq (INF\ i{\in}I.\ F\ i\ x)\} \in sets\ M$ **.**
qed
qed

lemma *borel_measurable_lfp*[*consumes 1*, *case_names continuity step*]:
fixes $F :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b{::}\{complete\_linorder, linorder\_topology, second\_countable\_topology\})$
assumes *sup_continuous F*
assumes $*$: $\bigwedge f.\ f \in borel\_measurable\ M \Longrightarrow F\ f \in borel\_measurable\ M$
shows *lfp F* $\in$ *borel_measurable M*
proof −
{ fix *i* **have** $((F \frown\frown i)\ bot) \in borel\_measurable\ M$
by (*induct i*) (*auto intro*!: $*$) **}**
then have $(\lambda x.\ SUP\ i.\ (F \frown\frown i)\ bot\ x) \in borel\_measurable\ M$
by *measurable*
also have $(\lambda x.\ SUP\ i.\ (F \frown\frown i)\ bot\ x) = (SUP\ i.\ (F \frown\frown i)\ bot)$
by (*auto simp add*: *image_comp*)
also have $(SUP\ i.\ (F \frown\frown i)\ bot) = lfp\ F$
by (*rule sup_continuous_lfp*[*symmetric*]) *fact*
finally show *?thesis* **.**
qed

lemma *borel_measurable_gfp*[*consumes 1*, *case_names continuity step*]:
fixes $F :: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b{::}\{complete\_linorder, linorder\_topology, second\_countable\_topology\})$
assumes *inf_continuous F*
assumes $*$: $\bigwedge f.\ f \in borel\_measurable\ M \Longrightarrow F\ f \in borel\_measurable\ M$
shows *gfp F* $\in$ *borel_measurable M*
proof −
{ fix *i* **have** $((F \frown\frown i)\ top) \in borel\_measurable\ M$
by (*induct i*) (*auto intro*!: $*$ *simp*: *bot_fun_def*) **}**
then have $(\lambda x.\ INF\ i.\ (F \frown\frown i)\ top\ x) \in borel\_measurable\ M$
by *measurable*
also have $(\lambda x.\ INF\ i.\ (F \frown\frown i)\ top\ x) = (INF\ i.\ (F \frown\frown i)\ top)$
by (*auto simp add*: *image_comp*)
also have $\ldots = gfp\ F$
by (*rule inf_continuous_gfp*[*symmetric*]) *fact*
finally show *?thesis* **.**
qed

lemma *borel_measurable_max*[*measurable* (*raw*)]:
$f \in borel\_measurable\ M \Longrightarrow g \in borel\_measurable\ M \Longrightarrow (\lambda x.\ max\ (g\ x)\ (f\ x)$

:: 'b::{*second_countable_topology*, *linorder_topology*}) ∈ *borel_measurable M*
by (*rule borel_measurableI_less*) *simp*

lemma *borel_measurable_min*[*measurable* (*raw*)]:
f ∈ *borel_measurable M* ⟹ *g* ∈ *borel_measurable M* ⟹ (λ*x*. *min* (*g x*) (*f x*)
:: 'b::{*second_countable_topology*, *linorder_topology*}) ∈ *borel_measurable M*
by (*rule borel_measurableI_greater*) *simp*

lemma *borel_measurable_Min*[*measurable* (*raw*)]:
finite I ⟹ (⋀*i*. *i* ∈ *I* ⟹ *f i* ∈ *borel_measurable M*) ⟹ (λ*x*. *Min* ((λ*i*. *f i x*) ' *I*)
:: 'b::{*second_countable_topology*, *linorder_topology*}) ∈ *borel_measurable M*
proof (*induct I rule*: *finite_induct*)
case (*insert i I*) **then show** *?case*
by (*cases I* = {}) *auto*
qed *auto*

lemma *borel_measurable_Max*[*measurable* (*raw*)]:
finite I ⟹ (⋀*i*. *i* ∈ *I* ⟹ *f i* ∈ *borel_measurable M*) ⟹ (λ*x*. *Max* ((λ*i*. *f*
i x) ' *I*) :: 'b::{*second_countable_topology*, *linorder_topology*}) ∈ *borel_measurable*
M
proof (*induct I rule*: *finite_induct*)
case (*insert i I*) **then show** *?case*
by (*cases I* = {}) *auto*
qed *auto*

lemma *borel_measurable_sup*[*measurable* (*raw*)]:
f ∈ *borel_measurable M* ⟹ *g* ∈ *borel_measurable M* ⟹ (λ*x*. *sup* (*g x*) (*f x*)
:: 'b::{*lattice*, *second_countable_topology*, *linorder_topology*}) ∈ *borel_measurable*
M
unfolding *sup_max* **by** *measurable*

lemma *borel_measurable_inf*[*measurable* (*raw*)]:
f ∈ *borel_measurable M* ⟹ *g* ∈ *borel_measurable M* ⟹ (λ*x*. *inf* (*g x*) (*f x*) ::
'b::{*lattice*, *second_countable_topology*, *linorder_topology*}) ∈ *borel_measurable M*
unfolding *inf_min* **by** *measurable*

lemma [*measurable* (*raw*)]:
fixes *f* :: *nat* ⇒ 'a ⇒ 'b::{*complete_linorder*, *second_countable_topology*, *linorder_topology*}
assumes ⋀*i*. *f i* ∈ *borel_measurable M*
shows *borel_measurable_liminf*: (λ*x*. *liminf* (λ*i*. *f i x*)) ∈ *borel_measurable M*
and *borel_measurable_limsup*: (λ*x*. *limsup* (λ*i*. *f i x*)) ∈ *borel_measurable M*
unfolding *liminf_SUP_INF limsup_INF_SUP* **using** *assms* **by** *auto*

lemma *measurable_convergent*[*measurable* (*raw*)]:
fixes *f* :: *nat* ⇒ 'a ⇒ 'b::{*complete_linorder*, *second_countable_topology*, *linorder_topology*}
assumes [*measurable*]: ⋀*i*. *f i* ∈ *borel_measurable M*
shows *Measurable.pred M* (λ*x*. *convergent* (λ*i*. *f i x*))
unfolding *convergent_ereal* **by** *measurable*

lemma *sets_Collect_convergent*[*measurable*]:
fixes $f :: nat \Rightarrow 'a \Rightarrow 'b$::{*complete_linorder*, *second_countable_topology*, *linorder_topology*}
assumes *f*[*measurable*]: $\bigwedge i.\ f\ i \in borel\_measurable\ M$
shows $\{x \in space\ M.\ convergent\ (\lambda i.\ f\ i\ x)\} \in sets\ M$
by *measurable*

lemma *borel_measurable_lim*[*measurable* (*raw*)]:
fixes $f :: nat \Rightarrow 'a \Rightarrow 'b$::{*complete_linorder*, *second_countable_topology*, *linorder_topology*}
assumes [*measurable*]: $\bigwedge i.\ f\ i \in borel\_measurable\ M$
shows $(\lambda x.\ lim\ (\lambda i.\ f\ i\ x)) \in borel\_measurable\ M$
proof −
have $\bigwedge x.\ lim\ (\lambda i.\ f\ i\ x) = (if\ convergent\ (\lambda i.\ f\ i\ x)\ then\ limsup\ (\lambda i.\ f\ i\ x)\ else\ (THE\ i.\ False))$
by (*simp add*: *lim_def convergent_def convergent_limsup_cl*)
then show *?thesis*
by *simp*
qed

lemma *borel_measurable_LIMSEQ_order*:
fixes $u :: nat \Rightarrow 'a \Rightarrow 'b$::{*complete_linorder*, *second_countable_topology*, *linorder_topology*}
assumes u': $\bigwedge x.\ x \in space\ M \Longrightarrow (\lambda i.\ u\ i\ x) \longrightarrow u'\ x$
and *u*: $\bigwedge i.\ u\ i \in borel\_measurable\ M$
shows $u' \in borel\_measurable\ M$
proof −
have $\bigwedge x.\ x \in space\ M \Longrightarrow u'\ x = liminf\ (\lambda n.\ u\ n\ x)$
using u' **by** (*simp add*: *lim_imp_Liminf*[*symmetric*])
with *u* **show** *?thesis* **by** (*simp cong*: *measurable_cong*)
qed

6.4.3 Borel spaces on topological monoids

lemma *borel_measurable_add*[*measurable* (*raw*)]:
fixes $f\ g :: 'a \Rightarrow 'b$::{*second_countable_topology*, *topological_monoid_add*}
assumes *f*: $f \in borel\_measurable\ M$
assumes *g*: $g \in borel\_measurable\ M$
shows $(\lambda x.\ f\ x + g\ x) \in borel\_measurable\ M$
using *f g* **by** (*rule borel_measurable_continuous_Pair*) (*intro continuous_intros*)

lemma *borel_measurable_sum*[*measurable* (*raw*)]:
fixes $f :: 'c \Rightarrow 'a \Rightarrow 'b$::{*second_countable_topology*, *topological_comm_monoid_add*}
assumes $\bigwedge i.\ i \in S \Longrightarrow f\ i \in borel\_measurable\ M$
shows $(\lambda x.\ \sum i \in S.\ f\ i\ x) \in borel\_measurable\ M$
proof *cases*
assume *finite S*
thus *?thesis* **using** *assms* **by** *induct auto*
qed *simp*

lemma *borel_measurable_suminf_order*[*measurable* (*raw*)]:
fixes $f :: nat \Rightarrow 'a \Rightarrow 'b$::{*complete_linorder*, *second_countable_topology*, *linorder_topology*,

topological_comm_monoid_add}
assumes *f*[*measurable*]: $\bigwedge i.\ f\ i \in$ *borel_measurable M*
shows $(\lambda x.\ suminf\ (\lambda i.\ f\ i\ x)) \in$ *borel_measurable M*
unfolding *suminf_def sums_def*[*abs_def*] *lim_def*[*symmetric*] **by** *simp*

6.4.4 Borel spaces on Euclidean spaces

lemma *borel_measurable_inner*[*measurable* (*raw*)]:
fixes *f g* :: $'a \Rightarrow 'b$::{*second_countable_topology*, *real_inner*}
assumes $f \in$ *borel_measurable M*
assumes $g \in$ *borel_measurable M*
shows $(\lambda x.\ f\ x \cdot g\ x) \in$ *borel_measurable M*
using *assms*
by (*rule borel_measurable_continuous_Pair*) (*intro continuous_intros*)

notation
eucl_less (**infix** $<e$ *50*)

lemma *box_oc*: $\{x.\ a <e\ x \wedge x \leq b\} = \{x.\ a <e\ x\} \cap \{..b\}$
and *box_co*: $\{x.\ a \leq x \wedge x <e\ b\} = \{a..\} \cap \{x.\ x <e\ b\}$
by *auto*

lemma *eucl_ivals*[*measurable*]:
fixes *a b* :: $'a$::*ordered_euclidean_space*
shows $\{x.\ x <e\ a\} \in$ *sets borel*
and $\{x.\ a <e\ x\} \in$ *sets borel*
and $\{..a\} \in$ *sets borel*
and $\{a..\} \in$ *sets borel*
and $\{a..b\} \in$ *sets borel*
and $\{x.\ a <e\ x \wedge x \leq b\} \in$ *sets borel*
and $\{x.\ a \leq x \wedge\ x <e\ b\} \in$ *sets borel*
unfolding *box_oc box_co*
by (*auto intro*: *borel_open borel_closed*)

lemma
fixes *i* :: $'a$::{*second_countable_topology*, *real_inner*}
shows *hafspace_less_borel*: $\{x.\ a < x \cdot i\} \in$ *sets borel*
and *hafspace_greater_borel*: $\{x.\ x \cdot i < a\} \in$ *sets borel*
and *hafspace_less_eq_borel*: $\{x.\ a \leq x \cdot i\} \in$ *sets borel*
and *hafspace_greater_eq_borel*: $\{x.\ x \cdot i \leq a\} \in$ *sets borel*
by *simp_all*

lemma *borel_eq_box*:
borel = *sigma UNIV* (*range* (λ (*a*, *b*). *box a b* :: $'a$ :: *euclidean_space set*))
(**is** _ = *?SIGMA*)
proof (*rule borel_eq_sigmaI1*[*OF borel_def*])
fix *M* :: $'a$ *set* **assume** $M \in \{S.\ open\ S\}$
then have *open M* **by** *simp*
show $M \in$ *?SIGMA*

```
    apply (subst open_UNION_box[OF ‹open M›])
    apply (safe intro!: sets.countable_UN' countable_PiE countable_Collect)
    apply (auto intro: countable_rat)
    done
qed (auto simp: box_def)

lemma halfspace_gt_in_halfspace:
  assumes i: i ∈ A
  shows {x::'a. a < x · i} ∈
    sigma_sets UNIV ((λ (a, i). {x::'a::euclidean_space. x · i < a}) ` (UNIV ×
A))
  (is ?set ∈ ?SIGMA)
proof −
  interpret sigma_algebra UNIV ?SIGMA
    by (intro sigma_algebra_sigma_sets) simp_all
  have ∗: ?set = (⋃ n. UNIV − {x::'a. x · i < a + 1 / real (Suc n)})
  proof (safe, simp_all add: not_less del: of_nat_Suc)
    fix x :: 'a assume a < x · i
    with reals_Archimedean[of x · i − a]
    obtain n where a + 1 / real (Suc n) < x · i
      by (auto simp: field_simps)
    then show ∃ n. a + 1 / real (Suc n) ≤ x · i
      by (blast intro: less_imp_le)
  next
    fix x n
    have a < a + 1 / real (Suc n) by auto
    also assume ... ≤ x
    finally show a < x .
  qed
  show ?set ∈ ?SIGMA unfolding ∗
    by (auto intro!: Diff sigma_sets_Inter i)
qed

lemma borel_eq_halfspace_less:
  borel = sigma UNIV ((λ(a, i). {x::'a::euclidean_space. x · i < a}) ` (UNIV ×
Basis))
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI2[OF borel_eq_box])
  fix a b :: 'a
  have box a b = {x∈space ?SIGMA. ∀ i∈Basis. a · i < x · i ∧ x · i < b · i}
    by (auto simp: box_def)
  also have ... ∈ sets ?SIGMA
   by (intro sets.sets_Collect_conj sets.sets_Collect_finite_All sets.sets_Collect_const)
       (auto intro!: halfspace_gt_in_halfspace countable_PiE countable_rat)
  finally show box a b ∈ sets ?SIGMA .
qed auto

lemma borel_eq_halfspace_le:
  borel = sigma UNIV ((λ (a, i). {x::'a::euclidean_space. x · i ≤ a}) ` (UNIV ×
```

```
Basis))
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI2[OF borel_eq_halfspace_less])
  fix a :: real and i :: 'a assume (a, i) ∈ UNIV × Basis
  then have i: i ∈ Basis by auto
  have ∗: {x::'a. x•i < a} = (⋃n. {x. x•i ≤ a − 1/real (Suc n)})
  proof (safe, simp_all del: of_nat_Suc)
    fix x::'a assume ∗: x•i < a
    with reals_Archimedean[of a − x•i]
    obtain n where x • i < a − 1 / (real (Suc n))
      by (auto simp: field_simps)
    then show ∃n. x • i ≤ a − 1 / (real (Suc n))
      by (blast intro: less_imp_le)
  next
    fix x::'a and n
    assume x•i ≤ a − 1 / real (Suc n)
    also have ... < a by auto
    finally show x•i < a .
  qed
  show {x. x•i < a} ∈ ?SIGMA unfolding ∗
    by (intro sets.countable_UN) (auto intro: i)
qed auto

lemma borel_eq_halfspace_ge:
  borel = sigma UNIV ((λ (a, i). {x::'a::euclidean_space. a ≤ x • i}) ‘ (UNIV ×
Basis))
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI2[OF borel_eq_halfspace_less])
  fix a :: real and i :: 'a assume i: (a, i) ∈ UNIV × Basis
  have ∗: {x::'a. x•i < a} = space ?SIGMA − {x::'a. a ≤ x•i} by auto
  show {x. x•i < a} ∈ ?SIGMA unfolding ∗
    using i by (intro sets.compl_sets) auto
qed auto

lemma borel_eq_halfspace_greater:
  borel = sigma UNIV ((λ (a, i). {x::'a::euclidean_space. a < x • i}) ‘ (UNIV ×
Basis))
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI2[OF borel_eq_halfspace_le])
  fix a :: real and i :: 'a assume (a, i) ∈ (UNIV × Basis)
  then have i: i ∈ Basis by auto
  have ∗: {x::'a. x•i ≤ a} = space ?SIGMA − {x::'a. a < x•i} by auto
  show {x. x•i ≤ a} ∈ ?SIGMA unfolding ∗
    by (intro sets.compl_sets) (auto intro: i)
qed auto

lemma borel_eq_atMost:
  borel = sigma UNIV (range (λa. {..a::'a::ordered_euclidean_space}))
  (is _ = ?SIGMA)
```

```
proof (rule borel_eq_sigmaI4[OF borel_eq_halfspace_le])
  fix a :: real and i :: 'a assume (a, i) ∈ UNIV × Basis
  then have i ∈ Basis by auto
  then have ∗: {x::'a. x•i ≤ a} = (⋃k::nat. {.. (∑ n∈Basis. (if n = i then a else
real k)∗R n)})
  proof (safe, simp_all add: eucl_le[where 'a='a] split: if_split_asm)
    fix x :: 'a
    obtain k where Max ((•) x ‘ Basis) ≤ real k
      using real_arch_simple by blast
    then have ⋀i. i ∈ Basis ⟹ x•i ≤ real k
      by (subst (asm) Max_le_iff) auto
    then show ∃k::nat. ∀ia∈Basis. ia ≠ i ⟶ x • ia ≤ real k
      by (auto intro!: exI[of _ k])
  qed
  show {x. x•i ≤ a} ∈ ?SIGMA unfolding ∗
    by (intro sets.countable_UN) auto
qed auto

lemma borel_eq_greaterThan:
  borel = sigma UNIV (range (λa::'a::ordered_euclidean_space. {x. a <e x}))
  (is _ = ?SIGMA)
proof (rule borel_eq_sigmaI4[OF borel_eq_halfspace_le])
  fix a :: real and i :: 'a assume (a, i) ∈ UNIV × Basis
  then have i: i ∈ Basis by auto
  have {x::'a. x•i ≤ a} = UNIV − {x::'a. a < x•i} by auto
  also have ∗: {x::'a. a < x•i} =
      (⋃k::nat. {x. (∑ n∈Basis. (if n = i then a else −real k) ∗R n) <e x}) using
i
  proof (safe, simp_all add: eucl_less_def split: if_split_asm)
    fix x :: 'a
    obtain k where k: Max ((•) (− x) ‘ Basis) < real k
      using reals_Archimedean2 by blast
    { fix i :: 'a assume i ∈ Basis
      then have −x•i < real k
        using k by (subst (asm) Max_less_iff) auto
      then have − real k < x•i by simp }
    then show ∃k::nat. ∀ia∈Basis. ia ≠ i ⟶ −real k < x • ia
      by (auto intro!: exI[of _ k])
  qed
  finally show {x. x•i ≤ a} ∈ ?SIGMA
    apply (simp only:)
    apply (intro sets.countable_UN sets.Diff)
    apply (auto intro: sigma_sets_top)
    done
qed auto

lemma borel_eq_lessThan:
  borel = sigma UNIV (range (λa::'a::ordered_euclidean_space. {x. x <e a}))
  (is _ = ?SIGMA)
```

proof (*rule borel_eq_sigmaI4*[*OF borel_eq_halfspace_ge*])
fix *a* :: *real* **and** *i* :: *'a* **assume** (*a*, *i*) ∈ *UNIV* × *Basis*
then have *i*: *i* ∈ *Basis* **by** *auto*
have {*x*::*'a*. *a* ≤ *x*•*i*} = *UNIV* − {*x*::*'a*. *x*•*i* < *a*} **by** *auto*
also have ∗: {*x*::*'a*. *x*•*i* < *a*} = (⋃*k*::*nat*. {*x*. *x* <*e* (∑ *n*∈*Basis*. (*if n* = *i then a else real k*) ∗*R n*)}) **using** ‹*i*∈ *Basis*›
proof (*safe*, *simp_all add*: *eucl_less_def split*: *if_split_asm*)
fix *x* :: *'a*
obtain *k* **where** *k*: *Max* ((•) *x* ‘ *Basis*) < *real k*
using *reals_Archimedean2* **by** *blast*
{ fix *i* :: *'a* **assume** *i* ∈ *Basis*
then have *x*•*i* < *real k*
using *k* **by** (*subst* (*asm*) *Max_less_iff*) *auto*
then have *x*•*i* < *real k* **by** *simp* **}**
then show ∃ *k*::*nat*. ∀ *ia*∈*Basis*. *ia* ≠ *i* ⟶ *x* • *ia* < *real k*
by (*auto intro*!: *exI*[*of* _ *k*])
qed
finally show {*x*. *a* ≤ *x*•*i*} ∈ *?SIGMA*
apply (*simp only*:)
apply (*intro sets.countable_UN sets.Diff*)
apply (*auto intro*: *sigma_sets_top*)
done
qed *auto*

lemma *borel_eq_atLeastAtMost*:
borel = *sigma UNIV* (*range* (λ(*a*,*b*). {*a*..*b*} ::*'a*::*ordered_euclidean_space set*))
(**is** _ = *?SIGMA*)
proof (*rule borel_eq_sigmaI5*[*OF borel_eq_atMost*])
fix *a*::*'a*
have ∗: {..*a*} = (⋃ *n*::*nat*. {− *real n* ∗*R One* .. *a*})
proof (*safe*, *simp_all add*: *eucl_le*[**where** *'a*=*'a*])
fix *x* :: *'a*
obtain *k* **where** *k*: *Max* ((•) (− *x*) ‘ *Basis*) ≤ *real k*
using *real_arch_simple* **by** *blast*
{ fix *i* :: *'a* **assume** *i* ∈ *Basis*
with *k* **have** − *x*•*i* ≤ *real k*
by (*subst* (*asm*) *Max_le_iff*) (*auto simp*: *field_simps*)
then have − *real k* ≤ *x*•*i* **by** *simp* **}**
then show ∃ *n*::*nat*. ∀ *i*∈*Basis*. − *real n* ≤ *x* • *i*
by (*auto intro*!: *exI*[*of* _ *k*])
qed
show {..*a*} ∈ *?SIGMA* **unfolding** ∗
by (*intro sets.countable_UN*)
(*auto intro*!: *sigma_sets_top*)
qed *auto*

lemma *borel_set_induct*[*consumes 1*, *case_names empty interval compl union*]:
assumes *A* ∈ *sets borel*
assumes *empty*: *P* {} **and** *int*: ⋀*a b*. *a* ≤ *b* ⟹ *P* {*a*..*b*} **and** *compl*: ⋀*A*. *A* ∈

sets borel $\Longrightarrow$ *P A* $\Longrightarrow$ *P* $(-A)$ **and**
un: $\bigwedge f$. *disjoint_family f* $\Longrightarrow$ $(\bigwedge i.\ f\ i \in$ *sets borel*$) \Longrightarrow$ $(\bigwedge i.\ P\ (f\ i)) \Longrightarrow$
P $(\bigcup i{::}nat.\ f\ i)$
shows *P* $(A{::}real\ set)$
proof −
let *?G* = *range* $(\lambda(a,b).\ \{a..b{::}real\})$
have *Int_stable ?G ?G* $\subseteq$ *Pow UNIV A* $\in$ *sigma_sets UNIV ?G*
using *assms*(*1*) **by** (*auto simp add*: *borel_eq_atLeastAtMost Int_stable_def*)
thus *?thesis*
proof (*induction rule*: *sigma_sets_induct_disjoint*)
case (*union f*)
from *union.hyps*(*2*) **have** $\bigwedge i.\ f\ i \in$ *sets borel* **by** (*auto simp*: *borel_eq_atLeastAtMost*)
with *union* **show** *?case* **by** (*auto intro*: *un*)
next
case (*basic A*)
then obtain *a b* **where** *A* = $\{a\ ..\ b\}$ **by** *auto*
then show *?case*
by (*cases a* $\leq$ *b*) (*auto intro*: *int empty*)
qed (*auto intro*: *empty compl simp*: *Compl_eq_Diff_UNIV*[*symmetric*] *borel_eq_atLeastAtMost*)
qed

lemma *borel_sigma_sets_Ioc*: *borel* = *sigma UNIV* (*range* $(\lambda(a,\ b).\ \{a <..\ b{::}real\})$)
proof (*rule borel_eq_sigmaI5*[*OF borel_eq_atMost*])
fix *i* :: *real*
have $\{..i\}$ = $(\bigcup j{::}nat.\ \{-j <..\ i\})$
by (*auto simp*: *minus_less_iff reals_Archimedean2*)
also have ... $\in$ *sets* (*sigma UNIV* (*range* $(\lambda(i,\ j).\ \{i<..j\})$))
by (*intro sets.countable_nat_UN*) *auto*
finally show $\{..i\} \in$ *sets* (*sigma UNIV* (*range* $(\lambda(i,\ j).\ \{i<..j\})$)) **.**
qed *simp*

lemma *eucl_lessThan*: $\{x{::}real.\ x <e\ a\}$ = *lessThan a*
by (*simp add*: *eucl_less_def lessThan_def*)

lemma *borel_eq_atLeastLessThan*:
borel = *sigma UNIV* (*range* $(\lambda(a,\ b).\ \{a\ ..<\ b :: real\})$) (**is** *_* = *?SIGMA*)
proof (*rule borel_eq_sigmaI5*[*OF borel_eq_lessThan*])
have *move_uminus*: $\bigwedge x\ y{::}real.\ -x \leq y \longleftrightarrow -y \leq x$ **by** *auto*
fix *x* :: *real*
have $\{..<x\}$ = $(\bigcup i{::}nat.\ \{-real\ i\ ..<\ x\})$
by (*auto simp*: *move_uminus real_arch_simple*)
then show $\{y.\ y <e\ x\} \in$ *?SIGMA*
by (*auto intro*: *sigma_sets.intros*(*2*−) *simp*: *eucl_lessThan*)
qed *auto*

lemma *borel_measurable_halfspacesI*:
fixes *f* :: *'a* $\Rightarrow$ *'c*::*euclidean_space*
assumes *F*: *borel* = *sigma UNIV* (*F* ‘ (*UNIV* $\times$ *Basis*))
and *S_eq*: $\bigwedge a\ i.\ S\ a\ i = f -‘ F\ (a,i) \cap$ *space M*

shows $f \in$ *borel_measurable* $M = (\forall i{\in}Basis.\ \forall a{::}real.\ S\ a\ i \in sets\ M)$
proof *safe*
fix $a :: real$ **and** $i :: 'b$ **assume** i: $i \in Basis$ **and** f: $f \in$ *borel_measurable* M
then show $S\ a\ i \in sets\ M$ **unfolding** *assms*
by (*auto intro*!: *measurable_sets simp*: *assms*(*1*))
next
assume a: $\forall i{\in}Basis.\ \forall a.\ S\ a\ i \in sets\ M$
then show $f \in$ *borel_measurable* M
by (*auto intro*!: *measurable_measure_of simp*: *S_eq F*)
qed

lemma *borel_measurable_iff_halfspace_le*:
fixes $f :: 'a \Rightarrow 'c{::}euclidean\_space$
shows $f \in$ *borel_measurable* $M = (\forall i{\in}Basis.\ \forall a.\ \{w \in space\ M.\ f\ w \cdot i \leq a\} \in sets\ M)$
by (*rule borel_measurable_halfspacesI*[*OF borel_eq_halfspace_le*]) *auto*

lemma *borel_measurable_iff_halfspace_less*:
fixes $f :: 'a \Rightarrow 'c{::}euclidean\_space$
shows $f \in$ *borel_measurable* $M \longleftrightarrow (\forall i{\in}Basis.\ \forall a.\ \{w \in space\ M.\ f\ w \cdot i < a\} \in sets\ M)$
by (*rule borel_measurable_halfspacesI*[*OF borel_eq_halfspace_less*]) *auto*

lemma *borel_measurable_iff_halfspace_ge*:
fixes $f :: 'a \Rightarrow 'c{::}euclidean\_space$
shows $f \in$ *borel_measurable* $M = (\forall i{\in}Basis.\ \forall a.\ \{w \in space\ M.\ a \leq f\ w \cdot i\} \in sets\ M)$
by (*rule borel_measurable_halfspacesI*[*OF borel_eq_halfspace_ge*]) *auto*

lemma *borel_measurable_iff_halfspace_greater*:
fixes $f :: 'a \Rightarrow 'c{::}euclidean\_space$
shows $f \in$ *borel_measurable* $M \longleftrightarrow (\forall i{\in}Basis.\ \forall a.\ \{w \in space\ M.\ a < f\ w \cdot i\} \in sets\ M)$
by (*rule borel_measurable_halfspacesI*[*OF borel_eq_halfspace_greater*]) *auto*

lemma *borel_measurable_iff_le*:
$(f{::}'a \Rightarrow real) \in$ *borel_measurable* $M = (\forall a.\ \{w \in space\ M.\ f\ w \leq a\} \in sets\ M)$
using *borel_measurable_iff_halfspace_le*[**where** $'c{=}real$] **by** *simp*

lemma *borel_measurable_iff_less*:
$(f{::}'a \Rightarrow real) \in$ *borel_measurable* $M = (\forall a.\ \{w \in space\ M.\ f\ w < a\} \in sets\ M)$
using *borel_measurable_iff_halfspace_less*[**where** $'c{=}real$] **by** *simp*

lemma *borel_measurable_iff_ge*:
$(f{::}'a \Rightarrow real) \in$ *borel_measurable* $M = (\forall a.\ \{w \in space\ M.\ a \leq f\ w\} \in sets\ M)$
using *borel_measurable_iff_halfspace_ge*[**where** $'c{=}real$]
by *simp*

lemma *borel_measurable_iff_greater*:

$(f::'a \Rightarrow real) \in borel\_measurable\ M = (\forall\, a.\ \{w \in space\ M.\ a < f\ w\} \in sets\ M)$
using *borel_measurable_iff_halfspace_greater*[**where** *'c=real*] **by** *simp*

lemma *borel_measurable_euclidean_space*:
fixes $f :: 'a \Rightarrow 'c::euclidean\_space$
shows $f \in borel\_measurable\ M \longleftrightarrow (\forall\, i{\in}Basis.\ (\lambda x.\ f\ x \cdot i) \in borel\_measurable\ M)$
proof *safe*
assume *f*: $\forall\, i{\in}Basis.\ (\lambda x.\ f\ x \cdot i) \in borel\_measurable\ M$
then show $f \in borel\_measurable\ M$
by (*subst borel_measurable_iff_halfspace_le*) *auto*
qed *auto*

6.4.5 Borel measurable operators

lemma *borel_measurable_norm*[*measurable*]: *norm* $\in$ *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_sgn* [*measurable*]: $(sgn::'a::real\_normed\_vector \Rightarrow 'a) \in borel\_measurable\ borel$
by (*rule borel_measurable_continuous_countable_exceptions*[**where** $X=\{0\}$])
(*auto intro*!: *continuous_on_sgn continuous_on_id*)

lemma *borel_measurable_uminus*[*measurable* (*raw*)]:
fixes $g :: 'a \Rightarrow 'b::\{second\_countable\_topology, real\_normed\_vector\}$
assumes *g*: $g \in borel\_measurable\ M$
shows $(\lambda x.\ -\ g\ x) \in borel\_measurable\ M$
by (*rule borel_measurable_continuous_on*[*OF _ g*]) (*intro continuous_intros*)

lemma *borel_measurable_diff*[*measurable* (*raw*)]:
fixes $f :: 'a \Rightarrow 'b::\{second\_countable\_topology, real\_normed\_vector\}$
assumes *f*: $f \in borel\_measurable\ M$
assumes *g*: $g \in borel\_measurable\ M$
shows $(\lambda x.\ f\ x - g\ x) \in borel\_measurable\ M$
using *borel_measurable_add* [*of f M − g*] *assms* **by** (*simp add*: *fun_Compl_def*)

lemma *borel_measurable_times*[*measurable* (*raw*)]:
fixes $f :: 'a \Rightarrow 'b::\{second\_countable\_topology, real\_normed\_algebra\}$
assumes *f*: $f \in borel\_measurable\ M$
assumes *g*: $g \in borel\_measurable\ M$
shows $(\lambda x.\ f\ x * g\ x) \in borel\_measurable\ M$
using *f g* **by** (*rule borel_measurable_continuous_Pair*) (*intro continuous_intros*)

lemma *borel_measurable_prod*[*measurable* (*raw*)]:
fixes $f :: 'c \Rightarrow 'a \Rightarrow 'b::\{second\_countable\_topology, real\_normed\_field\}$
assumes $\bigwedge i.\ i \in S \Longrightarrow f\ i \in borel\_measurable\ M$
shows $(\lambda x.\ \prod i{\in}S.\ f\ i\ x) \in borel\_measurable\ M$
proof *cases*
assume *finite S*

thus *?thesis* **using** *assms* **by** *induct auto*
qed *simp*

lemma *borel_measurable_dist*[*measurable* (*raw*)]:
fixes *g f* :: *'a* ⇒ *'b*::{*second_countable_topology*, *metric_space*}
assumes *f*: *f* ∈ *borel_measurable M*
assumes *g*: *g* ∈ *borel_measurable M*
shows (λ*x*. *dist* (*f x*) (*g x*)) ∈ *borel_measurable M*
using *f g* **by** (*rule borel_measurable_continuous_Pair*) (*intro continuous_intros*)

lemma *borel_measurable_scaleR*[*measurable* (*raw*)]:
fixes *g* :: *'a* ⇒ *'b*::{*second_countable_topology*, *real_normed_vector*}
assumes *f*: *f* ∈ *borel_measurable M*
assumes *g*: *g* ∈ *borel_measurable M*
shows (λ*x*. *f x* ∗*R* *g x*) ∈ *borel_measurable M*
using *f g* **by** (*rule borel_measurable_continuous_Pair*) (*intro continuous_intros*)

lemma *borel_measurable_uminus_eq* [*simp*]:
fixes *f* :: *'a* ⇒ *'b*::{*second_countable_topology*, *real_normed_vector*}
shows (λ*x*. − *f x*) ∈ *borel_measurable M* ⟷ *f* ∈ *borel_measurable M* (**is** *?l* = *?r*)
proof
assume *?l* **from** *borel_measurable_uminus*[*OF this*] **show** *?r* **by** *simp*
qed *auto*

lemma *affine_borel_measurable_vector*:
fixes *f* :: *'a* ⇒ *'x*::*real_normed_vector*
assumes *f* ∈ *borel_measurable M*
shows (λ*x*. *a* + *b* ∗*R* *f x*) ∈ *borel_measurable M*
proof (*rule borel_measurableI*)
fix *S* :: *'x set* **assume** *open S*
show (λ*x*. *a* + *b* ∗*R* *f x*) −‘ *S* ∩ *space M* ∈ *sets M*
proof *cases*
assume *b* ≠ *0*
with ‹*open S*› **have** *open* ((λ*x*. (− *a* + *x*) /*R* *b*) ‘ *S*) (**is** *open ?S*)
using *open_affinity* [*of S inverse b* − *a* /*R* *b*]
by (*auto simp*: *algebra_simps*)
hence *?S* ∈ *sets borel* **by** *auto*
moreover
from ‹*b* ≠ *0*› **have** (λ*x*. *a* + *b* ∗*R* *f x*) −‘ *S* = *f* −‘ *?S*
apply *auto* **by** (*rule_tac x*=*a* + *b* ∗*R* *f x* **in** *image_eqI*, *simp_all*)
ultimately show *?thesis* **using** *assms* **unfolding** *in_borel_measurable_borel*
by *auto*
qed *simp*
qed

lemma *borel_measurable_const_scaleR*[*measurable* (*raw*)]:
f ∈ *borel_measurable M* ⟹ (λ*x*. *b* ∗*R* *f x* ::*'a*::*real_normed_vector*) ∈ *borel_measurable M*

using *affine_borel_measurable_vector*[*of f M 0 b*] **by** *simp*

lemma *borel_measurable_const_add*[*measurable* (*raw*)]:
f ∈ *borel_measurable M* ⟹ (λ*x*. *a* + *f x* ::'*a*::*real_normed_vector*) ∈ *borel_measurable M*
using *affine_borel_measurable_vector*[*of f M a 1*] **by** *simp*

lemma *borel_measurable_inverse*[*measurable* (*raw*)]:
fixes *f* :: '*a* ⇒ '*b*::*real_normed_div_algebra*
assumes *f*: *f* ∈ *borel_measurable M*
shows (λ*x*. *inverse* (*f x*)) ∈ *borel_measurable M*
apply (*rule measurable_compose*[*OF f*])
apply (*rule borel_measurable_continuous_countable_exceptions*[*of* {*0*}])
apply (*auto intro*!: *continuous_on_inverse continuous_on_id*)
done

lemma *borel_measurable_divide*[*measurable* (*raw*)]:
f ∈ *borel_measurable M* ⟹ *g* ∈ *borel_measurable M* ⟹
(λ*x*. *f x* / *g x*::'*b*::{*second_countable_topology*, *real_normed_div_algebra*}) ∈ *borel_measurable M*
by (*simp add*: *divide_inverse*)

lemma *borel_measurable_abs*[*measurable* (*raw*)]:
f ∈ *borel_measurable M* ⟹ (λ*x*. |*f x* :: *real*|) ∈ *borel_measurable M*
unfolding *abs_real_def* **by** *simp*

lemma *borel_measurable_nth*[*measurable* (*raw*)]:
(λ*x*::*real*^'*n*. *x* $ *i*) ∈ *borel_measurable borel*
by (*simp add*: *cart_eq_inner_axis*)

lemma *convex_measurable*:
fixes *A* :: '*a* :: *euclidean_space set*
shows *X* ∈ *borel_measurable M* ⟹ *X* ' *space M* ⊆ *A* ⟹ *open A* ⟹ *convex_on A q* ⟹
(λ*x*. *q* (*X x*)) ∈ *borel_measurable M*
by (*rule measurable_compose*[**where** *f*=*X* **and** *N*=*restrict_space borel A*])
(*auto intro*!: *borel_measurable_continuous_on_restrict convex_on_continuous measurable_restrict_space2*)

lemma *borel_measurable_ln*[*measurable* (*raw*)]:
assumes *f*: *f* ∈ *borel_measurable M*
shows (λ*x*. *ln* (*f x* :: *real*)) ∈ *borel_measurable M*
apply (*rule measurable_compose*[*OF f*])
apply (*rule borel_measurable_continuous_countable_exceptions*[*of* {*0*}])
apply (*auto intro*!: *continuous_on_ln continuous_on_id*)
done

lemma *borel_measurable_log*[*measurable* (*raw*)]:
f ∈ *borel_measurable M* ⟹ *g* ∈ *borel_measurable M* ⟹ (λ*x*. *log* (*g x*) (*f x*))

$\in$ *borel_measurable M*
unfolding *log_def* **by** *auto*

lemma *borel_measurable_exp*[*measurable*]:
(*exp*::'*a*::{*real_normed_field*,*banach*}⇒'*a*) ∈ *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_at_imp_continuous_on ballI isCont_exp*)

lemma *measurable_real_floor*[*measurable*]:
(*floor* :: *real* ⇒ *int*) ∈ *measurable borel* (*count_space UNIV*)
proof −
have ⋀*a x*. ⌊*x*⌋ = *a* ⟷ (*real_of_int a* ≤ *x* ∧ *x* < *real_of_int* (*a* + *1*))
by (*auto intro*: *floor_eq2*)
then show *?thesis*
by (*auto simp*: *vimage_def measurable_count_space_eq2_countable*)
qed

lemma *measurable_real_ceiling*[*measurable*]:
(*ceiling* :: *real* ⇒ *int*) ∈ *measurable borel* (*count_space UNIV*)
unfolding *ceiling_def*[*abs_def*] **by** *simp*

lemma *borel_measurable_real_floor*: (λ*x*::*real*. *real_of_int* ⌊*x*⌋) ∈ *borel_measurable borel*
by *simp*

lemma *borel_measurable_root* [*measurable*]: *root n* ∈ *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_sqrt* [*measurable*]: *sqrt* ∈ *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_power* [*measurable* (*raw*)]:
fixes *f* :: _ ⇒ '*b*::{*power*,*real_normed_algebra*}
assumes *f*: *f* ∈ *borel_measurable M*
shows (λ*x*. (*f x*) ^ *n*) ∈ *borel_measurable M*
by (*intro borel_measurable_continuous_on* [*OF* _ *f*] *continuous_intros*)

lemma *borel_measurable_Re* [*measurable*]: *Re* ∈ *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_Im* [*measurable*]: *Im* ∈ *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_of_real* [*measurable*]: (*of_real* :: _ ⇒ (_::*real_normed_algebra*))
∈ *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_sin* [*measurable*]: (*sin* :: _ ⇒ (_::{*real_normed_field*,*banach*}))
∈ *borel_measurable borel*

by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_cos* [*measurable*]: (*cos* :: _ ⇒ (_::{*real_normed_field*,*banach*})) ∈ *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_arctan* [*measurable*]: *arctan* ∈ *borel_measurable borel*
by (*intro borel_measurable_continuous_onI continuous_intros*)

lemma *borel_measurable_complex_iff*:
f ∈ *borel_measurable M* ⟷
(λ*x*. *Re* (*f x*)) ∈ *borel_measurable M* ∧ (λ*x*. *Im* (*f x*)) ∈ *borel_measurable M*
apply *auto*
apply (*subst fun_complex_eq*)
apply (*intro borel_measurable_add*)
apply *auto*
done

lemma *powr_real_measurable* [*measurable*]:
assumes *f* ∈ *measurable M borel g* ∈ *measurable M borel*
shows (λ*x*. *f x powr g x* :: *real*) ∈ *measurable M borel*
using *assms* **by** (*simp_all add*: *powr_def*)

lemma *measurable_of_bool*[*measurable*]: *of_bool* ∈ *count_space UNIV* →$_M$ *borel*
by *simp*

6.4.6 Borel space on the extended reals

lemma *borel_measurable_ereal*[*measurable* (*raw*)]:
assumes *f*: *f* ∈ *borel_measurable M* **shows** (λ*x*. *ereal* (*f x*)) ∈ *borel_measurable M*
using *continuous_on_ereal f* **by** (*rule borel_measurable_continuous_on*) (*rule continuous_on_id*)

lemma *borel_measurable_real_of_ereal*[*measurable* (*raw*)]:
fixes *f* :: *'a* ⇒ *ereal*
assumes *f*: *f* ∈ *borel_measurable M*
shows (λ*x*. *real_of_ereal* (*f x*)) ∈ *borel_measurable M*
apply (*rule measurable_compose*[*OF f*])
apply (*rule borel_measurable_continuous_countable_exceptions*[*of* {∞, −∞ }])
apply (*auto intro*: *continuous_on_real simp*: *Compl_eq_Diff_UNIV*)
done

lemma *borel_measurable_ereal_cases*:
fixes *f* :: *'a* ⇒ *ereal*
assumes *f*: *f* ∈ *borel_measurable M*
assumes *H*: (λ*x*. *H* (*ereal* (*real_of_ereal* (*f x*)))) ∈ *borel_measurable M*
shows (λ*x*. *H* (*f x*)) ∈ *borel_measurable M*
proof −

let *?F* = λ*x*. *if f x* = ∞ *then H* ∞ *else if f x* = − ∞ *then H* (−∞) *else H* (*ereal* (*real_of_ereal* (*f x*)))
{ fix *x* **have** *H* (*f x*) = *?F x* **by** (*cases f x*) *auto* **}**
with *f H* **show** *?thesis* **by** *simp*
qed

lemma
fixes *f* :: *'a* ⇒ *ereal* **assumes** *f*[*measurable*]: *f* ∈ *borel_measurable M*
shows *borel_measurable_ereal_abs*[*measurable*(*raw*)]: (λ*x*. |*f x*|) ∈ *borel_measurable M*
and *borel_measurable_ereal_inverse*[*measurable*(*raw*)]: (λ*x*. *inverse* (*f x*) :: *ereal*) ∈ *borel_measurable M*
and *borel_measurable_uminus_ereal*[*measurable*(*raw*)]: (λ*x*. − *f x* :: *ereal*) ∈ *borel_measurable M*
by (*auto simp del*: *abs_real_of_ereal simp*: *borel_measurable_ereal_cases*[*OF f*] *measurable_If*)

lemma *borel_measurable_uminus_eq_ereal*[*simp*]:
(λ*x*. − *f x* :: *ereal*) ∈ *borel_measurable M* ⟷ *f* ∈ *borel_measurable M* (**is** *?l* = *?r*)
proof
assume *?l* **from** *borel_measurable_uminus_ereal*[*OF this*] **show** *?r* **by** *simp*
qed *auto*

lemma *set_Collect_ereal2*:
fixes *f g* :: *'a* ⇒ *ereal*
assumes *f*: *f* ∈ *borel_measurable M*
assumes *g*: *g* ∈ *borel_measurable M*
assumes *H*: {*x* ∈ *space M*. *H* (*ereal* (*real_of_ereal* (*f x*))) (*ereal* (*real_of_ereal* (*g x*)))} ∈ *sets M*
{*x* ∈ *space borel*. *H* (−∞) (*ereal x*)} ∈ *sets borel*
{*x* ∈ *space borel*. *H* (∞) (*ereal x*)} ∈ *sets borel*
{*x* ∈ *space borel*. *H* (*ereal x*) (−∞)} ∈ *sets borel*
{*x* ∈ *space borel*. *H* (*ereal x*) (∞)} ∈ *sets borel*
shows {*x* ∈ *space M*. *H* (*f x*) (*g x*)} ∈ *sets M*
proof −
let *?G* = λ*y x*. *if g x* = ∞ *then H y* ∞ *else if g x* = −∞ *then H y* (−∞) *else H y* (*ereal* (*real_of_ereal* (*g x*)))
let *?F* = λ*x*. *if f x* = ∞ *then ?G* ∞ *x else if f x* = −∞ *then ?G* (−∞) *x else ?G* (*ereal* (*real_of_ereal* (*f x*))) *x*
{ fix *x* **have** *H* (*f x*) (*g x*) = *?F x* **by** (*cases f x g x rule*: *ereal2_cases*) *auto* **}**
note ∗ = *this*
from *assms* **show** *?thesis*
by (*subst* ∗) (*simp del*: *space_borel split del*: *if_split*)
qed

lemma *borel_measurable_ereal_iff*:
shows (λ*x*. *ereal* (*f x*)) ∈ *borel_measurable M* ⟷ *f* ∈ *borel_measurable M*
proof

assume $(\lambda x.\ ereal\ (f\ x)) \in borel\_measurable\ M$
from *borel_measurable_real_of_ereal*[*OF this*]
show $f \in borel\_measurable\ M$ **by** *auto*
qed *auto*

lemma *borel_measurable_erealD*[*measurable_dest*]:
$(\lambda x.\ ereal\ (f\ x)) \in borel\_measurable\ M \Longrightarrow g \in measurable\ N\ M \Longrightarrow (\lambda x.\ f\ (g\ x)) \in borel\_measurable\ N$
unfolding *borel_measurable_ereal_iff* **by** *simp*

theorem *borel_measurable_ereal_iff_real*:
fixes $f :: 'a \Rightarrow ereal$
shows $f \in borel\_measurable\ M \longleftrightarrow$
$((\lambda x.\ real\_of\_ereal\ (f\ x)) \in borel\_measurable\ M \wedge f -` \{\infty\} \cap space\ M \in sets\ M \wedge f -` \{-\infty\} \cap space\ M \in sets\ M)$
proof *safe*
assume $*$: $(\lambda x.\ real\_of\_ereal\ (f\ x)) \in borel\_measurable\ M\ f -` \{\infty\} \cap space\ M \in sets\ M\ f -` \{-\infty\} \cap space\ M \in sets\ M$
have $f -` \{\infty\} \cap space\ M = \{x \in space\ M.\ f\ x = \infty\}\ f -` \{-\infty\} \cap space\ M = \{x \in space\ M.\ f\ x = -\infty\}$ **by** *auto*
with $*$ **have** $**$: $\{x \in space\ M.\ f\ x = \infty\} \in sets\ M\ \{x \in space\ M.\ f\ x = -\infty\} \in sets\ M$ **by** *simp_all*
let *?f* $= \lambda x.\ if\ f\ x = \infty\ then\ \infty\ else\ if\ f\ x = -\infty\ then\ -\infty\ else\ ereal\ (real\_of\_ereal\ (f\ x))$
have *?f* $\in borel\_measurable\ M$ **using** $*$ $**$ **by** (*intro measurable_If*) *auto*
also have *?f* $= f$ **by** (*auto simp*: *fun_eq_iff ereal_real*)
finally show $f \in borel\_measurable\ M$ **.**
qed *simp_all*

lemma *borel_measurable_ereal_iff_Iio*:
$(f::'a \Rightarrow ereal) \in borel\_measurable\ M \longleftrightarrow (\forall\ a.\ f -` \{..< a\} \cap space\ M \in sets\ M)$
by (*auto simp*: *borel_Iio measurable_iff_measure_of*)

lemma *borel_measurable_ereal_iff_Ioi*:
$(f::'a \Rightarrow ereal) \in borel\_measurable\ M \longleftrightarrow (\forall\ a.\ f -` \{a <..\} \cap space\ M \in sets\ M)$
by (*auto simp*: *borel_Ioi measurable_iff_measure_of*)

lemma *vimage_sets_compl_iff*:
$f -` A \cap space\ M \in sets\ M \longleftrightarrow f -` (-\ A) \cap space\ M \in sets\ M$
proof −
{ fix A **assume** $f -` A \cap space\ M \in sets\ M$
moreover have $f -` (-\ A) \cap space\ M = space\ M - f -` A \cap space\ M$ **by** *auto*
ultimately have $f -` (-\ A) \cap space\ M \in sets\ M$ **by** *auto* **}**
from *this*[*of A*] *this*[*of* $-A$] **show** *?thesis*
by (*metis double_complement*)
qed

lemma *borel_measurable_iff_Iic_ereal*:
(*f*::'*a*⇒*ereal*) ∈ *borel_measurable M* ⟷ (∀ *a*. *f* −‘ {..*a*} ∩ *space M* ∈ *sets M*)
unfolding *borel_measurable_ereal_iff_Ioi vimage_sets_compl_iff*[**where** *A*={*a* <..} **for** *a*] **by** *simp*

lemma *borel_measurable_iff_Ici_ereal*:
(*f*::'*a* ⇒ *ereal*) ∈ *borel_measurable M* ⟷ (∀ *a*. *f* −‘ {*a*..} ∩ *space M* ∈ *sets M*)
unfolding *borel_measurable_ereal_iff_Iio vimage_sets_compl_iff*[**where** *A*={..< *a*} **for** *a*] **by** *simp*

lemma *borel_measurable_ereal2*:
fixes *f g* :: '*a* ⇒ *ereal*
assumes *f*: *f* ∈ *borel_measurable M*
assumes *g*: *g* ∈ *borel_measurable M*
assumes *H*: (λ*x*. *H* (*ereal* (*real_of_ereal* (*f x*))) (*ereal* (*real_of_ereal* (*g x*)))) ∈ *borel_measurable M*
(λ*x*. *H* (−∞) (*ereal* (*real_of_ereal* (*g x*)))) ∈ *borel_measurable M*
(λ*x*. *H* (∞) (*ereal* (*real_of_ereal* (*g x*)))) ∈ *borel_measurable M*
(λ*x*. *H* (*ereal* (*real_of_ereal* (*f x*))) (−∞)) ∈ *borel_measurable M*
(λ*x*. *H* (*ereal* (*real_of_ereal* (*f x*))) (∞)) ∈ *borel_measurable M*
shows (λ*x*. *H* (*f x*) (*g x*)) ∈ *borel_measurable M*
proof −
let *?G* = λ*y x*. *if g x* = ∞ *then H y* ∞ *else if g x* = − ∞ *then H y* (−∞) *else H y* (*ereal* (*real_of_ereal* (*g x*)))
let *?F* = λ*x*. *if f x* = ∞ *then ?G* ∞ *x else if f x* = − ∞ *then ?G* (−∞) *x else ?G* (*ereal* (*real_of_ereal* (*f x*))) *x*
{ **fix** *x* **have** *H* (*f x*) (*g x*) = *?F x* **by** (*cases f x g x rule*: *ereal2_cases*) *auto* }
note ∗ = *this*
from *assms* **show** *?thesis* **unfolding** ∗ **by** *simp*
qed

lemma [*measurable*(*raw*)]:
fixes *f* :: '*a* ⇒ *ereal*
assumes [*measurable*]: *f* ∈ *borel_measurable M g* ∈ *borel_measurable M*
shows *borel_measurable_ereal_add*: (λ*x*. *f x* + *g x*) ∈ *borel_measurable M*
and *borel_measurable_ereal_times*: (λ*x*. *f x* ∗ *g x*) ∈ *borel_measurable M*
by (*simp_all add*: *borel_measurable_ereal2*)

lemma [*measurable*(*raw*)]:
fixes *f g* :: '*a* ⇒ *ereal*
assumes *f* ∈ *borel_measurable M*
assumes *g* ∈ *borel_measurable M*
shows *borel_measurable_ereal_diff*: (λ*x*. *f x* − *g x*) ∈ *borel_measurable M*
and *borel_measurable_ereal_divide*: (λ*x*. *f x* / *g x*) ∈ *borel_measurable M*
using *assms* **by** (*simp_all add*: *minus_ereal_def divide_ereal_def*)

lemma *borel_measurable_ereal_sum*[*measurable* (*raw*)]:
fixes *f* :: '*c* ⇒ '*a* ⇒ *ereal*

assumes $\bigwedge i.\ i \in S \Longrightarrow f\ i \in \mathit{borel\_measurable}\ M$
shows $(\lambda x.\ \sum i{\in}S.\ f\ i\ x) \in \mathit{borel\_measurable}\ M$
using *assms* **by** (*induction S rule*: *infinite_finite_induct*) *auto*

lemma *borel_measurable_ereal_prod*[*measurable* (*raw*)]:
fixes $f :: 'c \Rightarrow 'a \Rightarrow \mathit{ereal}$
assumes $\bigwedge i.\ i \in S \Longrightarrow f\ i \in \mathit{borel\_measurable}\ M$
shows $(\lambda x.\ \prod i{\in}S.\ f\ i\ x) \in \mathit{borel\_measurable}\ M$
using *assms* **by** (*induction S rule*: *infinite_finite_induct*) *auto*

lemma *borel_measurable_extreal_suminf*[*measurable* (*raw*)]:
fixes $f :: \mathit{nat} \Rightarrow 'a \Rightarrow \mathit{ereal}$
assumes [*measurable*]: $\bigwedge i.\ f\ i \in \mathit{borel\_measurable}\ M$
shows $(\lambda x.\ (\sum i.\ f\ i\ x)) \in \mathit{borel\_measurable}\ M$
unfolding *suminf_def sums_def*[*abs_def*] *lim_def*[*symmetric*] **by** *simp*

6.4.7 Borel space on the extended non-negative reals

ennreal is a topological monoid, so no rules for plus are required, also all order statements are usually done on type classes.

lemma *measurable_enn2ereal*[*measurable*]: $\mathit{enn2ereal} \in \mathit{borel} \rightarrow_M \mathit{borel}$
by (*intro borel_measurable_continuous_onI continuous_on_enn2ereal*)

lemma *measurable_e2ennreal*[*measurable*]: $\mathit{e2ennreal} \in \mathit{borel} \rightarrow_M \mathit{borel}$
by (*intro borel_measurable_continuous_onI continuous_on_e2ennreal*)

lemma *borel_measurable_enn2real*[*measurable* (*raw*)]:
$f \in M \rightarrow_M \mathit{borel} \Longrightarrow (\lambda x.\ \mathit{enn2real}\ (f\ x)) \in M \rightarrow_M \mathit{borel}$
unfolding *enn2real_def*[*abs_def*] **by** *measurable*

definition [*simp*]: $\mathit{is\_borel}\ f\ M \longleftrightarrow f \in \mathit{borel\_measurable}\ M$

lemma *is_borel_transfer*[*transfer_rule*]: *rel_fun* (*rel_fun* (=) *pcr_ennreal*) (=) *is_borel is_borel*
unfolding *is_borel_def*[*abs_def*]
proof (*safe intro*!: *rel_funI ext dest*!: *rel_fun_eq_pcr_ennreal*[*THEN iffD1*])
fix f **and** $M :: 'a\ \mathit{measure}$
show $f \in \mathit{borel\_measurable}\ M$ **if** *f*: $\mathit{enn2ereal} \circ f \in \mathit{borel\_measurable}\ M$
using *measurable_compose*[*OF f measurable_e2ennreal*] **by** *simp*
qed *simp*

context
includes *ennreal.lifting*
begin

lemma *measurable_ennreal*[*measurable*]: $\mathit{ennreal} \in \mathit{borel} \rightarrow_M \mathit{borel}$
unfolding *is_borel_def*[*symmetric*]
by *transfer simp*

lemma *borel_measurable_ennreal_iff*[*simp*]:
assumes [*simp*]: $\bigwedge x.\ x \in space\ M \Longrightarrow 0 \le f\ x$
shows $(\lambda x.\ ennreal\ (f\ x)) \in M \to_M borel \longleftrightarrow f \in M \to_M borel$
proof *safe*
assume $(\lambda x.\ ennreal\ (f\ x)) \in M \to_M borel$
then have $(\lambda x.\ enn2real\ (ennreal\ (f\ x))) \in M \to_M borel$
by *measurable*
then show $f \in M \to_M borel$
by (*rule measurable_cong*[*THEN iffD1*, *rotated*]) *auto*
qed *measurable*

lemma *borel_measurable_times_ennreal*[*measurable* (*raw*)]:
fixes $f\ g :: 'a \Rightarrow ennreal$
shows $f \in M \to_M borel \Longrightarrow g \in M \to_M borel \Longrightarrow (\lambda x.\ f\ x * g\ x) \in M \to_M borel$
unfolding *is_borel_def*[*symmetric*] **by** *transfer simp*

lemma *borel_measurable_inverse_ennreal*[*measurable* (*raw*)]:
fixes $f :: 'a \Rightarrow ennreal$
shows $f \in M \to_M borel \Longrightarrow (\lambda x.\ inverse\ (f\ x)) \in M \to_M borel$
unfolding *is_borel_def*[*symmetric*] **by** *transfer simp*

lemma *borel_measurable_divide_ennreal*[*measurable* (*raw*)]:
fixes $f :: 'a \Rightarrow ennreal$
shows $f \in M \to_M borel \Longrightarrow g \in M \to_M borel \Longrightarrow (\lambda x.\ f\ x\ /\ g\ x) \in M \to_M borel$
unfolding *divide_ennreal_def* **by** *simp*

lemma *borel_measurable_minus_ennreal*[*measurable* (*raw*)]:
fixes $f :: 'a \Rightarrow ennreal$
shows $f \in M \to_M borel \Longrightarrow g \in M \to_M borel \Longrightarrow (\lambda x.\ f\ x - g\ x) \in M \to_M borel$
unfolding *is_borel_def*[*symmetric*] **by** *transfer simp*

lemma *borel_measurable_power_ennreal* [*measurable* (*raw*)]:
fixes $f :: \_ \Rightarrow ennreal$
assumes *f*: $f \in borel\_measurable\ M$
shows $(\lambda x.\ (f\ x)\ \hat{}\ n) \in borel\_measurable\ M$
by (*induction n*) (*use f* **in** *auto*)

lemma *borel_measurable_prod_ennreal*[*measurable* (*raw*)]:
fixes $f :: 'c \Rightarrow 'a \Rightarrow ennreal$
assumes $\bigwedge i.\ i \in S \Longrightarrow f\ i \in borel\_measurable\ M$
shows $(\lambda x.\ \prod i{\in}S.\ f\ i\ x) \in borel\_measurable\ M$
using *assms* **by** (*induction S rule*: *infinite_finite_induct*) *auto*

end

hide_const (**open**) *is_borel*

6.4.8 LIMSEQ is borel measurable

lemma *borel_measurable_LIMSEQ_real*:
fixes $u :: nat \Rightarrow 'a \Rightarrow real$
assumes u': $\bigwedge x.\ x \in space\ M \Longrightarrow (\lambda i.\ u\ i\ x) \longrightarrow u'\ x$
and u: $\bigwedge i.\ u\ i \in borel\_measurable\ M$
shows $u' \in borel\_measurable\ M$
proof −
have $\bigwedge x.\ x \in space\ M \Longrightarrow liminf\ (\lambda n.\ ereal\ (u\ n\ x)) = ereal\ (u'\ x)$
using u' **by** (*simp add*: *lim_imp_Liminf*)
moreover from u **have** $(\lambda x.\ liminf\ (\lambda n.\ ereal\ (u\ n\ x))) \in borel\_measurable\ M$
by *auto*
ultimately show *?thesis* **by** (*simp cong*: *measurable_cong add*: *borel_measurable_ereal_iff*)
qed

lemma *borel_measurable_LIMSEQ_metric*:
fixes $f :: nat \Rightarrow 'a \Rightarrow 'b :: metric\_space$
assumes [*measurable*]: $\bigwedge i.\ f\ i \in borel\_measurable\ M$
assumes *lim*: $\bigwedge x.\ x \in space\ M \Longrightarrow (\lambda i.\ f\ i\ x) \longrightarrow g\ x$
shows $g \in borel\_measurable\ M$
unfolding *borel_eq_closed*
proof (*safe intro*!: *measurable_measure_of*)
fix $A :: 'b\ set$ **assume** *closed A*

have [*measurable*]: $(\lambda x.\ infdist\ (g\ x)\ A) \in borel\_measurable\ M$
proof (*rule borel_measurable_LIMSEQ_real*)
show $\bigwedge x.\ x \in space\ M \Longrightarrow (\lambda i.\ infdist\ (f\ i\ x)\ A) \longrightarrow infdist\ (g\ x)\ A$
by (*intro tendsto_infdist lim*)
show $\bigwedge i.\ (\lambda x.\ infdist\ (f\ i\ x)\ A) \in borel\_measurable\ M$
by (*intro borel_measurable_continuous_on*[**where** $f{=}\lambda x.\ infdist\ x\ A$]
continuous_at_imp_continuous_on ballI continuous_infdist continuous_ident)
auto
qed

show $g -` A \cap space\ M \in sets\ M$
proof *cases*
assume $A \neq \{\}$
then have $\bigwedge x.\ infdist\ x\ A = 0 \longleftrightarrow x \in A$
using ‹*closed A*› **by** (*simp add*: *in_closed_iff_infdist_zero*)
then have $g -` A \cap space\ M = \{x{\in}space\ M.\ infdist\ (g\ x)\ A = 0\}$
by *auto*
also have $\ldots \in sets\ M$
by *measurable*
finally show *?thesis* **.**
qed *simp*
qed *auto*

lemma *sets_Collect_Cauchy*[*measurable*]:
fixes $f :: nat \Rightarrow 'a => 'b{::}\{metric\_space, second\_countable\_topology\}$
assumes f[*measurable*]: $\bigwedge i.\ f\ i \in borel\_measurable\ M$

shows $\{x{\in}space\ M.\ Cauchy\ (\lambda i.\ f\ i\ x)\} \in sets\ M$
unfolding *metric_Cauchy_iff2* **using** f **by** *auto*

lemma *borel_measurable_lim_metric*[*measurable* (*raw*)]:
fixes $f :: nat \Rightarrow 'a \Rightarrow 'b{::}\{banach, second\_countable\_topology\}$
assumes *f*[*measurable*]: $\bigwedge i.\ f\ i \in borel\_measurable\ M$
shows $(\lambda x.\ lim\ (\lambda i.\ f\ i\ x)) \in borel\_measurable\ M$
proof −
define u' **where** $u'\ x = lim\ (\lambda i.\ if\ Cauchy\ (\lambda i.\ f\ i\ x)\ then\ f\ i\ x\ else\ 0)$ **for** x
then have $*$: $\bigwedge x.\ lim\ (\lambda i.\ f\ i\ x) = (if\ Cauchy\ (\lambda i.\ f\ i\ x)\ then\ u'\ x\ else\ (THE\ x.\ False))$
by (*auto simp*: *lim_def convergent_eq_Cauchy*[*symmetric*])
have $u' \in borel\_measurable\ M$
proof (*rule borel_measurable_LIMSEQ_metric*)
fix x
have *convergent* $(\lambda i.\ if\ Cauchy\ (\lambda i.\ f\ i\ x)\ then\ f\ i\ x\ else\ 0)$
by (*cases Cauchy* $(\lambda i.\ f\ i\ x)$)
(*auto simp add*: *convergent_eq_Cauchy*[*symmetric*] *convergent_def*)
then show $(\lambda i.\ if\ Cauchy\ (\lambda i.\ f\ i\ x)\ then\ f\ i\ x\ else\ 0) \longrightarrow u'\ x$
unfolding *u'_def*
by (*rule convergent_LIMSEQ_iff*[*THEN iffD1*])
qed *measurable*
then show *?thesis*
unfolding $*$ **by** *measurable*
qed

lemma *borel_measurable_suminf*[*measurable* (*raw*)]:
fixes $f :: nat \Rightarrow 'a \Rightarrow 'b{::}\{banach, second\_countable\_topology\}$
assumes *f*[*measurable*]: $\bigwedge i.\ f\ i \in borel\_measurable\ M$
shows $(\lambda x.\ suminf\ (\lambda i.\ f\ i\ x)) \in borel\_measurable\ M$
unfolding *suminf_def sums_def*[*abs_def*] *lim_def*[*symmetric*] **by** *simp*

lemma *Collect_closed_imp_pred_borel*: $closed\ \{x.\ P\ x\} \Longrightarrow Measurable.pred\ borel\ P$
by (*simp add*: *pred_def*)

lemma *isCont_borel_pred*[*measurable*]:
fixes $f :: 'b{::}metric\_space \Rightarrow 'a{::}metric\_space$
shows *Measurable.pred borel* (*isCont f*)
proof (*subst measurable_cong*)
let $?I = \lambda j.\ inverse(real\ (Suc\ j))$
show $isCont\ f\ x = (\forall i.\ \exists j.\ \forall y\ z.\ dist\ x\ y < ?I\ j \wedge dist\ x\ z < ?I\ j \longrightarrow dist\ (f\ y)\ (f\ z) \leq ?I\ i)$ **for** x
unfolding *continuous_at_eps_delta*
proof *safe*
fix i **assume** $\forall e{>}0.\ \exists d{>}0.\ \forall y.\ dist\ y\ x < d \longrightarrow dist\ (f\ y)\ (f\ x) < e$
moreover have $0 < ?I\ i\ /\ 2$
by *simp*

```
    ultimately obtain d where d: 0 < d ⋀y. dist x y < d ⟹ dist (f y) (f x) <
?I i / 2
      by (metis dist_commute)
    then obtain j where j: ?I j < d
      by (metis reals_Archimedean)

    show ∃j. ∀y z. dist x y < ?I j ∧ dist x z < ?I j ⟶ dist (f y) (f z) ≤ ?I i
    proof (safe intro!: exI[where x=j])
      fix y z assume *: dist x y < ?I j dist x z < ?I j
      have dist (f y) (f z) ≤ dist (f y) (f x) + dist (f z) (f x)
        by (rule dist_triangle2)
      also have ... < ?I i / 2 + ?I i / 2
        by (intro add_strict_mono d less_trans[OF _ j] *)
      also have ... ≤ ?I i
        by (simp add: field_simps)
      finally show dist (f y) (f z) ≤ ?I i
        by simp
    qed
  next
    fix e::real assume 0 < e
    then obtain n where n: ?I n < e
      by (metis reals_Archimedean)
    assume ∀i. ∃j. ∀y z. dist x y < ?I j ∧ dist x z < ?I j ⟶ dist (f y) (f z) ≤
?I i
    from this[THEN spec, of Suc n]
    obtain j where j: ⋀y z. dist x y < ?I j ⟹ dist x z < ?I j ⟹ dist (f y) (f
z) ≤ ?I (Suc n)
      by auto

    show ∃d>0. ∀y. dist y x < d ⟶ dist (f y) (f x) < e
    proof (safe intro!: exI[of _ ?I j])
      fix y assume dist y x < ?I j
      then have dist (f y) (f x) ≤ ?I (Suc n)
        by (intro j) (auto simp: dist_commute)
      also have ?I (Suc n) < ?I n
        by simp
      also note n
      finally show dist (f y) (f x) < e .
    qed simp
  qed
qed (intro pred_intros_countable closed_Collect_all closed_Collect_le open_Collect_less
        Collect_closed_imp_pred_borel closed_Collect_imp open_Collect_conj
continuous_intros)

lemma isCont_borel:
  fixes f :: 'b::metric_space ⇒ 'a::metric_space
  shows {x. isCont f x} ∈ sets borel
  by simp
```

lemma *is_real_interval*:
assumes *S*: *is_interval S*
shows ∃ *a b*::*real*. *S* = {} ∨ *S* = *UNIV* ∨ *S* = {..<*b*} ∨ *S* = {..*b*} ∨ *S* = {*a*<..} ∨ *S* = {*a*..} ∨
S = {*a*<..<*b*} ∨ *S* = {*a*<..*b*} ∨ *S* = {*a*..<*b*} ∨ *S* = {*a*..*b*}
using *S* **unfolding** *is_interval_1* **by** (*blast intro*: *interval_cases*)

lemma *real_interval_borel_measurable*:
assumes *is_interval* (*S*::*real set*)
shows *S* ∈ *sets borel*
proof −
from *assms is_real_interval* **have** ∃ *a b*::*real*. *S* = {} ∨ *S* = *UNIV* ∨ *S* = {..<*b*} ∨ *S* = {..*b*} ∨
S = {*a*<..} ∨ *S* = {*a*..} ∨ *S* = {*a*<..<*b*} ∨ *S* = {*a*<..*b*} ∨ *S* = {*a*..<*b*} ∨ *S* = {*a*..*b*} **by** *auto*
then show *?thesis*
by *auto*
qed

The next lemmas hold in any second countable linorder (including ennreal or ereal for instance), but in the current state they are restricted to reals.

lemma *borel_measurable_mono_on_fnc*:
fixes *f* :: *real* ⇒ *real* **and** *A* :: *real set*
assumes *mono_on A f*
shows *f* ∈ *borel_measurable* (*restrict_space borel A*)
apply (*rule measurable_restrict_countable*[*OF mono_on_ctble_discont*[*OF assms*]])
apply (*auto intro*!: *image_eqI*[**where** *x*={*x*} **for** *x*] *simp*: *sets_restrict_space*)
apply (*auto simp add*: *sets_restrict_restrict_space continuous_on_eq_continuous_within*
cong: *measurable_cong_sets*
intro!: *borel_measurable_continuous_on_restrict intro*: *continuous_within_subset*)
done

lemma *borel_measurable_piecewise_mono*:
fixes *f*::*real* ⇒ *real* **and** *C*::*real set set*
assumes *countable C* ⋀*c*. *c* ∈ *C* ⟹ *c* ∈ *sets borel* ⋀*c*. *c* ∈ *C* ⟹ *mono_on c f* (⋃ *C*) = *UNIV*
shows *f* ∈ *borel_measurable borel*
by (*rule measurable_piecewise_restrict*[*of C*], *auto intro*: *borel_measurable_mono_on_fnc simp*: *assms*)

lemma *borel_measurable_mono*:
fixes *f* :: *real* ⇒ *real*
shows *mono f* ⟹ *f* ∈ *borel_measurable borel*
using *borel_measurable_mono_on_fnc*[*of UNIV f*] **by** (*simp add*: *mono_def mono_on_def*)

lemma *measurable_bdd_below_real*[*measurable* (*raw*)]:
fixes *F* :: *'a* ⇒ *'i* ⇒ *real*

assumes [*simp*]: *countable I* **and** [*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow F\ i \in M \to_M borel$
shows *Measurable.pred M* $(\lambda x.\ bdd\_below\ ((\lambda i.\ F\ i\ x)`I))$
proof (*subst measurable_cong*)
show *bdd_below* $((\lambda i.\ F\ i\ x)`I) \longleftrightarrow (\exists\, q{\in}\mathbb{Z}.\ \forall\, i{\in}I.\ q \le F\ i\ x)$ **for** x
by (*auto simp*: *bdd_below_def intro*!: *bexI*[*of* _ *of_int* (*floor* _)] *intro*: *order_trans of_int_floor_le*)
show *Measurable.pred M* $(\lambda w.\ \exists\, q{\in}\mathbb{Z}.\ \forall\, i{\in}I.\ q \le F\ i\ w)$
using *countable_int* **by** *measurable*
qed

lemma *borel_measurable_cINF_real*[*measurable* (*raw*)]:
fixes $F :: \_ \Rightarrow \_ \Rightarrow real$
assumes [*simp*]: *countable I*
assumes *F*[*measurable*]: $\bigwedge i.\ i \in I \Longrightarrow F\ i \in borel\_measurable\ M$
shows $(\lambda x.\ INF\ i{\in}I.\ F\ i\ x) \in borel\_measurable\ M$
proof (*rule measurable_piecewise_restrict*)
let $?\Omega = \{x{\in}space\ M.\ bdd\_below\ ((\lambda i.\ F\ i\ x)`I)\}$
show *countable* $\{?\Omega, -\ ?\Omega\}$ *space M* $\subseteq \bigcup \{?\Omega, -\ ?\Omega\}$ $\bigwedge X.\ X \in \{?\Omega, -\ ?\Omega\} \Longrightarrow X \cap space\ M \in sets\ M$
by *auto*
fix *X* **assume** $X \in \{?\Omega, -\ ?\Omega\}$ **then show** $(\lambda x.\ INF\ i{\in}I.\ F\ i\ x) \in borel\_measurable\ (restrict\_space\ M\ X)$
proof *safe*
show $(\lambda x.\ INF\ i{\in}I.\ F\ i\ x) \in borel\_measurable\ (restrict\_space\ M\ ?\Omega)$
by (*intro borel_measurable_cINF measurable_restrict_space1 F*)
(*auto simp*: *space_restrict_space*)
show $(\lambda x.\ INF\ i{\in}I.\ F\ i\ x) \in borel\_measurable\ (restrict\_space\ M\ (-?\Omega))$
proof (*subst measurable_cong*)
fix *x* **assume** $x \in space\ (restrict\_space\ M\ (-?\Omega))$
then have $\neg\ (\forall\, i{\in}I.\ -\ F\ i\ x \le y)$ **for** *y*
by (*auto simp*: *space_restrict_space bdd_above_def bdd_above_uminus*[*symmetric*])
then show $(INF\ i{\in}I.\ F\ i\ x) = -\ (THE\ x.\ False)$
by (*auto simp*: *space_restrict_space Inf_real_def Sup_real_def Least_def simp del*: *Set.ball_simps*(*10*))
qed *simp*
qed
qed

lemma *borel_Ici*: *borel* = *sigma UNIV* (*range* $(\lambda x{::}real.\ \{x\ ..\})$)
proof (*safe intro*!: *borel_eq_sigmaI1*[*OF borel_Iio*])
fix *x* :: *real*
have *eq*: $\{..{<}x\} = space\ (sigma\ UNIV\ (range\ atLeast)) - \{x\ ..\}$
by *auto*
show $\{..{<}x\} \in sets\ (sigma\ UNIV\ (range\ atLeast))$
unfolding *eq* **by** (*intro sets.compl_sets*) *auto*
qed *auto*

lemma *borel_measurable_pred_less*[*measurable* (*raw*)]:
fixes *f* :: *'a* ⇒ *'b*::{*second_countable_topology*, *linorder_topology*}

shows $f \in$ *borel_measurable* $M \Longrightarrow g \in$ *borel_measurable* $M \Longrightarrow$ *Measurable.pred* M $(\lambda w.\ f\ w < g\ w)$
unfolding *Measurable.pred_def* **by** (*rule borel_measurable_less*)

no_notation
eucl_less (**infix** $<e$ *50*)

lemma *borel_measurable_Max2*[*measurable* (*raw*)]:
fixes f::_ $\Rightarrow$ _ $\Rightarrow$ 'a::{*second_countable_topology*, *dense_linorder*, *linorder_topology*}
assumes *finite I*
and [*measurable*]: $\bigwedge i.\ f\ i \in$ *borel_measurable* M
shows $(\lambda x.\ Max\{f\ i\ x\ |i.\ i \in I\}) \in$ *borel_measurable* M
by (*simp add*: *borel_measurable_Max*[*OF assms*(*1*), **where** *?f*=*f* **and** *?M*=*M*] *Setcompr_eq_image*)

lemma *measurable_compose_n* [*measurable* (*raw*)]:
assumes $T \in$ *measurable* M M
shows $(T\widehat{\ }n) \in$ *measurable* M M
by (*induction n*, *auto simp add*: *measurable_compose*[*OF* _ *assms*])

lemma *measurable_real_imp_nat*:
fixes f::'$a \Rightarrow$ *nat*
assumes [*measurable*]: $(\lambda x.\ real(f\ x)) \in$ *borel_measurable* M
shows $f \in$ *measurable* M (*count_space UNIV*)
proof −
let *?g* = $(\lambda x.\ real(f\ x))$
have $\bigwedge(n$::*nat*). *?g*−‘$(\{real\ n\}) \cap$ *space* $M = f-$‘$\{n\} \cap$ *space* M **by** *auto*
moreover have $\bigwedge(n$::*nat*). *?g*−‘$(\{real\ n\}) \cap$ *space* $M \in$ *sets* M **using** *assms* **by** *measurable*
ultimately have $\bigwedge(n$::*nat*). $f-$‘$\{n\} \cap$ *space* $M \in$ *sets* M **by** *simp*
then show *?thesis* **using** *measurable_count_space_eq2_countable* **by** *blast*
qed

lemma *measurable_equality_set* [*measurable*]:
fixes $f\ g$::_$\Rightarrow$ 'a::{*second_countable_topology*, *t2_space*}
assumes [*measurable*]: $f \in$ *borel_measurable* M $g \in$ *borel_measurable* M
shows $\{x \in$ *space* $M.\ f\ x = g\ x\} \in$ *sets* M

proof −
define A **where** $A = \{x \in$ *space* $M.\ f\ x = g\ x\}$
define B **where** $B = \{y.\ \exists x$::'$a.\ y = (x,x)\}$
have $A = (\lambda x.\ (f\ x,\ g\ x))-$‘$B \cap$ *space* M **unfolding** *A_def B_def* **by** *auto*
moreover have $(\lambda x.\ (f\ x,\ g\ x)) \in$ *borel_measurable* M **by** *simp*
moreover have $B \in$ *sets borel* **unfolding** *B_def* **by** (*simp add*: *closed_diagonal*)
ultimately have $A \in$ *sets* M **by** *simp*
then show *?thesis* **unfolding** *A_def* **by** *simp*
qed

lemma *measurable_inequality_set* [*measurable*]:

fixes $f\ g$::_ $\Rightarrow$ 'a::{*second_countable_topology*, *linorder_topology*}
assumes [*measurable*]: $f \in$ *borel_measurable M* $g \in$ *borel_measurable M*
shows $\{x \in$ *space M*. $f\ x \leq g\ x\} \in$ *sets M*
$\{x \in$ *space M*. $f\ x < g\ x\} \in$ *sets M*
$\{x \in$ *space M*. $f\ x \geq g\ x\} \in$ *sets M*
$\{x \in$ *space M*. $f\ x > g\ x\} \in$ *sets M*
proof −
define *F* **where** $F = (\lambda x.\ (f\ x,\ g\ x))$
have $*$ [*measurable*]: *F* $\in$ *borel_measurable M* **unfolding** *F_def* **by** *simp*

have $\{x \in$ *space M*. $f\ x \leq g\ x\} = F -` \{(x,\ y) \mid x\ y.\ x \leq y\} \cap$ *space M* **unfolding** *F_def* **by** *auto*
moreover have $\{(x,\ y) \mid x\ y.\ x \leq (y::'a)\} \in$ *sets borel* **using** *closed_subdiagonal borel_closed* **by** *blast*
ultimately show $\{x \in$ *space M*. $f\ x \leq g\ x\} \in$ *sets M* **using** $*$ **by** (*metis* (*mono_tags*, *lifting*) *measurable_sets*)

have $\{x \in$ *space M*. $f\ x < g\ x\} = F -` \{(x,\ y) \mid x\ y.\ x < y\} \cap$ *space M* **unfolding** *F_def* **by** *auto*
moreover have $\{(x,\ y) \mid x\ y.\ x < (y::'a)\} \in$ *sets borel* **using** *open_subdiagonal borel_open* **by** *blast*
ultimately show $\{x \in$ *space M*. $f\ x < g\ x\} \in$ *sets M* **using** $*$ **by** (*metis* (*mono_tags*, *lifting*) *measurable_sets*)

have $\{x \in$ *space M*. $f\ x \geq g\ x\} = F -` \{(x,\ y) \mid x\ y.\ x \geq y\} \cap$ *space M* **unfolding** *F_def* **by** *auto*
moreover have $\{(x,\ y) \mid x\ y.\ x \geq (y::'a)\} \in$ *sets borel* **using** *closed_superdiagonal borel_closed* **by** *blast*
ultimately show $\{x \in$ *space M*. $f\ x \geq g\ x\} \in$ *sets M* **using** $*$ **by** (*metis* (*mono_tags*, *lifting*) *measurable_sets*)

have $\{x \in$ *space M*. $f\ x > g\ x\} = F -` \{(x,\ y) \mid x\ y.\ x > y\} \cap$ *space M* **unfolding** *F_def* **by** *auto*
moreover have $\{(x,\ y) \mid x\ y.\ x > (y::'a)\} \in$ *sets borel* **using** *open_superdiagonal borel_open* **by** *blast*
ultimately show $\{x \in$ *space M*. $f\ x > g\ x\} \in$ *sets M* **using** $*$ **by** (*metis* (*mono_tags*, *lifting*) *measurable_sets*)
qed

proposition *measurable_limit* [*measurable*]:
fixes *f*::*nat* $\Rightarrow$ 'a $\Rightarrow$ 'b::*first_countable_topology*
assumes [*measurable*]: $\bigwedge n$::*nat*. *f n* $\in$ *borel_measurable M*
shows *Measurable.pred M* $(\lambda x.\ (\lambda n.\ f\ n\ x) \longrightarrow c)$
proof −
obtain *A* :: *nat* $\Rightarrow$ 'b *set* **where** *A*:
$\bigwedge i.$ *open* $(A\ i)$
$\bigwedge i.$ $c \in A\ i$
$\bigwedge S.$ *open S* $\Longrightarrow$ $c \in S \Longrightarrow$ *eventually* $(\lambda i.\ A\ i \subseteq S)$ *sequentially*
by (*rule countable_basis_at_decseq*) *blast*

```
  have [measurable]: ⋀N i. (f N)−‘(A i) ∩ space M ∈ sets M using A(1) by auto
  then have mes: (⋂i. ⋃n. ⋂N∈{n..}. (f N)−‘(A i) ∩ space M) ∈ sets M by
blast

  have (u ⟶ c) ⟷ (∀i. eventually (λn. u n ∈ A i) sequentially) for u::nat
⇒ 'b
  proof
    assume u ⟶ c
     then have eventually (λn. u n ∈ A i) sequentially for i using A(1)[of i]
A(2)[of i]
      by (simp add: topological_tendstoD)
    then show (∀i. eventually (λn. u n ∈ A i) sequentially) by auto
  next
    assume H: (∀i. eventually (λn. u n ∈ A i) sequentially)
    show (u ⟶ c)
    proof (rule topological_tendstoI)
      fix S assume open S c ∈ S
      with A(3)[OF this] obtain i where A i ⊆ S
        using eventually_False_sequentially eventually_mono by blast
      moreover have eventually (λn. u n ∈ A i) sequentially using H by simp
      ultimately show ∀F n in sequentially. u n ∈ S
        by (simp add: eventually_mono subset_eq)
    qed
  qed
  then have {x. (λn. f n x) ⟶ c} = (⋂i. ⋃n. ⋂N∈{n..}. (f N)−‘(A i))
    by (auto simp add: atLeast_def eventually_at_top_linorder)
  then have {x ∈ space M. (λn. f n x) ⟶ c} = (⋂i. ⋃n. ⋂N∈{n..}. (f
N)−‘(A i) ∩ space M)
    by auto
  then have {x ∈ space M. (λn. f n x) ⟶ c} ∈ sets M using mes by simp
  then show ?thesis by auto
qed

lemma measurable_limit2 [measurable]:
  fixes u::nat ⇒ 'a ⇒ real
  assumes [measurable]: ⋀n. u n ∈ borel_measurable M v ∈ borel_measurable M
  shows Measurable.pred M (λx. (λn. u n x) ⟶ v x)
proof −
  define w where w = (λn x. u n x − v x)
  have [measurable]: w n ∈ borel_measurable M for n unfolding w_def by auto
  have ((λn. u n x) ⟶ v x) ⟷ ((λn. w n x) ⟶ 0) for x
    unfolding w_def using Lim_null by auto
  then show ?thesis using measurable_limit by auto
qed

lemma measurable_P_restriction [measurable (raw)]:
  assumes [measurable]: Measurable.pred M P A ∈ sets M
  shows {x ∈ A. P x} ∈ sets M
```

proof −
have $A \subseteq$ *space* M **using** *sets.sets_into_space*[*OF assms*(*2*)]**.**
then have $\{x \in A.\ P\ x\} = A \cap \{x \in$ *space* $M.\ P\ x\}$ **by** *blast*
then show *?thesis* **by** *auto*
qed

lemma *measurable_sum_nat* [*measurable* (*raw*)]:
fixes $f :: 'c \Rightarrow 'a \Rightarrow$ *nat*
assumes $\bigwedge i.\ i \in S \Longrightarrow f\ i \in$ *measurable* M (*count_space UNIV*)
shows $(\lambda x.\ \sum i{\in}S.\ f\ i\ x) \in$ *measurable* M (*count_space UNIV*)
proof *cases*
assume *finite* S
then show *?thesis* **using** *assms* **by** *induct auto*
qed *simp*

lemma *measurable_abs_powr* [*measurable*]:
fixes p::*real*
assumes [*measurable*]: $f \in$ *borel_measurable* M
shows $(\lambda x.\ |f\ x|$ *powr* $p) \in$ *borel_measurable* M
by *simp*

The next one is a variation around *measurable_restrict_space*.

lemma *measurable_restrict_space3*:
assumes $f \in$ *measurable* $M\ N$ **and**
$f \in A \to B$
shows $f \in$ *measurable* (*restrict_space* $M\ A$) (*restrict_space* $N\ B$)
proof −
have $f \in$ *measurable* (*restrict_space* $M\ A$) N **using** *assms*(*1*) *measurable_restrict_space1*
by *auto*
then show *?thesis* **by** (*metis Int_iff funcsetI funcset_mem*
measurable_restrict_space2[*of f*, *of restrict_space M A*, *of B*, *of N*] *assms*(*2*)
space_restrict_space)
qed

lemma *measurable_restrict_mono*:
assumes *f*: $f \in$ *restrict_space* $M\ A \to_M N$ **and** $B \subseteq A$
shows $f \in$ *restrict_space* $M\ B \to_M N$
by (*rule measurable_compose*[*OF measurable_restrict_space3 f*])
(*insert* ‹$B \subseteq A$›, *auto*)

The next one is a variation around *measurable_piecewise_restrict*.

lemma *measurable_piecewise_restrict2*:
assumes [*measurable*]: $\bigwedge n.\ A\ n \in$ *sets* M
and *space* $M = (\bigcup (n{::}nat).\ A\ n)$
$\bigwedge n.\ \exists h \in$ *measurable* $M\ N.\ (\forall x \in A\ n.\ f\ x = h\ x)$
shows $f \in$ *measurable* $M\ N$
proof (*rule measurableI*)
fix B **assume** [*measurable*]: $B \in$ *sets* N

```
  {
    fix n::nat
    obtain h where [measurable]: h ∈ measurable M N and ∀ x ∈ A n. f x = h x
using assms(3) by blast
    then have ∗: f−‘B ∩ A n = h−‘B ∩ A n by auto
   have h−‘B ∩ A n = h−‘B ∩ space M ∩ A n using assms(2) sets.sets_into_space
by auto
    then have h−‘B ∩ A n ∈ sets M by simp
    then have f−‘B ∩ A n ∈ sets M using ∗ by simp
  }
  then have (⋃ n. f−‘B ∩ A n) ∈ sets M by measurable
  moreover have f−‘B ∩ space M = (⋃ n. f−‘B ∩ A n) using assms(2) by blast
  ultimately show f−‘B ∩ space M ∈ sets M by simp
next
  fix x assume x ∈ space M
  then obtain n where x ∈ A n using assms(2) by blast
  obtain h where [measurable]: h ∈ measurable M N and ∀ x ∈ A n. f x = h x
using assms(3) by blast
  then have f x = h x using ‹x ∈ A n› by blast
  moreover have h x ∈ space N by (metis measurable_space ‹x ∈ space M› ‹h ∈
measurable M N›)
  ultimately show f x ∈ space N by simp
qed

end
```

6.5 Lebesgue Integration for Nonnegative Functions

```
theory Nonnegative_Lebesgue_Integration
  imports Measure_Space Borel_Space
begin
```

6.5.1 Approximating functions

```
lemma AE_upper_bound_inf_ennreal:
  fixes F G::'a ⇒ ennreal
  assumes ⋀e. (e::real) > 0 ⟹ AE x in M. F x ≤ G x + e
  shows AE x in M. F x ≤ G x
proof −
  have AE x in M. ∀ n::nat. F x ≤ G x + ennreal (1 / Suc n)
    using assms by (auto simp: AE_all_countable)
  then show ?thesis
  proof (eventually_elim)
    fix x assume x: ∀ n::nat. F x ≤ G x + ennreal (1 / Suc n)
    show F x ≤ G x
    proof (rule ennreal_le_epsilon)
      fix e :: real assume 0 < e
```

```
      then obtain n where n: 1 / Suc n < e
        by (blast elim: nat_approx_posE)
      have F x ≤ G x + 1 / Suc n
        using x by simp
      also have ... ≤ G x + e
        using n by (intro add_mono ennreal_leI) auto
      finally show F x ≤ G x + ennreal e .
    qed
  qed
qed

lemma AE_upper_bound_inf:
  fixes F G::'a ⇒ real
  assumes ⋀e. e > 0 ⟹ AE x in M. F x ≤ G x + e
  shows AE x in M. F x ≤ G x
proof −
  have AE x in M. F x ≤ G x + 1/real (n+1) for n::nat
    by (rule assms, auto)
  then have AE x in M. ∀ n::nat ∈ UNIV. F x ≤ G x + 1/real (n+1)
    by (rule AE_ball_countable', auto)
  moreover
  {
    fix x assume i: ∀ n::nat ∈ UNIV. F x ≤ G x + 1/real (n+1)
    have (λn. G x + 1/real (n+1)) ⟶ G x + 0
    by (rule tendsto_add, simp, rule LIMSEQ_ignore_initial_segment[OF lim_1_over_n,
of 1])
    then have F x ≤ G x using i LIMSEQ_le_const by fastforce
  }
  ultimately show ?thesis by auto
qed

lemma not_AE_zero_ennreal_E:
  fixes f::'a ⇒ ennreal
  assumes ¬ (AE x in M. f x = 0) and [measurable]: f ∈ borel_measurable M
  shows ∃ A∈sets M. ∃ e::real>0. emeasure M A > 0 ∧ (∀ x ∈ A. f x ≥ e)
proof −
  { assume ¬ (∃ e::real>0. {x ∈ space M. f x ≥ e} ∉ null_sets M)
    then have 0 < e ⟹ AE x in M. f x ≤ e for e :: real
      by (auto simp: not_le less_imp_le dest!: AE_not_in)
    then have AE x in M. f x ≤ 0
      by (intro AE_upper_bound_inf_ennreal[where G=λ_. 0]) simp
    then have False
      using assms by auto }
  then obtain e::real where e: e > 0 {x ∈ space M. f x ≥ e} ∉ null_sets M by
auto
  define A where A = {x ∈ space M. f x ≥ e}
  have 1 [measurable]: A ∈ sets M unfolding A_def by auto
  have 2: emeasure M A > 0
    using e(2) A_def ‹A ∈ sets M› by auto
```

have *3*: $\bigwedge x.\ x \in A \Longrightarrow f\ x \geq e$ **unfolding** *A_def* **by** *auto*
show *?thesis* **using** *e(1) 1 2 3* **by** *blast*
qed

lemma *not_AE_zero_E*:
fixes f::$'a \Rightarrow$ *real*
assumes *AE x in M. f x* $\geq$ *0*
$\neg$(*AE x in M. f x* = *0*)
and [*measurable*]: $f \in$ *borel_measurable M*
shows $\exists A\ e.\ A \in$ *sets M* $\wedge$ *e*>*0* $\wedge$ *emeasure M A* > *0* $\wedge$ $(\forall x \in A.\ f\ x \geq e)$
proof −
have $\exists e.\ e > 0 \wedge \{x \in$ *space M*$.\ f\ x \geq e\} \notin$ *null_sets M*
proof (*rule ccontr*)
assume ∗: $\neg(\exists e.\ e > 0 \wedge \{x \in$ *space M*$.\ f\ x \geq e\} \notin$ *null_sets M*)
{
fix *e*::*real* **assume** *e* > *0*
then have $\{x \in$ *space M*$.\ f\ x \geq e\} \in$ *null_sets M* **using** ∗ **by** *blast*
then have *AE x in M. x* $\notin \{x \in$ *space M*$.\ f\ x \geq e\}$ **using** *AE_not_in* **by** *blast*
then have *AE x in M. f x* $\leq$ *e* **by** *auto*
}
then have *AE x in M. f x* $\leq$ *0* **by** (*rule AE_upper_bound_inf*, *auto*)
then have *AE x in M. f x* = *0* **using** *assms(1)* **by** *auto*
then show *False* **using** *assms(2)* **by** *auto*
qed
then obtain *e* **where** *e*: *e*>*0* $\{x \in$ *space M*$.\ f\ x \geq e\} \notin$ *null_sets M* **by** *auto*
define *A* **where** $A = \{x \in$ *space M*$.\ f\ x \geq e\}$
have *1* [*measurable*]: *A* $\in$ *sets M* **unfolding** *A_def* **by** *auto*
have *2*: *emeasure M A* > *0*
using *e(2) A_def* ‹*A* $\in$ *sets M*› **by** *auto*
have *3*: $\bigwedge x.\ x \in A \Longrightarrow f\ x \geq e$ **unfolding** *A_def* **by** *auto*
show *?thesis*
using *e(1) 1 2 3* **by** *blast*
qed

6.5.2 Simple function

Our simple functions are not restricted to nonnegative real numbers. Instead they are just functions with a finite range and are measurable when singleton sets are measurable.

definition *simple_function M g* ⟷
finite (*g* ‘ *space M*) ∧
$(\forall x \in g$ ‘ *space M*$.\ g -$‘ $\{x\} \cap$ *space M* $\in$ *sets M*)

lemma *simple_functionD*:
assumes *simple_function M g*
shows *finite* (*g* ‘ *space M*) **and** *g* −‘ *X* ∩ *space M* $\in$ *sets M*
proof −
show *finite* (*g* ‘ *space M*)

using *assms* **unfolding** *simple_function_def* **by** *auto*
have $g -` X \cap$ *space* $M = g -` (X \cap g`$*space* $M) \cap$ *space* M **by** *auto*
also have $\ldots = (\bigcup x \in X \cap g`$*space* $M.\ g-`\{x\} \cap$ *space* $M)$ **by** *auto*
finally show $g -` X \cap$ *space* $M \in$ *sets* M **using** *assms*
by (*auto simp del*: *UN_simps simp*: *simple_function_def*)
qed

lemma *measurable_simple_function*[*measurable_dest*]:
simple_function $M\ f \Longrightarrow f \in$ *measurable* M (*count_space UNIV*)
unfolding *simple_function_def measurable_def*
proof *safe*
fix A **assume** *finite* $(f`$ *space* $M) \forall x \in f`$ *space* $M.\ f -` \{x\} \cap$ *space* $M \in$ *sets* M
then have $(\bigcup x \in f`$ *space* $M.$ *if* $x \in A$ *then* $f -` \{x\} \cap$ *space* M *else* $\{\}) \in$ *sets* M
by (*intro sets.finite_UN*) *auto*
also have $(\bigcup x \in f`$ *space* $M.$ *if* $x \in A$ *then* $f -` \{x\} \cap$ *space* M *else* $\{\}) = f -` A \cap$ *space* M
by (*auto split*: *if_split_asm*)
finally show $f -` A \cap$ *space* $M \in$ *sets* M **.**
qed *simp*

lemma *borel_measurable_simple_function*:
simple_function $M\ f \Longrightarrow f \in$ *borel_measurable* M
by (*auto dest*!: *measurable_simple_function simp*: *measurable_def*)

lemma *simple_function_measurable2*[*intro*]:
assumes *simple_function* $M\ f$ *simple_function* $M\ g$
shows $f -` A \cap g -` B \cap$ *space* $M \in$ *sets* M
proof −
have $f -` A \cap g -` B \cap$ *space* $M = (f -` A \cap$ *space* $M) \cap (g -` B \cap$ *space* $M)$
by *auto*
then show *?thesis* **using** *assms*[*THEN simple_functionD*(*2*)] **by** *auto*
qed

lemma *simple_function_indicator_representation*:
fixes $f ::'a \Rightarrow$ *ennreal*
assumes f: *simple_function* $M\ f$ **and** x: $x \in$ *space* M
shows $f\ x = (\sum y \in f`$ *space* $M.\ y *$ *indicator* $(f -` \{y\} \cap$ *space* $M)\ x)$
(**is** *?l* = *?r*)
proof −
have *?r* $= (\sum y \in f`$ *space* $M.$
(*if* $y = f\ x$ *then* $y *$ *indicator* $(f -` \{y\} \cap$ *space* $M)\ x$ *else* *0*))
by (*auto intro*!: *sum.cong*)
also have ... = $f\ x *$ *indicator* $(f -` \{f\ x\} \cap$ *space* $M)\ x$
using *assms* **by** (*auto dest*: *simple_functionD*)
also have ... = $f\ x$ **using** x **by** (*auto simp*: *indicator_def*)
finally show *?thesis* **by** *auto*
qed

lemma *simple_function_notspace*:
simple_function M (λ*x*. *h x* ∗ *indicator* (− *space M*) *x*::*ennreal*) (**is** *simple_function M ?h*)
proof −
have *?h* ‘ *space M* ⊆ {*0*} **unfolding** *indicator_def* **by** *auto*
hence [*simp*, *intro*]: *finite* (*?h* ‘ *space M*) **by** (*auto intro*: *finite_subset*)
have *?h* −‘ {*0*} ∩ *space M* = *space M* **by** *auto*
thus *?thesis* **unfolding** *simple_function_def* **by** (*auto simp add*: *image_constant_conv*)
qed

lemma *simple_function_cong*:
assumes ⋀*t*. *t* ∈ *space M* ⟹ *f t* = *g t*
shows *simple_function M f* ⟷ *simple_function M g*
proof −
have ⋀*x*. *f* −‘ {*x*} ∩ *space M* = *g* −‘ {*x*} ∩ *space M*
using *assms* **by** *auto*
with *assms* **show** *?thesis*
by (*simp add*: *simple_function_def cong*: *image_cong*)
qed

lemma *simple_function_cong_algebra*:
assumes *sets N* = *sets M space N* = *space M*
shows *simple_function M f* ⟷ *simple_function N f*
unfolding *simple_function_def assms* **..**

lemma *simple_function_borel_measurable*:
fixes *f* :: *'a* ⇒ *'x*::{*t2_space*}
assumes *f* ∈ *borel_measurable M* **and** *finite* (*f* ‘ *space M*)
shows *simple_function M f*
using *assms* **unfolding** *simple_function_def*
by (*auto intro*: *borel_measurable_vimage*)

lemma *simple_function_iff_borel_measurable*:
fixes *f* :: *'a* ⇒ *'x*::{*t2_space*}
shows *simple_function M f* ⟷ *finite* (*f* ‘ *space M*) ∧ *f* ∈ *borel_measurable M*
by (*metis borel_measurable_simple_function simple_functionD*(*1*) *simple_function_borel_measurable*)

lemma *simple_function_eq_measurable*:
simple_function M f ⟷ *finite* (*f*‘*space M*) ∧ *f* ∈ *measurable M* (*count_space UNIV*)
using *measurable_simple_function*[*of M f*] **by** (*fastforce simp*: *simple_function_def*)

lemma *simple_function_const*[*intro*, *simp*]:
simple_function M (λ*x*. *c*)
by (*auto intro*: *finite_subset simp*: *simple_function_def*)
lemma *simple_function_compose*[*intro*, *simp*]:
assumes *simple_function M f*
shows *simple_function M* (*g* ∘ *f*)
unfolding *simple_function_def*

proof *safe*
show *finite* ((*g* ∘ *f*) ‘ *space M*)
using *assms* **unfolding** *simple_function_def image_comp* [*symmetric*] **by** *auto*
next
fix *x* **assume** *x* ∈ *space M*
let *?G* = *g* −‘ {*g* (*f x*)} ∩ (*f*‘*space M*)
have ∗: (*g* ∘ *f*) −‘ {(*g* ∘ *f*) *x*} ∩ *space M* =
(⋃*x*∈*?G*. *f* −‘ {*x*} ∩ *space M*) **by** *auto*
show (*g* ∘ *f*) −‘ {(*g* ∘ *f*) *x*} ∩ *space M* ∈ *sets M*
using *assms* **unfolding** *simple_function_def* ∗
by (*rule_tac sets.finite_UN*) *auto*
qed

lemma *simple_function_indicator*[*intro*, *simp*]:
assumes *A* ∈ *sets M*
shows *simple_function M* (*indicator A*)
proof −
have *indicator A* ‘ *space M* ⊆ {*0*, *1*} (**is** *?S* ⊆ _)
by (*auto simp*: *indicator_def*)
hence *finite ?S* **by** (*rule finite_subset*) *simp*
moreover have − *A* ∩ *space M* = *space M* − *A* **by** *auto*
ultimately show *?thesis* **unfolding** *simple_function_def*
using *assms* **by** (*auto simp*: *indicator_def* [*abs_def*])
qed

lemma *simple_function_Pair*[*intro*, *simp*]:
assumes *simple_function M f*
assumes *simple_function M g*
shows *simple_function M* (λ*x*. (*f x*, *g x*)) (**is** *simple_function M ?p*)
unfolding *simple_function_def*
proof *safe*
show *finite* (*?p* ‘ *space M*)
using *assms* **unfolding** *simple_function_def*
by (*rule_tac finite_subset*[*of* _ *f*‘*space M* × *g*‘*space M*]) *auto*
next
fix *x* **assume** *x* ∈ *space M*
have (λ*x*. (*f x*, *g x*)) −‘ {(*f x*, *g x*)} ∩ *space M* =
(*f* −‘ {*f x*} ∩ *space M*) ∩ (*g* −‘ {*g x*} ∩ *space M*)
by *auto*
with ‹*x* ∈ *space M*› **show** (λ*x*. (*f x*, *g x*)) −‘ {(*f x*, *g x*)} ∩ *space M* ∈ *sets M*
using *assms* **unfolding** *simple_function_def* **by** *auto*
qed

lemma *simple_function_compose1*:
assumes *simple_function M f*
shows *simple_function M* (λ*x*. *g* (*f x*))
using *simple_function_compose*[*OF assms*, *of g*]
by (*simp add*: *comp_def*)

lemma *simple_function_compose2*:
assumes *simple_function M f* **and** *simple_function M g*
shows *simple_function M* (λ*x*. *h* (*f x*) (*g x*))
proof −
have *simple_function M* ((λ(*x*, *y*). *h x y*) ∘ (λ*x*. (*f x*, *g x*)))
using *assms* **by** *auto*
thus *?thesis* **by** (*simp_all add*: *comp_def*)
qed

lemmas *simple_function_add*[*intro*, *simp*] = *simple_function_compose2*[**where** *h*=(+)]
and *simple_function_diff*[*intro*, *simp*] = *simple_function_compose2*[**where** *h*=(−)]
and *simple_function_uminus*[*intro*, *simp*] = *simple_function_compose*[**where** *g*=*uminus*]
and *simple_function_mult*[*intro*, *simp*] = *simple_function_compose2*[**where** *h*=(∗)]
and *simple_function_div*[*intro*, *simp*] = *simple_function_compose2*[**where** *h*=(/)]
and *simple_function_inverse*[*intro*, *simp*] = *simple_function_compose*[**where** *g*=*inverse*]
and *simple_function_max*[*intro*, *simp*] = *simple_function_compose2*[**where** *h*=*max*]

lemma *simple_function_sum*[*intro*, *simp*]:
assumes ⋀*i*. *i* ∈ *P* ⟹ *simple_function M* (*f i*)
shows *simple_function M* (λ*x*. ∑ *i*∈*P*. *f i x*)
proof *cases*
assume *finite P* **from** *this assms* **show** *?thesis* **by** *induct auto*
qed *auto*

lemma *simple_function_ennreal*[*intro*, *simp*]:
fixes *f g* :: *'a* ⇒ *real* **assumes** *sf*: *simple_function M f*
shows *simple_function M* (λ*x*. *ennreal* (*f x*))
by (*rule simple_function_compose1*[*OF sf*])

lemma *simple_function_real_of_nat*[*intro*, *simp*]:
fixes *f g* :: *'a* ⇒ *nat* **assumes** *sf*: *simple_function M f*
shows *simple_function M* (λ*x*. *real* (*f x*))
by (*rule simple_function_compose1*[*OF sf*])

lemma *borel_measurable_implies_simple_function_sequence*:
fixes *u* :: *'a* ⇒ *ennreal*
assumes *u*[*measurable*]: *u* ∈ *borel_measurable M*
shows ∃*f*. *incseq f* ∧ (∀ *i*. (∀ *x*. *f i x* < *top*) ∧ *simple_function M* (*f i*)) ∧ *u* = (*SUP i*. *f i*)
proof −
define *f* **where** [*abs_def*]:
f i x = *real_of_int* (*floor* (*enn2real* (*min i* (*u x*)) ∗ 2^*i*)) / 2^*i* **for** *i x*

have [*simp*]: *0* ≤ *f i x* **for** *i x*
by (*auto simp*: *f_def intro*!: *divide_nonneg_nonneg mult_nonneg_nonneg*

```
enn2real_nonneg)

  have *: 2^n * real_of_int x = real_of_int (2^n * x) for n x
    by simp

  have real_of_int ⌊real i * 2 ^ i⌋ = real_of_int ⌊i * 2 ^ i⌋ for i
    by (intro arg_cong[where f=real_of_int]) simp
  then have [simp]: real_of_int ⌊real i * 2 ^ i⌋ = i * 2 ^ i for i
    unfolding floor_of_nat by simp

  have incseq f
  proof (intro monoI le_funI)
    fix m n :: nat and x assume m ≤ n
    moreover
    { fix d :: nat
      have ⌊2^d::real⌋ * ⌊2^m * enn2real (min (of_nat m) (u x))⌋ ≤
        ⌊2^d * (2^m * enn2real (min (of_nat m) (u x)))⌋
        by (rule le_mult_floor) (auto)
       also have ... ≤ ⌊2^d * (2^m * enn2real (min (of_nat d + of_nat m) (u
x)))⌋
        by (intro floor_mono mult_mono enn2real_mono min.mono)
           (auto simp: min_less_iff_disj of_nat_less_top)
      finally have f m x ≤ f (m + d) x
        unfolding f_def
        by (auto simp: field_simps power_add * simp del: of_int_mult) }
    ultimately show f m x ≤ f n x
      by (auto simp add: le_iff_add)
  qed
  then have inc_f: incseq (λi. ennreal (f i x)) for x
    by (auto simp: incseq_def le_fun_def)
  then have incseq (λi x. ennreal (f i x))
    by (auto simp: incseq_def le_fun_def)
  moreover
  have simple_function M (f i) for i
  proof (rule simple_function_borel_measurable)
    have ⌊enn2real (min (of_nat i) (u x)) * 2 ^ i⌋ ≤ ⌊int i * 2 ^ i⌋ for x
      by (cases u x rule: ennreal_cases)
         (auto split: split_min intro!: floor_mono)
    then have f i ‘ space M ⊆ (λn. real_of_int n / 2^i) ‘ {0 .. of_nat i * 2^i}
      unfolding floor_of_int by (auto simp: f_def intro!: imageI)
    then show finite (f i ‘ space M)
      by (rule finite_subset) auto
    show f i ∈ borel_measurable M
      unfolding f_def enn2real_def by measurable
  qed
  moreover
  { fix x
    have (SUP i. ennreal (f i x)) = u x
    proof (cases u x rule: ennreal_cases)
```

```
    case top then show ?thesis
    by (simp add: f_def inf_min[symmetric] ennreal_of_nat_eq_real_of_nat[symmetric]
                  ennreal_SUP_of_nat_eq_top)
  next
    case (real r)
    obtain n where r ≤ of_nat n using real_arch_simple by auto
    then have min_eq_r: ∀F x in sequentially. min (real x) r = r
      by (auto simp: eventually_sequentially intro!: exI[of _ n] split: split_min)

    have (λi. real_of_int ⌊min (real i) r * 2^i⌋ / 2^i) ⟶ r
    proof (rule tendsto_sandwich)
      show (λn. r - (1/2)^n) ⟶ r
        by (auto intro!: tendsto_eq_intros LIMSEQ_power_zero)
      show ∀F n in sequentially. real_of_int ⌊min (real n) r * 2 ^ n⌋ / 2 ^ n ≤
r
        using min_eq_r by eventually_elim (auto simp: field_simps)
      have *: r * (2 ^ n * 2 ^ n) ≤ 2^n + 2^n * real_of_int ⌊r * 2 ^ n⌋ for n
       using real_of_int_floor_ge_diff_one[of r * 2^n, THEN mult_left_mono,
of 2^n]
        by (auto simp: field_simps)
      show ∀F n in sequentially. r - (1/2)^n ≤ real_of_int ⌊min (real n) r *
2 ^ n⌋ / 2 ^ n
        using min_eq_r by eventually_elim (insert *, auto simp: field_simps)
    qed auto
    then have (λi. ennreal (f i x)) ⟶ ennreal r
      by (simp add: real f_def ennreal_of_nat_eq_real_of_nat min_ennreal)
    from LIMSEQ_unique[OF LIMSEQ_SUP[OF inc_f] this]
    show ?thesis
      by (simp add: real)
  qed }
  ultimately show ?thesis
    by (intro exI [of _ λi x. ennreal (f i x)]) (auto simp add: image_comp)
qed

lemma borel_measurable_implies_simple_function_sequence':
  fixes u :: 'a ⇒ ennreal
  assumes u: u ∈ borel_measurable M
  obtains f where
    ⋀i. simple_function M (f i) incseq f ⋀i x. f i x < top ⋀x. (SUP i. f i x) = u x
  using borel_measurable_implies_simple_function_sequence [OF u]
  by (metis SUP_apply)

lemma simple_function_induct
    [consumes 1, case_names cong set mult add, induct set: simple_function]:
  fixes u :: 'a ⇒ ennreal
  assumes u: simple_function M u
  assumes cong: ⋀f g. simple_function M f ⟹ simple_function M g ⟹ (AE x
in M. f x = g x) ⟹ P f ⟹ P g
  assumes set: ⋀A. A ∈ sets M ⟹ P (indicator A)
```

```
  assumes mult: ⋀u c. P u ⟹ P (λx. c * u x)
  assumes add: ⋀u v. P u ⟹ P v ⟹ P (λx. v x + u x)
  shows P u
proof (rule cong)
  from AE_space show AE x in M. (∑ y∈u ‘ space M. y * indicator (u -‘ {y}
∩ space M) x) = u x
  proof eventually_elim
    fix x assume x: x ∈ space M
    from simple_function_indicator_representation[OF u x]
    show (∑ y∈u ‘ space M. y * indicator (u -‘ {y} ∩ space M) x) = u x ..
  qed
next
  from u have finite (u ‘ space M)
    unfolding simple_function_def by auto
  then show P (λx. ∑ y∈u ‘ space M. y * indicator (u -‘ {y} ∩ space M) x)
  proof induct
    case empty show ?case
      using set[of {}] by (simp add: indicator_def[abs_def])
  qed (auto intro!: add mult set simple_functionD u)
next
  show simple_function M (λx. (∑ y∈u ‘ space M. y * indicator (u -‘ {y} ∩
space M) x))
    apply (subst simple_function_cong)
    apply (rule simple_function_indicator_representation[symmetric])
    apply (auto intro: u)
    done
qed fact

lemma simple_function_induct_nn[consumes 1, case_names cong set mult add]:
  fixes u :: 'a ⇒ ennreal
  assumes u: simple_function M u
  assumes cong: ⋀f g. simple_function M f ⟹ simple_function M g ⟹ (⋀x. x
∈ space M ⟹ f x = g x) ⟹ P f ⟹ P g
  assumes set: ⋀A. A ∈ sets M ⟹ P (indicator A)
  assumes mult: ⋀u c. simple_function M u ⟹ P u ⟹ P (λx. c * u x)
  assumes add: ⋀u v. simple_function M u ⟹ P u ⟹ simple_function M v
⟹ (⋀x. x ∈ space M ⟹ u x = 0 ∨ v x = 0) ⟹ P v ⟹ P (λx. v x + u x)
  shows P u
proof -
  show ?thesis
  proof (rule cong)
    fix x assume x: x ∈ space M
    from simple_function_indicator_representation[OF u x]
    show (∑ y∈u ‘ space M. y * indicator (u -‘ {y} ∩ space M) x) = u x ..
  next
    show simple_function M (λx. (∑ y∈u ‘ space M. y * indicator (u -‘ {y} ∩
space M) x))
      apply (subst simple_function_cong)
      apply (rule simple_function_indicator_representation[symmetric])
```

```
    apply (auto intro: u)
    done
  next
    from u have finite (u ‘ space M)
      unfolding simple_function_def by auto
    then show P (λx. ∑ y∈u ‘ space M. y * indicator (u −‘ {y} ∩ space M) x)
    proof induct
      case empty show ?case
        using set[of {}] by (simp add: indicator_def[abs_def])
    next
      case (insert x S)
      { fix z have (∑ y∈S. y * indicator (u −‘ {y} ∩ space M) z) = 0 ∨
          x * indicator (u −‘ {x} ∩ space M) z = 0
          using insert by (subst sum_eq_0_iff) (auto simp: indicator_def) }
      note disj = this
      from insert show ?case
       by (auto intro!: add mult set simple_functionD u simple_function_sum disj)
    qed
  qed fact
qed

lemma borel_measurable_induct
    [consumes 1, case_names cong set mult add seq, induct set: borel_measurable]:
  fixes u :: 'a ⇒ ennreal
  assumes u: u ∈ borel_measurable M
  assumes cong: ⋀f g. f ∈ borel_measurable M ⟹ g ∈ borel_measurable M ⟹
(⋀x. x ∈ space M ⟹ f x = g x) ⟹ P g ⟹ P f
  assumes set: ⋀A. A ∈ sets M ⟹ P (indicator A)
  assumes mult': ⋀u c. c < top ⟹ u ∈ borel_measurable M ⟹ (⋀x. x ∈ space
M ⟹ u x < top) ⟹ P u ⟹ P (λx. c * u x)
  assumes add: ⋀u v. u ∈ borel_measurable M⟹ (⋀x. x ∈ space M ⟹ u x <
top) ⟹ P u ⟹ v ∈ borel_measurable M ⟹ (⋀x. x ∈ space M ⟹ v x < top)
⟹ (⋀x. x ∈ space M ⟹ u x = 0 ∨ v x = 0) ⟹ P v ⟹ P (λx. v x + u x)
  assumes seq: ⋀U. (⋀i. U i ∈ borel_measurable M) ⟹ (⋀i x. x ∈ space M ⟹
U i x < top) ⟹ (⋀i. P (U i)) ⟹ incseq U ⟹ u = (SUP i. U i) ⟹ P (SUP
i. U i)
  shows P u
  using u
proof (induct rule: borel_measurable_implies_simple_function_sequence')
  fix U assume U: ⋀i. simple_function M (U i) incseq U ⋀i x. U i x < top and
sup: ⋀x. (SUP i. U i x) = u x
  have u_eq: u = (SUP i. U i)
    using u by (auto simp add: image_comp sup)

  have not_inf: ⋀x i. x ∈ space M ⟹ U i x < top
    using U by (auto simp: image_iff eq_commute)

  from U have ⋀i. U i ∈ borel_measurable M
    by (simp add: borel_measurable_simple_function)
```

```
  show P u
    unfolding u_eq
  proof (rule seq)
    fix i show P (U i)
      using ‹simple_function M (U i)› not_inf[of _ i]
    proof (induct rule: simple_function_induct_nn)
      case (mult u c)
      show ?case
      proof cases
        assume c = 0 ∨ space M = {} ∨ (∀x∈space M. u x = 0)
        with mult(1) show ?thesis
          by (intro cong[of λx. c * u x indicator {}] set)
             (auto dest!: borel_measurable_simple_function)
      next
        assume ¬ (c = 0 ∨ space M = {} ∨ (∀x∈space M. u x = 0))
        then obtain x where space M ≠ {} and x: x ∈ space M u x ≠ 0 c ≠ 0
          by auto
        with mult(3)[of x] have c < top
          by (auto simp: ennreal_mult_less_top)
        then have u_fin: x' ∈ space M ⟹ u x' < top for x'
          using mult(3)[of x'] ‹c ≠ 0› by (auto simp: ennreal_mult_less_top)
        then have P u
          by (rule mult)
        with u_fin ‹c < top› mult(1) show ?thesis
          by (intro mult') (auto dest!: borel_measurable_simple_function)
      qed
    qed (auto intro: cong intro!: set add dest!: borel_measurable_simple_function)
  qed fact+
qed

lemma simple_function_If_set:
  assumes sf: simple_function M f simple_function M g and A: A ∩ space M ∈
sets M
  shows simple_function M (λx. if x ∈ A then f x else g x) (is simple_function
M ?IF)
proof −
  define F where F x = f −‘ {x} ∩ space M for x
  define G where G x = g −‘ {x} ∩ space M for x
  show ?thesis unfolding simple_function_def
  proof safe
    have ?IF ‘ space M ⊆ f ‘ space M ∪ g ‘ space M by auto
    from finite_subset[OF this] assms
    show finite (?IF ‘ space M) unfolding simple_function_def by auto
  next
    fix x assume x ∈ space M
    then have ∗: ?IF −‘ {?IF x} ∩ space M = (if x ∈ A
      then ((F (f x)) ∩ (A ∩ space M)) ∪ (G (f x) − (G (f x) ∩ (A ∩ space M))))
      else ((F (g x) ∩ (A ∩ space M)) ∪ (G (g x) − (G (g x) ∩ (A ∩ space M)))))
```

using *sets.sets_into_space*[*OF A*] **by** (*auto split*: *if_split_asm simp*: *G_def F_def*)
have [*intro*]: $\bigwedge x.\ F\ x \in sets\ M\ \bigwedge x.\ G\ x \in sets\ M$
unfolding *F_def G_def* **using** *sf*[*THEN simple_functionD*(2)] **by** *auto*
show *?IF* $-`$ {*?IF x*} $\cap$ *space M* $\in$ *sets M* **unfolding** $*$ **using** *A* **by** *auto*
qed
qed

lemma *simple_function_If*:
assumes *sf*: *simple_function M f simple_function M g* **and** *P*: {*x*∈*space M*. *P x*} ∈ *sets M*
shows *simple_function M* (λ*x*. *if P x then f x else g x*)
proof −
have {*x*∈*space M*. *P x*} = {*x*. *P x*} ∩ *space M* **by** *auto*
with *simple_function_If_set*[*OF sf*, *of* {*x*. *P x*}] *P* **show** *?thesis* **by** *simp*
qed

lemma *simple_function_subalgebra*:
assumes *simple_function N f*
and *N_subalgebra*: *sets N* ⊆ *sets M space N* = *space M*
shows *simple_function M f*
using *assms* **unfolding** *simple_function_def* **by** *auto*

lemma *simple_function_comp*:
assumes *T*: *T* ∈ *measurable M M′*
and *f*: *simple_function M′ f*
shows *simple_function M* (λ*x*. *f* (*T x*))
proof (*intro simple_function_def*[*THEN iffD2*] *conjI ballI*)
have (λ*x*. *f* (*T x*)) ‘ *space M* ⊆ *f* ‘ *space M′*
using *T* **unfolding** *measurable_def* **by** *auto*
then show *finite* ((λ*x*. *f* (*T x*)) ‘ *space M*)
using *f* **unfolding** *simple_function_def* **by** (*auto intro*: *finite_subset*)
fix *i* **assume** *i*: *i* ∈ (λ*x*. *f* (*T x*)) ‘ *space M*
then have *i* ∈ *f* ‘ *space M′*
using *T* **unfolding** *measurable_def* **by** *auto*
then have *f* $-`$ {*i*} ∩ *space M′* ∈ *sets M′*
using *f* **unfolding** *simple_function_def* **by** *auto*
then have *T* $-`$ (*f* $-`$ {*i*} ∩ *space M′*) ∩ *space M* ∈ *sets M*
using *T* **unfolding** *measurable_def* **by** *auto*
also have *T* $-`$ (*f* $-`$ {*i*} ∩ *space M′*) ∩ *space M* = (λ*x*. *f* (*T x*)) $-`$ {*i*} ∩ *space M*
using *T* **unfolding** *measurable_def* **by** *auto*
finally show (λ*x*. *f* (*T x*)) $-`$ {*i*} ∩ *space M* ∈ *sets M* **.**
qed

6.5.3 Simple integral

definition *simple_integral* :: *'a measure* ⇒ (*'a* ⇒ *ennreal*) ⇒ *ennreal* (*integral*^{S})
where

integral^S *M f* = (∑ *x* ∈ *f* ‘ *space M*. *x* ∗ *emeasure M* (*f* −‘ {*x*} ∩ *space M*))

syntax
_simple_integral :: *pttrn* ⇒ *ennreal* ⇒ *'a measure* ⇒ *ennreal* (∫^{S} _. _ ∂_ [*60*,*61*] *110*)

translations
∫^{S} *x*. *f* ∂*M* == *CONST simple_integral M* (%*x*. *f*)

lemma *simple_integral_cong*:
assumes ⋀*t*. *t* ∈ *space M* ⟹ *f t* = *g t*
shows *integral*^{S} *M f* = *integral*^{S} *M g*
proof −
have *f* ‘ *space M* = *g* ‘ *space M*
⋀*x*. *f* −‘ {*x*} ∩ *space M* = *g* −‘ {*x*} ∩ *space M*
using *assms* **by** (*auto intro*!: *image_eqI*)
thus *?thesis* **unfolding** *simple_integral_def* **by** *simp*
qed

lemma *simple_integral_const*[*simp*]:
(∫^{S}*x*. *c* ∂*M*) = *c* ∗ (*emeasure M*) (*space M*)
proof (*cases space M* = {})
case *True* **thus** *?thesis* **unfolding** *simple_integral_def* **by** *simp*
next
case *False* **hence** (λ*x*. *c*) ‘ *space M* = {*c*} **by** *auto*
thus *?thesis* **unfolding** *simple_integral_def* **by** *simp*
qed

lemma *simple_function_partition*:
assumes *f*: *simple_function M f* **and** *g*: *simple_function M g*
assumes *sub*: ⋀*x y*. *x* ∈ *space M* ⟹ *y* ∈ *space M* ⟹ *g x* = *g y* ⟹ *f x* = *f y*
assumes *v*: ⋀*x*. *x* ∈ *space M* ⟹ *f x* = *v* (*g x*)
shows *integral*^{S} *M f* = (∑ *y*∈*g* ‘ *space M*. *v y* ∗ *emeasure M* {*x*∈*space M*. *g x* = *y*})
(**is** _ = *?r*)
proof −
from *f g* **have** [*simp*]: *finite* (*f*‘*space M*) *finite* (*g*‘*space M*)
by (*auto simp*: *simple_function_def*)
from *f g* **have** [*measurable*]: *f* ∈ *measurable M* (*count_space UNIV*) *g* ∈ *measurable M* (*count_space UNIV*)
by (*auto intro*: *measurable_simple_function*)

{ fix *y* **assume** *y* ∈ *space M*
then have *f* ‘ *space M* ∩ {*i*. ∃ *x*∈*space M*. *i* = *f x* ∧ *g y* = *g x*} = {*v* (*g y*)}
by (*auto cong*: *sub simp*: *v*[*symmetric*]) **}**
note *eq* = *this*

have *integral*^{S} *M f* =
(∑ *y*∈*f*‘*space M*. *y* ∗ (∑ *z*∈*g*‘*space M*.

if $\exists x{\in}space\ M.\ y = f\ x \wedge z = g\ x$ *then emeasure* M $\{x{\in}space\ M.\ g\ x = z\}$ *else* 0))
unfolding *simple_integral_def*
proof (*safe intro*!: *sum.cong ennreal_mult_left_cong*)
fix y **assume** y: $y \in space\ M\ f\ y \neq 0$
have [*simp*]: g ‘ $space\ M \cap \{z.\ \exists x{\in}space\ M.\ f\ y = f\ x \wedge z = g\ x\} =$
$\{z.\ \exists x{\in}space\ M.\ f\ y = f\ x \wedge z = g\ x\}$
by *auto*
have *eq*:$(\bigcup i{\in}\{z.\ \exists x{\in}space\ M.\ f\ y = f\ x \wedge z = g\ x\}.\ \{x \in space\ M.\ g\ x = i\}) =$
$f\ -‘\ \{f\ y\} \cap space\ M$
by (*auto simp*: *eq_commute cong*: *sub rev_conj_cong*)
have *finite* (g‘$space\ M$) **by** *simp*
then have *finite* $\{z.\ \exists x{\in}space\ M.\ f\ y = f\ x \wedge z = g\ x\}$
by (*rule rev_finite_subset*) *auto*
then show *emeasure* M $(f\ -‘\ \{f\ y\} \cap space\ M) =$
$(\sum z{\in}g\ ‘\ space\ M.\ if\ \exists x{\in}space\ M.\ f\ y = f\ x \wedge z = g\ x\ then\ emeasure\ M\ \{x \in space\ M.\ g\ x = z\}\ else\ 0)$
apply (*simp add*: *sum.If_cases*)
apply (*subst sum_emeasure*)
apply (*auto simp*: *disjoint_family_on_def eq*)
done
qed
also have $\ldots = (\sum y{\in}f‘space\ M.\ (\sum z{\in}g‘space\ M.$
if $\exists x{\in}space\ M.\ y = f\ x \wedge z = g\ x$ *then* $y * emeasure\ M\ \{x{\in}space\ M.\ g\ x = z\}\ else\ 0))$
by (*auto intro*!: *sum.cong simp*: *sum_distrib_left*)
also have $\ldots = ?r$
by (*subst sum.swap*)
(*auto intro*!: *sum.cong simp*: *sum.If_cases scaleR_sum_right*[*symmetric*] *eq*)
finally show $integral^S\ M\ f = ?r$ **.**
qed

lemma *simple_integral_add*[*simp*]:
assumes f: *simple_function* $M\ f$ **and** $\bigwedge x.\ 0 \leq f\ x$ **and** g: *simple_function* $M\ g$ **and** $\bigwedge x.\ 0 \leq g\ x$
shows $(\int^S x.\ f\ x + g\ x\ \partial M) = integral^S\ M\ f + integral^S\ M\ g$
proof −
have $(\int^S x.\ f\ x + g\ x\ \partial M) =$
$(\sum y{\in}(\lambda x.\ (f\ x,\ g\ x))‘space\ M.\ (fst\ y + snd\ y) * emeasure\ M\ \{x{\in}space\ M.\ (f\ x,\ g\ x) = y\})$
by (*intro simple_function_partition*) (*auto intro*: *f g*)
also have $\ldots = (\sum y{\in}(\lambda x.\ (f\ x,\ g\ x))‘space\ M.\ fst\ y * emeasure\ M\ \{x{\in}space\ M.\ (f\ x,\ g\ x) = y\}) +$
$(\sum y{\in}(\lambda x.\ (f\ x,\ g\ x))‘space\ M.\ snd\ y * emeasure\ M\ \{x{\in}space\ M.\ (f\ x,\ g\ x) = y\})$
using *assms*(*2*,*4*) **by** (*auto intro*!: *sum.cong distrib_right simp*: *sum.distrib*[*symmetric*])
also have $(\sum y{\in}(\lambda x.\ (f\ x,\ g\ x))‘space\ M.\ fst\ y * emeasure\ M\ \{x{\in}space\ M.\ (f\ x,\ g\ x) = y\}) = (\int^S x.\ f\ x\ \partial M)$

```
    by (intro simple_function_partition[symmetric]) (auto intro: f g)
  also have (∑ y∈(λx. (f x, g x))‘space M. snd y * emeasure M {x∈space M. (f
x, g x) = y}) = (∫⁺x. g x ∂M)
    by (intro simple_function_partition[symmetric]) (auto intro: f g)
  finally show ?thesis .
qed

lemma simple_integral_sum[simp]:
  assumes ⋀i x. i ∈ P ⟹ 0 ≤ f i x
  assumes ⋀i. i ∈ P ⟹ simple_function M (f i)
  shows (∫⁺x. (∑ i∈P. f i x) ∂M) = (∑ i∈P. integral⁺ M (f i))
proof cases
  assume finite P
  from this assms show ?thesis
    by induct (auto)
qed auto

lemma simple_integral_mult[simp]:
  assumes f: simple_function M f
  shows (∫⁺x. c * f x ∂M) = c * integral⁺ M f
proof −
  have (∫⁺x. c * f x ∂M) = (∑ y∈f ‘ space M. (c * y) * emeasure M {x∈space
M. f x = y})
    using f by (intro simple_function_partition) auto
  also have ... = c * integral⁺ M f
    using f unfolding simple_integral_def
    by (subst sum_distrib_left) (auto simp: mult.assoc Int_def conj_commute)
  finally show ?thesis .
qed

lemma simple_integral_mono_AE:
 assumes f[measurable]: simple_function M f and g[measurable]: simple_function
M g
  and mono: AE x in M. f x ≤ g x
  shows integral⁺ M f ≤ integral⁺ M g
proof −
  let ?μ = λP. emeasure M {x∈space M. P x}
  have integral⁺ M f = (∑ y∈(λx. (f x, g x))‘space M. fst y * ?μ (λx. (f x, g x)
= y))
    using f g by (intro simple_function_partition) auto
  also have ... ≤ (∑ y∈(λx. (f x, g x))‘space M. snd y * ?μ (λx. (f x, g x) = y))
  proof (clarsimp intro!: sum_mono)
    fix x assume x ∈ space M
    let ?M = ?μ (λy. f y = f x ∧ g y = g x)
    show f x * ?M ≤ g x * ?M
    proof cases
      assume ?M ≠ 0
      then have 0 < ?M
        by (simp add: less_le)
```

```
    also have ... ≤ ?μ (λy. f x ≤ g x)
      using mono by (intro emeasure_mono_AE) auto
    finally have ¬ ¬ f x ≤ g x
      by (intro notI) auto
    then show ?thesis
      by (intro mult_right_mono) auto
  qed simp
 qed
 also have ... = integral^S M g
  using f g by (intro simple_function_partition[symmetric]) auto
 finally show ?thesis .
qed

lemma simple_integral_mono:
 assumes simple_function M f and simple_function M g
 and mono: ⋀ x. x ∈ space M ⟹ f x ≤ g x
 shows integral^S M f ≤ integral^S M g
 using assms by (intro simple_integral_mono_AE) auto

lemma simple_integral_cong_AE:
 assumes simple_function M f and simple_function M g
 and AE x in M. f x = g x
 shows integral^S M f = integral^S M g
 using assms by (auto simp: eq_iff intro!: simple_integral_mono_AE)

lemma simple_integral_cong':
 assumes sf: simple_function M f simple_function M g
 and mea: (emeasure M) {x∈space M. f x ≠ g x} = 0
 shows integral^S M f = integral^S M g
proof (intro simple_integral_cong_AE sf AE_I)
 show (emeasure M) {x∈space M. f x ≠ g x} = 0 by fact
 show {x ∈ space M. f x ≠ g x} ∈ sets M
  using sf[THEN borel_measurable_simple_function] by auto
qed simp

lemma simple_integral_indicator:
 assumes A: A ∈ sets M
 assumes f: simple_function M f
 shows (∫^S x. f x * indicator A x ∂M) =
   (∑ x ∈ f ‘ space M. x * emeasure M (f -‘ {x} ∩ space M ∩ A))
proof -
 have eq: (λx. (f x, indicator A x)) ‘ space M ∩ {x. snd x = 1} = (λx. (f x,
1::ennreal))‘A
  using A[THEN sets.sets_into_space] by (auto simp: indicator_def image_iff
split: if_split_asm)
 have eq2: ⋀x. f x ∉ f ‘ A ⟹ f -‘ {f x} ∩ space M ∩ A = {}
  by (auto simp: image_iff)

 have (∫^S x. f x * indicator A x ∂M) =
```

$(\sum y{\in}(\lambda x.\ (f\ x,\ indicator\ A\ x))\ `space\ M.\ (fst\ y * snd\ y) * emeasure\ M\ \{x{\in}space\ M.\ (f\ x,\ indicator\ A\ x) = y\})$
using *assms* **by** (*intro simple_function_partition*) *auto*
also have $\ldots = (\sum y{\in}(\lambda x.\ (f\ x,\ indicator\ A\ x{::}ennreal))\ `space\ M.$
*if snd y = 1 then fst y * emeasure M (f −` {fst y} ∩ space M ∩ A) else 0*)
by (*auto simp*: *indicator_def split*: *if_split_asm intro*!: *arg_cong2*[**where** *f*=(∗)] *arg_cong2*[**where** *f*=*emeasure*] *sum.cong*)
also have $\ldots = (\sum y{\in}(\lambda x.\ (f\ x,\ 1{::}ennreal))\ `A.\ fst\ y * emeasure\ M\ (f -` \{fst\ y\} \cap space\ M \cap A))$
using *assms* **by** (*subst sum.If_cases*) (*auto intro*!: *simple_functionD*(*1*) *simp*: *eq*)
also have $\ldots = (\sum y{\in}fst`(\lambda x.\ (f\ x,\ 1{::}ennreal))\ `A.\ y * emeasure\ M\ (f -` \{y\} \cap space\ M \cap A))$
by (*subst sum.reindex* [*of fst*]) (*auto simp*: *inj_on_def*)
also have $\ldots = (\sum x \in f\ `\ space\ M.\ x * emeasure\ M\ (f -` \{x\} \cap space\ M \cap A))$
using *A*[*THEN sets.sets_into_space*]
by (*intro sum.mono_neutral_cong_left simple_functionD f*) (*auto simp*: *image_comp comp_def eq2*)
finally show *?thesis* **.**
qed

lemma *simple_integral_indicator_only*[*simp*]:
assumes $A \in sets\ M$
shows $integral^S\ M\ (indicator\ A) = emeasure\ M\ A$
using *simple_integral_indicator*[*OF assms*, *of* λ*x. 1*] *sets.sets_into_space*[*OF assms*]
by (*simp_all add*: *image_constant_conv Int_absorb1 split*: *if_split_asm*)

lemma *simple_integral_null_set*:
assumes *simple_function M u* $\bigwedge x.\ 0 \le u\ x$ **and** $N \in null\_sets\ M$
shows $(\int^S x.\ u\ x * indicator\ N\ x\ \partial M) = 0$
proof −
have *AE x in M. indicator N x* = (*0* :: *ennreal*)
using ‹$N \in null\_sets\ M$› **by** (*auto simp*: *indicator_def intro*!: *AE_I*[*of* _ _ *N*])
then have $(\int^S x.\ u\ x * indicator\ N\ x\ \partial M) = (\int^S x.\ 0\ \partial M)$
using *assms* **apply** (*intro simple_integral_cong_AE*) **by** *auto*
then show *?thesis* **by** *simp*
qed

lemma *simple_integral_cong_AE_mult_indicator*:
assumes *sf*: *simple_function M f* **and** *eq*: *AE x in M.* $x \in S$ **and** $S \in sets\ M$
shows $integral^S\ M\ f = (\int^S x.\ f\ x * indicator\ S\ x\ \partial M)$
using *assms* **by** (*intro simple_integral_cong_AE*) *auto*

lemma *simple_integral_cmult_indicator*:
assumes *A*: $A \in sets\ M$
shows $(\int^S x.\ c * indicator\ A\ x\ \partial M) = c * emeasure\ M\ A$

using *simple_integral_mult*[*OF simple_function_indicator*[*OF A*]]
unfolding *simple_integral_indicator_only*[*OF A*] **by** *simp*

lemma *simple_integral_nonneg*:
assumes *f*: *simple_function M f* **and** *ae*: *AE x in M*. *0* ≤ *f x*
shows *0* ≤ *integral*^{S} *M f*
proof −
have *integral*^{S} *M* (λ*x*. *0*) ≤ *integral*^{S} *M f*
using *simple_integral_mono_AE*[*OF _ f ae*] **by** *auto*
then show *?thesis* **by** *simp*
qed

6.5.4 Integral on nonnegative functions

definition *nn_integral* :: *'a measure* ⇒ (*'a* ⇒ *ennreal*) ⇒ *ennreal* (*integral*^{N})
where
integral^N *M f* = (*SUP g* ∈ {*g*. *simple_function M g* ∧ *g* ≤ *f*}. *integral*^{S} *M g*)

syntax
_nn_integral :: *pttrn* ⇒ *ennreal* ⇒ *'a measure* ⇒ *ennreal* (∫$^+$((*2* _./ _)/ ∂_) [*60*,*61*] *110*)

translations
∫$^+$*x*. *f* ∂*M* == *CONST nn_integral M* (λ*x*. *f*)

lemma *nn_integral_def_finite*:
integral^N *M f* = (*SUP g* ∈ {*g*. *simple_function M g* ∧ *g* ≤ *f* ∧ (∀ *x*. *g x* < *top*)}. *integral*^{S} *M g*)
(**is** _ = *Sup* (*?A* ‘ *?f*))
unfolding *nn_integral_def*
proof (*safe intro*!: *antisym SUP_least*)
fix *g* **assume** *g*[*measurable*]: *simple_function M g g* ≤ *f*

show *integral*^{S} *M g* ≤ *Sup* (*?A* ‘ *?f*)
proof *cases*
assume *ae*: *AE x in M*. *g x* ≠ *top*
let *?G* = {*x* ∈ *space M*. *g x* ≠ *top*}
have *integral*^{S} *M g* = *integral*^{S} *M* (λ*x*. *g x* ∗ *indicator ?G x*)
proof (*rule simple_integral_cong_AE*)
show *AE x in M*. *g x* = *g x* ∗ *indicator ?G x*
using *ae AE_space* **by** *eventually_elim auto*
qed (*insert g*, *auto*)
also have ... ≤ *Sup* (*?A* ‘ *?f*)
using *g* **by** (*intro SUP_upper*) (*auto simp*: *le_fun_def less_top split*: *split_indicator*)
finally show *?thesis* .
next
assume *nAE*: ¬ (*AE x in M*. *g x* ≠ *top*)
then have *emeasure M* {*x*∈*space M*. *g x* = *top*} ≠ *0* (**is** *emeasure M ?G* ≠ *0*)

```
    by (subst (asm) AE_iff_measurable[OF _ refl]) auto
  then have top = (SUP n. (∫⁺x. of_nat n * indicator ?G x ∂M))
  by (simp add: ennreal_SUP_of_nat_eq_top ennreal_top_eq_mult_iff SUP_mult_right_ennreal[sy
  also have ... ≤ Sup (?A ‘ ?f)
    using g
    by (safe intro!: SUP_least SUP_upper)
       (auto simp: le_fun_def of_nat_less_top top_unique[symmetric] split:
split_indicator
          intro: order_trans[of _ g x f x for x, OF order_trans[of _ top]])
  finally show ?thesis
    by (simp add: top_unique del: SUP_eq_top_iff Sup_eq_top_iff)
 qed
qed (auto intro: SUP_upper)

lemma nn_integral_mono_AE:
 assumes ae: AE x in M. u x ≤ v x shows integral^N M u ≤ integral^N M v
 unfolding nn_integral_def
proof (safe intro!: SUP_mono)
 fix n assume n: simple_function M n n ≤ u
 from ae[THEN AE_E] obtain N
   where N: {x ∈ space M. ¬ u x ≤ v x} ⊆ N emeasure M N = 0 N ∈ sets M
   by auto
 then have ae_N: AE x in M. x ∉ N by (auto intro: AE_not_in)
 let ?n = λx. n x * indicator (space M - N) x
 have AE x in M. n x ≤ ?n x simple_function M ?n
   using n N ae_N by auto
 moreover
 { fix x have ?n x ≤ v x
   proof cases
     assume x: x ∈ space M - N
     with N have u x ≤ v x by auto
     with n(2)[THEN le_funD, of x] x show ?thesis
       by (auto simp: max_def split: if_split_asm)
   qed simp }
 then have ?n ≤ v by (auto simp: le_funI)
 moreover have integral^S M n ≤ integral^S M ?n
   using ae_N N n by (auto intro!: simple_integral_mono_AE)
 ultimately show ∃m∈{g. simple_function M g ∧ g ≤ v}. integral^S M n ≤
integral^S M m
   by force
qed

lemma nn_integral_mono:
 (⋀x. x ∈ space M ⟹ u x ≤ v x) ⟹ integral^N M u ≤ integral^N M v
 by (auto intro: nn_integral_mono_AE)

lemma mono_nn_integral: mono F ⟹ mono (λx. integral^N M (F x))
 by (auto simp add: mono_def le_fun_def intro!: nn_integral_mono)
```

lemma *nn_integral_cong_AE*:
AE x in M. u x = v x $\Longrightarrow$ $integral^N\ M\ u = integral^N\ M\ v$
by (*auto simp*: *eq_iff intro*!: *nn_integral_mono_AE*)

lemma *nn_integral_cong*:
$(\bigwedge x.\ x \in space\ M \Longrightarrow u\ x = v\ x) \Longrightarrow integral^N\ M\ u = integral^N\ M\ v$
by (*auto intro*: *nn_integral_cong_AE*)

lemma *nn_integral_cong_simp*:
$(\bigwedge x.\ x \in space\ M =simp=> u\ x = v\ x) \Longrightarrow integral^N\ M\ u = integral^N\ M\ v$
by (*auto intro*: *nn_integral_cong simp*: *simp_implies_def*)

lemma *incseq_nn_integral*:
assumes *incseq f* **shows** *incseq* $(\lambda i.\ integral^N\ M\ (f\ i))$
proof −
have $\bigwedge i\ x.\ f\ i\ x \le f\ (Suc\ i)\ x$
using *assms* **by** (*auto dest*!: *incseq_SucD simp*: *le_fun_def*)
then show *?thesis*
by (*auto intro*!: *incseq_SucI nn_integral_mono*)
qed

lemma *nn_integral_eq_simple_integral*:
assumes *f*: *simple_function M f* **shows** $integral^N\ M\ f = integral^S\ M\ f$
proof −
let *?f* $= \lambda x.\ f\ x * indicator\ (space\ M)\ x$
have *f'*: *simple_function M ?f* **using** *f* **by** *auto*
have $integral^N\ M\ ?f \le integral^S\ M\ ?f$ **using** *f'*
by (*force intro*!: *SUP_least simple_integral_mono simp*: *le_fun_def nn_integral_def*)
moreover have $integral^S\ M\ ?f \le integral^N\ M\ ?f$
unfolding *nn_integral_def*
using *f'* **by** (*auto intro*!: *SUP_upper*)
ultimately show *?thesis*
by (*simp cong*: *nn_integral_cong simple_integral_cong*)
qed

Beppo-Levi monotone convergence theorem

lemma *nn_integral_monotone_convergence_SUP*:
assumes *f*: *incseq f* **and** [*measurable*]: $\bigwedge i.\ f\ i \in borel\_measurable\ M$
shows $(\int^+ x.\ (SUP\ i.\ f\ i\ x)\ \partial M) = (SUP\ i.\ integral^N\ M\ (f\ i))$
proof (*rule antisym*)
show $(\int^+ x.\ (SUP\ i.\ f\ i\ x)\ \partial M) \le (SUP\ i.\ (\int^+ x.\ f\ i\ x\ \partial M))$
unfolding *nn_integral_def_finite*[*of* _ $\lambda x.\ SUP\ i.\ f\ i\ x$]
proof (*safe intro*!: *SUP_least*)
fix *u* **assume** *sf_u*[*simp*]: *simple_function M u* **and**
u: $u \le (\lambda x.\ SUP\ i.\ f\ i\ x)$ **and** *u_range*: $\forall x.\ u\ x < top$
note *sf_u*[*THEN borel_measurable_simple_function*, *measurable*]
show $integral^S\ M\ u \le (SUP\ j.\ \int^+ x.\ f\ j\ x\ \partial M)$
proof (*rule ennreal_approx_unit*)
fix *a* :: *ennreal* **assume** *a* < *1*

```
    let ?au = λx. a * u x

    let ?B = λc i. {x∈space M. ?au x = c ∧ c ≤ f i x}
    have integral^S M ?au = (∑ c∈?au`space M. c * (SUP i. emeasure M (?B c i)))
      unfolding simple_integral_def
    proof (intro sum.cong ennreal_mult_left_cong refl)
      fix c assume c ∈ ?au ` space M c ≠ 0
      { fix x' assume x': x' ∈ space M ?au x' = c
        with ‹c ≠ 0› u_range have ?au x' < 1 * u x'
         by (intro ennreal_mult_strict_right_mono ‹a < 1›) (auto simp: less_le)
        also have ... ≤ (SUP i. f i x')
          using u by (auto simp: le_fun_def)
        finally have ∃i. ?au x' ≤ f i x'
          by (auto simp: less_SUP_iff intro: less_imp_le) }
      then have *: ?au -` {c} ∩ space M = (⋃ i. ?B c i)
        by auto
      show emeasure M (?au -` {c} ∩ space M) = (SUP i. emeasure M (?B c i))
        unfolding * using f
        by (intro SUP_emeasure_incseq[symmetric])
           (auto simp: incseq_def le_fun_def intro: order_trans)
    qed
    also have ... = (SUP i. ∑ c∈?au`space M. c * emeasure M (?B c i))
      unfolding SUP_mult_left_ennreal using f
      by (intro ennreal_SUP_sum[symmetric])
         (auto intro!: mult_mono emeasure_mono simp: incseq_def le_fun_def intro: order_trans)
    also have ... ≤ (SUP i. integral^N M (f i))
    proof (intro SUP_subset_mono order_refl)
      fix i
      have (∑ c∈?au`space M. c * emeasure M (?B c i)) =
        (∫^S x. (a * u x) * indicator {x∈space M. a * u x ≤ f i x} x ∂M)
        by (subst simple_integral_indicator)
           (auto intro!: sum.cong ennreal_mult_left_cong arg_cong2[where f=emeasure])
      also have ... = (∫^+ x. (a * u x) * indicator {x∈space M. a * u x ≤ f i x} x ∂M)
        by (rule nn_integral_eq_simple_integral[symmetric]) simp
      also have ... ≤ (∫^+ x. f i x ∂M)
        by (intro nn_integral_mono) (auto split: split_indicator)
      finally show (∑ c∈?au`space M. c * emeasure M (?B c i)) ≤ (∫^+ x. f i x ∂M) .
    qed
    finally show a * integral^S M u ≤ (SUP i. integral^N M (f i))
      by simp
  qed
 qed
qed (auto intro!: SUP_least SUP_upper nn_integral_mono)
```

lemma *sup_continuous_nn_integral*[*order_continuous_intros*]:
assumes *f*: ⋀*y*. *sup_continuous* (*f y*)
assumes [*measurable*]: ⋀*x*. (λ*y*. *f y x*) ∈ *borel_measurable M*
shows *sup_continuous* (λ*x*. (∫⁺*y*. *f y x* ∂*M*))
unfolding *sup_continuous_def*
proof *safe*
fix *C* :: *nat* ⇒ *'b* **assume** *C*: *incseq C*
with *sup_continuous_mono*[*OF f*] **show** (∫⁺ *y*. *f y* (*Sup* (*C* ‘ *UNIV*)) ∂*M*) = (*SUP i*. ∫⁺ *y*. *f y* (*C i*) ∂*M*)
unfolding *sup_continuousD*[*OF f C*]
by (*subst nn_integral_monotone_convergence_SUP*) (*auto simp*: *mono_def le_fun_def*)
qed

theorem *nn_integral_monotone_convergence_SUP_AE*:
assumes *f*: ⋀*i*. *AE x in M*. *f i x* ≤ *f* (*Suc i*) *x* ⋀*i*. *f i* ∈ *borel_measurable M*
shows (∫⁺ *x*. (*SUP i*. *f i x*) ∂*M*) = (*SUP i*. *integral*^{N} *M* (*f i*))
proof −
from *f* **have** *AE x in M*. ∀*i*. *f i x* ≤ *f* (*Suc i*) *x*
by (*simp add*: *AE_all_countable*)
from *this*[*THEN AE_E*] **obtain** *N*
where *N*: {*x* ∈ *space M*. ¬ (∀*i*. *f i x* ≤ *f* (*Suc i*) *x*)} ⊆ *N emeasure M N* = *0* *N* ∈ *sets M*
by *auto*
let *?f* = λ*i x*. *if x* ∈ *space M* − *N then f i x else 0*
have *f_eq*: *AE x in M*. ∀*i*. *?f i x* = *f i x* **using** *N* **by** (*auto intro*!: *AE_I*[*of _ _ N*])
then have (∫⁺ *x*. (*SUP i*. *f i x*) ∂*M*) = (∫⁺ *x*. (*SUP i*. *?f i x*) ∂*M*)
by (*auto intro*!: *nn_integral_cong_AE*)
also have ... = (*SUP i*. (∫⁺ *x*. *?f i x* ∂*M*))
proof (*rule nn_integral_monotone_convergence_SUP*)
show *incseq ?f* **using** *N*(*1*) **by** (*force intro*!: *incseq_SucI le_funI*)
{ **fix** *i* **show** (λ*x*. *if x* ∈ *space M* − *N then f i x else 0*) ∈ *borel_measurable M*
using *f N*(*3*) **by** (*intro measurable_If_set*) *auto* }
qed
also have ... = (*SUP i*. (∫⁺ *x*. *f i x* ∂*M*))
using *f_eq* **by** (*force intro*!: *arg_cong*[**where** *f* = λ*f*. *Sup* (*range f*)] *nn_integral_cong_AE ext*)
finally show *?thesis* .
qed

lemma *nn_integral_monotone_convergence_simple*:
incseq f ⟹ (⋀*i*. *simple_function M* (*f i*)) ⟹ (*SUP i*. ∫^{S} *x*. *f i x* ∂*M*) = (∫⁺*x*. (*SUP i*. *f i x*) ∂*M*)
using *nn_integral_monotone_convergence_SUP*[*of f M*]
by (*simp add*: *nn_integral_eq_simple_integral*[*symmetric*] *borel_measurable_simple_function*)

lemma *SUP_simple_integral_sequences*:

assumes f: *incseq f* $\bigwedge i$. *simple_function M* (*f i*)
and g: *incseq g* $\bigwedge i$. *simple_function M* (*g i*)
and *eq*: *AE x in M*. (*SUP i. f i x*) = (*SUP i. g i x*)
shows (*SUP i. integral*^{S} *M* (*f i*)) = (*SUP i. integral*^{S} *M* (*g i*))
(**is** *Sup* (*?F* ‘ _) = *Sup* (*?G* ‘ _))
proof −
have (*SUP i. integral*^{S} *M* (*f i*)) = ($\int^+ x$. (*SUP i. f i x*) *∂M*)
using *f* **by** (*rule nn_integral_monotone_convergence_simple*)
also have ... = ($\int^+ x$. (*SUP i. g i x*) *∂M*)
unfolding *eq*[*THEN nn_integral_cong_AE*] **..**
also have ... = (*SUP i. ?G i*)
using *g* **by** (*rule nn_integral_monotone_convergence_simple*[*symmetric*])
finally show *?thesis* **by** *simp*
qed

lemma *nn_integral_const*[*simp*]: ($\int^+ x$. *c ∂M*) = *c* ∗ *emeasure M* (*space M*)
by (*subst nn_integral_eq_simple_integral*) *auto*

lemma *nn_integral_linear*:
assumes *f*: *f* ∈ *borel_measurable M* **and** *g*: *g* ∈ *borel_measurable M*
shows ($\int^+ x$. *a* ∗ *f x* + *g x ∂M*) = *a* ∗ *integral*^{N} *M f* + *integral*^{N} *M g*
(**is** *integral*^{N} *M ?L* = _)
proof −
obtain *u*
where $\bigwedge i$. *simple_function M* (*u i*) *incseq u* $\bigwedge i\ x$. *u i x* < *top* $\bigwedge x$. (*SUP i. u i x*) = *f x*
using *borel_measurable_implies_simple_function_sequence′ f*(*1*)
by *auto*
note *u* = *nn_integral_monotone_convergence_simple*[*OF this*(*2*,*1*)] *this*

obtain *v* **where**
$\bigwedge i$. *simple_function M* (*v i*) *incseq v* $\bigwedge i\ x$. *v i x* < *top* $\bigwedge x$. (*SUP i. v i x*) = *g x*
using *borel_measurable_implies_simple_function_sequence′ g*(*1*)
by *auto*
note *v* = *nn_integral_monotone_convergence_simple*[*OF this*(*2*,*1*)] *this*

let *?L′* = $\lambda i\ x$. *a* ∗ *u i x* + *v i x*

have *?L* ∈ *borel_measurable M* **using** *assms* **by** *auto*
from *borel_measurable_implies_simple_function_sequence′*[*OF this*]
obtain *l* **where** $\bigwedge i$. *simple_function M* (*l i*) *incseq l* $\bigwedge i\ x$. *l i x* < *top* $\bigwedge x$. (*SUP i. l i x*) = *a* ∗ *f x* + *g x*
by *auto*
note *l* = *nn_integral_monotone_convergence_simple*[*OF this*(*2*,*1*)] *this*

have *inc*: *incseq* (λi. *a* ∗ *integral*^{S} *M* (*u i*)) *incseq* (λi. *integral*^{S} *M* (*v i*))
using *u v* **by** (*auto simp*: *incseq_Suc_iff le_fun_def intro*!: *add_mono mult_left_mono simple_integral_mono*)

have *l'*: (*SUP i. integral*^{S} *M* (*l i*)) = (*SUP i. integral*^{S} *M* (*?L' i*))
proof (*rule SUP_simple_integral_sequences*[*OF l*(*3*,*2*)])
show *incseq ?L'* ⋀*i. simple_function M* (*?L' i*)
using *u v* **unfolding** *incseq_Suc_iff le_fun_def*
by (*auto intro*!: *add_mono mult_left_mono*)
{ fix *x*
have (*SUP i. a* ∗ *u i x* + *v i x*) = *a* ∗ (*SUP i. u i x*) + (*SUP i. v i x*)
using *u*(*3*) *v*(*3*) *u*(*4*)[*of* _ *x*] *v*(*4*)[*of* _ *x*] **unfolding** *SUP_mult_left_ennreal*
by (*auto intro*!: *ennreal_SUP_add simp*: *incseq_Suc_iff le_fun_def add_mono mult_left_mono*) **}**
then show *AE x in M*. (*SUP i. l i x*) = (*SUP i. ?L' i x*)
unfolding *l*(*5*) **using** *u*(*5*) *v*(*5*) **by** (*intro AE_I2*) *auto*
qed
also have ... = (*SUP i. a* ∗ *integral*^{S} *M* (*u i*) + *integral*^{S} *M* (*v i*))
using *u*(*2*) *v*(*2*) **by** *auto*
finally show *?thesis*
unfolding *l*(*5*)[*symmetric*] *l*(*1*)[*symmetric*]
by (*simp add*: *ennreal_SUP_add*[*OF inc*] *v u SUP_mult_left_ennreal*[*symmetric*])
qed

lemma *nn_integral_cmult*: *f* ∈ *borel_measurable M* ⟹ (∫$^+$ *x*. *c* ∗ *f x* ∂*M*) = *c* ∗ *integral*^{N} *M f*
using *nn_integral_linear*[*of f M* λ*x*. *0 c*] **by** *simp*

lemma *nn_integral_multc*: *f* ∈ *borel_measurable M* ⟹ (∫$^+$ *x*. *f x* ∗ *c* ∂*M*) = *integral*^{N} *M f* ∗ *c*
unfolding *mult.commute*[*of* _ *c*] *nn_integral_cmult* **by** *simp*

lemma *nn_integral_divide*: *f* ∈ *borel_measurable M* ⟹ (∫$^+$ *x*. *f x* / *c* ∂*M*) = (∫$^+$ *x*. *f x* ∂*M*) / *c*
unfolding *divide_ennreal_def* **by** (*rule nn_integral_multc*)

lemma *nn_integral_indicator*[*simp*]: *A* ∈ *sets M* ⟹ (∫$^+$ *x*. *indicator A x*∂*M*) = (*emeasure M*) *A*
by (*subst nn_integral_eq_simple_integral*) (*auto simp*: *simple_integral_indicator*)

lemma *nn_integral_cmult_indicator*: *A* ∈ *sets M* ⟹ (∫$^+$ *x*. *c* ∗ *indicator A x* ∂*M*) = *c* ∗ *emeasure M A*
by (*subst nn_integral_eq_simple_integral*) (*auto*)

lemma *nn_integral_indicator'*:
assumes [*measurable*]: *A* ∩ *space M* ∈ *sets M*
shows (∫$^+$ *x*. *indicator A x* ∂*M*) = *emeasure M* (*A* ∩ *space M*)
proof −
have (∫$^+$ *x*. *indicator A x* ∂*M*) = (∫$^+$ *x*. *indicator* (*A* ∩ *space M*) *x* ∂*M*)
by (*intro nn_integral_cong*) (*simp split*: *split_indicator*)
also have ... = *emeasure M* (*A* ∩ *space M*)
by *simp*

finally show *?thesis* **.**
qed

lemma *nn_integral_indicator_singleton*[*simp*]:
assumes [*measurable*]: $\{y\} \in$ *sets M* **shows** $(\int^+ x.\ f\ x * \mathit{indicator}\ \{y\}\ x\ \partial M) = f\ y * \mathit{emeasure}\ M\ \{y\}$
proof −
have $(\int^+ x.\ f\ x * \mathit{indicator}\ \{y\}\ x\ \partial M) = (\int^+ x.\ f\ y * \mathit{indicator}\ \{y\}\ x\ \partial M)$
by (*auto intro*!: *nn_integral_cong split*: *split_indicator*)
then show *?thesis*
by (*simp add*: *nn_integral_cmult*)
qed

lemma *nn_integral_set_ennreal*:
$(\int^+ x.\ \mathit{ennreal}\ (f\ x) * \mathit{indicator}\ A\ x\ \partial M) = (\int^+ x.\ \mathit{ennreal}\ (f\ x * \mathit{indicator}\ A\ x)\ \partial M)$
by (*rule nn_integral_cong*) (*simp split*: *split_indicator*)

lemma *nn_integral_indicator_singleton′*[*simp*]:
assumes [*measurable*]: $\{y\} \in$ *sets M*
shows $(\int^+ x.\ \mathit{ennreal}\ (f\ x * \mathit{indicator}\ \{y\}\ x)\ \partial M) = f\ y * \mathit{emeasure}\ M\ \{y\}$
by (*subst nn_integral_set_ennreal*[*symmetric*]) (*simp*)

lemma *nn_integral_add*:
$f \in \mathit{borel\_measurable}\ M \Longrightarrow g \in \mathit{borel\_measurable}\ M \Longrightarrow (\int^+ x.\ f\ x + g\ x\ \partial M) = \mathit{integral}^N\ M\ f + \mathit{integral}^N\ M\ g$
using *nn_integral_linear*[*of f M g 1*] **by** *simp*

lemma *nn_integral_sum*:
$(\bigwedge i.\ i \in P \Longrightarrow f\ i \in \mathit{borel\_measurable}\ M) \Longrightarrow (\int^+ x.\ (\sum i{\in}P.\ f\ i\ x)\ \partial M) = (\sum i{\in}P.\ \mathit{integral}^N\ M\ (f\ i))$
by (*induction P rule*: *infinite_finite_induct*) (*auto simp*: *nn_integral_add*)

theorem *nn_integral_suminf*:
assumes *f*: $\bigwedge i.\ f\ i \in \mathit{borel\_measurable}\ M$
shows $(\int^+ x.\ (\sum i.\ f\ i\ x)\ \partial M) = (\sum i.\ \mathit{integral}^N\ M\ (f\ i))$
proof −
have *all_pos*: *AE x in M*. $\forall i.\ 0 \le f\ i\ x$
using *assms* **by** (*auto simp*: *AE_all_countable*)
have $(\sum i.\ \mathit{integral}^N\ M\ (f\ i)) = (\mathit{SUP}\ n.\ \sum i{<}n.\ \mathit{integral}^N\ M\ (f\ i))$
by (*rule suminf_eq_SUP*)
also have $\ldots = (\mathit{SUP}\ n.\ \int^+ x.\ (\sum i{<}n.\ f\ i\ x)\ \partial M)$
unfolding *nn_integral_sum*[*OF f*] **..**
also have $\ldots = \int^+ x.\ (\mathit{SUP}\ n.\ \sum i{<}n.\ f\ i\ x)\ \partial M$ **using** *f all_pos*
by (*intro nn_integral_monotone_convergence_SUP_AE*[*symmetric*])
(*elim AE_mp*, *auto simp*: *sum_nonneg simp del*: *sum.lessThan_Suc intro*!: *AE_I2 sum_mono2*)
also have $\ldots = \int^+ x.\ (\sum i.\ f\ i\ x)\ \partial M$ **using** *all_pos*
by (*intro nn_integral_cong_AE*) (*auto simp*: *suminf_eq_SUP*)

finally show *?thesis* **by** *simp*
qed

lemma *nn_integral_bound_simple_function*:
assumes *bnd*: $\bigwedge x.\ x \in space\ M \Longrightarrow f\ x < \infty$
assumes *f*[*measurable*]: *simple_function M f*
assumes *supp*: *emeasure M* $\{x \in space\ M.\ f\ x \neq 0\} < \infty$
shows *nn_integral M f* $< \infty$
proof *cases*
assume *space M* $= \{\}$
then have *nn_integral M f* $= (\int^+ x.\ 0\ \partial M)$
by (*intro nn_integral_cong*) *auto*
then show *?thesis* **by** *simp*
next
assume *space M* $\neq \{\}$
with *simple_functionD*(1)[*OF f*] *bnd* **have** *bnd*: $0 \leq Max\ (f`space\ M) \wedge Max\ (f`space\ M) < \infty$
by (*subst Max_less_iff*) (*auto simp*: *Max_ge_iff*)

have *nn_integral M f* $\leq (\int^+ x.\ Max\ (f`space\ M) * indicator\ \{x \in space\ M.\ f\ x \neq 0\}\ x\ \partial M)$
proof (*rule nn_integral_mono*)
fix *x* **assume** *x* $\in$ *space M*
with *f* **show** *f x* $\leq$ *Max* (*f* ` *space M*) $*$ *indicator* $\{x \in space\ M.\ f\ x \neq 0\}\ x$
by (*auto split*: *split_indicator intro*!: *Max_ge simple_functionD*)
qed
also have $\ldots < \infty$
using *bnd supp* **by** (*subst nn_integral_cmult*) (*auto simp*: *ennreal_mult_less_top*)
finally show *?thesis* **.**
qed

theorem *nn_integral_Markov_inequality*:
assumes *u*: $(\lambda x.\ u\ x * indicator\ A\ x) \in borel\_measurable\ M$ **and** $A \in sets\ M$
shows (*emeasure M*) $(\{x \in A.\ 1 \leq c * u\ x\}) \leq c * (\int^+ x.\ u\ x * indicator\ A\ x\ \partial M)$
(**is** (*emeasure M*) *?A* $\leq$ _ $*$ *?PI*)
proof −
define u' **where** $u' = (\lambda x.\ u\ x * indicator\ A\ x)$
have [*measurable*]: $u' \in borel\_measurable\ M$
using *u* **unfolding** u'_def **.**
have $\{x \in space\ M.\ c * u'\ x \geq 1\} \in sets\ M$
by *measurable*
also have $\{x \in space\ M.\ c * u'\ x \geq 1\} = ?A$
using *sets.sets_into_space*[*OF* ‹*A* $\in$ *sets M*›] **by** (*auto simp*: u'_def *indicator_def*)
finally have (*emeasure M*) *?A* $= (\int^+ x.\ indicator\ ?A\ x\ \partial M)$
using *nn_integral_indicator* **by** *simp*
also have $\ldots \leq (\int^+ x.\ c * (u\ x * indicator\ A\ x)\ \partial M)$
using *u* **by** (*auto intro*!: *nn_integral_mono_AE simp*: *indicator_def*)

```
  also have ... = c * (∫⁺ x. u x * indicator A x ∂M)
    using assms by (auto intro!: nn_integral_cmult)
  finally show ?thesis .
qed

lemma Chernoff_ineq_nn_integral_ge:
  assumes s: s > 0 and [measurable]: A ∈ sets M
  assumes [measurable]: (λx. f x * indicator A x) ∈ borel_measurable M
  shows   emeasure M {x∈A. f x ≥ a} ≤
          ennreal (exp (-s * a)) * nn_integral M (λx. ennreal (exp (s * f x)) *
indicator A x)
proof -
  define f' where f' = (λx. f x * indicator A x)
  have [measurable]: f' ∈ borel_measurable M
    using assms(3) unfolding f'_def by assumption
  have (λx. ennreal (exp (s * f' x)) * indicator A x) ∈ borel_measurable M
    by simp
  also have (λx. ennreal (exp (s * f' x)) * indicator A x) =
            (λx. ennreal (exp (s * f x)) * indicator A x)
    by (auto simp: f'_def indicator_def fun_eq_iff)
  finally have meas: ... ∈ borel_measurable M .

  have {x∈A. f x ≥ a} = {x∈A. ennreal (exp (-s * a)) * ennreal (exp (s * f x))
≥ 1}
    using s by (auto simp: exp_minus field_simps simp flip: ennreal_mult)
  also have emeasure M ... ≤ ennreal (exp (-s * a)) *
              (∫⁺x. ennreal (exp (s * f x)) * indicator A x ∂M)
    by (intro order.trans[OF nn_integral_Markov_inequality] meas) auto
  finally show ?thesis .
qed

lemma Chernoff_ineq_nn_integral_le:
  assumes s: s > 0 and [measurable]: A ∈ sets M
  assumes [measurable]: f ∈ borel_measurable M
  shows   emeasure M {x∈A. f x ≤ a} ≤
          ennreal (exp (s * a)) * nn_integral M (λx. ennreal (exp (-s * f x)) *
indicator A x)
  using Chernoff_ineq_nn_integral_ge[of s A M λx. -f x -a] assms by simp

lemma nn_integral_noteq_infinite:
  assumes g: g ∈ borel_measurable M and integral^N M g ≠ ∞
  shows AE x in M. g x ≠ ∞
proof (rule ccontr)
  assume c: ¬ (AE x in M. g x ≠ ∞)
  have (emeasure M) {x∈space M. g x = ∞} ≠ 0
    using c g by (auto simp add: AE_iff_null)
  then have 0 < (emeasure M) {x∈space M. g x = ∞}
    by (auto simp: zero_less_iff_neq_zero)
  then have ∞ = ∞ * (emeasure M) {x∈space M. g x = ∞}
```

```
    by (auto simp: ennreal_top_eq_mult_iff)
  also have ... ≤ (∫⁺x. ∞ * indicator {x∈space M. g x = ∞} x ∂M)
    using g by (subst nn_integral_cmult_indicator) auto
  also have ... ≤ integral^N M g
    using assms by (auto intro!: nn_integral_mono_AE simp: indicator_def)
  finally show False
    using ‹integral^N M g ≠ ∞› by (auto simp: top_unique)
qed

lemma nn_integral_PInf:
  assumes f: f ∈ borel_measurable M and not_Inf: integral^N M f ≠ ∞
  shows emeasure M (f -` {∞} ∩ space M) = 0
proof -
  have ∞ * emeasure M (f -` {∞} ∩ space M) = (∫⁺ x. ∞ * indicator (f -`
{∞} ∩ space M) x ∂M)
    using f by (subst nn_integral_cmult_indicator) (auto simp: measurable_sets)
  also have ... ≤ integral^N M f
    by (auto intro!: nn_integral_mono simp: indicator_def)
  finally have ∞ * (emeasure M) (f -` {∞} ∩ space M) ≤ integral^N M f
    by simp
  then show ?thesis
    using assms by (auto simp: ennreal_top_mult top_unique split: if_split_asm)
qed

lemma simple_integral_PInf:
  simple_function M f ⟹ integral^S M f ≠ ∞ ⟹ emeasure M (f -` {∞} ∩ space
M) = 0
 by (rule nn_integral_PInf) (auto simp: nn_integral_eq_simple_integral borel_measurable_simple_function)

lemma nn_integral_PInf_AE:
  assumes f ∈ borel_measurable M integral^N M f ≠ ∞ shows AE x in M. f x ≠
∞
proof (rule AE_I)
  show (emeasure M) (f -` {∞} ∩ space M) = 0
    by (rule nn_integral_PInf[OF assms])
  show f -` {∞} ∩ space M ∈ sets M
    using assms by (auto intro: borel_measurable_vimage)
qed auto

lemma nn_integral_diff:
  assumes f: f ∈ borel_measurable M
  and g: g ∈ borel_measurable M
  and fin: integral^N M g ≠ ∞
  and mono: AE x in M. g x ≤ f x
  shows (∫⁺ x. f x - g x ∂M) = integral^N M f - integral^N M g
proof -
  have diff: (λx. f x - g x) ∈ borel_measurable M
    using assms by auto
  have AE x in M. f x = f x - g x + g x
```

using *diff_add_cancel_ennreal mono nn_integral_noteq_infinite*[*OF g fin*] *assms* **by** *auto*
then have $**$: $integral^N\ M\ f = (\int^+x.\ f\ x - g\ x\ \partial M) + integral^N\ M\ g$
unfolding *nn_integral_add*[*OF diff g, symmetric*]
by (*rule nn_integral_cong_AE*)
show *?thesis* **unfolding** $**$
using *fin*
by (*cases rule*: *ennreal2_cases*[*of* $\int^+ x.\ f\ x - g\ x\ \partial M\ integral^N\ M\ g$]) *auto*
qed

lemma *nn_integral_mult_bounded_inf*:
assumes *f*: $f \in borel\_measurable\ M$ $(\int^+x.\ f\ x\ \partial M) < \infty$ **and** *c*: $c \neq \infty$ **and** *ae*: $AE\ x\ in\ M.\ g\ x \leq c * f\ x$
shows $(\int^+x.\ g\ x\ \partial M) < \infty$
proof −
have $(\int^+x.\ g\ x\ \partial M) \leq (\int^+x.\ c * f\ x\ \partial M)$
by (*intro nn_integral_mono_AE ae*)
also have $(\int^+x.\ c * f\ x\ \partial M) < \infty$
using *c f* **by** (*subst nn_integral_cmult*) (*auto simp*: *ennreal_mult_less_top top_unique not_less*)
finally show *?thesis* **.**
qed

Fatou's lemma: convergence theorem on limes inferior

lemma *nn_integral_monotone_convergence_INF_AE'*:
assumes *f*: $\bigwedge i.\ AE\ x\ in\ M.\ f\ (Suc\ i)\ x \leq f\ i\ x$ **and** [*measurable*]: $\bigwedge i.\ f\ i \in borel\_measurable\ M$
and $*$: $(\int^+ x.\ f\ 0\ x\ \partial M) < \infty$
shows $(\int^+ x.\ (INF\ i.\ f\ i\ x)\ \partial M) = (INF\ i.\ integral^N\ M\ (f\ i))$
proof (*rule ennreal_minus_cancel*)
have $integral^N\ M\ (f\ 0) - (\int^+ x.\ (INF\ i.\ f\ i\ x)\ \partial M) = (\int^+x.\ f\ 0\ x - (INF\ i.\ f\ i\ x)\ \partial M)$
proof (*rule nn_integral_diff*[*symmetric*])
have $(\int^+ x.\ (INF\ i.\ f\ i\ x)\ \partial M) \leq (\int^+ x.\ f\ 0\ x\ \partial M)$
by (*intro nn_integral_mono INF_lower*) *simp*
with $*$ **show** $(\int^+ x.\ (INF\ i.\ f\ i\ x)\ \partial M) \neq \infty$
by *simp*
qed (*auto intro*: *INF_lower*)
also have $\ldots = (\int^+x.\ (SUP\ i.\ f\ 0\ x - f\ i\ x)\ \partial M)$
by (*simp add*: *ennreal_INF_const_minus*)
also have $\ldots = (SUP\ i.\ (\int^+x.\ f\ 0\ x - f\ i\ x\ \partial M))$
proof (*intro nn_integral_monotone_convergence_SUP_AE*)
show $AE\ x\ in\ M.\ f\ 0\ x - f\ i\ x \leq f\ 0\ x - f\ (Suc\ i)\ x$ **for** *i*
using *f*[*of i*] **by** *eventually_elim* (*auto simp*: *ennreal_mono_minus*)
qed *simp*
also have $\ldots = (SUP\ i.\ nn\_integral\ M\ (f\ 0) - (\int^+x.\ f\ i\ x\ \partial M))$
proof (*subst nn_integral_diff*[*symmetric*])
fix *i*
have *dec*: $AE\ x\ in\ M.\ \forall i.\ f\ (Suc\ i)\ x \leq f\ i\ x$

unfolding *AE_all_countable* **using** *f* **by** *auto*
then show *AE x in M. f i x ≤ f 0 x*
using *dec* **by** *eventually_elim* (*auto intro*: *lift_Suc_antimono_le*[*of* λ*i. f i x 0 i* **for** *x*])
then have (∫⁺ *x. f i x ∂M*) ≤ (∫⁺ *x. f 0 x ∂M*)
by (*rule nn_integral_mono_AE*)
with ∗ **show** (∫⁺ *x. f i x ∂M*) ≠ ∞
by *simp*
qed (*insert f*, *auto simp*: *decseq_def le_fun_def*)
finally show *integral*^{N} *M* (*f 0*) − (∫⁺ *x.* (*INF i. f i x*) *∂M*) =
integral^N *M* (*f 0*) − (*INF i.* ∫⁺ *x. f i x ∂M*)
by (*simp add*: *ennreal_INF_const_minus*)
qed (*insert* ∗, *auto intro*!: *nn_integral_mono intro*: *INF_lower*)

theorem *nn_integral_monotone_convergence_INF_AE*:
fixes *f* :: *nat* ⇒ *'a* ⇒ *ennreal*
assumes *f*: ⋀*i. AE x in M. f* (*Suc i*) *x* ≤ *f i x*
and [*measurable*]: ⋀*i. f i* ∈ *borel_measurable M*
and *fin*: (∫⁺ *x. f i x ∂M*) < ∞
shows (∫⁺ *x.* (*INF i. f i x*) *∂M*) = (*INF i. integral*^{N} *M* (*f i*))
proof −
{ fix *f* :: *nat* ⇒ *ennreal* **and** *j* **assume** *decseq f*
then have (*INF i. f i*) = (*INF i. f* (*i* + *j*))
apply (*intro INF_eq*)
apply (*rule_tac x*=*i* **in** *bexI*)
apply (*auto simp*: *decseq_def le_fun_def*)
done }
note *INF_shift* = *this*
have *mono*: *AE x in M.* ∀ *i. f* (*Suc i*) *x* ≤ *f i x*
using *f* **by** (*auto simp*: *AE_all_countable*)
then have *AE x in M.* (*INF i. f i x*) = (*INF n. f* (*n* + *i*) *x*)
by *eventually_elim* (*auto intro*!: *decseq_SucI INF_shift*)
then have (∫⁺ *x.* (*INF i. f i x*) *∂M*) = (∫⁺ *x.* (*INF n. f* (*n* + *i*) *x*) *∂M*)
by (*rule nn_integral_cong_AE*)
also have ... = (*INF n.* (∫⁺ *x. f* (*n* + *i*) *x ∂M*))
by (*rule nn_integral_monotone_convergence_INF_AE'*) (*insert assms*, *auto*)
also have ... = (*INF n.* (∫⁺ *x. f n x ∂M*))
by (*intro INF_shift*[*symmetric*] *decseq_SucI nn_integral_mono_AE f*)
finally show *?thesis* **.**
qed

lemma *nn_integral_monotone_convergence_INF_decseq*:
assumes *f*: *decseq f* **and** ∗: ⋀*i. f i* ∈ *borel_measurable M* (∫⁺ *x. f i x ∂M*) < ∞
shows (∫⁺ *x.* (*INF i. f i x*) *∂M*) = (*INF i. integral*^{N} *M* (*f i*))
using *nn_integral_monotone_convergence_INF_AE*[*of f M i*, *OF* _ ∗] *f* **by** (*simp add*: *decseq_SucD le_funD*)

theorem *nn_integral_liminf*:

fixes $u :: nat \Rightarrow 'a \Rightarrow ennreal$
assumes u: $\bigwedge i.\ u\ i \in borel\_measurable\ M$
shows $(\int^+ x.\ liminf\ (\lambda n.\ u\ n\ x)\ \partial M) \leq liminf\ (\lambda n.\ integral^N\ M\ (u\ n))$
proof −
have $(\int^+ x.\ liminf\ (\lambda n.\ u\ n\ x)\ \partial M) = (SUP\ n.\ \int^+ x.\ (INF\ i{\in}\{n..\}.\ u\ i\ x)\ \partial M)$
unfolding *liminf_SUP_INF* **using** u
by (*intro nn_integral_monotone_convergence_SUP_AE*)
(*auto intro*!: *AE_I2 intro*: *INF_greatest INF_superset_mono*)
also have $\ldots \leq liminf\ (\lambda n.\ integral^N\ M\ (u\ n))$
by (*auto simp*: *liminf_SUP_INF intro*!: *SUP_mono INF_greatest nn_integral_mono INF_lower*)
finally show *?thesis* **.**
qed

theorem *nn_integral_limsup*:
fixes $u :: nat \Rightarrow 'a \Rightarrow ennreal$
assumes [*measurable*]: $\bigwedge i.\ u\ i \in borel\_measurable\ M\ w \in borel\_measurable\ M$
assumes *bounds*: $\bigwedge i.\ AE\ x\ in\ M.\ u\ i\ x \leq w\ x$ **and** w: $(\int^+ x.\ w\ x\ \partial M) < \infty$
shows $limsup\ (\lambda n.\ integral^N\ M\ (u\ n)) \leq (\int^+ x.\ limsup\ (\lambda n.\ u\ n\ x)\ \partial M)$
proof −
have *bnd*: $AE\ x\ in\ M.\ \forall i.\ u\ i\ x \leq w\ x$
using *bounds* **by** (*auto simp*: *AE_all_countable*)
then have $(\int^+ x.\ (SUP\ n.\ u\ n\ x)\ \partial M) \leq (\int^+ x.\ w\ x\ \partial M)$
by (*auto intro*!: *nn_integral_mono_AE elim*: *eventually_mono intro*: *SUP_least*)
then have $(\int^+ x.\ limsup\ (\lambda n.\ u\ n\ x)\ \partial M) = (INF\ n.\ \int^+ x.\ (SUP\ i{\in}\{n..\}.\ u\ i\ x)\ \partial M)$
unfolding *limsup_INF_SUP* **using** *bnd w*
by (*intro nn_integral_monotone_convergence_INF_AE'*)
(*auto intro*!: *AE_I2 intro*: *SUP_least SUP_subset_mono*)
also have $\ldots \geq limsup\ (\lambda n.\ integral^N\ M\ (u\ n))$
by (*auto simp*: *limsup_INF_SUP intro*!: *INF_mono SUP_least exI nn_integral_mono SUP_upper*)
finally (*xtrans*) **show** *?thesis* **.**
qed

lemma *nn_integral_LIMSEQ*:
assumes f: $incseq\ f\ \bigwedge i.\ f\ i \in borel\_measurable\ M$
and u: $\bigwedge x.\ (\lambda i.\ f\ i\ x) \longrightarrow u\ x$
shows $(\lambda n.\ integral^N\ M\ (f\ n)) \longrightarrow integral^N\ M\ u$
proof −
have $(\lambda n.\ integral^N\ M\ (f\ n)) \longrightarrow (SUP\ n.\ integral^N\ M\ (f\ n))$
using f **by** (*intro LIMSEQ_SUP*[*of* $\lambda n.\ integral^N\ M\ (f\ n)$] *incseq_nn_integral*)
also have $(SUP\ n.\ integral^N\ M\ (f\ n)) = integral^N\ M\ (\lambda x.\ SUP\ n.\ f\ n\ x)$
using f **by** (*intro nn_integral_monotone_convergence_SUP*[*symmetric*])
also have $integral^N\ M\ (\lambda x.\ SUP\ n.\ f\ n\ x) = integral^N\ M\ (\lambda x.\ u\ x)$
using f **by** (*subst LIMSEQ_SUP*[*THEN LIMSEQ_unique*, *OF* _ u]) (*auto simp*: *incseq_def le_fun_def*)
finally show *?thesis* **.**
qed

theorem *nn_integral_dominated_convergence*:
assumes [*measurable*]:
⋀*i*. *u i* ∈ *borel_measurable M u′* ∈ *borel_measurable M w* ∈ *borel_measurable M*
and *bound*: ⋀*j*. *AE x in M*. *u j x* ≤ *w x*
and *w*: (∫⁺*x*. *w x* ∂*M*) < ∞
and *u′*: *AE x in M*. (λ*i*. *u i x*) ⟶ *u′ x*
shows (λ*i*. (∫⁺*x*. *u i x* ∂*M*)) ⟶ (∫⁺*x*. *u′ x* ∂*M*)
proof −
have *limsup* (λ*n*. *integral*^{N} *M* (*u n*)) ≤ (∫⁺ *x*. *limsup* (λ*n*. *u n x*) ∂*M*)
by (*intro nn_integral_limsup*[*OF* _ _ *bound w*]) *auto*
moreover have (∫⁺ *x*. *limsup* (λ*n*. *u n x*) ∂*M*) = (∫⁺ *x*. *u′ x* ∂*M*)
using *u′* **by** (*intro nn_integral_cong_AE*, *eventually_elim*) (*metis tendsto_iff_Liminf_eq_Limsup sequentially_bot*)
moreover have (∫⁺ *x*. *liminf* (λ*n*. *u n x*) ∂*M*) = (∫⁺ *x*. *u′ x* ∂*M*)
using *u′* **by** (*intro nn_integral_cong_AE*, *eventually_elim*) (*metis tendsto_iff_Liminf_eq_Limsup sequentially_bot*)
moreover have (∫⁺*x*. *liminf* (λ*n*. *u n x*) ∂*M*) ≤ *liminf* (λ*n*. *integral*^{N} *M* (*u n*))
by (*intro nn_integral_liminf*) *auto*
moreover have *liminf* (λ*n*. *integral*^{N} *M* (*u n*)) ≤ *limsup* (λ*n*. *integral*^{N} *M* (*u n*))
by (*intro Liminf_le_Limsup sequentially_bot*)
ultimately show *?thesis*
by (*intro Liminf_eq_Limsup*) *auto*
qed

lemma *inf_continuous_nn_integral*[*order_continuous_intros*]:
assumes *f*: ⋀*y*. *inf_continuous* (*f y*)
assumes [*measurable*]: ⋀*x*. (λ*y*. *f y x*) ∈ *borel_measurable M*
assumes *bnd*: ⋀*x*. (∫⁺ *y*. *f y x* ∂*M*) ≠ ∞
shows *inf_continuous* (λ*x*. (∫⁺*y*. *f y x* ∂*M*))
unfolding *inf_continuous_def*
proof *safe*
fix *C* :: *nat* ⇒ *'b* **assume** *C*: *decseq C*
then show (∫⁺ *y*. *f y* (*Inf* (*C* ‘ *UNIV*)) ∂*M*) = (*INF i*. ∫⁺ *y*. *f y* (*C i*) ∂*M*)
using *inf_continuous_mono*[*OF f*] *bnd*
by (*auto simp add*: *inf_continuousD*[*OF f C*] *fun_eq_iff monotone_def le_fun_def less_top*
intro!: *nn_integral_monotone_convergence_INF_decseq*)
qed

lemma *nn_integral_null_set*:
assumes *N* ∈ *null_sets M* **shows** (∫⁺ *x*. *u x* ∗ *indicator N x* ∂*M*) = *0*
proof −
have (∫⁺ *x*. *u x* ∗ *indicator N x* ∂*M*) = (∫⁺ *x*. *0* ∂*M*)
proof (*intro nn_integral_cong_AE AE_I*)
show {*x* ∈ *space M*. *u x* ∗ *indicator N x* ≠ *0*} ⊆ *N*

```
    by (auto simp: indicator_def)
   show (emeasure M) N = 0 N ∈ sets M
    using assms by auto
 qed
 then show ?thesis by simp
qed

lemma nn_integral_0_iff:
 assumes u [measurable]: u ∈ borel_measurable M
 shows integral^N M u = 0 ⟷ emeasure M {x∈space M. u x ≠ 0} = 0
   (is _ ⟷ (emeasure M) ?A = 0)
proof −
 have u_eq: (∫⁺ x. u x ∗ indicator ?A x ∂M) = integral^N M u
   by (auto intro!: nn_integral_cong simp: indicator_def)
 show ?thesis
 proof
   assume (emeasure M) ?A = 0
   with nn_integral_null_set[of ?A M u] u
   show integral^N M u = 0 by (simp add: u_eq null_sets_def)
 next
   assume ∗: integral^N M u = 0
   let ?M = λn. {x ∈ space M. 1 ≤ real (n::nat) ∗ u x}
   have 0 = (SUP n. (emeasure M) (?M n ∩ ?A))
   proof −
     { fix n :: nat
       have emeasure M {x ∈ ?A. 1 ≤ of_nat n ∗ u x} ≤
             of_nat n ∗ ∫⁺ x. u x ∗ indicator ?A x ∂M
         by (intro nn_integral_Markov_inequality) auto
       also have {x ∈ ?A. 1 ≤ of_nat n ∗ u x} = (?M n ∩ ?A)
         by (auto simp: ennreal_of_nat_eq_real_of_nat u_eq ∗ )
       finally have emeasure M (?M n ∩ ?A) ≤ 0
         by (simp add: ennreal_of_nat_eq_real_of_nat u_eq ∗ )
       moreover have 0 ≤ (emeasure M) (?M n ∩ ?A) using u by auto
       ultimately have (emeasure M) (?M n ∩ ?A) = 0 by auto }
     thus ?thesis by simp
   qed
   also have ... = (emeasure M) (⋃ n. ?M n ∩ ?A)
   proof (safe intro!: SUP_emeasure_incseq)
     fix n show ?M n ∩ ?A ∈ sets M
       using u by (auto intro!: sets.Int)
   next
     show incseq (λn. {x ∈ space M. 1 ≤ real n ∗ u x} ∩ {x ∈ space M. u x ≠
0})
     proof (safe intro!: incseq_SucI)
       fix n :: nat and x
       assume ∗: 1 ≤ real n ∗ u x
       also have real n ∗ u x ≤ real (Suc n) ∗ u x
         by (auto intro!: mult_right_mono)
       finally show 1 ≤ real (Suc n) ∗ u x by auto
```

```
    qed
  qed
  also have ... = (emeasure M) {x∈space M. 0 < u x}
  proof (safe intro!: arg_cong[where f=(emeasure M)])
    fix x assume 0 < u x and [simp, intro]: x ∈ space M
    show x ∈ (⋃ n. ?M n ∩ ?A)
    proof (cases u x rule: ennreal_cases)
      case (real r) with ‹0 < u x› have 0 < r by auto
      obtain j :: nat where 1 / r ≤ real j using real_arch_simple ..
      hence 1 / r * r ≤ real j * r unfolding mult_le_cancel_right using ‹0 <
r› by auto
      hence 1 ≤ real j * r using real ‹0 < r› by auto
      thus ?thesis using ‹0 < r› real
        by (auto simp: ennreal_of_nat_eq_real_of_nat ennreal_1[symmetric]
ennreal_mult[symmetric]
            simp del: ennreal_1)
    qed (insert ‹0 < u x›, auto simp: ennreal_mult_top)
  qed (auto simp: zero_less_iff_neq_zero)
  finally show emeasure M ?A = 0
    by (simp add: zero_less_iff_neq_zero)
 qed
qed

lemma nn_integral_0_iff_AE:
 assumes u: u ∈ borel_measurable M
 shows integral^N M u = 0 ⟷ (AE x in M. u x = 0)
proof −
 have sets: {x∈space M. u x ≠ 0} ∈ sets M
   using u by auto
 show integral^N M u = 0 ⟷ (AE x in M. u x = 0)
   using nn_integral_0_iff[of u] AE_iff_null[OF sets] u by auto
qed

lemma AE_iff_nn_integral:
 {x∈space M. P x} ∈ sets M ⟹ (AE x in M. P x) ⟷ integral^N M (indicator
{x. ¬ P x}) = 0
 by (subst nn_integral_0_iff_AE) (auto simp: indicator_def[abs_def])

lemma nn_integral_less:
 assumes [measurable]: f ∈ borel_measurable M g ∈ borel_measurable M
 assumes f: (∫⁺x. f x ∂M) ≠ ∞
 assumes ord: AE x in M. f x ≤ g x ¬ (AE x in M. g x ≤ f x)
 shows (∫⁺x. f x ∂M) < (∫⁺x. g x ∂M)
proof −
 have 0 < (∫⁺x. g x − f x ∂M)
 proof (intro order_le_neq_trans notI)
   assume 0 = (∫⁺x. g x − f x ∂M)
   then have AE x in M. g x − f x = 0
     using nn_integral_0_iff_AE[of λx. g x − f x M] by simp
```

```
    with ord(1) have AE x in M. g x ≤ f x
      by eventually_elim (auto simp: ennreal_minus_eq_0)
    with ord show False
      by simp
  qed simp
  also have ... = (∫⁺x. g x ∂M) - (∫⁺x. f x ∂M)
    using f by (subst nn_integral_diff) (auto simp: ord)
  finally show ?thesis
    using f by (auto dest!: ennreal_minus_pos_iff[rotated] simp: less_top)
qed

lemma nn_integral_subalgebra:
  assumes f: f ∈ borel_measurable N
  and N: sets N ⊆ sets M space N = space M ⋀A. A ∈ sets N ⟹ emeasure N
A = emeasure M A
  shows integral^N N f = integral^N M f
proof -
 have [simp]: ⋀f :: 'a ⇒ ennreal. f ∈ borel_measurable N ⟹ f ∈ borel_measurable
M
    using N by (auto simp: measurable_def)
  have [simp]: ⋀P. (AE x in N. P x) ⟹ (AE x in M. P x)
    using N by (auto simp add: eventually_ae_filter null_sets_def subset_eq)
  have [simp]: ⋀A. A ∈ sets N ⟹ A ∈ sets M
    using N by auto
  from f show ?thesis
    apply induct
   apply (simp_all add: nn_integral_add nn_integral_cmult nn_integral_monotone_convergence_SUP
N image_comp)
   apply (auto intro!: nn_integral_cong cong: nn_integral_cong simp: N(2)[symmetric])
    done
qed

lemma nn_integral_nat_function:
  fixes f :: 'a ⇒ nat
  assumes f ∈ measurable M (count_space UNIV)
  shows (∫⁺x. of_nat (f x) ∂M) = (∑ t. emeasure M {x∈space M. t < f x})
proof -
  define F where F i = {x∈space M. i < f x} for i
  with assms have [measurable]: ⋀i. F i ∈ sets M
    by auto

  { fix x assume x ∈ space M
    have (λi. if i < f x then 1 else 0) sums (of_nat (f x)::real)
      using sums_If_finite[of λi. i < f x λ_. 1::real] by simp
    then have (λi. ennreal (if i < f x then 1 else 0)) sums of_nat(f x)
      unfolding ennreal_of_nat_eq_real_of_nat
      by (subst sums_ennreal) auto
    moreover have ⋀i. ennreal (if i < f x then 1 else 0) = indicator (F i) x
      using ‹x ∈ space M› by (simp add: one_ennreal_def F_def)
```

ultimately have *of_nat* $(f\ x) = (\sum i.\ indicator\ (F\ i)\ x :: ennreal)$
by (*simp add*: *sums_iff*) **}**
then have $(\int^+ x.\ of\_nat\ (f\ x)\ \partial M) = (\int^+ x.\ (\sum i.\ indicator\ (F\ i)\ x)\ \partial M)$
by (*simp cong*: *nn_integral_cong*)
also have $\ldots = (\sum i.\ emeasure\ M\ (F\ i))$
by (*simp add*: *nn_integral_suminf*)
finally show *?thesis*
by (*simp add*: *F_def*)
qed

theorem *nn_integral_lfp*:
assumes *sets*[*simp*]: $\bigwedge s.\ sets\ (M\ s) = sets\ N$
assumes *f*: *sup_continuous f*
assumes *g*: *sup_continuous g*
assumes *meas*: $\bigwedge F.\ F \in borel\_measurable\ N \Longrightarrow f\ F \in borel\_measurable\ N$
assumes *step*: $\bigwedge F\ s.\ F \in borel\_measurable\ N \Longrightarrow integral^N\ (M\ s)\ (f\ F) = g\ (\lambda s.\ integral^N\ (M\ s)\ F)\ s$
shows $(\int^+ \omega.\ lfp\ f\ \omega\ \partial M\ s) = lfp\ g\ s$
proof (*subst lfp_transfer_bounded*[**where** α=$\lambda F\ s.\ \int^+ x.\ F\ x\ \partial M\ s$ **and** *g*=*g* **and** *f*=*f* **and** *P*=$\lambda f.\ f \in borel\_measurable\ N$, *symmetric*])
fix $C :: nat \Rightarrow 'b \Rightarrow ennreal$ **assume** *incseq C* $\bigwedge i.\ C\ i \in borel\_measurable\ N$
then show $(\lambda s.\ \int^+ x.\ (SUP\ i.\ C\ i)\ x\ \partial M\ s) = (SUP\ i.\ (\lambda s.\ \int^+ x.\ C\ i\ x\ \partial M\ s))$
unfolding *SUP_apply*[*abs_def*]
by (*subst nn_integral_monotone_convergence_SUP*)
(*auto simp*: *mono_def fun_eq_iff intro*!: *arg_cong2*[**where** *f*=*emeasure*] *cong*: *measurable_cong_sets*)
qed (*auto simp add*: *step le_fun_def SUP_apply*[*abs_def*] *bot_fun_def bot_ennreal intro*!: *meas f g*)

theorem *nn_integral_gfp*:
assumes *sets*[*simp*]: $\bigwedge s.\ sets\ (M\ s) = sets\ N$
assumes *f*: *inf_continuous f* **and** *g*: *inf_continuous g*
assumes *meas*: $\bigwedge F.\ F \in borel\_measurable\ N \Longrightarrow f\ F \in borel\_measurable\ N$
assumes *bound*: $\bigwedge F\ s.\ F \in borel\_measurable\ N \Longrightarrow (\int^+ x.\ f\ F\ x\ \partial M\ s) < \infty$
assumes *non_zero*: $\bigwedge s.\ emeasure\ (M\ s)\ (space\ (M\ s)) \neq 0$
assumes *step*: $\bigwedge F\ s.\ F \in borel\_measurable\ N \Longrightarrow integral^N\ (M\ s)\ (f\ F) = g\ (\lambda s.\ integral^N\ (M\ s)\ F)\ s$
shows $(\int^+ \omega.\ gfp\ f\ \omega\ \partial M\ s) = gfp\ g\ s$
proof (*subst gfp_transfer_bounded*[**where** α=$\lambda F\ s.\ \int^+ x.\ F\ x\ \partial M\ s$ **and** *g*=*g* **and** *f*=*f*
and *P*=$\lambda F.\ F \in borel\_measurable\ N \wedge (\forall s.\ (\int^+ x.\ F\ x\ \partial M\ s) < \infty)$, *symmetric*])
fix $C :: nat \Rightarrow 'b \Rightarrow ennreal$ **assume** *decseq C* $\bigwedge i.\ C\ i \in borel\_measurable\ N \wedge (\forall s.\ integral^N\ (M\ s)\ (C\ i) < \infty)$
then show $(\lambda s.\ \int^+ x.\ (INF\ i.\ C\ i)\ x\ \partial M\ s) = (INF\ i.\ (\lambda s.\ \int^+ x.\ C\ i\ x\ \partial M\ s))$
unfolding *INF_apply*[*abs_def*]
by (*subst nn_integral_monotone_convergence_INF_decseq*)
(*auto simp*: *mono_def fun_eq_iff intro*!: *arg_cong2*[**where** *f*=*emeasure*] *cong*: *measurable_cong_sets*)

next
show $\bigwedge x.\ g\ x \leq (\lambda s.\ integral^N\ (M\ s)\ (f\ top))$
by (*subst step*)
(*auto simp add*: *top_fun_def less_le non_zero le_fun_def ennreal_top_mult*
cong del: *if_weak_cong intro*!: *monoD*[*OF inf_continuous_mono*[*OF g*],
THEN le_funD])
next
fix *C* **assume** $\bigwedge i$::*nat*. *C i* ∈ *borel_measurable N* ∧ (∀ *s*. *integral*^{N} (*M s*) (*C i*)
< ∞) *decseq C*
with *bound* **show** *Inf* (*C* ‘ *UNIV*) ∈ *borel_measurable N* ∧ (∀ *s*. *integral*^{N} (*M*
s) (*Inf* (*C* ‘ *UNIV*)) < ∞)
unfolding *INF_apply*[*abs_def*]
by (*subst nn_integral_monotone_convergence_INF_decseq*)
(*auto simp*: *INF_less_iff cong*: *measurable_cong_sets intro*!: *borel_measurable_INF*)
next
show $\bigwedge x.\ x \in borel\_measurable\ N \wedge (\forall\, s.\ integral^N\ (M\ s)\ x < \infty) \Longrightarrow$
$(\lambda s.\ integral^N\ (M\ s)\ (f\ x)) = g\ (\lambda s.\ integral^N\ (M\ s)\ x)$
by (*subst step*) *auto*
qed (*insert bound*, *auto simp add*: *le_fun_def INF_apply*[*abs_def*] *top_fun_def*
intro!: *meas f g*)

Cauchy–Schwarz inequality for *integral*^{N}

lemma *sum_of_squares_ge_ennreal*:
fixes *a b* :: *ennreal*
shows $2 * a * b \leq a^2 + b^2$
proof (*cases a*; *cases b*)
fix *x y*
assume *xy*: $x \geq 0\ y \geq 0$ **and** [*simp*]: $a = ennreal\ x\ b = ennreal\ y$
have $0 \leq (x - y)^2$
by *simp*
also have $\ldots = x^2 + y^2 - 2 * x * y$
by (*simp add*: *algebra_simps power2_eq_square*)
finally have $2 * x * y \leq x^2 + y^2$
by *simp*
hence *ennreal* $(2 * x * y) \leq ennreal\ (x^2 + y^2)$
by (*intro ennreal_leI*)
thus *?thesis* **using** *xy*
by (*simp add*: *ennreal_mult ennreal_power*)
qed *auto*

lemma *Cauchy_Schwarz_nn_integral*:
assumes [*measurable*]: $f \in borel\_measurable\ M\ g \in borel\_measurable\ M$
shows $(\int^+ x.\ f\ x * g\ x\ \partial M)^2 \leq (\int^+ x.\ f\ x\ \hat{}\ 2\ \partial M) * (\int^+ x.\ g\ x\ \hat{}\ 2\ \partial M)$
proof (*cases* $(\int^+ x.\ f\ x * g\ x\ \partial M) = 0$)
case *False*
define *F* **where** $F = nn\_integral\ M\ (\lambda x.\ f\ x\ \hat{}\ 2)$
define *G* **where** $G = nn\_integral\ M\ (\lambda x.\ g\ x\ \hat{}\ 2)$
from *False* **have** ¬(*AE x in M*. $f\ x = 0 \vee g\ x = 0$)
by (*auto simp*: *nn_integral_0_iff_AE*)

hence $\neg(AE\ x\ in\ M.\ f\ x = 0)$ **and** $\neg(AE\ x\ in\ M.\ g\ x = 0)$
by (*auto intro*: *AE_disjI1 AE_disjI2*)
hence *nz*: $F \neq 0\ G \neq 0$
by (*auto simp*: *nn_integral_0_iff_AE F_def G_def*)

show *?thesis*
proof (*cases* $F = \infty \vee G = \infty$)
case *True*
thus *?thesis* **using** *nz*
by (*auto simp*: *F_def G_def*)
next
case *False*
define F' **where** $F' = ennreal\ (sqrt\ (enn2real\ F))$
define G' **where** $G' = ennreal\ (sqrt\ (enn2real\ G))$
from *False* **have** *fin*: $F < top\ G < top$
by (*simp_all add*: *top.not_eq_extremum*)
have *F'_sqr*: $F'^2 = F$
using *False* **by** (*cases F*) (*auto simp*: *F'_def ennreal_power*)
have *G'_sqr*: $G'^2 = G$
using *False* **by** (*cases G*) (*auto simp*: *G'_def ennreal_power*)
have *nz'*: $F' \neq 0\ G' \neq 0$ **and** *fin'*: $F' \neq \infty\ G' \neq \infty$
using *F'_sqr G'_sqr nz fin* **by** *auto*
from *fin'* **have** *fin''*: $F' < top\ G' < top$
by (*auto simp*: *top.not_eq_extremum*)

have $2 * (F' / F') * (G' / G') * (\int^+ x.\ f\ x * g\ x\ \partial M) =$
$F' * G' * (\int^+ x.\ 2 * (f\ x / F') * (g\ x / G')\ \partial M)$
using *nz' fin''*
by (*simp add*: *divide_ennreal_def algebra_simps ennreal_inverse_mult flip*: *nn_integral_cmult*)
also have $F'/ F' = 1$
using *nz' fin''* **by** *simp*
also have $G'/ G' = 1$
using *nz' fin''* **by** *simp*
also have $2 * 1 * 1 = (2 :: ennreal)$ **by** *simp*
also have $F' * G' * (\int^+ x.\ 2 * (f\ x / F') * (g\ x / G')\ \partial M) \leq$
$F' * G' * (\int^+ x.\ (f\ x / F')^2 + (g\ x / G')^2\ \partial M)$
by (*intro mult_left_mono nn_integral_mono sum_of_squares_ge_ennreal*) *auto*
also have $\ldots = F' * G' * (F / F'^2 + G / G'^2)$ **using** *nz*
by (*auto simp*: *nn_integral_add algebra_simps nn_integral_divide F_def G_def*)
also have $F / F'^2 = 1$
using *nz F'_sqr fin* **by** *simp*
also have $G / G'^2 = 1$
using *nz G'_sqr fin* **by** *simp*
also have $F' * G' * (1 + 1) = 2 * (F' * G')$
by (*simp add*: *mult_ac*)
finally have $(\int^+ x.\ f\ x * g\ x\ \partial M) \leq F' * G'$

by (*subst* (*asm*) *ennreal_mult_le_mult_iff*) *auto*
hence $(\int^+ x.\ f\ x * g\ x\ \partial M)^2 \leq (F' * G')^2$
by (*intro power_mono_ennreal*)
also have $\ldots = F * G$
by (*simp add*: *algebra_simps F'_sqr G'_sqr*)
finally show *?thesis*
by (*simp add*: *F_def G_def*)
qed
qed *auto*

6.5.5 Integral under concrete measures

lemma *nn_integral_mono_measure*:
assumes *sets M* = *sets N M* ≤ *N* **shows** *nn_integral M f* ≤ *nn_integral N f*
unfolding *nn_integral_def*
proof (*intro SUP_subset_mono*)
note ‹*sets M* = *sets N*›[*simp*] ‹*sets M* = *sets N*›[*THEN sets_eq_imp_space_eq*, *simp*]
show {*g*. *simple_function M g* ∧ *g* ≤ *f*} ⊆ {*g*. *simple_function N g* ∧ *g* ≤ *f*}
by (*simp add*: *simple_function_def*)
show *integral*^{S} *M x* ≤ *integral*^{S} *N x* **for** *x*
using *le_measureD3*[*OF* ‹*M* ≤ *N*›]
by (*auto simp add*: *simple_integral_def intro*!: *sum_mono mult_mono*)
qed

lemma *nn_integral_empty*:
assumes *space M* = {}
shows *nn_integral M f* = *0*
proof −
have $(\int^+ x.\ f\ x\ \partial M) = (\int^+ x.\ 0\ \partial M)$
by(*rule nn_integral_cong*)(*simp add*: *assms*)
thus *?thesis* **by** *simp*
qed

lemma *nn_integral_bot*[*simp*]: *nn_integral bot f* = *0*
by (*simp add*: *nn_integral_empty*)

Distributions

lemma *nn_integral_distr*:
assumes *T*: *T* ∈ *measurable M M'* **and** *f*: *f* ∈ *borel_measurable* (*distr M M' T*)
shows *integral*^{N} (*distr M M' T*) *f* = $(\int^+ x.\ f\ (T\ x)\ \partial M)$
using *f*
proof *induct*
case (*cong f g*)
with *T* **show** *?case*
apply (*subst nn_integral_cong*[*of* _ *f g*])
apply *simp*
apply (*subst nn_integral_cong*[*of* _ λ*x*. *f* (*T x*) λ*x*. *g* (*T x*)])

apply (*simp add*: *measurable_def Pi_iff*)
apply *simp*
done
next
case (*set A*)
then have *eq*: $\bigwedge x.\ x \in space\ M \Longrightarrow indicator\ A\ (T\ x) = indicator\ (T\ -`\ A \cap space\ M)\ x$
by (*auto simp*: *indicator_def*)
from *set T* **show** *?case*
by (*subst nn_integral_cong*[*OF eq*])
(*auto simp add*: *emeasure_distr intro*!: *nn_integral_indicator*[*symmetric*] *measurable_sets*)
qed (*simp_all add*: *measurable_compose*[*OF T*] *T nn_integral_cmult nn_integral_add nn_integral_monotone_convergence_SUP le_fun_def incseq_def image_comp*)

Counting space

lemma *simple_function_count_space*[*simp*]:
simple_function (*count_space A*) *f* $\longleftrightarrow$ *finite* (*f* ` *A*)
unfolding *simple_function_def* **by** *simp*

lemma *nn_integral_count_space*:
assumes *A*: *finite* {*a*∈*A*. *0* < *f a*}
shows $integral^N\ (count\_space\ A)\ f = (\sum a|a{\in}A \wedge 0 < f\ a.\ f\ a)$
proof −
have ∗: $(\int^+ x.\ max\ 0\ (f\ x)\ \partial count\_space\ A) =$
$(\int^+ x.\ (\sum a|a{\in}A \wedge 0 < f\ a.\ f\ a * indicator\ \{a\}\ x)\ \partial count\_space\ A)$
by (*auto intro*!: *nn_integral_cong simp add*: *indicator_def of_bool_def if_distrib sum.If_cases*[*OF A*] *max_def le_less*)
also have ... = $(\sum a|a{\in}A \wedge 0 < f\ a.\ \int^+ x.\ f\ a * indicator\ \{a\}\ x\ \partial count\_space\ A)$
by (*subst nn_integral_sum*) (*simp_all add*: *AE_count_space less_imp_le*)
also have ... = $(\sum a|a{\in}A \wedge 0 < f\ a.\ f\ a)$
by (*auto intro*!: *sum.cong simp*: *one_ennreal_def*[*symmetric*] *max_def*)
finally show *?thesis* **by** (*simp add*: *max.absorb2*)
qed

lemma *nn_integral_count_space_finite*:
finite A $\Longrightarrow (\int^+ x.\ f\ x\ \partial count\_space\ A) = (\sum a{\in}A.\ f\ a)$
by (*auto intro*!: *sum.mono_neutral_left simp*: *nn_integral_count_space less_le*)

lemma *nn_integral_count_space'*:
assumes *finite A* $\bigwedge x.\ x \in B \Longrightarrow x \notin A \Longrightarrow f\ x = 0$ $A \subseteq B$
shows $(\int^+ x.\ f\ x\ \partial count\_space\ B) = (\sum x{\in}A.\ f\ x)$
proof −
have $(\int^+ x.\ f\ x\ \partial count\_space\ B) = (\sum a \mid a \in B \wedge 0 < f\ a.\ f\ a)$
using *assms*(*2*,*3*)

by (*intro nn_integral_count_space finite_subset*[*OF* _ ‹*finite A*›]) (*auto simp*: *less_le*)
also have ... = ($\sum a{\in}A.\ f\ a$)
using *assms* **by** (*intro sum.mono_neutral_cong_left*) (*auto simp*: *less_le*)
finally show *?thesis* **.**
qed

lemma *nn_integral_bij_count_space*:
assumes *g*: *bij_betw g A B*
shows ($\int^+ x.\ f\ (g\ x)\ \partial count\_space\ A$) = ($\int^+ x.\ f\ x\ \partial count\_space\ B$)
using *g*[*THEN bij_betw_imp_funcset*]
by (*subst distr_bij_count_space*[*OF g*, *symmetric*])
(*auto intro*!: *nn_integral_distr*[*symmetric*])

lemma *nn_integral_indicator_finite*:
fixes *f* :: *'a* ⇒ *ennreal*
assumes *f*: *finite A* **and** [*measurable*]: $\bigwedge a.\ a \in A \Longrightarrow \{a\} \in sets\ M$
shows ($\int^+ x.\ f\ x * indicator\ A\ x\ \partial M$) = ($\sum x{\in}A.\ f\ x * emeasure\ M\ \{x\}$)
proof −
from *f* **have** ($\int^+ x.\ f\ x * indicator\ A\ x\ \partial M$) = ($\int^+ x.\ (\sum a{\in}A.\ f\ a * indicator\ \{a\}\ x)\ \partial M$)
by (*intro nn_integral_cong*) (*auto simp*: *indicator_def if_distrib*[**where** *f*=λ*a*. *x* ∗ *a* **for** *x*] *sum.If_cases*)
also have ... = ($\sum a{\in}A.\ f\ a * emeasure\ M\ \{a\}$)
by (*subst nn_integral_sum*) *auto*
finally show *?thesis* **.**
qed

lemma *nn_integral_count_space_nat*:
fixes *f* :: *nat* ⇒ *ennreal*
shows ($\int^+ i.\ f\ i\ \partial count\_space\ UNIV$) = ($\sum i.\ f\ i$)
proof −
have ($\int^+ i.\ f\ i\ \partial count\_space\ UNIV$) =
($\int^+ i.\ (\sum j.\ f\ j * indicator\ \{j\}\ i)\ \partial count\_space\ UNIV$)
proof (*intro nn_integral_cong*)
fix *i*
have *f i* = ($\sum j{\in}\{i\}.\ f\ j * indicator\ \{j\}\ i$)
by *simp*
also have ... = ($\sum j.\ f\ j * indicator\ \{j\}\ i$)
by (*rule suminf_finite*[*symmetric*]) *auto*
finally show *f i* = ($\sum j.\ f\ j * indicator\ \{j\}\ i$) **.**
qed
also have ... = ($\sum j.\ (\int^+ i.\ f\ j * indicator\ \{j\}\ i\ \partial count\_space\ UNIV)$)
by (*rule nn_integral_suminf*) *auto*
finally show *?thesis*
by *simp*
qed

lemma *nn_integral_enat_function*:

assumes *f*: $f \in$ *measurable M* (*count_space UNIV*)
shows $(\int^+ x.$ *ennreal_of_enat* $(f\ x)\ \partial M) = (\sum t.$ *emeasure M* $\{x \in$ *space M*. $t < f\ x\})$
proof −
define *F* **where** $F\ i = \{x \in$ *space M*. $i < f\ x\}$ **for** $i ::$ *nat*
with *assms* **have** [*measurable*]: $\bigwedge i.$ $F\ i \in$ *sets M*
by *auto*

{ **fix** *x* **assume** $x \in$ *space M*
have $(\lambda i{::}nat.$ *if* $i < f\ x$ *then 1 else 0*) *sums ennreal_of_enat* $(f\ x)$
using *sums_If_finite*[*of* $\lambda r.\ r < f\ x\ \lambda\_.\ 1 ::$ *ennreal*]
by (*cases f x*) (*simp_all add*: *sums_def of_nat_tendsto_top_ennreal*)
also have $(\lambda i.$ (*if* $i < f\ x$ *then 1 else 0*)) = $(\lambda i.$ *indicator* $(F\ i)\ x)$
using ‹$x \in$ *space M*› **by** (*simp add*: *one_ennreal_def F_def fun_eq_iff*)
finally have *ennreal_of_enat* $(f\ x) = (\sum i.$ *indicator* $(F\ i)\ x)$
by (*simp add*: *sums_iff*) }
then have $(\int^+ x.$ *ennreal_of_enat* $(f\ x)\ \partial M) = (\int^+ x.$ $(\sum i.$ *indicator* $(F\ i)\ x)$ $\partial M)$
by (*simp cong*: *nn_integral_cong*)
also have $\ldots = (\sum i.$ *emeasure M* $(F\ i))$
by (*simp add*: *nn_integral_suminf*)
finally show *?thesis*
by (*simp add*: *F_def*)
qed

lemma *nn_integral_count_space_nn_integral*:
fixes *f* :: $'i \Rightarrow 'a \Rightarrow$ *ennreal*
assumes *countable I* **and** [*measurable*]: $\bigwedge i.$ $i \in I \Longrightarrow f\ i \in$ *borel_measurable M*
shows $(\int^+ x.$ $\int^+ i.$ $f\ i\ x\ \partial$*count_space I* $\partial M) = (\int^+ i.$ $\int^+ x.$ $f\ i\ x\ \partial M\ \partial$*count_space I*)
proof *cases*
assume *finite I* **then show** *?thesis*
by (*simp add*: *nn_integral_count_space_finite nn_integral_sum*)
next
assume *infinite I*
then have [*simp*]: $I \neq \{\}$
by *auto*
note $* =$ *bij_betw_from_nat_into*[*OF* ‹*countable I*› ‹*infinite I*›]
have $**$: $\bigwedge f.$ $(\bigwedge i.\ 0 \leq f\ i) \Longrightarrow (\int^+ i.\ f\ i\ \partial$*count_space I*$) = (\sum n.\ f$ (*from_nat_into I n*))
by (*simp add*: *nn_integral_bij_count_space*[*symmetric*, *OF* $*$] *nn_integral_count_space_nat*)
show *?thesis*
by (*simp add*: $**$ *nn_integral_suminf from_nat_into*)
qed

lemma *of_bool_Bex_eq_nn_integral*:
assumes *unique*: $\bigwedge x\ y.\ x \in X \Longrightarrow y \in X \Longrightarrow P\ x \Longrightarrow P\ y \Longrightarrow x = y$
shows *of_bool* $(\exists y{\in}X.\ P\ y) = (\int^+ y.$ *of_bool* $(P\ y)\ \partial$*count_space X*$)$
proof *cases*

```
  assume ∃ y∈X. P y
  then obtain y where P y y ∈ X by auto
  then show ?thesis
    by (subst nn_integral_count_space'[where A={y}]) (auto dest: unique)
qed (auto cong: nn_integral_cong_simp)

lemma emeasure_UN_countable:
  assumes sets[measurable]: ⋀i. i ∈ I ⟹ X i ∈ sets M and I[simp]: countable I
  assumes disj: disjoint_family_on X I
  shows emeasure M (⋃(X ` I)) = (∫⁺i. emeasure M (X i) ∂count_space I)
proof −
  have eq: ⋀x. indicator (⋃(X ` I)) x = ∫⁺ i. indicator (X i) x ∂count_space I
  proof cases
    fix x assume x: x ∈ ⋃(X ` I)
    then obtain j where j: x ∈ X j j ∈ I
      by auto
    with disj have ⋀i. i ∈ I ⟹ indicator (X i) x = (indicator {j} i::ennreal)
      by (auto simp: disjoint_family_on_def split: split_indicator)
    with x j show ?thesis x
      by (simp cong: nn_integral_cong_simp)
  qed (auto simp: nn_integral_0_iff_AE)

  note sets.countable_UN'[unfolded subset_eq, measurable]
  have emeasure M (⋃(X ` I)) = (∫⁺x. indicator (⋃(X ` I)) x ∂M)
    by simp
  also have ... = (∫⁺i. ∫⁺x. indicator (X i) x ∂M ∂count_space I)
    by (simp add: eq nn_integral_count_space_nn_integral)
  finally show ?thesis
    by (simp cong: nn_integral_cong_simp)
qed

lemma emeasure_countable_singleton:
  assumes sets: ⋀x. x ∈ X ⟹ {x} ∈ sets M and X: countable X
  shows emeasure M X = (∫⁺x. emeasure M {x} ∂count_space X)
proof −
  have emeasure M (⋃i∈X. {i}) = (∫⁺x. emeasure M {x} ∂count_space X)
   using assms by (intro emeasure_UN_countable) (auto simp: disjoint_family_on_def)
  also have (⋃i∈X. {i}) = X by auto
  finally show ?thesis .
qed

lemma measure_eqI_countable:
  assumes [simp]: sets M = Pow A sets N = Pow A and A: countable A
  assumes eq: ⋀a. a ∈ A ⟹ emeasure M {a} = emeasure N {a}
  shows M = N
proof (rule measure_eqI)
  fix X assume X ∈ sets M
  then have X: X ⊆ A by auto
  moreover from A X have countable X by (auto dest: countable_subset)
```

ultimately have
emeasure M X = (∫⁺*a*. *emeasure M* {*a*} ∂*count_space X*)
emeasure N X = (∫⁺*a*. *emeasure N* {*a*} ∂*count_space X*)
by (*auto intro*!: *emeasure_countable_singleton*)
moreover have (∫⁺*a*. *emeasure M* {*a*} ∂*count_space X*) = (∫⁺*a*. *emeasure N* {*a*} ∂*count_space X*)
using *X* **by** (*intro nn_integral_cong eq*) *auto*
ultimately show *emeasure M X* = *emeasure N X*
by *simp*
qed *simp*

lemma *measure_eqI_countable_AE*:
assumes [*simp*]: *sets M* = *UNIV sets N* = *UNIV*
assumes *ae*: *AE x in M*. *x* ∈ Ω *AE x in N*. *x* ∈ Ω **and** [*simp*]: *countable* Ω
assumes *eq*: ⋀*x*. *x* ∈ Ω ⟹ *emeasure M* {*x*} = *emeasure N* {*x*}
shows *M* = *N*
proof (*rule measure_eqI*)
fix *A*
have *emeasure N A* = *emeasure N* {*x*∈Ω. *x* ∈ *A*}
using *ae* **by** (*intro emeasure_eq_AE*) *auto*
also have ... = (∫⁺*x*. *emeasure N* {*x*} ∂*count_space* {*x*∈Ω. *x* ∈ *A*})
by (*intro emeasure_countable_singleton*) *auto*
also have ... = (∫⁺*x*. *emeasure M* {*x*} ∂*count_space* {*x*∈Ω. *x* ∈ *A*})
by (*intro nn_integral_cong eq*[*symmetric*]) *auto*
also have ... = *emeasure M* {*x*∈Ω. *x* ∈ *A*}
by (*intro emeasure_countable_singleton*[*symmetric*]) *auto*
also have ... = *emeasure M A*
using *ae* **by** (*intro emeasure_eq_AE*) *auto*
finally show *emeasure M A* = *emeasure N A* **..**
qed *simp*

lemma *nn_integral_monotone_convergence_SUP_nat*:
fixes *f* :: ′*a* ⇒ *nat* ⇒ *ennreal*
assumes *chain*: *Complete_Partial_Order.chain* (≤) (*f* ‘ *Y*)
and *nonempty*: *Y* ≠ {}
shows (∫⁺ *x*. (*SUP i*∈*Y*. *f i x*) ∂*count_space UNIV*) = (*SUP i*∈*Y*. (∫⁺ *x*. *f i x* ∂*count_space UNIV*))
(**is** *?lhs* = *?rhs* **is** *integral*^{N} *?M* _ = _)
proof (*rule order_class.order.antisym*)
show *?rhs* ≤ *?lhs*
by (*auto intro*!: *SUP_least SUP_upper nn_integral_mono*)
next
have ∃*g*. *incseq g* ∧ *range g* ⊆ (λ*i*. *f i x*) ‘ *Y* ∧ (*SUP i*∈*Y*. *f i x*) = (*SUP i*. *g i*) **for** *x*
by (*rule ennreal_Sup_countable_SUP*) (*simp add*: *nonempty*)
then obtain *g* **where** *incseq*: ⋀*x*. *incseq* (*g x*)
and *range*: ⋀*x*. *range* (*g x*) ⊆ (λ*i*. *f i x*) ‘ *Y*
and *sup*: ⋀*x*. (*SUP i*∈*Y*. *f i x*) = (*SUP i*. *g x i*) **by** *moura*
from *incseq* **have** *incseq*′: *incseq* (λ*i x*. *g x i*)

```
    by(blast intro: incseq_SucI le_funI dest: incseq_SucD)

  have ?lhs = ∫⁺ x. (SUP i. g x i) ∂?M by(simp add: sup)
  also have ... = (SUP i. ∫⁺ x. g x i ∂?M) using incseq'
    by(rule nn_integral_monotone_convergence_SUP) simp
  also have ... ≤ (SUP i∈Y. ∫⁺ x. f i x ∂?M)
  proof(rule SUP_least)
    fix n
    have ⋀x. ∃ i. g x n = f i x ∧ i ∈ Y using range by blast
    then obtain I where I: ⋀x. g x n = f (I x) x ⋀x. I x ∈ Y by moura

    have (∫⁺ x. g x n ∂count_space UNIV) = (∑ x. g x n)
      by(rule nn_integral_count_space_nat)
    also have ... = (SUP m. ∑ x<m. g x n)
      by(rule suminf_eq_SUP)
    also have ... ≤ (SUP i∈Y. ∫⁺ x. f i x ∂?M)
    proof(rule SUP_mono)
      fix m
      show ∃ m'∈Y. (∑ x<m. g x n) ≤ (∫⁺ x. f m' x ∂?M)
      proof(cases m > 0)
        case False
        thus ?thesis using nonempty by auto
      next
        case True
        let ?Y = I ‘ {..<m}
        have f ‘ ?Y ⊆ f ‘ Y using I by auto
      with chain have chain': Complete_Partial_Order.chain (≤) (f ‘ ?Y) by(rule
chain_subset)
        hence Sup (f ‘ ?Y) ∈ f ‘ ?Y
       by(rule ccpo_class.in_chain_finite)(auto simp add: True lessThan_empty_iff)
        then obtain m' where m' < m and m': (SUP i∈?Y. f i) = f (I m') by
auto
        have I m' ∈ Y using I by blast
        have (∑ x<m. g x n) ≤ (∑ x<m. f (I m') x)
        proof(rule sum_mono)
          fix x
          assume x ∈ {..<m}
          hence x < m by simp
          have g x n = f (I x) x by(simp add: I)
         also have ... ≤ (SUP i∈?Y. f i) x unfolding Sup_fun_def image_image
            using ‹x ∈ {..<m}› by (rule Sup_upper [OF imageI])
          also have ... = f (I m') x unfolding m' by simp
          finally show g x n ≤ f (I m') x .
        qed
        also have ... ≤ (SUP m. (∑ x<m. f (I m') x))
          by(rule SUP_upper) simp
        also have ... = (∑ x. f (I m') x)
          by(rule suminf_eq_SUP[symmetric])
        also have ... = (∫⁺ x. f (I m') x ∂?M)
```

by(*rule nn_integral_count_space_nat*[*symmetric*])
finally show *?thesis* **using** ‹*I m'* ∈ *Y*› **by** *blast*
qed
qed
finally show (∫⁺ *x*. *g x n* ∂*count_space UNIV*) ≤ ... **.**
qed
finally show *?lhs* ≤ *?rhs* **.**
qed

lemma *power_series_tendsto_at_left*:
assumes *nonneg*: ⋀*i*. 0 ≤ *f i* **and** *summable*: ⋀*z*. 0 ≤ *z* ⟹ *z* < 1 ⟹ *summable* (λ*n*. *f n* ∗ *z*^*n*)
shows ((λ*z*. *ennreal* (∑ *n*. *f n* ∗ *z*^*n*)) ⟶ (∑ *n*. *ennreal* (*f n*))) (*at_left* (1::*real*))
proof (*intro tendsto_at_left_sequentially*)
show 0 < (1::*real*) **by** *simp*
fix *S* :: *nat* ⇒ *real* **assume** *S*: ⋀*n*. *S n* < 1 ⋀*n*. 0 < *S n S* ⟶ 1 *incseq S*
then have *S_nonneg*: ⋀*i*. 0 ≤ *S i* **by** (*auto intro*: *less_imp_le*)

have (λ*i*. (∫⁺*n*. *f n* ∗ *S i*^*n* ∂*count_space UNIV*)) ⟶ (∫⁺*n*. *ennreal* (*f n*) ∂*count_space UNIV*)
proof (*rule nn_integral_LIMSEQ*)
show *incseq* (λ*i n*. *ennreal* (*f n* ∗ *S i*^*n*))
using *S* **by** (*auto intro*!: *mult_mono power_mono nonneg ennreal_leI*
simp: *incseq_def le_fun_def less_imp_le*)
fix *n* **have** (λ*i*. *ennreal* (*f n* ∗ *S i*^*n*)) ⟶ *ennreal* (*f n* ∗ 1^*n*)
by (*intro tendsto_intros tendsto_ennrealI S*)
then show (λ*i*. *ennreal* (*f n* ∗ *S i*^*n*)) ⟶ *ennreal* (*f n*)
by *simp*
qed (*auto simp*: *S_nonneg intro*!: *mult_nonneg_nonneg nonneg*)
also have (λ*i*. (∫⁺*n*. *f n* ∗ *S i*^*n* ∂*count_space UNIV*)) = (λ*i*. ∑ *n*. *f n* ∗ *S i*^*n*)
by (*subst nn_integral_count_space_nat*)
(*intro ext suminf_ennreal2 mult_nonneg_nonneg nonneg S_nonneg*
zero_le_power summable S)+
also have (∫⁺*n*. *ennreal* (*f n*) ∂*count_space UNIV*) = (∑ *n*. *ennreal* (*f n*))
by (*simp add*: *nn_integral_count_space_nat nonneg*)
finally show (λ*n*. *ennreal* (∑ *na*. *f na* ∗ *S n* ^ *na*)) ⟶ (∑ *n*. *ennreal* (*f n*))
.
qed

Measures with Restricted Space

lemma *simple_function_restrict_space_ennreal*:
fixes *f* :: 'a ⇒ *ennreal*
assumes Ω ∩ *space M* ∈ *sets M*
shows *simple_function* (*restrict_space M* Ω) *f* ⟷ *simple_function M* (λ*x*. *f x* ∗ *indicator* Ω *x*)
proof −

```
  { assume finite (f ‘ space (restrict_space M Ω))
    then have finite (f ‘ space (restrict_space M Ω) ∪ {0}) by simp
    then have finite ((λx. f x ∗ indicator Ω x) ‘ space M)
    by (rule rev_finite_subset) (auto split: split_indicator simp: space_restrict_space)
}
  moreover
  { assume finite ((λx. f x ∗ indicator Ω x) ‘ space M)
    then have finite (f ‘ space (restrict_space M Ω))
    by (rule rev_finite_subset) (auto split: split_indicator simp: space_restrict_space)
}
  ultimately show ?thesis
    unfolding
    simple_function_iff_borel_measurable borel_measurable_restrict_space_iff_ennreal[OF
assms]
    by auto
qed

lemma simple_function_restrict_space:
  fixes f :: 'a ⇒ 'b::real_normed_vector
  assumes Ω ∩ space M ∈ sets M
   shows simple_function (restrict_space M Ω) f ⟷ simple_function M (λx.
indicator Ω x ∗R f x)
proof −
  { assume finite (f ‘ space (restrict_space M Ω))
    then have finite (f ‘ space (restrict_space M Ω) ∪ {0}) by simp
    then have finite ((λx. indicator Ω x ∗R f x) ‘ space M)
    by (rule rev_finite_subset) (auto split: split_indicator simp: space_restrict_space)
}
  moreover
  { assume finite ((λx. indicator Ω x ∗R f x) ‘ space M)
    then have finite (f ‘ space (restrict_space M Ω))
    by (rule rev_finite_subset) (auto split: split_indicator simp: space_restrict_space)
}
  ultimately show ?thesis
    unfolding simple_function_iff_borel_measurable
      borel_measurable_restrict_space_iff[OF assms]
    by auto
qed

lemma simple_integral_restrict_space:
  assumes Ω: Ω ∩ space M ∈ sets M simple_function (restrict_space M Ω) f
  shows simple_integral (restrict_space M Ω) f = simple_integral M (λx. f x ∗
indicator Ω x)
  using simple_function_restrict_space_ennreal[THEN iffD1, OF Ω, THEN sim-
ple_functionD(1)]
 by (auto simp add: space_restrict_space emeasure_restrict_space[OF Ω(1)] le_infI2
simple_integral_def
          split: split_indicator split_indicator_asm
       intro!: sum.mono_neutral_cong_left ennreal_mult_left_cong arg_cong2[where
```

f=emeasure])

lemma *nn_integral_restrict_space*:
assumes Ω[*simp*]: Ω ∩ *space M* ∈ *sets M*
shows *nn_integral* (*restrict_space M* Ω) *f* = *nn_integral M* (λ*x*. *f x* ∗ *indicator* Ω *x*)
proof −
let *?R* = *restrict_space M* Ω **and** *?X* = λ*M f*. {*s*. *simple_function M s* ∧ *s* ≤ *f* ∧ (∀ *x*. *s x* < *top*)}
have *integral*^{S} *?R* ' *?X ?R f* = *integral*^{S} *M* ' *?X M* (λ*x*. *f x* ∗ *indicator* Ω *x*)
proof (*safe intro*!: *image_eqI*)
fix *s* **assume** *s*: *simple_function ?R s s* ≤ *f* ∀ *x*. *s x* < *top*
from *s* **show** *integral*^{S} (*restrict_space M* Ω) *s* = *integral*^{S} *M* (λ*x*. *s x* ∗ *indicator* Ω *x*)
by (*intro simple_integral_restrict_space*) *auto*
from *s* **show** *simple_function M* (λ*x*. *s x* ∗ *indicator* Ω *x*)
by (*simp add*: *simple_function_restrict_space_ennreal*)
from *s* **show** (λ*x*. *s x* ∗ *indicator* Ω *x*) ≤ (λ*x*. *f x* ∗ *indicator* Ω *x*)
⋀*x*. *s x* ∗ *indicator* Ω *x* < *top*
by (*auto split*: *split_indicator simp*: *le_fun_def image_subset_iff*)
next
fix *s* **assume** *s*: *simple_function M s s* ≤ (λ*x*. *f x* ∗ *indicator* Ω *x*) ∀ *x*. *s x* < *top*
then have *simple_function M* (λ*x*. *s x* ∗ *indicator* (Ω ∩ *space M*) *x*) (**is** *?s'*)
by (*intro simple_function_mult simple_function_indicator*) *auto*
also have *?s'* ⟷ *simple_function M* (λ*x*. *s x* ∗ *indicator* Ω *x*)
by (*rule simple_function_cong*) (*auto split*: *split_indicator*)
finally show *sf*: *simple_function* (*restrict_space M* Ω) *s*
by (*simp add*: *simple_function_restrict_space_ennreal*)

from *s* **have** *s_eq*: *s* = (λ*x*. *s x* ∗ *indicator* Ω *x*)
by (*auto simp add*: *fun_eq_iff le_fun_def image_subset_iff*
split: *split_indicator split_indicator_asm*
intro: *antisym*)

show *integral*^{S} *M s* = *integral*^{S} (*restrict_space M* Ω) *s*
by (*subst s_eq*) (*rule simple_integral_restrict_space*[*symmetric*, *OF* Ω *sf*])
show ⋀*x*. *s x* < *top*
using *s* **by** (*auto simp*: *image_subset_iff*)
from *s* **show** *s* ≤ *f*
by (*subst s_eq*) (*auto simp*: *image_subset_iff le_fun_def split*: *split_indicator split_indicator_asm*)
qed
then show *?thesis*
unfolding *nn_integral_def_finite* **by** (*simp cong del*: *SUP_cong_simp*)
qed

lemma *nn_integral_count_space_indicator*:
assumes *NO_MATCH* (*UNIV*::'*a set*) (*X*::'*a set*)

shows $(\int^+ x.\ f\ x\ \partial count\_space\ X) = (\int^+ x.\ f\ x * indicator\ X\ x\ \partial count\_space\ UNIV)$
by (*simp add*: *nn_integral_restrict_space*[*symmetric*] *restrict_count_space*)

lemma *nn_integral_count_space_eq*:
$(\bigwedge x.\ x \in A - B \Longrightarrow f\ x = 0) \Longrightarrow (\bigwedge x.\ x \in B - A \Longrightarrow f\ x = 0) \Longrightarrow$
$(\int^+ x.\ f\ x\ \partial count\_space\ A) = (\int^+ x.\ f\ x\ \partial count\_space\ B)$
by (*auto simp*: *nn_integral_count_space_indicator intro*!: *nn_integral_cong split*: *split_indicator*)

lemma *nn_integral_ge_point*:
assumes $x \in A$
shows $p\ x \leq \int^+ x.\ p\ x\ \partial count\_space\ A$
proof −
from *assms* **have** $p\ x \leq \int^+ x.\ p\ x\ \partial count\_space\ \{x\}$
by(*auto simp add*: *nn_integral_count_space_finite max_def*)
also have $\ldots = \int^+ x'.\ p\ x' * indicator\ \{x\}\ x'\ \partial count\_space\ A$
using *assms* **by**(*auto simp add*: *nn_integral_count_space_indicator indicator_def intro*!: *nn_integral_cong*)
also have $\ldots \leq \int^+ x.\ p\ x\ \partial count\_space\ A$
by(*rule nn_integral_mono*)(*simp add*: *indicator_def*)
finally show *?thesis* **.**
qed

Measure spaces with an associated density

definition *density* :: *'a measure* ⇒ (*'a* ⇒ *ennreal*) ⇒ *'a measure* **where**
density M f = *measure_of* (*space M*) (*sets M*) (λ*A*. $\int^+ x.\ f\ x * indicator\ A\ x\ \partial M$)

lemma
shows *sets_density*[*simp*, *measurable_cong*]: *sets* (*density M f*) = *sets M*
and *space_density*[*simp*]: *space* (*density M f*) = *space M*
by (*auto simp*: *density_def*)

lemma *space_density_imp*[*measurable_dest*]:
$\bigwedge x\ M\ f.\ x \in space\ (density\ M\ f) \Longrightarrow x \in space\ M$ **by** *auto*

lemma
shows *measurable_density_eq1*[*simp*]: $g \in measurable\ (density\ Mg\ f)\ Mg' \longleftrightarrow g \in measurable\ Mg\ Mg'$
and *measurable_density_eq2*[*simp*]: $h \in measurable\ Mh\ (density\ Mh'\ f) \longleftrightarrow h \in measurable\ Mh\ Mh'$
and *simple_function_density_eq*[*simp*]: $simple\_function\ (density\ Mu\ f)\ u \longleftrightarrow simple\_function\ Mu\ u$
unfolding *measurable_def simple_function_def* **by** *simp_all*

lemma *density_cong*: $f \in borel\_measurable\ M \Longrightarrow f' \in borel\_measurable\ M \Longrightarrow$

(*AE x in M. f x = f' x*) ⟹ *density M f = density M f'*
unfolding *density_def* **by** (*auto intro*!: *measure_of_eq nn_integral_cong_AE sets.space_closed*)

lemma *emeasure_density*:
assumes *f*[*measurable*]: *f* ∈ *borel_measurable M* **and** *A*[*measurable*]: *A* ∈ *sets M*
shows *emeasure* (*density M f*) *A* = (∫⁺ *x*. *f x* ∗ *indicator A x* ∂*M*)
(**is** *_ = ?μ A*)
unfolding *density_def*
proof (*rule emeasure_measure_of_sigma*)
show *sigma_algebra* (*space M*) (*sets M*) **..**
show *positive* (*sets M*) *?μ*
using *f* **by** (*auto simp*: *positive_def*)
show *countably_additive* (*sets M*) *?μ*
proof (*intro countably_additiveI*)
fix *A* :: *nat* ⇒ *'a set* **assume** *range A* ⊆ *sets M*
then have ⋀*i*. *A i* ∈ *sets M* **by** *auto*
then have ∗: ⋀*i*. (λ*x*. *f x* ∗ *indicator* (*A i*) *x*) ∈ *borel_measurable M*
by *auto*
assume *disj*: *disjoint_family A*
then have (∑ *n*. *?μ* (*A n*)) = (∫⁺ *x*. (∑ *n*. *f x* ∗ *indicator* (*A n*) *x*) ∂*M*)
using *f* ∗ **by** (*subst nn_integral_suminf*) *auto*
also have (∫⁺ *x*. (∑ *n*. *f x* ∗ *indicator* (*A n*) *x*) ∂*M*) = (∫⁺ *x*. *f x* ∗ (∑ *n*. *indicator* (*A n*) *x*) ∂*M*)
using *f* **by** (*auto intro*!: *ennreal_suminf_cmult nn_integral_cong_AE*)
also have ... = (∫⁺ *x*. *f x* ∗ *indicator* (⋃ *n*. *A n*) *x* ∂*M*)
unfolding *suminf_indicator*[*OF disj*] **..**
finally show (∑ *i*. ∫⁺ *x*. *f x* ∗ *indicator* (*A i*) *x* ∂*M*) = ∫⁺ *x*. *f x* ∗ *indicator* (⋃ *i*. *A i*) *x* ∂*M* **.**
qed
qed *fact*

lemma *null_sets_density_iff*:
assumes *f*: *f* ∈ *borel_measurable M*
shows *A* ∈ *null_sets* (*density M f*) ⟷ *A* ∈ *sets M* ∧ (*AE x in M*. *x* ∈ *A* ⟶ *f x* = *0*)
proof −
{ assume *A* ∈ *sets M*
have (∫⁺*x*. *f x* ∗ *indicator A x* ∂*M*) = *0* ⟷ *emeasure M* {*x* ∈ *space M*. *f x* ∗ *indicator A x* ≠ *0*} = *0*
using *f* ‹*A* ∈ *sets M*› **by** (*intro nn_integral_0_iff*) *auto*
also have ... ⟷ (*AE x in M*. *f x* ∗ *indicator A x* = *0*)
using *f* ‹*A* ∈ *sets M*› **by** (*intro AE_iff_measurable*[*OF _ refl*, *symmetric*]) *auto*
also have (*AE x in M*. *f x* ∗ *indicator A x* = *0*) ⟷ (*AE x in M*. *x* ∈ *A* ⟶ *f x* ≤ *0*)
by (*auto simp add*: *indicator_def max_def split*: *if_split_asm*)
finally have (∫⁺*x*. *f x* ∗ *indicator A x* ∂*M*) = *0* ⟷ (*AE x in M*. *x* ∈ *A* ⟶ *f x* ≤ *0*) **. }**

with *f* **show** *?thesis*
by (*simp add*: *null_sets_def emeasure_density cong*: *conj_cong*)
qed

lemma *AE_density*:
assumes *f*: $f \in$ *borel_measurable M*
shows (*AE x in density M f*. *P x*) $\longleftrightarrow$ (*AE x in M*. $0 < f\ x \longrightarrow$ *P x*)
proof
assume *AE x in density M f*. *P x*
with *f* **obtain** *N* **where** $\{x \in$ *space M*. $\neg$ *P x*$\} \subseteq N\ N \in$ *sets M* **and** *ae*: *AE x in M*. $x \in N \longrightarrow f\ x = 0$
by (*auto simp*: *eventually_ae_filter null_sets_density_iff*)
then have *AE x in M*. $x \notin N \longrightarrow$ *P x* **by** *auto*
with *ae* **show** *AE x in M*. $0 < f\ x \longrightarrow$ *P x*
by (*rule eventually_elim2*) *auto*
next
fix *N* **assume** *ae*: *AE x in M*. $0 < f\ x \longrightarrow$ *P x*
then obtain *N* **where** $\{x \in$ *space M*. $\neg$ $(0 < f\ x \longrightarrow$ *P x*$)\} \subseteq N\ N \in$ *null_sets M*
by (*auto simp*: *eventually_ae_filter*)
then have $*$: $\{x \in$ *space* (*density M f*). $\neg$ *P x*$\} \subseteq N \cup \{x \in$*space M*. $f\ x = 0\}$ $N \cup \{x \in$*space M*. $f\ x = 0\} \in$ *sets M* **and** *ae2*: *AE x in M*. $x \notin N$
using *f* **by** (*auto simp*: *subset_eq zero_less_iff_neq_zero intro*!: *AE_not_in*)
show *AE x in density M f*. *P x*
using *ae2*
unfolding *eventually_ae_filter*[*of* _ *density M f*] *Bex_def null_sets_density_iff*[*OF f*]
by (*intro exI*[*of* _ $N \cup \{x \in$*space M*. $f\ x = 0\}$] *conjI* $*$) (*auto elim*: *eventually_elim2*)
qed

lemma *nn_integral_density*:
assumes *f*: $f \in$ *borel_measurable M*
assumes *g*: $g \in$ *borel_measurable M*
shows $integral^N$ (*density M f*) $g = (\int^+ x.\ f\ x * g\ x\ \partial M)$
using *g* **proof** *induct*
case (*cong u v*)
then show *?case*
apply (*subst nn_integral_cong*[*OF cong*(*3*)])
apply (*simp_all cong*: *nn_integral_cong*)
done
next
case (*set A*) **then show** *?case*
by (*simp add*: *emeasure_density f*)
next
case (*mult u c*)
moreover have $\bigwedge x.\ f\ x * (c * u\ x) = c * (f\ x * u\ x)$ **by** (*simp add*: *field_simps*)
ultimately show *?case*
using *f* **by** (*simp add*: *nn_integral_cmult*)

next
case (*add u v*)
then have $\bigwedge x.\ f\ x * (v\ x + u\ x) = f\ x * v\ x + f\ x * u\ x$
by (*simp add*: *distrib_left*)
with *add f* **show** *?case*
by (*auto simp add*: *nn_integral_add intro*!: *nn_integral_add*[*symmetric*])
next
case (*seq U*)
have *eq*: *AE x in M*. $f\ x * (SUP\ i.\ U\ i\ x) = (SUP\ i.\ f\ x * U\ i\ x)$
by *eventually_elim* (*simp add*: *SUP_mult_left_ennreal seq*)
from *seq f* **show** *?case*
apply (*simp add*: *nn_integral_monotone_convergence_SUP image_comp*)
apply (*subst nn_integral_cong_AE*[*OF eq*])
apply (*subst nn_integral_monotone_convergence_SUP_AE*)
apply (*auto simp*: *incseq_def le_fun_def intro*!: *mult_left_mono*)
done
qed

lemma *density_distr*:
assumes [*measurable*]: $f \in borel\_measurable\ N\ X \in measurable\ M\ N$
shows $density\ (distr\ M\ N\ X)\ f = distr\ (density\ M\ (\lambda x.\ f\ (X\ x)))\ N\ X$
by (*intro measure_eqI*)
(*auto simp add*: *emeasure_density nn_integral_distr emeasure_distr*
split: *split_indicator intro*!: *nn_integral_cong*)

lemma *emeasure_restricted*:
assumes *S*: $S \in sets\ M$ **and** *X*: $X \in sets\ M$
shows $emeasure\ (density\ M\ (indicator\ S))\ X = emeasure\ M\ (S \cap X)$
proof −
have $emeasure\ (density\ M\ (indicator\ S))\ X = (\int^+ x.\ indicator\ S\ x * indicator\ X\ x\ \partial M)$
using *S X* **by** (*simp add*: *emeasure_density*)
also have $\ldots = (\int^+ x.\ indicator\ (S \cap X)\ x\ \partial M)$
by (*auto intro*!: *nn_integral_cong simp*: *indicator_def*)
also have $\ldots = emeasure\ M\ (S \cap X)$
using *S X* **by** (*simp add*: *sets.Int*)
finally show *?thesis* **.**
qed

lemma *measure_restricted*:
$S \in sets\ M \Longrightarrow X \in sets\ M \Longrightarrow measure\ (density\ M\ (indicator\ S))\ X = measure\ M\ (S \cap X)$
by (*simp add*: *emeasure_restricted measure_def*)

lemma (**in** *finite_measure*) *finite_measure_restricted*:
$S \in sets\ M \Longrightarrow finite\_measure\ (density\ M\ (indicator\ S))$
by *standard* (*simp add*: *emeasure_restricted*)

lemma *emeasure_density_const*:

$A \in sets\ M \Longrightarrow emeasure\ (density\ M\ (\lambda\_.\ c))\ A = c * emeasure\ M\ A$
by (*auto simp*: *nn_integral_cmult_indicator emeasure_density*)

lemma *measure_density_const*:
$A \in sets\ M \Longrightarrow c \neq \infty \Longrightarrow measure\ (density\ M\ (\lambda\_.\ c))\ A = enn2real\ c * measure\ M\ A$
by (*auto simp*: *emeasure_density_const measure_def enn2real_mult*)

lemma *density_density_eq*:
$f \in borel\_measurable\ M \Longrightarrow g \in borel\_measurable\ M \Longrightarrow$
$density\ (density\ M\ f)\ g = density\ M\ (\lambda x.\ f\ x * g\ x)$
by (*auto intro*!: *measure_eqI simp*: *emeasure_density nn_integral_density ac_simps*)

lemma *distr_density_distr*:
assumes T: $T \in measurable\ M\ M'$ **and** T': $T' \in measurable\ M'\ M$
and *inv*: $\forall x \in space\ M.\ T'\ (T\ x) = x$
assumes f: $f \in borel\_measurable\ M'$
shows $distr\ (density\ (distr\ M\ M'\ T)\ f)\ M\ T' = density\ M\ (f \circ T)$ (**is** *?R* = *?L*)
proof (*rule measure_eqI*)
fix A **assume** A: $A \in sets\ ?R$
{ fix x **assume** $x \in space\ M$
with *sets.sets_into_space*[*OF A*]
have $indicator\ (T' -` A \cap space\ M')\ (T\ x) = (indicator\ A\ x :: ennreal)$
using *T inv* **by** (*auto simp*: *indicator_def measurable_space*) **}**
with *A T T' f* **show** *emeasure ?R A* = *emeasure ?L A*
by (*simp add*: *measurable_comp emeasure_density emeasure_distr*
nn_integral_distr measurable_sets cong: *nn_integral_cong*)
qed *simp*

lemma *density_density_divide*:
fixes $f\ g :: 'a \Rightarrow real$
assumes f: $f \in borel\_measurable\ M\ AE\ x\ in\ M.\ 0 \leq f\ x$
assumes g: $g \in borel\_measurable\ M\ AE\ x\ in\ M.\ 0 \leq g\ x$
assumes *ac*: $AE\ x\ in\ M.\ f\ x = 0 \longrightarrow g\ x = 0$
shows $density\ (density\ M\ f)\ (\lambda x.\ g\ x\ /\ f\ x) = density\ M\ g$
proof −
have $density\ M\ g = density\ M\ (\lambda x.\ ennreal\ (f\ x) * ennreal\ (g\ x\ /\ f\ x))$
using *f g ac* **by** (*auto intro*!: *density_cong measurable_If simp*: *ennreal_mult*[*symmetric*])
then show *?thesis*
using *f g* **by** (*subst density_density_eq*) *auto*
qed

lemma *density_1*: $density\ M\ (\lambda\_.\ 1) = M$
by (*intro measure_eqI*) (*auto simp*: *emeasure_density*)

lemma *emeasure_density_add*:
assumes X: $X \in sets\ M$
assumes *Mf*[*measurable*]: $f \in borel\_measurable\ M$

assumes *Mg*[*measurable*]: $g \in$ *borel_measurable M*
shows *emeasure* (*density M f*) *X* + *emeasure* (*density M g*) *X* =
emeasure (*density M* (λ*x*. *f x* + *g x*)) *X*
using *assms*
apply (*subst* (*1 2 3*) *emeasure_density*, *simp_all*) []
apply (*subst nn_integral_add*[*symmetric*], *simp_all*) []
apply (*intro nn_integral_cong*, *simp split*: *split_indicator*)
done

Point measure

definition *point_measure* :: *'a set* ⇒ (*'a* ⇒ *ennreal*) ⇒ *'a measure* **where**
point_measure A f = *density* (*count_space A*) *f*

lemma
shows *space_point_measure*: *space* (*point_measure A f*) = *A*
and *sets_point_measure*: *sets* (*point_measure A f*) = *Pow A*
by (*auto simp*: *point_measure_def*)

lemma *sets_point_measure_count_space*[*measurable_cong*]: *sets* (*point_measure A f*) = *sets* (*count_space A*)
by (*simp add*: *sets_point_measure*)

lemma *measurable_point_measure_eq1*[*simp*]:
g ∈ *measurable* (*point_measure A f*) *M* ⟷ *g* ∈ *A* → *space M*
unfolding *point_measure_def* **by** *simp*

lemma *measurable_point_measure_eq2_finite*[*simp*]:
finite A ⟹
g ∈ *measurable M* (*point_measure A f*) ⟷
(*g* ∈ *space M* → *A* ∧ (∀ *a*∈*A*. *g* −‘ {*a*} ∩ *space M* ∈ *sets M*))
unfolding *point_measure_def* **by** (*simp add*: *measurable_count_space_eq2*)

lemma *simple_function_point_measure*[*simp*]:
simple_function (*point_measure A f*) *g* ⟷ *finite* (*g* ‘ *A*)
by (*simp add*: *point_measure_def*)

lemma *emeasure_point_measure*:
assumes *A*: *finite* {*a*∈*X*. 0 < *f a*} *X* ⊆ *A*
shows *emeasure* (*point_measure A f*) *X* = (∑ *a*|*a*∈*X* ∧ 0 < *f a*. *f a*)
proof −
have {*a*. (*a* ∈ *X* ⟶ *a* ∈ *A* ∧ 0 < *f a*) ∧ *a* ∈ *X*} = {*a*∈*X*. 0 < *f a*}
using ‹*X* ⊆ *A*› **by** *auto*
with *A* **show** *?thesis*
by (*simp add*: *emeasure_density nn_integral_count_space point_measure_def indicator_def of_bool_def*)
qed

lemma *emeasure_point_measure_finite*:

finite A ⟹ X ⊆ A ⟹ emeasure (point_measure A f) X = (∑ a∈X. f a)
by (*subst emeasure_point_measure*) (*auto dest*: *finite_subset intro*!: *sum.mono_neutral_left simp*: *less_le*)

lemma *emeasure_point_measure_finite2*:
X ⊆ A ⟹ finite X ⟹ emeasure (point_measure A f) X = (∑ a∈X. f a)
by (*subst emeasure_point_measure*)
(*auto dest*: *finite_subset intro*!: *sum.mono_neutral_left simp*: *less_le*)

lemma *null_sets_point_measure_iff*:
X ∈ null_sets (point_measure A f) ⟷ X ⊆ A ∧ (∀ x∈X. f x = 0)
by (*auto simp*: *AE_count_space null_sets_density_iff point_measure_def*)

lemma *AE_point_measure*:
(AE x in point_measure A f. P x) ⟷ (∀ x∈A. 0 < f x ⟶ P x)
unfolding *point_measure_def*
by (*subst AE_density*) (*auto simp*: *AE_density AE_count_space point_measure_def*)

lemma *nn_integral_point_measure*:
finite {a∈A. 0 < f a ∧ 0 < g a} ⟹
integral^N (point_measure A f) g = (∑ a|a∈A ∧ 0 < f a ∧ 0 < g a. f a ∗ g a)
unfolding *point_measure_def*
by (*subst nn_integral_density*)
(*simp_all add*: *nn_integral_density nn_integral_count_space ennreal_zero_less_mult_iff*)

lemma *nn_integral_point_measure_finite*:
finite A ⟹ integral^N (point_measure A f) g = (∑ a∈A. f a ∗ g a)
by (*subst nn_integral_point_measure*) (*auto intro*!: *sum.mono_neutral_left simp*: *less_le*)

Uniform measure

definition *uniform_measure M A = density M (λx. indicator A x / emeasure M A)*

lemma
shows *sets_uniform_measure*[*simp*, *measurable_cong*]: *sets (uniform_measure M A) = sets M*
and *space_uniform_measure*[*simp*]: *space (uniform_measure M A) = space M*
by (*auto simp*: *uniform_measure_def*)

lemma *emeasure_uniform_measure*[*simp*]:
assumes *A*: *A ∈ sets M* **and** *B*: *B ∈ sets M*
shows *emeasure (uniform_measure M A) B = emeasure M (A ∩ B) / emeasure M A*
proof −
from *A B* **have** *emeasure (uniform_measure M A) B = (∫⁺x. (1 / emeasure M A) ∗ indicator (A ∩ B) x ∂M)*
by (*auto simp add*: *uniform_measure_def emeasure_density divide_ennreal_def*

split: *split_indicator*
intro!: *nn_integral_cong*)
also have ... = *emeasure M* (*A* ∩ *B*) / *emeasure M A*
using *A B*
by (*subst nn_integral_cmult_indicator*) (*simp_all add*: *sets.Int divide_ennreal_def mult.commute*)
finally show *?thesis* **.**
qed

lemma *measure_uniform_measure*[*simp*]:
assumes *A*: *emeasure M A* ≠ *0 emeasure M A* ≠ ∞ **and** *B*: *B* ∈ *sets M*
shows *measure* (*uniform_measure M A*) *B* = *measure M* (*A* ∩ *B*) / *measure M A*
using *emeasure_uniform_measure*[*OF emeasure_neq_0_sets*[*OF A*(*1*)] *B*] *A*
by (*cases emeasure M A emeasure M* (*A* ∩ *B*) *rule*: *ennreal2_cases*)
(*simp_all add*: *measure_def divide_ennreal top_ennreal.rep_eq top_ereal_def ennreal_top_divide*)

lemma *AE_uniform_measureI*:
A ∈ *sets M* ⟹ (*AE x in M*. *x* ∈ *A* ⟶ *P x*) ⟹ (*AE x in uniform_measure M A*. *P x*)
unfolding *uniform_measure_def* **by** (*auto simp*: *AE_density divide_ennreal_def*)

lemma *emeasure_uniform_measure_1*:
emeasure M S ≠ *0* ⟹ *emeasure M S* ≠ ∞ ⟹ *emeasure* (*uniform_measure M S*) *S* = *1*
by (*subst emeasure_uniform_measure*)
(*simp_all add*: *emeasure_neq_0_sets emeasure_eq_ennreal_measure divide_ennreal zero_less_iff_neq_zero*[*symmetric*])

lemma *nn_integral_uniform_measure*:
assumes *f*[*measurable*]: *f* ∈ *borel_measurable M* **and** *S*[*measurable*]: *S* ∈ *sets M*
shows (∫⁺*x*. *f x* ∂*uniform_measure M S*) = (∫⁺*x*. *f x* ∗ *indicator S x* ∂*M*) / *emeasure M S*
proof −
{ assume *emeasure M S* = ∞
then have *?thesis*
by (*simp add*: *uniform_measure_def nn_integral_density f*) **}**
moreover
{ assume [*simp*]: *emeasure M S* = *0*
then have *ae*: *AE x in M*. *x* ∉ *S*
using *sets.sets_into_space*[*OF S*]
by (*subst AE_iff_measurable*[*OF _ refl*]) (*simp_all add*: *subset_eq cong*: *rev_conj_cong*)
from *ae* **have** (∫⁺ *x*. *indicator S x* / *0* ∗ *f x* ∂*M*) = *0*
by (*subst nn_integral_0_iff_AE*) *auto*
moreover from *ae* **have** (∫⁺ *x*. *f x* ∗ *indicator S x* ∂*M*) = *0*
by (*subst nn_integral_0_iff_AE*) *auto*
ultimately have *?thesis*

```
    by (simp add: uniform_measure_def nn_integral_density f) }
  moreover have emeasure M S ≠ 0 ⟹ emeasure M S ≠ ∞ ⟹ ?thesis
    unfolding uniform_measure_def
    by (subst nn_integral_density)
     (auto simp: ennreal_times_divide f nn_integral_divide[symmetric] mult.commute)
  ultimately show ?thesis by blast
qed

lemma AE_uniform_measure:
  assumes emeasure M A ≠ 0 emeasure M A < ∞
  shows (AE x in uniform_measure M A. P x) ⟷ (AE x in M. x ∈ A ⟶ P x)
proof −
  have A ∈ sets M
    using ‹emeasure M A ≠ 0› by (metis emeasure_notin_sets)
  moreover have ⋀x. 0 < indicator A x / emeasure M A ⟷ x ∈ A
    using assms
   by (cases emeasure M A) (auto split: split_indicator simp: ennreal_zero_less_divide)
  ultimately show ?thesis
    unfolding uniform_measure_def by (simp add: AE_density)
qed
```

Null measure

```
lemma null_measure_eq_density: null_measure M = density M (λ_. 0)
  by (intro measure_eqI) (simp_all add: emeasure_density)

lemma nn_integral_null_measure[simp]: (∫⁺x. f x ∂null_measure M) = 0
  by (auto simp add: nn_integral_def simple_integral_def SUP_constant bot_ennreal_def
le_fun_def
          intro!: exI[of _ λx. 0])

lemma density_null_measure[simp]: density (null_measure M) f = null_measure
M
proof (intro measure_eqI)
  fix A show emeasure (density (null_measure M) f) A = emeasure (null_measure
M) A
    by (simp add: density_def) (simp only: null_measure_def[symmetric] emea-
sure_null_measure)
qed simp
```

Uniform count measure

```
definition uniform_count_measure A = point_measure A (λx. 1 / card A)

lemma
  shows space_uniform_count_measure: space (uniform_count_measure A) = A
    and sets_uniform_count_measure: sets (uniform_count_measure A) = Pow
A
   unfolding uniform_count_measure_def by (auto simp: space_point_measure
sets_point_measure)
```

lemma *sets_uniform_count_measure_count_space*[*measurable_cong*]:
sets (*uniform_count_measure A*) = *sets* (*count_space A*)
by (*simp add*: *sets_uniform_count_measure*)

lemma *emeasure_uniform_count_measure*:
finite A ⟹ *X* ⊆ *A* ⟹ *emeasure* (*uniform_count_measure A*) *X* = *card X* / *card A*
by (*simp add*: *emeasure_point_measure_finite uniform_count_measure_def divide_inverse ennreal_mult*
ennreal_of_nat_eq_real_of_nat)

lemma *measure_uniform_count_measure*:
finite A ⟹ *X* ⊆ *A* ⟹ *measure* (*uniform_count_measure A*) *X* = *card X* / *card A*
by (*simp add*: *emeasure_point_measure_finite uniform_count_measure_def measure_def enn2real_mult*)

lemma *space_uniform_count_measure_empty_iff* [*simp*]:
space (*uniform_count_measure X*) = {} ⟷ *X* = {}
by(*simp add*: *space_uniform_count_measure*)

lemma *sets_uniform_count_measure_eq_UNIV* [*simp*]:
sets (*uniform_count_measure UNIV*) = *UNIV* ⟷ *True*
UNIV = *sets* (*uniform_count_measure UNIV*) ⟷ *True*
by(*simp_all add*: *sets_uniform_count_measure*)

Scaled measure

lemma *nn_integral_scale_measure*:
assumes *f*: *f* ∈ *borel_measurable M*
shows *nn_integral* (*scale_measure r M*) *f* = *r* ∗ *nn_integral M f*
using *f*
proof *induction*
case (*cong f g*)
thus *?case*
by(*simp add*: *cong.hyps space_scale_measure cong*: *nn_integral_cong_simp*)
next
case (*mult f c*)
thus *?case*
by(*simp add*: *nn_integral_cmult max_def mult.assoc mult.left_commute*)
next
case (*add f g*)
thus *?case*
by(*simp add*: *nn_integral_add distrib_left*)
next
case (*seq U*)
thus *?case*
by(*simp add*: *nn_integral_monotone_convergence_SUP SUP_mult_left_ennreal*

image_comp)
qed *simp*

end

6.6 Binary Product Measure

theory *Binary_Product_Measure*
imports *Nonnegative_Lebesgue_Integration*
begin

lemma *Pair_vimage_times*[*simp*]: *Pair x* $-`$ $(A \times B) = (\mathit{if}\ x \in A\ \mathit{then}\ B\ \mathit{else}\ \{\})$
by *auto*

lemma *rev_Pair_vimage_times*[*simp*]: $(\lambda x.\ (x,\ y))$ $-`$ $(A \times B) = (\mathit{if}\ y \in B\ \mathit{then}\ A\ \mathit{else}\ \{\})$
by *auto*

6.6.1 Binary products

definition *pair_measure* (**infixr** $\bigotimes_M$ *80*) **where**
$A \bigotimes_M B = \mathit{measure\_of}\ (\mathit{space}\ A \times \mathit{space}\ B)$
$\{a \times b \mid a\ b.\ a \in \mathit{sets}\ A \wedge b \in \mathit{sets}\ B\}$
$(\lambda X.\ \int^+ x.\ (\int^+ y.\ \mathit{indicator}\ X\ (x,y)\ \partial B)\ \partial A)$

lemma *pair_measure_closed*: $\{a \times b \mid a\ b.\ a \in \mathit{sets}\ A \wedge b \in \mathit{sets}\ B\} \subseteq \mathit{Pow}\ (\mathit{space}\ A \times \mathit{space}\ B)$
using *sets.space_closed*[*of A*] *sets.space_closed*[*of B*] **by** *auto*

lemma *space_pair_measure*:
space $(A \bigotimes_M B) = \mathit{space}\ A \times \mathit{space}\ B$
unfolding *pair_measure_def* **using** *pair_measure_closed*[*of A B*]
by (*rule space_measure_of*)

lemma *SIGMA_Collect_eq*: $(\mathit{SIGMA}\ x{:}\mathit{space}\ M.\ \{y{\in}\mathit{space}\ N.\ P\ x\ y\}) = \{x{\in}\mathit{space}\ (M \bigotimes_M N).\ P\ (\mathit{fst}\ x)\ (\mathit{snd}\ x)\}$
by (*auto simp*: *space_pair_measure*)

lemma *sets_pair_measure*:
sets $(A \bigotimes_M B) = \mathit{sigma\_sets}\ (\mathit{space}\ A \times \mathit{space}\ B)\ \{a \times b \mid a\ b.\ a \in \mathit{sets}\ A \wedge b \in \mathit{sets}\ B\}$
unfolding *pair_measure_def* **using** *pair_measure_closed*[*of A B*]
by (*rule sets_measure_of*)

lemma *sets_pair_measure_cong*[*measurable_cong*, *cong*]:
sets M1 $=$ *sets M1'* $\Longrightarrow$ *sets M2* $=$ *sets M2'* $\Longrightarrow$ *sets* $(M1 \bigotimes_M M2) = \mathit{sets}\ (M1' \bigotimes_M M2')$
unfolding *sets_pair_measure* **by** (*simp cong*: *sets_eq_imp_space_eq*)

lemma *pair_measureI*[*intro*, *simp*, *measurable*]:
  $x \in sets\ A \Longrightarrow y \in sets\ B \Longrightarrow x \times y \in sets\ (A \bigotimes_M B)$
  by (*auto simp*: *sets_pair_measure*)

lemma *sets_Pair*: $\{x\} \in sets\ M1 \Longrightarrow \{y\} \in sets\ M2 \Longrightarrow \{(x, y)\} \in sets\ (M1 \bigotimes_M M2)$
  using *pair_measureI*[*of* $\{x\}$ *M1* $\{y\}$ *M2*] **by** *simp*

lemma *measurable_pair_measureI*:
  assumes *1*: $f \in space\ M \to space\ M1 \times space\ M2$
  assumes *2*: $\bigwedge A\ B.\ A \in sets\ M1 \Longrightarrow B \in sets\ M2 \Longrightarrow f -` (A \times B) \cap space\ M \in sets\ M$
  shows $f \in measurable\ M\ (M1 \bigotimes_M M2)$
  unfolding *pair_measure_def* **using** *1 2*
  by (*intro measurable_measure_of*) (*auto dest*: *sets.sets_into_space*)

lemma *measurable_split_replace*[*measurable* (*raw*)]:
  $(\lambda x.\ f\ x\ (fst\ (g\ x))\ (snd\ (g\ x))) \in measurable\ M\ N \Longrightarrow (\lambda x.\ case\_prod\ (f\ x)\ (g\ x)) \in measurable\ M\ N$
  unfolding *split_beta'* **.**

lemma *measurable_Pair*[*measurable* (*raw*)]:
  assumes *f*: $f \in measurable\ M\ M1$ **and** *g*: $g \in measurable\ M\ M2$
  shows $(\lambda x.\ (f\ x,\ g\ x)) \in measurable\ M\ (M1 \bigotimes_M M2)$
proof (*rule measurable_pair_measureI*)
  show $(\lambda x.\ (f\ x,\ g\ x)) \in space\ M \to space\ M1 \times space\ M2$
    using *f g* **by** (*auto simp*: *measurable_def*)
  fix *A B* **assume** $*$: $A \in sets\ M1\ B \in sets\ M2$
  have $(\lambda x.\ (f\ x,\ g\ x)) -` (A \times B) \cap space\ M = (f -` A \cap space\ M) \cap (g -` B \cap space\ M)$
    by *auto*
  also have $\ldots \in sets\ M$
    by (*rule sets.Int*) (*auto intro*!: *measurable_sets* $*$ *f g*)
  finally show $(\lambda x.\ (f\ x,\ g\ x)) -` (A \times B) \cap space\ M \in sets\ M$ **.**
qed

lemma *measurable_fst*[*intro*!, *simp*, *measurable*]: $fst \in measurable\ (M1 \bigotimes_M M2)\ M1$
  by (*auto simp*: *fst_vimage_eq_Times space_pair_measure sets.sets_into_space Times_Int_Times*
    measurable_def)

lemma *measurable_snd*[*intro*!, *simp*, *measurable*]: $snd \in measurable\ (M1 \bigotimes_M M2)\ M2$
  by (*auto simp*: *snd_vimage_eq_Times space_pair_measure sets.sets_into_space Times_Int_Times*
    measurable_def)

lemma *measurable_Pair_compose_split*[*measurable_dest*]:

assumes f: *case_prod f* ∈ *measurable* (*M1* ⨂$_M$ *M2*) *N*
assumes g: *g* ∈ *measurable M M1* **and** h: *h* ∈ *measurable M M2*
shows (λ*x*. *f* (*g x*) (*h x*)) ∈ *measurable M N*
using *measurable_compose*[*OF measurable_Pair f*, *OF g h*] **by** *simp*

lemma *measurable_Pair1_compose*[*measurable_dest*]:
assumes *f*: (λ*x*. (*f x*, *g x*)) ∈ *measurable M* (*M1* ⨂$_M$ *M2*)
assumes [*measurable*]: *h* ∈ *measurable N M*
shows (λ*x*. *f* (*h x*)) ∈ *measurable N M1*
using *measurable_compose*[*OF f measurable_fst*] **by** *simp*

lemma *measurable_Pair2_compose*[*measurable_dest*]:
assumes *f*: (λ*x*. (*f x*, *g x*)) ∈ *measurable M* (*M1* ⨂$_M$ *M2*)
assumes [*measurable*]: *h* ∈ *measurable N M*
shows (λ*x*. *g* (*h x*)) ∈ *measurable N M2*
using *measurable_compose*[*OF f measurable_snd*] **by** *simp*

lemma *measurable_pair*:
assumes (*fst* ∘ *f*) ∈ *measurable M M1* (*snd* ∘ *f*) ∈ *measurable M M2*
shows *f* ∈ *measurable M* (*M1* ⨂$_M$ *M2*)
using *measurable_Pair*[*OF assms*] **by** *simp*

lemma
assumes *f*[*measurable*]: *f* ∈ *measurable M* (*N* ⨂$_M$ *P*)
shows *measurable_fst'*: (λ*x*. *fst* (*f x*)) ∈ *measurable M N*
and *measurable_snd'*: (λ*x*. *snd* (*f x*)) ∈ *measurable M P*
by *simp_all*

lemma
assumes *f*[*measurable*]: *f* ∈ *measurable M N*
shows *measurable_fst''*: (λ*x*. *f* (*fst x*)) ∈ *measurable* (*M* ⨂$_M$ *P*) *N*
and *measurable_snd''*: (λ*x*. *f* (*snd x*)) ∈ *measurable* (*P* ⨂$_M$ *M*) *N*
by *simp_all*

lemma *sets_pair_in_sets*:
assumes ⋀*a b*. *a* ∈ *sets A* ⟹ *b* ∈ *sets B* ⟹ *a* × *b* ∈ *sets N*
shows *sets* (*A* ⨂$_M$ *B*) ⊆ *sets N*
unfolding *sets_pair_measure*
by (*intro sets.sigma_sets_subset'*) (*auto intro*!: *assms*)

lemma *sets_pair_eq_sets_fst_snd*:
sets (*A* ⨂$_M$ *B*) = *sets* (*Sup* {*vimage_algebra* (*space A* × *space B*) *fst A*, *vimage_algebra* (*space A* × *space B*) *snd B*})
(**is** *?P* = *sets* (*Sup* {*?fst*, *?snd*}))
proof −
{ fix *a b* **assume** *ab*: *a* ∈ *sets A b* ∈ *sets B*
then have *a* × *b* = (*fst* −‘ *a* ∩ (*space A* × *space B*)) ∩ (*snd* −‘ *b* ∩ (*space A* × *space B*))
by (*auto dest*: *sets.sets_into_space*)

```
    also have ... ∈ sets (Sup {?fst, ?snd})
      apply (rule sets.Int)
      apply (rule in_sets_Sup)
      apply auto []
      apply (rule insertI1)
      apply (auto intro: ab in_vimage_algebra) []
      apply (rule in_sets_Sup)
      apply auto []
      apply (rule insertI2)
      apply (auto intro: ab in_vimage_algebra)
      done
    finally have a × b ∈ sets (Sup {?fst, ?snd}) . }
  moreover have sets ?fst ⊆ sets (A ⨂M B)
    by (rule sets_image_in_sets) (auto simp: space_pair_measure[symmetric])
  moreover have sets ?snd ⊆ sets (A ⨂M B)
    by (rule sets_image_in_sets) (auto simp: space_pair_measure)
  ultimately show ?thesis
    apply (intro antisym[of sets A for A] sets_Sup_in_sets sets_pair_in_sets)
    apply simp
    apply simp
    apply simp
    apply (elim disjE)
    apply (simp add: space_pair_measure)
    apply (simp add: space_pair_measure)
    apply (auto simp add: space_pair_measure)
    done
qed

lemma measurable_pair_iff:
  f ∈ measurable M (M1 ⨂M M2) ⟷ (fst ∘ f) ∈ measurable M M1 ∧ (snd ∘
f) ∈ measurable M M2
  by (auto intro: measurable_pair[of f M M1 M2])

lemma measurable_split_conv:
  (λ(x, y). f x y) ∈ measurable A B ⟷ (λx. f (fst x) (snd x)) ∈ measurable A B
  by (intro arg_cong2[where f=(∈)]) auto

lemma measurable_pair_swap': (λ(x,y). (y, x)) ∈ measurable (M1 ⨂M M2) (M2
⨂M M1)
  by (auto intro!: measurable_Pair simp: measurable_split_conv)

lemma measurable_pair_swap:
  assumes f: f ∈ measurable (M1 ⨂M M2) M shows (λ(x,y). f (y, x)) ∈ mea-
surable (M2 ⨂M M1) M
  using measurable_comp[OF measurable_Pair f] by (auto simp: measurable_split_conv
comp_def)

lemma measurable_pair_swap_iff:
  f ∈ measurable (M2 ⨂M M1) M ⟷ (λ(x,y). f (y,x)) ∈ measurable (M1 ⨂M
```

M2) *M*
by (*auto dest*: *measurable_pair_swap*)

lemma *measurable_Pair1'*: $x \in$ *space M1* $\Longrightarrow$ *Pair x* $\in$ *measurable M2* (*M1* $\bigotimes_M$ *M2*)
by *simp*

lemma *sets_Pair1*[*measurable* (*raw*)]:
assumes *A*: $A \in$ *sets* (*M1* $\bigotimes_M$ *M2*) **shows** *Pair x* $-`$ $A \in$ *sets M2*
proof −
have *Pair x* $-`$ $A =$ (*if x* $\in$ *space M1 then Pair x* $-`$ $A \cap$ *space M2 else* {})
using *A*[*THEN sets.sets_into_space*] **by** (*auto simp*: *space_pair_measure*)
also have ... $\in$ *sets M2*
using *A* **by** (*auto simp add*: *measurable_Pair1' intro*!: *measurable_sets split*: *if_split_asm*)
finally show *?thesis* **.**
qed

lemma *measurable_Pair2'*: $y \in$ *space M2* $\Longrightarrow$ $(\lambda x.\ (x,\ y)) \in$ *measurable M1* (*M1* $\bigotimes_M$ *M2*)
by (*auto intro*!: *measurable_Pair*)

lemma *sets_Pair2*: **assumes** *A*: $A \in$ *sets* (*M1* $\bigotimes_M$ *M2*) **shows** $(\lambda x.\ (x,\ y))$ $-`$ $A \in$ *sets M1*
proof −
have $(\lambda x.\ (x,\ y))$ $-`$ $A =$ (*if y* $\in$ *space M2 then* $(\lambda x.\ (x,\ y))$ $-`$ $A \cap$ *space M1 else* {})
using *A*[*THEN sets.sets_into_space*] **by** (*auto simp*: *space_pair_measure*)
also have ... $\in$ *sets M1*
using *A* **by** (*auto simp add*: *measurable_Pair2' intro*!: *measurable_sets split*: *if_split_asm*)
finally show *?thesis* **.**
qed

lemma *measurable_Pair2*:
assumes *f*: $f \in$ *measurable* (*M1* $\bigotimes_M$ *M2*) *M* **and** *x*: $x \in$ *space M1*
shows $(\lambda y.\ f\ (x,\ y)) \in$ *measurable M2 M*
using *measurable_comp*[*OF measurable_Pair1' f*, *OF x*]
by (*simp add*: *comp_def*)

lemma *measurable_Pair1*:
assumes *f*: $f \in$ *measurable* (*M1* $\bigotimes_M$ *M2*) *M* **and** *y*: $y \in$ *space M2*
shows $(\lambda x.\ f\ (x,\ y)) \in$ *measurable M1 M*
using *measurable_comp*[*OF measurable_Pair2' f*, *OF y*]
by (*simp add*: *comp_def*)

lemma *Int_stable_pair_measure_generator*: *Int_stable* $\{a \times b \mid a\ b.\ a \in$ *sets A* $\wedge\ b \in$ *sets B*$\}$
unfolding *Int_stable_def*

by *safe* (*auto simp add*: *Times_Int_Times*)

lemma (**in** *finite_measure*) *finite_measure_cut_measurable*:
assumes [*measurable*]: $Q \in sets\ (N \bigotimes_M M)$
shows $(\lambda x.\ emeasure\ M\ (Pair\ x -`\ Q)) \in borel\_measurable\ N$
(**is** *?s Q* $\in$ _)
using *Int_stable_pair_measure_generator pair_measure_closed assms*
unfolding *sets_pair_measure*
proof (*induct rule*: *sigma_sets_induct_disjoint*)
case (*compl A*)
with *sets.sets_into_space* **have** $\bigwedge x.\ emeasure\ M\ (Pair\ x -`\ ((space\ N \times space\ M) - A)) =$
$(if\ x \in space\ N\ then\ emeasure\ M\ (space\ M) - ?s\ A\ x\ else\ 0)$
unfolding *sets_pair_measure*[*symmetric*]
by (*auto intro*!: *emeasure_compl simp*: *vimage_Diff sets_Pair1*)
with *compl sets.top* **show** *?case*
by (*auto intro*!: *measurable_If simp*: *space_pair_measure*)
next
case (*union F*)
then have $\bigwedge x.\ emeasure\ M\ (Pair\ x -`\ (\bigcup i.\ F\ i)) = (\sum i.\ ?s\ (F\ i)\ x)$
by (*simp add*: *suminf_emeasure disjoint_family_on_vimageI subset_eq vimage_UN sets_pair_measure*[*symmetric*])
with *union* **show** *?case*
unfolding *sets_pair_measure*[*symmetric*] **by** *simp*
qed (*auto simp add*: *if_distrib Int_def*[*symmetric*] *intro*!: *measurable_If*)

lemma (**in** *sigma_finite_measure*) *measurable_emeasure_Pair*:
assumes *Q*: $Q \in sets\ (N \bigotimes_M M)$ **shows** $(\lambda x.\ emeasure\ M\ (Pair\ x -`\ Q)) \in borel\_measurable\ N$ (**is** *?s Q* $\in$ _)
proof −
obtain $F :: nat \Rightarrow 'a\ set$ **where** *F*:
$range\ F \subseteq sets\ M$
$\bigcup (range\ F) = space\ M$
$\bigwedge i.\ emeasure\ M\ (F\ i) \neq \infty$
$disjoint\_family\ F$ **by** (*blast intro*: *sigma_finite_disjoint*)
then have *F_sets*: $\bigwedge i.\ F\ i \in sets\ M$ **by** *auto*
let $?C = \lambda x\ i.\ F\ i \cap Pair\ x -`\ Q$
{ fix *i*
have [*simp*]: $space\ N \times F\ i \cap space\ N \times space\ M = space\ N \times F\ i$
using *F sets.sets_into_space* **by** *auto*
let $?R = density\ M\ (indicator\ (F\ i))$
have *finite_measure ?R*
using *F* **by** (*intro finite_measureI*) (*auto simp*: *emeasure_restricted subset_eq*)
then have $(\lambda x.\ emeasure\ ?R\ (Pair\ x -`\ (space\ N \times space\ ?R \cap Q))) \in borel\_measurable\ N$
by (*rule finite_measure.finite_measure_cut_measurable*) (*auto intro*: *Q*)
moreover have $\bigwedge x.\ emeasure\ ?R\ (Pair\ x -`\ (space\ N \times space\ ?R \cap Q))$
$= emeasure\ M\ (F\ i \cap Pair\ x -`\ (space\ N \times space\ ?R \cap Q))$

```
    using Q F_sets by (intro emeasure_restricted) (auto intro: sets_Pair1)
  moreover have ⋀x. F i ∩ Pair x -` (space N × space ?R ∩ Q) = ?C x i
    using sets.sets_into_space[OF Q] by (auto simp: space_pair_measure)
  ultimately have (λx. emeasure M (?C x i)) ∈ borel_measurable N
    by simp }
 moreover
 { fix x
  have (∑ i. emeasure M (?C x i)) = emeasure M (⋃ i. ?C x i)
  proof (intro suminf_emeasure)
    show range (?C x) ⊆ sets M
      using F ‹Q ∈ sets (N ⨂M M)› by (auto intro!: sets_Pair1)
    have disjoint_family F using F by auto
    show disjoint_family (?C x)
      by (rule disjoint_family_on_bisimulation[OF ‹disjoint_family F›]) auto
  qed
  also have (⋃ i. ?C x i) = Pair x -` Q
    using F sets.sets_into_space[OF ‹Q ∈ sets (N ⨂M M)›]
    by (auto simp: space_pair_measure)
  finally have emeasure M (Pair x -` Q) = (∑ i. emeasure M (?C x i))
    by simp }
 ultimately show ?thesis using ‹Q ∈ sets (N ⨂M M)› F_sets
  by auto
qed

lemma (in sigma_finite_measure) measurable_emeasure[measurable (raw)]:
 assumes space: ⋀x. x ∈ space N ⟹ A x ⊆ space M
 assumes A: {x∈space (N ⨂M M). snd x ∈ A (fst x)} ∈ sets (N ⨂M M)
 shows (λx. emeasure M (A x)) ∈ borel_measurable N
proof -
 from space have ⋀x. x ∈ space N ⟹ Pair x -` {x ∈ space (N ⨂M M). snd
x ∈ A (fst x)} = A x
   by (auto simp: space_pair_measure)
 with measurable_emeasure_Pair[OF A] show ?thesis
   by (auto cong: measurable_cong)
qed

lemma (in sigma_finite_measure) emeasure_pair_measure:
 assumes X ∈ sets (N ⨂M M)
 shows emeasure (N ⨂M M) X = (∫+ x. ∫+ y. indicator X (x, y) ∂M ∂N)
(is _ = ?μ X)
proof (rule emeasure_measure_of[OF pair_measure_def])
 show positive (sets (N ⨂M M)) ?μ
   by (auto simp: positive_def)
 have eq[simp]: ⋀A x y. indicator A (x, y) = indicator (Pair x -` A) y
   by (auto simp: indicator_def)
 show countably_additive (sets (N ⨂M M)) ?μ
 proof (rule countably_additiveI)
   fix F :: nat ⇒ ('b × 'a) set assume F: range F ⊆ sets (N ⨂M M) dis-
joint_family F
```

```
    from F have *: ⋀i. F i ∈ sets (N ⨂M M) by auto
    moreover have ⋀x. disjoint_family (λi. Pair x -` F i)
      by (intro disjoint_family_on_bisimulation[OF F(2)]) auto
    moreover have ⋀x. range (λi. Pair x -` F i) ⊆ sets M
      using F by (auto simp: sets_Pair1)
    ultimately show (∑ n. ?μ (F n)) = ?μ (⋃ i. F i)
    by (auto simp add: nn_integral_suminf[symmetric] vimage_UN suminf_emeasure
             intro!: nn_integral_cong nn_integral_indicator[symmetric])
  qed
  show {a × b |a b. a ∈ sets N ∧ b ∈ sets M} ⊆ Pow (space N × space M)
    using sets.space_closed[of N] sets.space_closed[of M] by auto
qed fact

lemma (in sigma_finite_measure) emeasure_pair_measure_alt:
  assumes X: X ∈ sets (N ⨂M M)
  shows emeasure (N ⨂M M) X = (∫+x. emeasure M (Pair x -` X) ∂N)
proof -
  have [simp]: ⋀x y. indicator X (x, y) = indicator (Pair x -` X) y
    by (auto simp: indicator_def)
  show ?thesis
     using X by (auto intro!: nn_integral_cong simp: emeasure_pair_measure
sets_Pair1)
qed

proposition (in sigma_finite_measure) emeasure_pair_measure_Times:
  assumes A: A ∈ sets N and B: B ∈ sets M
  shows emeasure (N ⨂M M) (A × B) = emeasure N A * emeasure M B
proof -
  have emeasure (N ⨂M M) (A × B) = (∫+x. emeasure M B * indicator A x
∂N)
   using A B by (auto intro!: nn_integral_cong simp: emeasure_pair_measure_alt)
  also have ... = emeasure M B * emeasure N A
    using A by (simp add: nn_integral_cmult_indicator)
  finally show ?thesis
    by (simp add: ac_simps)
qed

lemma (in sigma_finite_measure) times_in_null_sets1 [intro]:
  assumes A ∈ null_sets N B ∈ sets M
  shows   A × B ∈ null_sets (N ⨂M M)
  using assms by (simp add: null_sets_def emeasure_pair_measure_Times)

lemma (in sigma_finite_measure) times_in_null_sets2 [intro]:
  assumes A ∈ sets N B ∈ null_sets M
  shows   A × B ∈ null_sets (N ⨂M M)
  using assms by (simp add: null_sets_def emeasure_pair_measure_Times)
```

6.6.2 Binary products of σ-finite emeasure spaces

locale *pair_sigma_finite* = *M1?*: *sigma_finite_measure M1* + *M2?*: *sigma_finite_measure M2*
  for *M1* :: *'a measure* **and** *M2* :: *'b measure*

lemma (**in** *pair_sigma_finite*) *measurable_emeasure_Pair1*:
  $Q \in$ *sets* (*M1* $\bigotimes_M$ *M2*) $\Longrightarrow$ (λx. *emeasure M2* (*Pair x* −‘ *Q*)) ∈ *borel_measurable M1*
  using *M2.measurable_emeasure_Pair* .

lemma (**in** *pair_sigma_finite*) *measurable_emeasure_Pair2*:
  assumes *Q*: $Q \in$ *sets* (*M1* $\bigotimes_M$ *M2*) **shows** (λy. *emeasure M1* ((λx. (*x*, *y*)) −‘ *Q*)) ∈ *borel_measurable M2*
proof −
  have (λ(*x*, *y*). (*y*, *x*)) −‘ *Q* ∩ *space* (*M2* $\bigotimes_M$ *M1*) ∈ *sets* (*M2* $\bigotimes_M$ *M1*)
    using *Q measurable_pair_swap'* **by** (*auto intro*: *measurable_sets*)
  note *M1.measurable_emeasure_Pair*[*OF this*]
  moreover have $\bigwedge y$. *Pair y* −‘ ((λ(*x*, *y*). (*y*, *x*)) −‘ *Q* ∩ *space* (*M2* $\bigotimes_M$ *M1*)) = (λx. (*x*, *y*)) −‘ *Q*
    using *Q*[*THEN sets.sets_into_space*] **by** (*auto simp*: *space_pair_measure*)
  ultimately show *?thesis* **by** *simp*
qed

proposition (**in** *pair_sigma_finite*) *sigma_finite_up_in_pair_measure_generator*:
  defines *E* ≡ {*A* × *B* | *A B*. *A* ∈ *sets M1* ∧ *B* ∈ *sets M2*}
  shows ∃ *F*::*nat* ⇒ (*'a* × *'b*) *set*. *range F* ⊆ *E* ∧ *incseq F* ∧ ($\bigcup i$. *F i*) = *space M1* × *space M2* ∧
    (∀ *i*. *emeasure* (*M1* $\bigotimes_M$ *M2*) (*F i*) ≠ ∞)
proof −
  obtain *F1* **where** *F1*: *range F1* ⊆ *sets M1*
    $\bigcup$ (*range F1*) = *space M1*
    $\bigwedge i$. *emeasure M1* (*F1 i*) ≠ ∞
    incseq F1
    by (*rule M1.sigma_finite_incseq*) *blast*
  obtain *F2* **where** *F2*: *range F2* ⊆ *sets M2*
    $\bigcup$ (*range F2*) = *space M2*
    $\bigwedge i$. *emeasure M2* (*F2 i*) ≠ ∞
    incseq F2
    by (*rule M2.sigma_finite_incseq*) *blast*
  from *F1 F2* **have** *space*: *space M1* = ($\bigcup i$. *F1 i*) *space M2* = ($\bigcup i$. *F2 i*) **by** *auto*
  let *?F* = λi. *F1 i* × *F2 i*
  show *?thesis*
  proof (*intro exI*[*of* _ *?F*] *conjI allI*)
   show *range ?F* ⊆ *E* **using** *F1 F2* **by** (*auto simp*: *E_def*) (*metis range_subsetD*)
  next
    have *space M1* × *space M2* ⊆ ($\bigcup i$. *?F i*)
    proof (*intro subsetI*)
      fix *x* **assume** *x* ∈ *space M1* × *space M2*

```
      then obtain i j where fst x ∈ F1 i snd x ∈ F2 j
        by (auto simp: space)
      then have fst x ∈ F1 (max i j) snd x ∈ F2 (max j i)
        using ‹incseq F1› ‹incseq F2› unfolding incseq_def
        by (force split: split_max)+
      then have (fst x, snd x) ∈ F1 (max i j) × F2 (max i j)
        by (intro SigmaI) (auto simp add: max.commute)
      then show x ∈ (⋃ i. ?F i) by auto
    qed
    then show (⋃ i. ?F i) = space M1 × space M2
      using space by (auto simp: space)
  next
    fix i show incseq (λi. F1 i × F2 i)
      using ‹incseq F1› ‹incseq F2› unfolding incseq_Suc_iff by auto
  next
    fix i
    from F1 F2 have F1 i ∈ sets M1 F2 i ∈ sets M2 by auto
    with F1 F2 show emeasure (M1 ⨂M M2) (F1 i × F2 i) ≠ ∞
    by (auto simp add: emeasure_pair_measure_Times ennreal_mult_eq_top_iff)
  qed
qed

sublocale pair_sigma_finite ⊆ P?: sigma_finite_measure M1 ⨂M M2
proof
  obtain F1 :: 'a set set and F2 :: 'b set set where
      countable F1 ∧ F1 ⊆ sets M1 ∧ ⋃ F1 = space M1 ∧ (∀ a∈F1. emeasure M1
a ≠ ∞)
      countable F2 ∧ F2 ⊆ sets M2 ∧ ⋃ F2 = space M2 ∧ (∀ a∈F2. emeasure M2
a ≠ ∞)
    using M1.sigma_finite_countable M2.sigma_finite_countable by auto
  then show
    ∃ A. countable A ∧ A ⊆ sets (M1 ⨂M M2) ∧ ⋃ A = space (M1 ⨂M M2) ∧
(∀ a∈A. emeasure (M1 ⨂M M2) a ≠ ∞)
    by (intro exI[of _ (λ(a, b). a × b) ‘ (F1 × F2)] conjI)
     (auto simp: M2.emeasure_pair_measure_Times space_pair_measure set_eq_iff
subset_eq ennreal_mult_eq_top_iff)
qed

lemma sigma_finite_pair_measure:
  assumes A: sigma_finite_measure A and B: sigma_finite_measure B
  shows sigma_finite_measure (A ⨂M B)
proof −
  interpret A: sigma_finite_measure A by fact
  interpret B: sigma_finite_measure B by fact
  interpret AB: pair_sigma_finite A B ..
  show ?thesis ..
qed

lemma sets_pair_swap:
```

assumes $A \in sets\ (M1 \bigotimes_M M2)$
shows $(\lambda(x, y).\ (y, x)) -` A \cap space\ (M2 \bigotimes_M M1) \in sets\ (M2 \bigotimes_M M1)$
using *measurable_pair_swap'* *assms* **by** (*rule measurable_sets*)

lemma (**in** *pair_sigma_finite*) *distr_pair_swap*:
$M1 \bigotimes_M M2 = distr\ (M2 \bigotimes_M M1)\ (M1 \bigotimes_M M2)\ (\lambda(x, y).\ (y, x))$ (**is** *?P* = *?D*)
proof −
let *?E* = $\{a \times b \mid a\ b.\ a \in sets\ M1 \wedge b \in sets\ M2\}$
obtain $F :: nat \Rightarrow ('a \times 'b)\ set$ **where** F: $range\ F \subseteq$ *?E*
incseq $F \bigcup (range\ F) = space\ M1 \times space\ M2 \wedge i.\ emeasure\ (M1 \bigotimes_M M2)\ (F\ i) \neq \infty$
using *sigma_finite_up_in_pair_measure_generator* **by** *auto*
show *?thesis*
proof (*rule measure_eqI_generator_eq*[*OF Int_stable_pair_measure_generator*[*of M1 M2*]])
show *?E* $\subseteq Pow\ (space\ ?P)$
using *sets.space_closed*[*of M1*] *sets.space_closed*[*of M2*] **by** (*auto simp*: *space_pair_measure*)
show *sets ?P* = *sigma_sets* (*space ?P*) *?E*
by (*simp add*: *sets_pair_measure space_pair_measure*)
then show *sets ?D* = *sigma_sets* (*space ?P*) *?E*
by *simp*
from F **show** $range\ F \subseteq$ *?E* $(\bigcup i.\ F\ i) = space\ ?P \bigwedge i.\ emeasure\ ?P\ (F\ i) \neq \infty$
by (*auto simp*: *space_pair_measure*)
next
fix X **assume** $X \in$ *?E*
then obtain $A\ B$ **where** X[*simp*]: $X = A \times B$ **and** A: $A \in sets\ M1$ **and** B: $B \in sets\ M2$ **by** *auto*
have $(\lambda(y, x).\ (x, y)) -` X \cap space\ (M2 \bigotimes_M M1) = B \times A$
using *sets.sets_into_space*[*OF A*] *sets.sets_into_space*[*OF B*] **by** (*auto simp*: *space_pair_measure*)
with $A\ B$ **show** $emeasure\ (M1 \bigotimes_M M2)\ X = emeasure\ ?D\ X$
by (*simp add*: *M2.emeasure_pair_measure_Times M1.emeasure_pair_measure_Times emeasure_distr*
measurable_pair_swap' ac_simps)
qed
qed

lemma (**in** *pair_sigma_finite*) *emeasure_pair_measure_alt2*:
assumes A: $A \in sets\ (M1 \bigotimes_M M2)$
shows $emeasure\ (M1 \bigotimes_M M2)\ A = (\int^+ y.\ emeasure\ M1\ ((\lambda x.\ (x, y)) -` A)\ \partial M2)$
(**is** $\_ = ?\nu\ A$)
proof −
have [*simp*]: $\bigwedge y.\ (Pair\ y -` ((\lambda(x, y).\ (y, x)) -` A \cap space\ (M2 \bigotimes_M M1))) = (\lambda x.\ (x, y)) -` A$
using *sets.sets_into_space*[*OF A*] **by** (*auto simp*: *space_pair_measure*)

show *?thesis* **using** *A*
by (*subst distr_pair_swap*)
(*simp_all del*: *vimage_Int add*: *measurable_sets*[*OF measurable_pair_swap'*]
M1.emeasure_pair_measure_alt emeasure_distr[*OF measurable_pair_swap' A*])
qed

lemma (**in** *pair_sigma_finite*) *AE_pair*:
assumes *AE x in* (*M1* $\bigotimes_M$ *M2*). *Q x*
shows *AE x in M1*. (*AE y in M2*. *Q* (*x*, *y*))
proof −
obtain *N* **where** *N*: *N* ∈ *sets* (*M1* $\bigotimes_M$ *M2*) *emeasure* (*M1* $\bigotimes_M$ *M2*) *N* = *0*
{*x*∈*space* (*M1* $\bigotimes_M$ *M2*). ¬ *Q x*} ⊆ *N*
using *assms* **unfolding** *eventually_ae_filter* **by** *auto*
show *?thesis*
proof (*rule AE_I*)
from *N measurable_emeasure_Pair1*[*OF* ‹*N* ∈ *sets* (*M1* $\bigotimes_M$ *M2*)›]
show *emeasure M1* {*x*∈*space M1*. *emeasure M2* (*Pair x* −‘ *N*) ≠ *0*} = *0*
by (*auto simp*: *M2.emeasure_pair_measure_alt nn_integral_0_iff*)
show {*x* ∈ *space M1*. *emeasure M2* (*Pair x* −‘ *N*) ≠ *0*} ∈ *sets M1*
by (*intro borel_measurable_eq measurable_emeasure_Pair1 N sets.sets_Collect_neg N*) *simp*
{ **fix** *x* **assume** *x* ∈ *space M1 emeasure M2* (*Pair x* −‘ *N*) = *0*
have *AE y in M2*. *Q* (*x*, *y*)
proof (*rule AE_I*)
show *emeasure M2* (*Pair x* −‘ *N*) = *0* **by** *fact*
show *Pair x* −‘ *N* ∈ *sets M2* **using** *N*(*1*) **by** (*rule sets_Pair1*)
show {*y* ∈ *space M2*. ¬ *Q* (*x*, *y*)} ⊆ *Pair x* −‘ *N*
using *N* ‹*x* ∈ *space M1*› **unfolding** *space_pair_measure* **by** *auto*
qed }
then show {*x* ∈ *space M1*. ¬ (*AE y in M2*. *Q* (*x*, *y*))} ⊆ {*x* ∈ *space M1*. *emeasure M2* (*Pair x* −‘ *N*) ≠ *0*}
by *auto*
qed
qed

lemma (**in** *pair_sigma_finite*) *AE_pair_measure*:
assumes {*x*∈*space* (*M1* $\bigotimes_M$ *M2*). *P x*} ∈ *sets* (*M1* $\bigotimes_M$ *M2*)
assumes *ae*: *AE x in M1*. *AE y in M2*. *P* (*x*, *y*)
shows *AE x in M1* $\bigotimes_M$ *M2*. *P x*
proof (*subst AE_iff_measurable*[*OF _ refl*])
show {*x*∈*space* (*M1* $\bigotimes_M$ *M2*). ¬ *P x*} ∈ *sets* (*M1* $\bigotimes_M$ *M2*)
by (*rule sets.sets_Collect*) *fact*
then have *emeasure* (*M1* $\bigotimes_M$ *M2*) {*x* ∈ *space* (*M1* $\bigotimes_M$ *M2*). ¬ *P x*} =
(∫$^+$ *x*. ∫$^+$ *y*. *indicator* {*x* ∈ *space* (*M1* $\bigotimes_M$ *M2*). ¬ *P x*} (*x*, *y*) ∂*M2* ∂*M1*)
by (*simp add*: *M2.emeasure_pair_measure*)
also have ... = (∫$^+$ *x*. ∫$^+$ *y*. *0* ∂*M2* ∂*M1*)
using *ae*
apply (*safe intro*!: *nn_integral_cong_AE*)

apply (*intro AE_I2*)
apply (*safe intro*!: *nn_integral_cong_AE*)
apply *auto*
done
finally show *emeasure* (*M1* $\bigotimes_M$ *M2*) {*x* ∈ *space* (*M1* $\bigotimes_M$ *M2*). ¬ *P x*} = *0*
by *simp*
qed

lemma (**in** *pair_sigma_finite*) *AE_pair_iff*:
{*x*∈*space* (*M1* $\bigotimes_M$ *M2*). *P* (*fst x*) (*snd x*)} ∈ *sets* (*M1* $\bigotimes_M$ *M2*) ⟹
(*AE x in M1*. *AE y in M2*. *P x y*) ⟷ (*AE x in* (*M1* $\bigotimes_M$ *M2*). *P* (*fst x*) (*snd x*))
using *AE_pair*[*of* λ*x*. *P* (*fst x*) (*snd x*)] *AE_pair_measure*[*of* λ*x*. *P* (*fst x*) (*snd x*)] **by** *auto*

lemma (**in** *pair_sigma_finite*) *AE_commute*:
assumes *P*: {*x*∈*space* (*M1* $\bigotimes_M$ *M2*). *P* (*fst x*) (*snd x*)} ∈ *sets* (*M1* $\bigotimes_M$ *M2*)
shows (*AE x in M1*. *AE y in M2*. *P x y*) ⟷ (*AE y in M2*. *AE x in M1*. *P x y*)
proof −
interpret *Q*: *pair_sigma_finite M2 M1* **..**
have [*simp*]: ⋀*x*. (*fst* (*case x of* (*x*, *y*) ⇒ (*y*, *x*))) = *snd x* ⋀*x*. (*snd* (*case x of* (*x*, *y*) ⇒ (*y*, *x*))) = *fst x*
by *auto*
have {*x* ∈ *space* (*M2* $\bigotimes_M$ *M1*). *P* (*snd x*) (*fst x*)} =
(λ(*x*, *y*). (*y*, *x*)) −‘ {*x* ∈ *space* (*M1* $\bigotimes_M$ *M2*). *P* (*fst x*) (*snd x*)} ∩ *space* (*M2* $\bigotimes_M$ *M1*)
by (*auto simp*: *space_pair_measure*)
also have ... ∈ *sets* (*M2* $\bigotimes_M$ *M1*)
by (*intro sets_pair_swap P*)
finally show *?thesis*
apply (*subst AE_pair_iff*[*OF P*])
apply (*subst distr_pair_swap*)
apply (*subst AE_distr_iff*[*OF measurable_pair_swap*′ *P*])
apply (*subst Q.AE_pair_iff*)
apply *simp_all*
done
qed

6.6.3 Fubinis theorem

lemma *measurable_compose_Pair1*:
x ∈ *space M1* ⟹ *g* ∈ *measurable* (*M1* $\bigotimes_M$ *M2*) *L* ⟹ (λ*y*. *g* (*x*, *y*)) ∈ *measurable M2 L*
by *simp*

lemma (**in** *sigma_finite_measure*) *borel_measurable_nn_integral_fst*:
assumes *f*: *f* ∈ *borel_measurable* (*M1* $\bigotimes_M$ *M*)
shows (λ*x*. $\int^+$ *y*. *f* (*x*, *y*) ∂*M*) ∈ *borel_measurable M1*

using *f* **proof** *induct*
case (*cong u v*)
then have $\bigwedge w\ x.\ w \in space\ M1 \Longrightarrow x \in space\ M \Longrightarrow u\ (w, x) = v\ (w, x)$
by (*auto simp*: *space_pair_measure*)
show *?case*
apply (*subst measurable_cong*)
apply (*rule nn_integral_cong*)
apply *fact+*
done
next
case (*set Q*)
have [*simp*]: $\bigwedge x\ y.\ indicator\ Q\ (x, y) = indicator\ (Pair\ x -` Q)\ y$
by (*auto simp*: *indicator_def*)
have $\bigwedge x.\ x \in space\ M1 \Longrightarrow emeasure\ M\ (Pair\ x -` Q) = \int^+ y.\ indicator\ Q\ (x, y)\ \partial M$
by (*simp add*: *sets_Pair1*[*OF set*])
from *this measurable_emeasure_Pair*[*OF set*] **show** *?case*
by (*rule measurable_cong*[*THEN iffD1*])
qed (*simp_all add*: *nn_integral_add nn_integral_cmult measurable_compose_Pair1*
nn_integral_monotone_convergence_SUP incseq_def le_fun_def image_comp
cong: *measurable_cong*)

lemma (**in** *sigma_finite_measure*) *nn_integral_fst*:
assumes *f*: $f \in borel\_measurable\ (M1 \bigotimes_M M)$
shows $(\int^+ x.\ \int^+ y.\ f\ (x, y)\ \partial M\ \partial M1) = integral^N\ (M1 \bigotimes_M M)\ f$ (**is** *?I f* = _)
using *f* **proof** *induct*
case (*cong u v*)
then have *?I u* = *?I v*
by (*intro nn_integral_cong*) (*auto simp*: *space_pair_measure*)
with *cong* **show** *?case*
by (*simp cong*: *nn_integral_cong*)
qed (*simp_all add*: *emeasure_pair_measure nn_integral_cmult nn_integral_add*
nn_integral_monotone_convergence_SUP measurable_compose_Pair1
borel_measurable_nn_integral_fst nn_integral_mono incseq_def
le_fun_def image_comp
cong: *nn_integral_cong*)

lemma (**in** *sigma_finite_measure*) *borel_measurable_nn_integral*[*measurable* (*raw*)]:
case_prod f $\in borel\_measurable\ (N \bigotimes_M M) \Longrightarrow (\lambda x.\ \int^+ y.\ f\ x\ y\ \partial M) \in borel\_measurable\ N$
using *borel_measurable_nn_integral_fst*[*of case_prod f N*] **by** *simp*

proposition (**in** *pair_sigma_finite*) *nn_integral_snd*:
assumes *f*[*measurable*]: $f \in borel\_measurable\ (M1 \bigotimes_M M2)$
shows $(\int^+ y.\ (\int^+ x.\ f\ (x, y)\ \partial M1)\ \partial M2) = integral^N\ (M1 \bigotimes_M M2)\ f$
proof −
note *measurable_pair_swap*[*OF f*]

from *M1.nn_integral_fst*[*OF this*]
have $(\int^+ y.\ (\int^+ x.\ f\ (x,\ y)\ \partial M1)\ \partial M2) = (\int^+ (x,\ y).\ f\ (y,\ x)\ \partial(M2 \bigotimes_M M1))$
by *simp*
also have $(\int^+ (x,\ y).\ f\ (y,\ x)\ \partial(M2 \bigotimes_M M1)) = integral^N\ (M1 \bigotimes_M M2)\ f$
by (*subst distr_pair_swap*) (*auto simp add*: *nn_integral_distr intro*!: *nn_integral_cong*)
finally show *?thesis* **.**
qed

theorem (**in** *pair_sigma_finite*) *Fubini*:
assumes *f*: $f \in borel\_measurable\ (M1 \bigotimes_M M2)$
shows $(\int^+ y.\ (\int^+ x.\ f\ (x,\ y)\ \partial M1)\ \partial M2) = (\int^+ x.\ (\int^+ y.\ f\ (x,\ y)\ \partial M2)\ \partial M1)$
unfolding *nn_integral_snd*[*OF assms*] *M2.nn_integral_fst*[*OF assms*] **..**

theorem (**in** *pair_sigma_finite*) *Fubini'*:
assumes *f*: $case\_prod\ f \in borel\_measurable\ (M1 \bigotimes_M M2)$
shows $(\int^+ y.\ (\int^+ x.\ f\ x\ y\ \partial M1)\ \partial M2) = (\int^+ x.\ (\int^+ y.\ f\ x\ y\ \partial M2)\ \partial M1)$
using *Fubini*[*OF f*] **by** *simp*

6.6.4 Products on counting spaces, densities and distributions

proposition *sigma_prod*:
assumes *X_cover*: $\exists E{\subseteq}A.\ countable\ E \wedge X = \bigcup E$ **and** *A*: $A \subseteq Pow\ X$
assumes *Y_cover*: $\exists E{\subseteq}B.\ countable\ E \wedge Y = \bigcup E$ **and** *B*: $B \subseteq Pow\ Y$
shows $sigma\ X\ A \bigotimes_M sigma\ Y\ B = sigma\ (X \times Y)\ \{a \times b \mid a\ b.\ a \in A \wedge b \in B\}$
(**is** *?P* = *?S*)
proof (*rule measure_eqI*)
have [*simp*]: $snd \in X \times Y \to Y\ fst \in X \times Y \to X$
by *auto*
let *?XY* = $\{\{fst -` a \cap X \times Y \mid a.\ a \in A\}, \{snd -` b \cap X \times Y \mid b.\ b \in B\}\}$
have *sets ?P* = $sets\ (SUP\ xy{\in}?XY.\ sigma\ (X \times Y)\ xy)$
by (*simp add*: *vimage_algebra_sigma sets_pair_eq_sets_fst_snd A B*)
also have ... = $sets\ (sigma\ (X \times Y)\ (\bigcup ?XY))$
by (*intro Sup_sigma arg_cong*[**where** *f*=*sets*]) *auto*
also have ... = *sets ?S*
proof (*intro arg_cong*[**where** *f*=*sets*] *sigma_eqI sigma_sets_eqI*)
show $\bigcup ?XY \subseteq Pow\ (X \times Y)\ \{a \times b \mid a\ b.\ a \in A \wedge b \in B\} \subseteq Pow\ (X \times Y)$
using *A B* **by** *auto*
next
interpret *XY*: $sigma\_algebra\ X \times Y\ sigma\_sets\ (X \times Y)\ \{a \times b \mid a\ b.\ a \in A \wedge b \in B\}$
using *A B* **by** (*intro sigma_algebra_sigma_sets*) *auto*
fix *Z* **assume** $Z \in \bigcup ?XY$
then show $Z \in sigma\_sets\ (X \times Y)\ \{a \times b \mid a\ b.\ a \in A \wedge b \in B\}$
proof *safe*
fix *a* **assume** $a \in A$

```
    from Y_cover obtain E where E: E ⊆ B countable E and Y = ⋃ E
      by auto
    with ‹a ∈ A› A have eq: fst −‘ a ∩ X × Y = (⋃e∈E. a × e)
      by auto
    show fst −‘ a ∩ X × Y ∈ sigma_sets (X × Y) {a × b |a b. a ∈ A ∧ b ∈ B}
      using ‹a ∈ A› E unfolding eq by (auto intro!: XY.countable_UN′)
  next
    fix b assume b ∈ B
    from X_cover obtain E where E: E ⊆ A countable E and X = ⋃ E
      by auto
    with ‹b ∈ B› B have eq: snd −‘ b ∩ X × Y = (⋃e∈E. e × b)
      by auto
    show snd −‘ b ∩ X × Y ∈ sigma_sets (X × Y) {a × b |a b. a ∈ A ∧ b ∈
B}
      using ‹b ∈ B› E unfolding eq by (auto intro!: XY.countable_UN′)
  qed
 next
  fix Z assume Z ∈ {a × b |a b. a ∈ A ∧ b ∈ B}
  then obtain a b where Z = a × b and ab: a ∈ A b ∈ B
    by auto
  then have Z: Z = (fst −‘ a ∩ X × Y) ∩ (snd −‘ b ∩ X × Y)
    using A B by auto
  interpret XY: sigma_algebra X × Y sigma_sets (X × Y) (⋃ ?XY)
    by (intro sigma_algebra_sigma_sets) auto
  show Z ∈ sigma_sets (X × Y) (⋃ ?XY)
    unfolding Z by (rule XY.Int) (blast intro: ab)+
 qed
 finally show sets ?P = sets ?S .
next
 interpret finite_measure sigma X A for X A
  proof qed (simp add: emeasure_sigma)
 fix A assume A ∈ sets ?P then show emeasure ?P A = emeasure ?S A
  by (simp add: emeasure_pair_measure_alt emeasure_sigma)
qed

lemma sigma_sets_pair_measure_generator_finite:
 assumes finite A and finite B
 shows sigma_sets (A × B) { a × b | a b. a ⊆ A ∧ b ⊆ B} = Pow (A × B)
 (is sigma_sets ?prod ?sets = _)
proof safe
 have fin: finite (A × B) using assms by (rule finite_cartesian_product)
 fix x assume subset: x ⊆ A × B
 hence finite x using fin by (rule finite_subset)
 from this subset show x ∈ sigma_sets ?prod ?sets
 proof (induct x)
  case empty show ?case by (rule sigma_sets.Empty)
 next
  case (insert a x)
  hence {a} ∈ sigma_sets ?prod ?sets by auto
```

moreover have $x \in$ *sigma_sets ?prod ?sets* **using** *insert* **by** *auto*
ultimately show *?case* **unfolding** *insert_is_Un*[*of a x*] **by** (*rule sigma_sets_Un*)
qed
next
fix *x a b*
assume $x \in$ *sigma_sets ?prod ?sets* **and** $(a, b) \in x$
from *sigma_sets_into_sp*[*OF _ this*(*1*)] *this*(*2*)
show $a \in A$ **and** $b \in B$ **by** *auto*
qed

proposition *sets_pair_eq*:
assumes *Ea*: $Ea \subseteq Pow\ (space\ A)$ $sets\ A = sigma\_sets\ (space\ A)\ Ea$
and *Ca*: *countable Ca* $Ca \subseteq Ea$ $\bigcup Ca = space\ A$
and *Eb*: $Eb \subseteq Pow\ (space\ B)$ $sets\ B = sigma\_sets\ (space\ B)\ Eb$
and *Cb*: *countable Cb* $Cb \subseteq Eb$ $\bigcup Cb = space\ B$
shows $sets\ (A \bigotimes_M B) = sets\ (sigma\ (space\ A \times space\ B)\ \{\ a \times b \mid a\ b.\ a \in Ea \wedge b \in Eb\ \})$
(**is** $\_ = sets\ (sigma\ ?\Omega\ ?E)$)
proof
show $sets\ (sigma\ ?\Omega\ ?E) \subseteq sets\ (A \bigotimes_M B)$
using *Ea*(*1*) *Eb*(*1*) **by** (*subst sigma_le_sets*) (*auto simp*: *Ea*(*2*) *Eb*(*2*))
have $?E \subseteq Pow\ ?\Omega$
using *Ea*(*1*) *Eb*(*1*) **by** *auto*
then have *E*: $a \in Ea \Longrightarrow b \in Eb \Longrightarrow a \times b \in sets\ (sigma\ ?\Omega\ ?E)$ **for** *a b*
by *auto*
have $sets\ (A \bigotimes_M B) \subseteq sets\ (Sup\ \{vimage\_algebra\ ?\Omega\ fst\ A,\ vimage\_algebra\ ?\Omega\ snd\ B\})$
unfolding *sets_pair_eq_sets_fst_snd* **..**
also have $vimage\_algebra\ ?\Omega\ fst\ A = vimage\_algebra\ ?\Omega\ fst\ (sigma\ (space\ A)\ Ea)$
by (*intro vimage_algebra_cong*[*OF refl refl*]) (*simp add*: *Ea*)
also have $\ldots = sigma\ ?\Omega\ \{fst -` A \cap ?\Omega \mid A.\ A \in Ea\}$
by (*intro Ea vimage_algebra_sigma*) *auto*
also have $vimage\_algebra\ ?\Omega\ snd\ B = vimage\_algebra\ ?\Omega\ snd\ (sigma\ (space\ B)\ Eb)$
by (*intro vimage_algebra_cong*[*OF refl refl*]) (*simp add*: *Eb*)
also have $\ldots = sigma\ ?\Omega\ \{snd -` A \cap ?\Omega \mid A.\ A \in Eb\}$
by (*intro Eb vimage_algebra_sigma*) *auto*
also have $\{sigma\ ?\Omega\ \{fst -` Aa \cap ?\Omega \mid Aa.\ Aa \in Ea\},\ sigma\ ?\Omega\ \{snd -` Aa \cap ?\Omega \mid Aa.\ Aa \in Eb\}\} =$
$sigma\ ?\Omega\ `\ \{\{fst -` Aa \cap ?\Omega \mid Aa.\ Aa \in Ea\},\ \{snd -` Aa \cap ?\Omega \mid Aa.\ Aa \in Eb\}\}$
by *auto*
also have $sets\ (SUP\ S \in \{\{fst -` Aa \cap ?\Omega \mid Aa.\ Aa \in Ea\},\ \{snd -` Aa \cap ?\Omega \mid Aa.\ Aa \in Eb\}\}.\ sigma\ ?\Omega\ S) =$
$sets\ (sigma\ ?\Omega\ (\bigcup \{\{fst -` Aa \cap ?\Omega \mid Aa.\ Aa \in Ea\},\ \{snd -` Aa \cap ?\Omega \mid Aa.\ Aa \in Eb\}\}))$
using *Ea*(*1*) *Eb*(*1*) **by** (*intro sets_Sup_sigma*) *auto*
also have $\ldots \subseteq sets\ (sigma\ ?\Omega\ ?E)$
proof (*subst sigma_le_sets*, *safe intro*!: *space_in_measure_of*)

```
    fix a assume a ∈ Ea
    then have fst -` a ∩ ?Ω = (⋃b∈Cb. a × b)
      using Cb(3)[symmetric] Ea(1) by auto
    then show fst -` a ∩ ?Ω ∈ sets (sigma ?Ω ?E)
      using Cb ‹a ∈ Ea› by (auto intro!: sets.countable_UN' E)
  next
    fix b assume b ∈ Eb
    then have snd -` b ∩ ?Ω = (⋃a∈Ca. a × b)
      using Ca(3)[symmetric] Eb(1) by auto
    then show snd -` b ∩ ?Ω ∈ sets (sigma ?Ω ?E)
      using Ca ‹b ∈ Eb› by (auto intro!: sets.countable_UN' E)
  qed
  finally show sets (A ⨂M B) ⊆ sets (sigma ?Ω ?E) .
qed

proposition borel_prod:
  (borel ⨂M borel) = (borel :: ('a::second_countable_topology × 'b::second_countable_topology)
measure)
  (is ?P = ?B)
proof -
  have ?B = sigma UNIV {A × B | A B. open A ∧ open B}
    by (rule second_countable_borel_measurable[OF open_prod_generated])
  also have ... = ?P
    unfolding borel_def
    by (subst sigma_prod) (auto intro!: exI[of _ {UNIV}])
  finally show ?thesis ..
qed

proposition pair_measure_count_space:
  assumes A: finite A and B: finite B
  shows count_space A ⨂M count_space B = count_space (A × B) (is ?P =
?C)
proof (rule measure_eqI)
  interpret A: finite_measure count_space A by (rule finite_measure_count_space)
fact
  interpret B: finite_measure count_space B by (rule finite_measure_count_space)
fact
  interpret P: pair_sigma_finite count_space A count_space B ..
  show eq: sets ?P = sets ?C
    by (simp add: sets_pair_measure sigma_sets_pair_measure_generator_finite
A B)
  fix X assume X: X ∈ sets ?P
  with eq have X_subset: X ⊆ A × B by simp
  with A B have fin_Pair: ⋀x. finite (Pair x -` X)
    by (intro finite_subset[OF _ B]) auto
  have fin_X: finite X using X_subset by (rule finite_subset) (auto simp: A B)
  have card: 0 < card (Pair a -` X) if (a, b) ∈ X for a b
    using card_gt_0_iff fin_Pair that by auto
  then have emeasure ?P X = ∫+ x. emeasure (count_space B) (Pair x -` X)
```

```
          ∂count_space A
    by (simp add: B.emeasure_pair_measure_alt X)
  also have ... = emeasure ?C X
    apply (subst emeasure_count_space)
    using card X_subset A fin_Pair fin_X
    apply (auto simp add: nn_integral_count_space
                          of_nat_sum[symmetric] card_SigmaI[symmetric]
                simp del:  card_SigmaI
                intro!: arg_cong[where f=card])
    done
  finally show emeasure ?P X = emeasure ?C X .
qed


lemma emeasure_prod_count_space:
  assumes A: A ∈ sets (count_space UNIV ⨂M M) (is A ∈ sets (?A ⨂M ?B))
  shows emeasure (?A ⨂M ?B) A = (∫+ x. ∫+ y. indicator A (x, y) ∂?B ∂?A)
  by (rule emeasure_measure_of[OF pair_measure_def])
     (auto simp: countably_additive_def positive_def suminf_indicator A
                 nn_integral_suminf[symmetric] dest: sets.sets_into_space)

lemma emeasure_prod_count_space_single[simp]: emeasure (count_space UNIV
⨂M count_space UNIV) {x} = 1
proof −
  have [simp]: ⋀a b x y. indicator {(a, b)} (x, y) = (indicator {a} x ∗ indicator
{b} y::ennreal)
    by (auto split: split_indicator)
  show ?thesis
   by (cases x) (auto simp: emeasure_prod_count_space nn_integral_cmult sets_Pair)
qed

lemma emeasure_count_space_prod_eq:
  fixes A :: ('a × 'b) set
  assumes A: A ∈ sets (count_space UNIV ⨂M count_space UNIV) (is A ∈
sets (?A ⨂M ?B))
  shows emeasure (?A ⨂M ?B) A = emeasure (count_space UNIV) A
proof −
  { fix A :: ('a × 'b) set assume countable A
    then have emeasure (?A ⨂M ?B) (⋃a∈A. {a}) = (∫+a. emeasure (?A ⨂M
?B) {a} ∂count_space A)
     by (intro emeasure_UN_countable) (auto simp: sets_Pair disjoint_family_on_def)
    also have ... = (∫+a. indicator A a ∂count_space UNIV)
      by (subst nn_integral_count_space_indicator) auto
    finally have emeasure (?A ⨂M ?B) A = emeasure (count_space UNIV) A
      by simp }
  note ∗ = this

  show ?thesis
  proof cases
```

```
    assume finite A then show ?thesis
      by (intro * countable_finite)
  next
    assume infinite A
    then obtain C where countable C and infinite C and C ⊆ A
      by (auto dest: infinite_countable_subset')
    with A have emeasure (?A ⨂M ?B) C ≤ emeasure (?A ⨂M ?B) A
      by (intro emeasure_mono) auto
    also have emeasure (?A ⨂M ?B) C = emeasure (count_space UNIV) C
      using ‹countable C› by (rule *)
    finally show ?thesis
      using ‹infinite C› ‹infinite A› by (simp add: top_unique)
  qed
qed

lemma nn_integral_count_space_prod_eq:
   nn_integral (count_space UNIV ⨂M count_space UNIV) f = nn_integral
(count_space UNIV) f
    (is nn_integral ?P f = _)
proof cases
  assume cntbl: countable {x. f x ≠ 0}
  have [simp]: ⋀x. card ({x} ∩ {x. f x ≠ 0}) = (indicator {x. f x ≠ 0} x::ennreal)
    by (auto split: split_indicator)
  have [measurable]: ⋀y. (λx. indicator {y} x) ∈ borel_measurable ?P
    by (rule measurable_discrete_difference[of λx. 0 _ borel {y} λx. indicator {y}
x for y])
       (auto intro: sets_Pair)

  have (∫+x. f x ∂?P) = (∫+x. ∫+x'. f x * indicator {x} x' ∂count_space {x. f
x ≠ 0} ∂?P)
    by (auto simp add: nn_integral_cmult nn_integral_indicator' intro!: nn_integral_cong
split: split_indicator)
  also have ... = (∫+x. ∫+x'. f x' * indicator {x'} x ∂count_space {x. f x ≠ 0}
∂?P)
    by (auto intro!: nn_integral_cong split: split_indicator)
  also have ... = (∫+x'. ∫+x. f x' * indicator {x'} x ∂?P ∂count_space {x. f x
≠ 0})
    by (intro nn_integral_count_space_nn_integral cntbl) auto
  also have ... = (∫+x'. f x' ∂count_space {x. f x ≠ 0})
    by (intro nn_integral_cong) (auto simp: nn_integral_cmult sets_Pair)
  finally show ?thesis
    by (auto simp add: nn_integral_count_space_indicator intro!: nn_integral_cong
split: split_indicator)
next
  { fix x assume f x ≠ 0
    then have (∃r≥0. 0 < r ∧ f x = ennreal r) ∨ f x = ∞
      by (cases f x rule: ennreal_cases) (auto simp: less_le)
    then have ∃n. ennreal (1 / real (Suc n)) ≤ f x
      by (auto elim!: nat_approx_posE intro!: less_imp_le) }
```

```
  note ∗ = this

  assume cntbl: uncountable {x. f x ≠ 0}
  also have {x. f x ≠ 0} = (⋃ n. {x. 1/Suc n ≤ f x})
    using ∗ by auto
  finally obtain n where infinite {x. 1/Suc n ≤ f x}
    by (meson countableI_type countable_UN uncountable_infinite)
  then obtain C where C: C ⊆ {x. 1/Suc n ≤ f x} and countable C infinite C
    by (metis infinite_countable_subset')

  have [measurable]: C ∈ sets ?P
    using sets.countable[OF _ ‹countable C›, of ?P] by (auto simp: sets_Pair)

  have (∫⁺x. ennreal (1/Suc n) ∗ indicator C x ∂?P) ≤ nn_integral ?P f
   using C by (intro nn_integral_mono) (auto split: split_indicator simp: zero_ereal_def[symmetric])
  moreover have (∫⁺x. ennreal (1/Suc n) ∗ indicator C x ∂?P) = ∞
   using ‹infinite C› by (simp add: nn_integral_cmult emeasure_count_space_prod_eq
ennreal_mult_top)
  moreover have (∫⁺x. ennreal (1/Suc n) ∗ indicator C x ∂count_space UNIV)
≤ nn_integral (count_space UNIV) f
   using C by (intro nn_integral_mono) (auto split: split_indicator simp: zero_ereal_def[symmetric])
  moreover have (∫⁺x. ennreal (1/Suc n) ∗ indicator C x ∂count_space UNIV)
= ∞
    using ‹infinite C› by (simp add: nn_integral_cmult ennreal_mult_top)
  ultimately show ?thesis
    by (simp add: top_unique)
qed

theorem pair_measure_density:
  assumes f: f ∈ borel_measurable M1
  assumes g: g ∈ borel_measurable M2
  assumes sigma_finite_measure M2 sigma_finite_measure (density M2 g)
  shows density M1 f ⨂M density M2 g = density (M1 ⨂M M2) (λ(x,y). f x ∗
g y) (is ?L = ?R)
proof (rule measure_eqI)
  interpret M2: sigma_finite_measure M2 by fact
  interpret D2: sigma_finite_measure density M2 g by fact

  fix A assume A: A ∈ sets ?L
  with f g have (∫⁺ x. f x ∗ ∫⁺ y. g y ∗ indicator A (x, y) ∂M2 ∂M1) =
    (∫⁺ x. ∫⁺ y. f x ∗ g y ∗ indicator A (x, y) ∂M2 ∂M1)
    by (intro nn_integral_cong_AE)
       (auto simp add: nn_integral_cmult[symmetric] ac_simps)
  with A f g show emeasure ?L A = emeasure ?R A
   by (simp add: D2.emeasure_pair_measure emeasure_density nn_integral_density
                 M2.nn_integral_fst[symmetric]
             cong: nn_integral_cong)
qed simp
```

lemma *sigma_finite_measure_distr*:
assumes *sigma_finite_measure* (*distr M N f*) **and** *f*: $f \in$ *measurable M N*
shows *sigma_finite_measure M*
proof −
interpret *sigma_finite_measure distr M N f* **by** *fact*
obtain *A* **where** *A*: *countable A* $A \subseteq$ *sets* (*distr M N f*)
$\bigcup A =$ *space* (*distr M N f*) $\forall a{\in}A.$ *emeasure* (*distr M N f*) $a \neq \infty$
using *sigma_finite_countable* **by** *auto*
show *?thesis*
proof
show $\exists A.$ *countable* $A \wedge A \subseteq$ *sets* $M \wedge \bigcup A =$ *space* $M \wedge (\forall a{\in}A.$ *emeasure* $M\ a \neq \infty)$
using *A f*
by (*intro exI*[*of* _ ($\lambda a.$ *f* −‘ *a* ∩ *space M*) ‘ *A*])
(*auto simp*: *emeasure_distr set_eq_iff subset_eq intro*: *measurable_space*)
qed
qed

lemma *pair_measure_distr*:
assumes *f*: $f \in$ *measurable M S* **and** *g*: $g \in$ *measurable N T*
assumes *sigma_finite_measure* (*distr N T g*)
shows *distr M S f* $\bigotimes_M$ *distr N T g* = *distr* (*M* $\bigotimes_M$ *N*) (*S* $\bigotimes_M$ *T*) ($\lambda(x, y).$ (*f x*, *g y*)) (**is** *?P* = *?D*)
proof (*rule measure_eqI*)
interpret *T*: *sigma_finite_measure distr N T g* **by** *fact*
interpret *N*: *sigma_finite_measure N* **by** (*rule sigma_finite_measure_distr*) *fact*+

fix *A* **assume** *A*: $A \in$ *sets ?P*
with *f g* **show** *emeasure ?P A* = *emeasure ?D A*
by (*auto simp add*: *N.emeasure_pair_measure_alt space_pair_measure emeasure_distr*
T.emeasure_pair_measure_alt nn_integral_distr
intro!: *nn_integral_cong arg_cong*[**where** *f*=*emeasure N*])
qed *simp*

lemma *pair_measure_eqI*:
assumes *sigma_finite_measure M1 sigma_finite_measure M2*
assumes *sets*: *sets* (*M1* $\bigotimes_M$ *M2*) = *sets M*
assumes *emeasure*: $\bigwedge A\ B.\ A \in$ *sets M1* $\Longrightarrow$ $B \in$ *sets M2* $\Longrightarrow$ *emeasure M1 A* $*$ *emeasure M2 B* = *emeasure M* ($A \times B$)
shows *M1* $\bigotimes_M$ *M2* = *M*
proof −
interpret *M1*: *sigma_finite_measure M1* **by** *fact*
interpret *M2*: *sigma_finite_measure M2* **by** *fact*
interpret *pair_sigma_finite M1 M2* **..**
let *?E* = $\{a \times b \mid a\ b.\ a \in$ *sets M1* $\wedge\ b \in$ *sets M2*$\}$
let *?P* = *M1* $\bigotimes_M$ *M2*
obtain *F* :: *nat* ⇒ ('*a* × '*b*) *set* **where** *F*:

range F ⊆ ?E incseq F $\bigcup$ *(range F) = space M1 × space M2* $\forall$ *i. emeasure ?P (F i) ≠* ∞
using *sigma_finite_up_in_pair_measure_generator*
by *blast*
show *?thesis*
proof (*rule measure_eqI_generator_eq*[*OF Int_stable_pair_measure_generator*[*of M1 M2*]])
show *?E ⊆ Pow (space ?P)*
using *sets.space_closed*[*of M1*] *sets.space_closed*[*of M2*] **by** (*auto simp*: *space_pair_measure*)
show *sets ?P = sigma_sets (space ?P) ?E*
by (*simp add*: *sets_pair_measure space_pair_measure*)
then show *sets M = sigma_sets (space ?P) ?E*
using *sets*[*symmetric*] **by** *simp*
next
show *range F ⊆ ?E* ($\bigcup$ *i. F i*) *= space ?P* $\bigwedge$*i. emeasure ?P (F i) ≠* ∞
using *F* **by** (*auto simp*: *space_pair_measure*)
next
fix *X* **assume** *X ∈ ?E*
then obtain *A B* **where** *X*[*simp*]: *X = A × B* **and** *A*: *A ∈ sets M1* **and** *B*: *B ∈ sets M2* **by** *auto*
then have *emeasure ?P X = emeasure M1 A * emeasure M2 B*
by (*simp add*: *M2.emeasure_pair_measure_Times*)
also have *... = emeasure M (A × B)*
using *A B emeasure* **by** *auto*
finally show *emeasure ?P X = emeasure M X*
by *simp*
qed
qed

lemma *sets_pair_countable*:
assumes *countable S1 countable S2*
assumes *M*: *sets M = Pow S1* **and** *N*: *sets N = Pow S2*
shows *sets* ($M \bigotimes_M N$) *= Pow (S1 × S2)*
proof *auto*
fix *x a b* **assume** *x*: *x ∈ sets* ($M \bigotimes_M N$) *(a, b) ∈ x*
from *sets.sets_into_space*[*OF x(1)*] *x(2)*
sets_eq_imp_space_eq[*of N count_space S2*] *sets_eq_imp_space_eq*[*of M count_space S1*] *M N*
show *a ∈ S1 b ∈ S2*
by (*auto simp*: *space_pair_measure*)
next
fix *X* **assume** *X*: *X ⊆ S1 × S2*
then have *countable X*
by (*metis countable_subset ‹countable S1› ‹countable S2› countable_SIGMA*)
have *X =* ($\bigcup$*(a, b)∈X. {a} × {b}*) **by** *auto*
also have *... ∈ sets* ($M \bigotimes_M N$)
using *X*
by (*safe intro*!: *sets.countable_UN' ‹countable X› subsetI pair_measureI*) (*auto*

simp: *M N*)
finally show $X \in sets\ (M \bigotimes_M N)$ **.**
qed

lemma *pair_measure_countable*:
assumes *countable S1 countable S2*
shows *count_space S1* $\bigotimes_M$ *count_space S2* = *count_space* (*S1* × *S2*)
proof (*rule pair_measure_eqI*)
show *sigma_finite_measure* (*count_space S1*) *sigma_finite_measure* (*count_space S2*)
using *assms* **by** (*auto intro*!: *sigma_finite_measure_count_space_countable*)
show *sets* (*count_space S1* $\bigotimes_M$ *count_space S2*) = *sets* (*count_space* (*S1* × *S2*))
by (*subst sets_pair_countable*[*OF assms*]) *auto*
next
fix *A B* **assume** *A* ∈ *sets* (*count_space S1*) *B* ∈ *sets* (*count_space S2*)
then show *emeasure* (*count_space S1*) *A* ∗ *emeasure* (*count_space S2*) *B* = *emeasure* (*count_space* (*S1* × *S2*)) (*A* × *B*)
by (*subst* (*1 2 3*) *emeasure_count_space*) (*auto simp*: *finite_cartesian_product_iff ennreal_mult_top ennreal_top_mult*)
qed

proposition *nn_integral_fst_count_space*:
(∫⁺ *x*. ∫⁺ *y*. *f* (*x*, *y*) ∂*count_space UNIV* ∂*count_space UNIV*) = *integral*^{N} (*count_space UNIV*) *f*
(**is** *?lhs* = *?rhs*)
proof(*cases*)
assume ∗: *countable* {*xy*. *f xy* ≠ *0*}
let *?A* = *fst* ‘ {*xy*. *f xy* ≠ *0*}
let *?B* = *snd* ‘ {*xy*. *f xy* ≠ *0*}
from ∗ **have** [*simp*]: *countable ?A countable ?B* **by**(*rule countable_image*)+
have *?lhs* = (∫⁺ *x*. ∫⁺ *y*. *f* (*x*, *y*) ∂*count_space UNIV* ∂*count_space ?A*)
by(*rule nn_integral_count_space_eq*)
(*auto simp add*: *nn_integral_0_iff_AE AE_count_space not_le intro*: *rev_image_eqI*)
also have ... = (∫⁺ *x*. ∫⁺ *y*. *f* (*x*, *y*) ∂*count_space ?B* ∂*count_space ?A*)
by(*intro nn_integral_count_space_eq nn_integral_cong*)(*auto intro*: *rev_image_eqI*)
also have ... = (∫⁺ *xy*. *f xy* ∂*count_space* (*?A* × *?B*))
by(*subst sigma_finite_measure.nn_integral_fst*)
(*simp_all add*: *sigma_finite_measure_count_space_countable pair_measure_countable*)
also have ... = *?rhs*
by(*rule nn_integral_count_space_eq*)(*auto intro*: *rev_image_eqI*)
finally show *?thesis* **.**
next
{ fix *xy* **assume** *f xy* ≠ *0*
then have (∃ *r*≥*0*. *0* < *r* ∧ *f xy* = *ennreal r*) ∨ *f xy* = ∞
by (*cases f xy rule*: *ennreal_cases*) (*auto simp*: *less_le*)
then have ∃ *n*. *ennreal* (*1* / *real* (*Suc n*)) ≤ *f xy*
by (*auto elim*!: *nat_approx_posE intro*!: *less_imp_le*) **}**
note ∗ = *this*

assume *cntbl*: *uncountable* $\{xy.\ f\ xy \neq 0\}$
also have $\{xy.\ f\ xy \neq 0\} = (\bigcup n.\ \{xy.\ 1/Suc\ n \leq f\ xy\})$
using $*$ **by** *auto*
finally obtain n **where** *infinite* $\{xy.\ 1/Suc\ n \leq f\ xy\}$
by (*meson countableI_type countable_UN uncountable_infinite*)
then obtain C **where** C: $C \subseteq \{xy.\ 1/Suc\ n \leq f\ xy\}$ **and** *countable C infinite C*
by (*metis infinite_countable_subset'*)

have $\infty = (\int^+ xy.\ ennreal\ (1\ /\ Suc\ n) * indicator\ C\ xy\ \partial count\_space\ UNIV)$
using ‹*infinite C*› **by**(*simp add*: *nn_integral_cmult ennreal_mult_top*)
also have $\ldots \leq$ *?rhs* **using** *C*
by(*intro nn_integral_mono*)(*auto split*: *split_indicator*)
finally have *?rhs* $= \infty$ **by** (*simp add*: *top_unique*)
moreover have *?lhs* $= \infty$
proof(*cases finite* (*fst* ‘ *C*))
case *True*
then obtain $x\ C'$ **where** x: $x \in fst\ `\ C$
and C': $C' = fst\ -`\ \{x\} \cap C$
and *infinite* C'
using ‹*infinite C*› **by**(*auto elim*!: *inf_img_fin_domE'*)
from $x\ C\ C'$ **have** $**$: $C' \subseteq \{xy.\ 1\ /\ Suc\ n \leq f\ xy\}$ **by** *auto*

from C' ‹*infinite C'*› **have** *infinite* (*snd* ‘ *C'*)
by(*auto dest*!: *finite_imageD simp add*: *inj_on_def*)
then have $\infty = (\int^+ y.\ ennreal\ (1\ /\ Suc\ n) * indicator\ (snd\ `\ C')\ y\ \partial count\_space\ UNIV)$
by(*simp add*: *nn_integral_cmult ennreal_mult_top*)
also have $\ldots = (\int^+ y.\ ennreal\ (1\ /\ Suc\ n) * indicator\ C'\ (x,\ y)\ \partial count\_space\ UNIV)$
by(*rule nn_integral_cong*)(*force split*: *split_indicator intro*: *rev_image_eqI simp add*: *C'*)
also have $\ldots = (\int^+ x'.\ (\int^+ y.\ ennreal\ (1\ /\ Suc\ n) * indicator\ C'\ (x,\ y)\ \partial count\_space\ UNIV) * indicator\ \{x\}\ x'\ \partial count\_space\ UNIV)$
by(*simp add*: *one_ereal_def*[*symmetric*])
also have $\ldots \leq (\int^+ x.\ \int^+ y.\ ennreal\ (1\ /\ Suc\ n) * indicator\ C'\ (x,\ y)\ \partial count\_space\ UNIV\ \partial count\_space\ UNIV)$
by(*rule nn_integral_mono*)(*simp split*: *split_indicator*)
also have $\ldots \leq$ *?lhs* **using** $**$
by(*intro nn_integral_mono*)(*auto split*: *split_indicator*)
finally show *?thesis* **by** (*simp add*: *top_unique*)
next
case *False*
define C' **where** $C' = fst\ `\ C$
have $\infty = \int^+ x.\ ennreal\ (1\ /\ Suc\ n) * indicator\ C'\ x\ \partial count\_space\ UNIV$
using *C'_def False* **by**(*simp add*: *nn_integral_cmult ennreal_mult_top*)
also have $\ldots = \int^+ x.\ \int^+ y.\ ennreal\ (1\ /\ Suc\ n) * indicator\ C'\ x * indicator\ \{SOME\ y.\ (x,\ y) \in C\}\ y\ \partial count\_space\ UNIV\ \partial count\_space\ UNIV$

by(*auto simp add*: *one_ereal_def*[*symmetric*] *max_def intro*: *nn_integral_cong*)
also have ... $\leq \int^+ x.\ \int^+ y.\ ennreal\ (1\ /\ Suc\ n) * indicator\ C\ (x,\ y)$ *∂count_space UNIV ∂count_space UNIV*
by(*intro nn_integral_mono*)(*auto simp add*: *C′_def split*: *split_indicator intro*: *someI*)
also have ... $\leq$ *?lhs* **using** *C*
by(*intro nn_integral_mono*)(*auto split*: *split_indicator*)
finally show *?thesis* **by** (*simp add*: *top_unique*)
qed
ultimately show *?thesis* **by** *simp*
qed

proposition *nn_integral_snd_count_space*:
$(\int^+ y.\ \int^+ x.\ f\ (x,\ y)$ *∂count_space UNIV ∂count_space UNIV*$) = integral^N$ *(count_space UNIV) f*
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = $(\int^+ y.\ \int^+ x.\ (\lambda(y,\ x).\ f\ (x,\ y))\ (y,\ x)$ *∂count_space UNIV ∂count_space UNIV*)
by(*simp*)
also have ... = $\int^+ yx.\ (\lambda(y,\ x).\ f\ (x,\ y))\ yx$ *∂count_space UNIV*
by(*rule nn_integral_fst_count_space*)
also have ... = $\int^+ xy.\ f\ xy$ *∂count_space* ((λ(*x*, *y*). (*y*, *x*)) ‘ *UNIV*)
by(*subst nn_integral_bij_count_space*[*OF inj_on_imp_bij_betw*, *symmetric*])
(*simp_all add*: *inj_on_def split_def*)
also have ... = *?rhs* **by**(*rule nn_integral_count_space_eq*) *auto*
finally show *?thesis* **.**
qed

lemma *measurable_pair_measure_countable1*:
assumes *countable A*
and [*measurable*]: $\bigwedge x.\ x \in A \Longrightarrow (\lambda y.\ f\ (x,\ y)) \in measurable\ N\ K$
shows $f \in measurable\ (count\_space\ A \bigotimes_M N)\ K$
using _ _ *assms*(*1*)
by(*rule measurable_compose_countable′*[**where** *f*=λ*a b*. *f* (*a*, *snd b*) **and** *g*=*fst* **and** *I*=*A*, *simplified*])*simp_all*

6.6.5 Product of Borel spaces

theorem *borel_Times*:
fixes *A* :: *′a*::*topological_space set* **and** *B* :: *′b*::*topological_space set*
assumes *A*: $A \in sets\ borel$ **and** *B*: $B \in sets\ borel$
shows $A \times B \in sets\ borel$
proof −
have $A \times B = (A \times UNIV) \cap (UNIV \times B)$
by *auto*
moreover
{ have $A \in sigma\_sets\ UNIV\ \{S.\ open\ S\}$ **using** *A* **by** (*simp add*: *sets_borel*)
then have $A \times UNIV \in sets\ borel$

```
    proof (induct A)
      case (Basic S) then show ?case
        by (auto intro!: borel_open open_Times)
    next
      case (Compl A)
      moreover have *: (UNIV - A) × UNIV = UNIV - (A × UNIV)
        by auto
      ultimately show ?case
        unfolding * by auto
    next
      case (Union A)
     moreover have *: (⋃(A ` UNIV)) × UNIV = ⋃((λi. A i × UNIV) ` UNIV)
        by auto
      ultimately show ?case
        unfolding * by auto
    qed simp }
  moreover
  { have B ∈ sigma_sets UNIV {S. open S} using B by (simp add: sets_borel)
    then have UNIV×B ∈ sets borel
    proof (induct B)
      case (Basic S) then show ?case
        by (auto intro!: borel_open open_Times)
    next
      case (Compl B)
      moreover have *: UNIV × (UNIV - B) = UNIV - (UNIV × B)
        by auto
      ultimately show ?case
        unfolding * by auto
    next
      case (Union B)
     moreover have *: UNIV × (⋃(B ` UNIV)) = ⋃((λi. UNIV × B i) ` UNIV)
        by auto
      ultimately show ?case
        unfolding * by auto
    qed simp }
  ultimately show ?thesis
    by auto
qed

lemma finite_measure_pair_measure:
  assumes finite_measure M finite_measure N
  shows finite_measure (N ⨂M M)
proof (rule finite_measureI)
  interpret M: finite_measure M by fact
  interpret N: finite_measure N by fact
  show emeasure (N ⨂M M) (space (N ⨂M M)) ≠ ∞
    by (auto simp: space_pair_measure M.emeasure_pair_measure_Times en-
nreal_mult_eq_top_iff)
qed
```

end

6.7 Finite Product Measure

theory *Finite_Product_Measure*
imports *Binary_Product_Measure Function_Topology*
begin

lemma *PiE_choice*: $(\exists f{\in}Pi_E\ I\ F.\ \forall i{\in}I.\ P\ i\ (f\ i)) \longleftrightarrow (\forall i{\in}I.\ \exists x{\in}F\ i.\ P\ i\ x)$
by (*auto simp*: *Bex_def PiE_iff Ball_def dest*!: *choice_iff'*[*THEN iffD1*])
(*force intro*: *exI*[*of* _ *restrict f I* **for** *f*])

lemma *case_prod_const*: $(\lambda(i, j).\ c) = (\lambda\_.\ c)$
by *auto*

6.7.1 More about Function restricted by *extensional*

definition
merge I J $= (\lambda(x, y)\ i.$ *if* $i \in I$ *then* $x\ i$ *else if* $i \in J$ *then* $y\ i$ *else undefined*)

lemma *merge_apply*[*simp*]:
$I \cap J = \{\} \Longrightarrow i \in I \Longrightarrow$ *merge I J* $(x, y)\ i = x\ i$
$I \cap J = \{\} \Longrightarrow i \in J \Longrightarrow$ *merge I J* $(x, y)\ i = y\ i$
$J \cap I = \{\} \Longrightarrow i \in I \Longrightarrow$ *merge I J* $(x, y)\ i = x\ i$
$J \cap I = \{\} \Longrightarrow i \in J \Longrightarrow$ *merge I J* $(x, y)\ i = y\ i$
$i \notin I \Longrightarrow i \notin J \Longrightarrow$ *merge I J* $(x, y)\ i =$ *undefined*
unfolding *merge_def* **by** *auto*

lemma *merge_commute*:
$I \cap J = \{\} \Longrightarrow$ *merge I J* $(x, y) =$ *merge J I* (y, x)
by (*force simp*: *merge_def*)

lemma *Pi_cancel_merge_range*[*simp*]:
$I \cap J = \{\} \Longrightarrow x \in Pi\ I\ (merge\ I\ J\ (A, B)) \longleftrightarrow x \in Pi\ I\ A$
$I \cap J = \{\} \Longrightarrow x \in Pi\ I\ (merge\ J\ I\ (B, A)) \longleftrightarrow x \in Pi\ I\ A$
$J \cap I = \{\} \Longrightarrow x \in Pi\ I\ (merge\ I\ J\ (A, B)) \longleftrightarrow x \in Pi\ I\ A$
$J \cap I = \{\} \Longrightarrow x \in Pi\ I\ (merge\ J\ I\ (B, A)) \longleftrightarrow x \in Pi\ I\ A$
by (*auto simp*: *Pi_def*)

lemma *Pi_cancel_merge*[*simp*]:
$I \cap J = \{\} \Longrightarrow$ *merge I J* $(x, y) \in Pi\ I\ B \longleftrightarrow x \in Pi\ I\ B$
$J \cap I = \{\} \Longrightarrow$ *merge I J* $(x, y) \in Pi\ I\ B \longleftrightarrow x \in Pi\ I\ B$
$I \cap J = \{\} \Longrightarrow$ *merge I J* $(x, y) \in Pi\ J\ B \longleftrightarrow y \in Pi\ J\ B$
$J \cap I = \{\} \Longrightarrow$ *merge I J* $(x, y) \in Pi\ J\ B \longleftrightarrow y \in Pi\ J\ B$
by (*auto simp*: *Pi_def*)

lemma *extensional_merge*[*simp*]: *merge I J* $(x, y) \in$ *extensional* $(I \cup J)$
by (*auto simp*: *extensional_def*)

lemma *restrict_merge*[*simp*]:
  $I \cap J = \{\} \Longrightarrow$ *restrict* (*merge I J* (*x*, *y*)) *I* = *restrict x I*
  $I \cap J = \{\} \Longrightarrow$ *restrict* (*merge I J* (*x*, *y*)) *J* = *restrict y J*
  $J \cap I = \{\} \Longrightarrow$ *restrict* (*merge I J* (*x*, *y*)) *I* = *restrict x I*
  $J \cap I = \{\} \Longrightarrow$ *restrict* (*merge I J* (*x*, *y*)) *J* = *restrict y J*
  by (*auto simp*: *restrict_def*)

lemma *split_merge*: *P* (*merge I J* (*x*,*y*) *i*) $\longleftrightarrow$ (*i* $\in$ *I* $\longrightarrow$ *P* (*x i*)) $\wedge$ (*i* $\in$ *J* $-$ *I* $\longrightarrow$ *P* (*y i*)) $\wedge$ (*i* $\notin$ *I* $\cup$ *J* $\longrightarrow$ *P undefined*)
  unfolding *merge_def* **by** *auto*

lemma *PiE_cancel_merge*[*simp*]:
  $I \cap J = \{\} \Longrightarrow$
    merge I J (*x*, *y*) $\in$ *Pi*$_E$ (*I* $\cup$ *J*) *B* $\longleftrightarrow$ *x* $\in$ *Pi I B* $\wedge$ *y* $\in$ *Pi J B*
  by (*auto simp*: *PiE_def restrict_Pi_cancel*)

lemma *merge_singleton*[*simp*]: *i* $\notin$ *I* $\Longrightarrow$ *merge I* {*i*} (*x*,*y*) = *restrict* (*x*(*i* := *y i*)) (*insert i I*)
  unfolding *merge_def* **by** (*auto simp*: *fun_eq_iff*)

lemma *extensional_merge_sub*: *I* $\cup$ *J* $\subseteq$ *K* $\Longrightarrow$ *merge I J* (*x*, *y*) $\in$ *extensional K*
  unfolding *merge_def extensional_def* **by** *auto*

lemma *merge_restrict*[*simp*]:
  merge I J (*restrict x I*, *y*) = *merge I J* (*x*, *y*)
  merge I J (*x*, *restrict y J*) = *merge I J* (*x*, *y*)
  unfolding *merge_def* **by** *auto*

lemma *merge_x_x_eq_restrict*[*simp*]:
  merge I J (*x*, *x*) = *restrict x* (*I* $\cup$ *J*)
  unfolding *merge_def* **by** *auto*

lemma *injective_vimage_restrict*:
  assumes *J*: *J* $\subseteq$ *I*
  and *sets*: *A* $\subseteq$ (Π_E *i*$\in$*J*. *S i*) *B* $\subseteq$ (Π_E *i*$\in$*J*. *S i*) **and** *ne*: (Π_E *i*$\in$*I*. *S i*) $\neq$ {}
  and *eq*: (λ*x*. *restrict x J*) $-`$ *A* $\cap$ ($\Pi_E$ *i*$\in$*I*. *S i*) = ($\lambda$*x*. *restrict x J*) $-`$ *B* $\cap$ (Π_E *i*$\in$*I*. *S i*)
  shows *A* = *B*
proof (*intro set_eqI*)
  fix *x*
  from *ne* **obtain** *y* **where** *y*: $\bigwedge$*i*. *i* $\in$ *I* $\Longrightarrow$ *y i* $\in$ *S i* **by** *auto*
  have *J* $\cap$ (*I* $-$ *J*) = {} **by** *auto*
  show *x* $\in$ *A* $\longleftrightarrow$ *x* $\in$ *B*
  proof *cases*
    assume *x*: *x* $\in$ (Π_E *i*$\in$*J*. *S i*)
    have *x* $\in$ *A* $\longleftrightarrow$ *merge J* (*I* $-$ *J*) (*x*,*y*) $\in$ (λ*x*. *restrict x J*) $-`$ *A* $\cap$ (Π_E *i*$\in$*I*. *S i*)
      using *y x* ‹*J* $\subseteq$ *I*› *PiE_cancel_merge*[*of J I* $-$ *J x y S*]

by (*auto simp del*: *PiE_cancel_merge simp add*: *Un_absorb1*)
then show $x \in A \longleftrightarrow x \in B$
using *y x* ‹$J \subseteq I$› *PiE_cancel_merge*[*of J I − J x y S*]
by (*auto simp del*: *PiE_cancel_merge simp add*: *Un_absorb1 eq*)
qed (*insert sets*, *auto*)
qed

lemma *restrict_vimage*:
$I \cap J = \{\} \Longrightarrow$
$(\lambda x.\ (\mathit{restrict}\ x\ I,\ \mathit{restrict}\ x\ J)) -` (\mathit{Pi}_E\ I\ E \times \mathit{Pi}_E\ J\ F) = \mathit{Pi}\ (I \cup J)\ (\mathit{merge}\ I\ J\ (E,\ F))$
by (*auto simp*: *restrict_Pi_cancel PiE_def*)

lemma *merge_vimage*:
$I \cap J = \{\} \Longrightarrow \mathit{merge}\ I\ J -` \mathit{Pi}_E\ (I \cup J)\ E = \mathit{Pi}\ I\ E \times \mathit{Pi}\ J\ E$
by (*auto simp*: *restrict_Pi_cancel PiE_def*)

6.7.2 Finite product spaces

definition *prod_emb* **where**
$\mathit{prod\_emb}\ I\ M\ K\ X = (\lambda x.\ \mathit{restrict}\ x\ K) -` X \cap (\Pi_E\ i \in I.\ \mathit{space}\ (M\ i))$

lemma *prod_emb_iff*:
$f \in \mathit{prod\_emb}\ I\ M\ K\ X \longleftrightarrow f \in \mathit{extensional}\ I \wedge (\mathit{restrict}\ f\ K \in X) \wedge (\forall i \in I.\ f\ i \in \mathit{space}\ (M\ i))$
unfolding *prod_emb_def PiE_def* **by** *auto*

lemma
shows *prod_emb_empty*[*simp*]: $\mathit{prod\_emb}\ M\ L\ K\ \{\} = \{\}$
and *prod_emb_Un*[*simp*]: $\mathit{prod\_emb}\ M\ L\ K\ (A \cup B) = \mathit{prod\_emb}\ M\ L\ K\ A \cup \mathit{prod\_emb}\ M\ L\ K\ B$
and *prod_emb_Int*: $\mathit{prod\_emb}\ M\ L\ K\ (A \cap B) = \mathit{prod\_emb}\ M\ L\ K\ A \cap \mathit{prod\_emb}\ M\ L\ K\ B$
and *prod_emb_UN*[*simp*]: $\mathit{prod\_emb}\ M\ L\ K\ (\bigcup i \in I.\ F\ i) = (\bigcup i \in I.\ \mathit{prod\_emb}\ M\ L\ K\ (F\ i))$
and *prod_emb_INT*[*simp*]: $I \neq \{\} \Longrightarrow \mathit{prod\_emb}\ M\ L\ K\ (\bigcap i \in I.\ F\ i) = (\bigcap i \in I.\ \mathit{prod\_emb}\ M\ L\ K\ (F\ i))$
and *prod_emb_Diff*[*simp*]: $\mathit{prod\_emb}\ M\ L\ K\ (A - B) = \mathit{prod\_emb}\ M\ L\ K\ A - \mathit{prod\_emb}\ M\ L\ K\ B$
by (*auto simp*: *prod_emb_def*)

lemma *prod_emb_PiE*: $J \subseteq I \Longrightarrow (\bigwedge i.\ i \in J \Longrightarrow E\ i \subseteq \mathit{space}\ (M\ i)) \Longrightarrow$
$\mathit{prod\_emb}\ I\ M\ J\ (\Pi_E\ i \in J.\ E\ i) = (\Pi_E\ i \in I.\ \mathit{if}\ i \in J\ \mathit{then}\ E\ i\ \mathit{else}\ \mathit{space}\ (M\ i))$
by (*force simp*: *prod_emb_def PiE_iff if_split_mem2*)

lemma *prod_emb_PiE_same_index*[*simp*]:
$(\bigwedge i.\ i \in I \Longrightarrow E\ i \subseteq \mathit{space}\ (M\ i)) \Longrightarrow \mathit{prod\_emb}\ I\ M\ I\ (\mathit{Pi}_E\ I\ E) = \mathit{Pi}_E\ I\ E$
by (*auto simp*: *prod_emb_def PiE_iff*)

lemma *prod_emb_trans*[*simp*]:
$J \subseteq K \Longrightarrow K \subseteq L \Longrightarrow$ *prod_emb L M K* (*prod_emb K M J X*) = *prod_emb L M J X*
by (*auto simp add*: *Int_absorb1 prod_emb_def PiE_def*)

lemma *prod_emb_Pi*:
assumes $X \in (\Pi\ j{\in}J.\ sets\ (M\ j))$ $J \subseteq K$
shows *prod_emb K M J* (Pi_E *J X*) = (Π_E $i{\in}K.$ *if* $i \in J$ *then X i else space* (*M i*))
using *assms sets.space_closed*
by (*auto simp*: *prod_emb_def PiE_iff split*: *if_split_asm*) *blast+*

lemma *prod_emb_id*:
$B \subseteq (\Pi_E\ i{\in}L.\ space\ (M\ i)) \Longrightarrow$ *prod_emb L M L B* = *B*
by (*auto simp*: *prod_emb_def subset_eq extensional_restrict*)

lemma *prod_emb_mono*:
$F \subseteq G \Longrightarrow$ *prod_emb A M B F* $\subseteq$ *prod_emb A M B G*
by (*auto simp*: *prod_emb_def*)

definition *PiM* :: *'i set* $\Rightarrow$ (*'i* $\Rightarrow$ *'a measure*) $\Rightarrow$ (*'i* $\Rightarrow$ *'a*) *measure* **where**
PiM I M = *extend_measure* (Π_E $i{\in}I.$ *space* (*M i*))
$\{(J, X).\ (J \neq \{\} \vee I = \{\}) \wedge$ *finite* $J \wedge J \subseteq I \wedge X \in (\Pi\ j{\in}J.\ sets\ (M\ j))\}$
$(\lambda(J, X).$ *prod_emb I M J* (Π_E $j{\in}J.$ *X j*))
$(\lambda(J, X).\ \prod j{\in}J \cup \{i{\in}I.\ emeasure\ (M\ i)\ (space\ (M\ i)) \neq 1\}.$ *if* $j \in J$ *then emeasure* (*M j*) (*X j*) *else emeasure* (*M j*) (*space* (*M j*)))

definition *prod_algebra* :: *'i set* $\Rightarrow$ (*'i* $\Rightarrow$ *'a measure*) $\Rightarrow$ (*'i* $\Rightarrow$ *'a*) *set set* **where**
prod_algebra I M = $(\lambda(J, X).$ *prod_emb I M J* (Π_E $j{\in}J.$ *X j*)) ‘
$\{(J, X).\ (J \neq \{\} \vee I = \{\}) \wedge$ *finite* $J \wedge J \subseteq I \wedge X \in (\Pi\ j{\in}J.\ sets\ (M\ j))\}$

abbreviation
Pi_M *I M* $\equiv$ *PiM I M*

syntax
_PiM :: *pttrn* $\Rightarrow$ *'i set* $\Rightarrow$ *'a measure* $\Rightarrow$ (*'i => 'a*) *measure* ((3Π_M *_*$\in$*_./ _*) *10*)
translations
Π_M $x{\in}I.$ *M* == *CONST PiM I* (%$x.$ *M*)

lemma *extend_measure_cong*:
assumes $\Omega = \Omega'$ $I = I'$ $G = G'$ $\bigwedge i.\ i \in I' \Longrightarrow \mu\ i = \mu'\ i$
shows *extend_measure* Ω *I G* μ = *extend_measure* Ω' I' G' μ'
unfolding *extend_measure_def* **by** (*auto simp add*: *assms*)

lemma *Pi_cong_sets*:
$[\![I = J;\ \bigwedge x.\ x \in I \Longrightarrow M\ x = N\ x]\!] \Longrightarrow$ *Pi I M* = *Pi J N*
unfolding *Pi_def* **by** *auto*

lemma *PiM_cong*:
assumes $I = J$ $\bigwedge x.\ x \in I \Longrightarrow M\ x = N\ x$
shows *PiM I M* = *PiM J N*
unfolding *PiM_def*
proof (*rule extend_measure_cong*, *goal_cases*)
case *1*
show *?case* **using** *assms*
by (*subst assms*(*1*), *intro PiE_cong*[*of J* $\lambda i.$ *space* (*M i*) $\lambda i.$ *space* (*N i*)])
simp_all
next
case *2*
have $\bigwedge K.\ K \subseteq J \Longrightarrow (\Pi\ j{\in}K.\ sets\ (M\ j)) = (\Pi\ j{\in}K.\ sets\ (N\ j))$
using *assms* **by** (*intro Pi_cong_sets*) *auto*
thus *?case* **by** (*auto simp*: *assms*)
next
case *3*
show *?case* **using** *assms*
by (*intro ext*) (*auto simp*: *prod_emb_def dest*: *PiE_mem*)
next
case (*4 x*)
thus *?case* **using** *assms*
by (*auto intro*!: *prod.cong split*: *if_split_asm*)
qed

lemma *prod_algebra_sets_into_space*:
prod_algebra I M $\subseteq$ *Pow* (Π_E *i*$\in$*I*. *space* (*M i*))
by (*auto simp*: *prod_emb_def prod_algebra_def*)

lemma *prod_algebra_eq_finite*:
assumes *I*: *finite I*
shows *prod_algebra I M* = {(Π_E *i*$\in$*I*. *X i*) |*X*. *X* $\in$ (Π *j*$\in$*I*. *sets* (*M j*))} (**is** *?L* = *?R*)
proof (*intro iffI set_eqI*)
fix *A* **assume** *A* $\in$ *?L*
then obtain *J E* **where** *J*: *J* $\neq$ {} $\vee$ *I* = {} *finite J J* $\subseteq$ *I* $\forall$ *i*$\in$*J*. *E i* $\in$ *sets* (*M i*)
and *A*: *A* = *prod_emb I M J* (Π_E *j*$\in$*J*. *E j*)
by (*auto simp*: *prod_algebra_def*)
let *?A* = Π_E *i*$\in$*I*. *if i* $\in$ *J then E i else space* (*M i*)
have *A*: *A* = *?A*
unfolding *A* **using** *J* **by** (*intro prod_emb_PiE sets.sets_into_space*) *auto*
show *A* $\in$ *?R* **unfolding** *A* **using** *J sets.top*
by (*intro CollectI exI*[*of* _ $\lambda i.$ *if i* $\in$ *J then E i else space* (*M i*)]) *simp*
next
fix *A* **assume** *A* $\in$ *?R*
then obtain *X* **where** *A*: *A* = (Π_E *i*$\in$*I*. *X i*) **and** *X*: *X* $\in$ (Π *j*$\in$*I*. *sets* (*M j*))
by *auto*
then have *A*: *A* = *prod_emb I M I* (Π_E *i*$\in$*I*. *X i*)

by (*simp add*: *prod_emb_PiE_same_index*[*OF sets.sets_into_space*] *Pi_iff*)
from *X I* **show** *A* ∈ *?L* **unfolding** *A*
by (*auto simp*: *prod_algebra_def*)
qed

lemma *prod_algebraI*:
finite J ⟹ (*J* ≠ {} ∨ *I* = {}) ⟹ *J* ⊆ *I* ⟹ (⋀*i*. *i* ∈ *J* ⟹ *E i* ∈ *sets* (*M i*))
⟹ *prod_emb I M J* (Π$_E$ *j*∈*J*. *E j*) ∈ *prod_algebra I M*
by (*auto simp*: *prod_algebra_def*)

lemma *prod_algebraI_finite*:
finite I ⟹ (∀ *i*∈*I*. *E i* ∈ *sets* (*M i*)) ⟹ (*Pi*$_E$ *I E*) ∈ *prod_algebra I M*
using *prod_algebraI*[*of I I E M*] *prod_emb_PiE_same_index*[*of I E M*, *OF sets.sets_into_space*] **by** *simp*

lemma *Int_stable_PiE*: *Int_stable* {*Pi*$_E$ *J E* | *E*. ∀ *i*∈*I*. *E i* ∈ *sets* (*M i*)}
proof (*safe intro*!: *Int_stableI*)
fix *E F* **assume** ∀ *i*∈*I*. *E i* ∈ *sets* (*M i*) ∀ *i*∈*I*. *F i* ∈ *sets* (*M i*)
then show ∃ *G*. *Pi*$_E$ *J E* ∩ *Pi*$_E$ *J F* = *Pi*$_E$ *J G* ∧ (∀ *i*∈*I*. *G i* ∈ *sets* (*M i*))
by (*auto intro*!: *exI*[*of* _ λ*i*. *E i* ∩ *F i*] *simp*: *PiE_Int*)
qed

lemma *prod_algebraE*:
assumes *A*: *A* ∈ *prod_algebra I M*
obtains *J E* **where** *A* = *prod_emb I M J* (Π$_E$ *j*∈*J*. *E j*)
finite J J ≠ {} ∨ *I* = {} *J* ⊆ *I* ⋀*i*. *i* ∈ *J* ⟹ *E i* ∈ *sets* (*M i*)
using *A* **by** (*auto simp*: *prod_algebra_def*)

lemma *prod_algebraE_all*:
assumes *A*: *A* ∈ *prod_algebra I M*
obtains *E* **where** *A* = *Pi*$_E$ *I E E* ∈ (Π *i*∈*I*. *sets* (*M i*))
proof −
from *A* **obtain** *E J* **where** *A*: *A* = *prod_emb I M J* (*Pi*$_E$ *J E*)
and *J*: *J* ⊆ *I* **and** *E*: *E* ∈ (Π *i*∈*J*. *sets* (*M i*))
by (*auto simp*: *prod_algebra_def*)
from *E* **have** ⋀*i*. *i* ∈ *J* ⟹ *E i* ⊆ *space* (*M i*)
using *sets.sets_into_space* **by** *auto*
then have *A* = (Π$_E$ *i*∈*I*. *if i*∈*J then E i else space* (*M i*))
using *A J* **by** (*auto simp*: *prod_emb_PiE*)
moreover have (λ*i*. *if i*∈*J then E i else space* (*M i*)) ∈ (Π *i*∈*I*. *sets* (*M i*))
using *sets.top E* **by** *auto*
ultimately show *?thesis* **using** *that* **by** *auto*
qed

lemma *Int_stable_prod_algebra*: *Int_stable* (*prod_algebra I M*)
unfolding *Int_stable_def*
proof *safe*
fix *A* **assume** *A* ∈ *prod_algebra I M*
from *prod_algebraE*[*OF this*] **obtain** *J E* **where** *A*:

$A = prod\_emb\ I\ M\ J\ (Pi_E\ J\ E)$
$finite\ J$
$J \neq \{\} \vee I = \{\}$
$J \subseteq I$
$\bigwedge i.\ i \in J \Longrightarrow E\ i \in sets\ (M\ i)$
by *auto*
fix B **assume** $B \in prod\_algebra\ I\ M$
from $prod\_algebraE[OF\ this]$ **obtain** $K\ F$ **where** B:
$B = prod\_emb\ I\ M\ K\ (Pi_E\ K\ F)$
$finite\ K$
$K \neq \{\} \vee I = \{\}$
$K \subseteq I$
$\bigwedge i.\ i \in K \Longrightarrow F\ i \in sets\ (M\ i)$
by *auto*
have $A \cap B = prod\_emb\ I\ M\ (J \cup K)\ (\Pi_E\ i{\in}J \cup K.\ (if\ i \in J\ then\ E\ i\ else\ space\ (M\ i)) \cap$
$(if\ i \in K\ then\ F\ i\ else\ space\ (M\ i)))$
unfolding $A\ B$ **using** $A(2,3,4)\ A(5)[THEN\ sets.sets\_into\_space]\ B(2,3,4)$
$B(5)[THEN\ sets.sets\_into\_space]$
apply $(subst\ (1\ 2\ 3)\ prod\_emb\_PiE)$
apply $(simp\_all\ add{:}\ subset\_eq\ PiE\_Int)$
apply *blast*
apply $(intro\ PiE\_cong)$
apply *auto*
done
also have $\ldots \in prod\_algebra\ I\ M$
using $A\ B$ **by** $(auto\ intro!{:}\ prod\_algebraI)$
finally show $A \cap B \in prod\_algebra\ I\ M$ **.**
qed

proposition *prod_algebra_mono*:
assumes *space*: $\bigwedge i.\ i \in I \Longrightarrow space\ (E\ i) = space\ (F\ i)$
assumes *sets*: $\bigwedge i.\ i \in I \Longrightarrow sets\ (E\ i) \subseteq sets\ (F\ i)$
shows $prod\_algebra\ I\ E \subseteq prod\_algebra\ I\ F$
proof
fix A **assume** $A \in prod\_algebra\ I\ E$
then obtain $J\ G$ **where** $J{:}\ J \neq \{\} \vee I = \{\}\ finite\ J\ J \subseteq I$
and $A{:}\ A = prod\_emb\ I\ E\ J\ (\Pi_E\ i{\in}J.\ G\ i)$
and $G{:}\ \bigwedge i.\ i \in J \Longrightarrow G\ i \in sets\ (E\ i)$
by $(auto\ simp{:}\ prod\_algebra\_def)$
moreover
from *space* **have** $(\Pi_E\ i{\in}I.\ space\ (E\ i)) = (\Pi_E\ i{\in}I.\ space\ (F\ i))$
by $(rule\ PiE\_cong)$
with A **have** $A = prod\_emb\ I\ F\ J\ (\Pi_E\ i{\in}J.\ G\ i)$
by $(simp\ add{:}\ prod\_emb\_def)$
moreover
from *sets* $G\ J$ **have** $\bigwedge i.\ i \in J \Longrightarrow G\ i \in sets\ (F\ i)$
by *auto*
ultimately show $A \in prod\_algebra\ I\ F$

apply (*simp add*: *prod_algebra_def image_iff*)
apply (*intro exI*[*of* _ *J*] *exI*[*of* _ *G*] *conjI*)
apply *auto*
done
qed
proposition *prod_algebra_cong*:
assumes $I = J$ **and** *sets*: $(\bigwedge i.\ i \in I \Longrightarrow sets\ (M\ i) = sets\ (N\ i))$
shows *prod_algebra I M* = *prod_algebra J N*
proof −
have *space*: $\bigwedge i.\ i \in I \Longrightarrow space\ (M\ i) = space\ (N\ i)$
using *sets_eq_imp_space_eq*[*OF sets*] **by** *auto*
with *sets* **show** *?thesis* **unfolding** ‹$I = J$›
by (*intro antisym prod_algebra_mono*) *auto*
qed

lemma *space_in_prod_algebra*:
$(\Pi_E\ i{\in}I.\ space\ (M\ i)) \in prod\_algebra\ I\ M$
proof *cases*
assume $I = \{\}$ **then show** *?thesis*
by (*auto simp add*: *prod_algebra_def image_iff prod_emb_def*)
next
assume $I \neq \{\}$
then obtain *i* **where** $i \in I$ **by** *auto*
then have $(\Pi_E\ i{\in}I.\ space\ (M\ i)) = prod\_emb\ I\ M\ \{i\}\ (\Pi_E\ i{\in}\{i\}.\ space\ (M\ i))$
by (*auto simp*: *prod_emb_def*)
also have $\ldots \in prod\_algebra\ I\ M$
using ‹$i \in I$› **by** (*intro prod_algebraI*) *auto*
finally show *?thesis* **.**
qed

lemma *space_PiM*: $space\ (\Pi_M\ i{\in}I.\ M\ i) = (\Pi_E\ i{\in}I.\ space\ (M\ i))$
using *prod_algebra_sets_into_space* **unfolding** *PiM_def prod_algebra_def* **by** (*intro space_extend_measure*) *simp*

lemma *prod_emb_subset_PiM*[*simp*]: $prod\_emb\ I\ M\ K\ X \subseteq space\ (PiM\ I\ M)$
by (*auto simp*: *prod_emb_def space_PiM*)

lemma *space_PiM_empty_iff*[*simp*]: $space\ (PiM\ I\ M) = \{\} \longleftrightarrow\ (\exists\, i{\in}I.\ space\ (M\ i) = \{\})$
by (*auto simp*: *space_PiM PiE_eq_empty_iff*)

lemma *undefined_in_PiM_empty*[*simp*]: $(\lambda x.\ undefined) \in space\ (PiM\ \{\}\ M)$
by (*auto simp*: *space_PiM*)

lemma *sets_PiM*: $sets\ (\Pi_M\ i{\in}I.\ M\ i) = sigma\_sets\ (\Pi_E\ i{\in}I.\ space\ (M\ i))\ (prod\_algebra\ I\ M)$
using *prod_algebra_sets_into_space* **unfolding** *PiM_def prod_algebra_def* **by** (*intro sets_extend_measure*) *simp*

proposition *sets_PiM_single*: *sets* (*PiM I M*) =
sigma_sets (Π_E *i*∈*I*. *space* (*M i*)) {{*f*∈Π_E *i*∈*I*. *space* (*M i*). *f i* ∈ *A*} | *i A*. *i* ∈ *I* ∧ *A* ∈ *sets* (*M i*)}
(**is** _ = *sigma_sets* *?Ω* *?R*)
unfolding *sets_PiM*
proof (*rule sigma_sets_eqI*)
interpret *R*: *sigma_algebra* *?Ω* *sigma_sets* *?Ω* *?R* **by** (*rule sigma_algebra_sigma_sets*) *auto*
fix *A* **assume** *A* ∈ *prod_algebra I M*
from *prod_algebraE*[*OF this*] **obtain** *J X* **where** *X*:
A = *prod_emb I M J* (Pi_E *J X*)
finite J
J ≠ {} ∨ *I* = {}
J ⊆ *I*
⋀*i*. *i* ∈ *J* ⟹ *X i* ∈ *sets* (*M i*)
by *auto*
show *A* ∈ *sigma_sets* *?Ω* *?R*
proof *cases*
assume *I* = {}
with *X* **have** *A* = {λ*x*. *undefined*} **by** (*auto simp*: *prod_emb_def*)
with ‹*I* = {}› **show** *?thesis* **by** (*auto intro*!: *sigma_sets_top*)
next
assume *I* ≠ {}
with *X* **have** *A* = (⋂*j*∈*J*. {*f*∈(Π_E *i*∈*I*. *space* (*M i*)). *f j* ∈ *X j*})
by (*auto simp*: *prod_emb_def*)
also have ... ∈ *sigma_sets* *?Ω* *?R*
using *X* ‹*I* ≠ {}› **by** (*intro R.finite_INT sigma_sets.Basic*) *auto*
finally show *A* ∈ *sigma_sets* *?Ω* *?R* **.**
qed
next
fix *A* **assume** *A* ∈ *?R*
then obtain *i B* **where** *A*: *A* = {*f*∈Π_E *i*∈*I*. *space* (*M i*). *f i* ∈ *B*} *i* ∈ *I B* ∈ *sets* (*M i*)
by *auto*
then have *A* = *prod_emb I M* {*i*} (Π_E *i*∈{*i*}. *B*)
by (*auto simp*: *prod_emb_def*)
also have ... ∈ *sigma_sets* *?Ω* (*prod_algebra I M*)
using *A* **by** (*intro sigma_sets.Basic prod_algebraI*) *auto*
finally show *A* ∈ *sigma_sets* *?Ω* (*prod_algebra I M*) **.**
qed

lemma *sets_PiM_eq_proj*:
I ≠ {} ⟹ *sets* (*PiM I M*) = *sets* (*SUP i*∈*I*. *vimage_algebra* (Π_E *i*∈*I*. *space* (*M i*)) (λ*x*. *x i*) (*M i*))
apply (*simp add*: *sets_PiM_single*)
apply (*subst sets_Sup_eq*[**where** *X*=Π_E *i*∈*I*. *space* (*M i*)])
apply *auto* []
apply *auto* []
apply *simp*

apply (*subst arg_cong* [*of* _ _ *Sup*, *OF image_cong*, *OF refl*])
apply (*rule sets_vimage_algebra2*)
apply (*auto intro*!: *arg_cong2*[**where** *f*=*sigma_sets*])
done

lemma
shows *space_PiM_empty*: *space* (*Pi*$_M$ {} *M*) = {λ*k*. *undefined*}
and *sets_PiM_empty*: *sets* (*Pi*$_M$ {} *M*) = { {}, {λ*k*. *undefined*} }
by (*simp_all add*: *space_PiM sets_PiM_single image_constant sigma_sets_empty_eq*)

proposition *sets_PiM_sigma*:
assumes Ω_*cover*: ⋀*i*. *i* ∈ *I* ⟹ ∃ *S*⊆*E i*. *countable S* ∧ Ω *i* = ⋃ *S*
assumes *E*: ⋀*i*. *i* ∈ *I* ⟹ *E i* ⊆ *Pow* (Ω *i*)
assumes *J*: ⋀*j*. *j* ∈ *J* ⟹ *finite j* ⋃ *J* = *I*
defines *P* ≡ {{*f*∈(Π$_E$ *i*∈*I*. Ω *i*). ∀ *i*∈*j*. *f i* ∈ *A i*} | *A j*. *j* ∈ *J* ∧ *A* ∈ *Pi j E*}
shows *sets* (Π$_M$ *i*∈*I*. *sigma* (Ω *i*) (*E i*)) = *sets* (*sigma* (Π$_E$ *i*∈*I*. Ω *i*) *P*)
proof *cases*
assume *I* = {}
with ‹⋃ *J* = *I*› **have** *P* = {{λ_. *undefined*}} ∨ *P* = {}
by (*auto simp*: *P_def*)
with ‹*I* = {}› **show** *?thesis*
by (*auto simp add*: *sets_PiM_empty sigma_sets_empty_eq*)
next
let *?F* = λ*i*. {(λ*x*. *x i*) −‘ *A* ∩ *Pi*$_E$ *I* Ω |*A*. *A* ∈ *E i*}
assume *I* ≠ {}
then have *sets* (*Pi*$_M$ *I* (λ*i*. *sigma* (Ω *i*) (*E i*))) =
sets (*SUP i*∈*I*. *vimage_algebra* (Π$_E$ *i*∈*I*. Ω *i*) (λ*x*. *x i*) (*sigma* (Ω *i*) (*E i*)))
by (*subst sets_PiM_eq_proj*) (*auto simp*: *space_measure_of_conv*)
also have ... = *sets* (*SUP i*∈*I*. *sigma* (*Pi*$_E$ *I* Ω) (*?F i*))
using *E* **by** (*intro sets_SUP_cong arg_cong*[**where** *f*=*sets*] *vimage_algebra_sigma*) *auto*
also have ... = *sets* (*sigma* (*Pi*$_E$ *I* Ω) (⋃*i*∈*I*. *?F i*))
using ‹*I* ≠ {}› **by** (*intro arg_cong*[**where** *f*=*sets*] *SUP_sigma_sigma*) *auto*
also have ... = *sets* (*sigma* (*Pi*$_E$ *I* Ω) *P*)
proof (*intro arg_cong*[**where** *f*=*sets*] *sigma_eqI sigma_sets_eqI*)
show (⋃*i*∈*I*. *?F i*) ⊆ *Pow* (*Pi*$_E$ *I* Ω) *P* ⊆ *Pow* (*Pi*$_E$ *I* Ω)
by (*auto simp*: *P_def*)
next
interpret *P*: *sigma_algebra* Π$_E$ *i*∈*I*. Ω *i sigma_sets* (Π$_E$ *i*∈*I*. Ω *i*) *P*
by (*auto intro*!: *sigma_algebra_sigma_sets simp*: *P_def*)

fix *Z* **assume** *Z* ∈ (⋃*i*∈*I*. *?F i*)
then obtain *i A* **where** *i*: *i* ∈ *I A* ∈ *E i* **and** *Z_def*: *Z* = (λ*x*. *x i*) −‘ *A* ∩ *Pi*$_E$ *I* Ω
by *auto*
from ‹*i* ∈ *I*› *J* **obtain** *j* **where** *j*: *i* ∈ *j j* ∈ *J j* ⊆ *I finite j*
by *auto*
obtain *S* **where** *S*: ⋀*i*. *i* ∈ *j* ⟹ *S i* ⊆ *E i* ⋀*i*. *i* ∈ *j* ⟹ *countable* (*S i*)
⋀*i*. *i* ∈ *j* ⟹ Ω *i* = ⋃ (*S i*)

by (*metis subset_eq Ω_cover* ‹$j \subseteq I$›)
define A' **where** $A'\ n = n(i := A)$ **for** n
then have A'_i: $\bigwedge n.\ A'\ n\ i = A$
by *simp*
{ fix n **assume** $n \in Pi_E\ (j - \{i\})\ S$
then have $A'\ n \in Pi\ j\ E$
unfolding *PiE_def Pi_def* **using** $S(1)$ **by** (*auto simp*: A'*_def* ‹$A \in E\ i$›)
with ‹$j \in J$› **have** $\{f \in Pi_E\ I\ \Omega.\ \forall i{\in}j.\ f\ i \in A'\ n\ i\} \in P$
by (*auto simp*: *P_def*) **}**
note A'_in_P = *this*

{ fix x **assume** $x\ i \in A\ x \in Pi_E\ I\ \Omega$
with $S(3)$ ‹$j \subseteq I$› **have** $\forall i{\in}j.\ \exists s{\in}S\ i.\ x\ i \in s$
by (*auto simp*: *PiE_def Pi_def*)
then obtain s **where** s: $\bigwedge i.\ i \in j \Longrightarrow s\ i \in S\ i\ \bigwedge i.\ i \in j \Longrightarrow x\ i \in s\ i$
by *metis*
with ‹$x\ i \in A$› **have** $\exists n{\in}Pi_E\ (j{-}\{i\})\ S.\ \forall i{\in}j.\ x\ i \in A'\ n\ i$
by (*intro bexI*[*of* _ *restrict* (*s*(*i* := *A*)) (*j*−{*i*})]) (*auto simp*: A'*_def split*: *if_splits*) **}**
then have $Z = (\bigcup n{\in}Pi_E\ (j{-}\{i\})\ S.\ \{f{\in}(\Pi_E\ i{\in}I.\ \Omega\ i).\ \forall i{\in}j.\ f\ i \in A'\ n\ i\})$
unfolding *Z_def*
by (*auto simp add*: *set_eq_iff ball_conj_distrib* ‹$i{\in}j$› A'_i *dest*: *bspec*[*OF* _ ‹$i{\in}j$›]
cong: *conj_cong*)
also have $\ldots \in$ *sigma_sets* $(\Pi_E\ i{\in}I.\ \Omega\ i)\ P$
using ‹*finite j*› $S(2)$
by (*intro P.countable_UN' countable_PiE*) (*simp_all add*: *image_subset_iff* A'_in_P)
finally show $Z \in$ *sigma_sets* $(\Pi_E\ i{\in}I.\ \Omega\ i)\ P$ **.**
next
interpret F: *sigma_algebra* $\Pi_E\ i{\in}I.\ \Omega\ i$ *sigma_sets* $(\Pi_E\ i{\in}I.\ \Omega\ i)$ $(\bigcup i{\in}I.\ ?F\ i)$
by (*auto intro*!: *sigma_algebra_sigma_sets*)

fix b **assume** $b \in P$
then obtain $A\ j$ **where** b: $b = \{f{\in}(\Pi_E\ i{\in}I.\ \Omega\ i).\ \forall i{\in}j.\ f\ i \in A\ i\}\ j \in J\ A \in Pi\ j\ E$
by (*auto simp*: *P_def*)
show $b \in$ *sigma_sets* $(Pi_E\ I\ \Omega)$ $(\bigcup i{\in}I.\ ?F\ i)$
proof *cases*
assume $j = \{\}$
with b **have** $b = (\Pi_E\ i{\in}I.\ \Omega\ i)$
by *auto*
then show *?thesis*
by *blast*
next
assume $j \neq \{\}$
with $J\ b(2{,}3)$ **have** *eq*: $b = (\bigcap i{\in}j.\ ((\lambda x.\ x\ i)\ -`\ A\ i \cap Pi_E\ I\ \Omega))$
unfolding $b(1)$

```
      by (auto simp: PiE_def Pi_def)
    show ?thesis
      unfolding eq using ‹A ∈ Pi j E› ‹j ∈ J› J(2)
      by (intro F.finite_INT J ‹j ∈ J› ‹j ≠ {}› sigma_sets.Basic) blast
  qed
 qed
 finally show ?thesis .
qed

lemma sets_PiM_in_sets:
 assumes space: space N = (Π_E i∈I. space (M i))
 assumes sets: ⋀i A. i ∈ I ⟹ A ∈ sets (M i) ⟹ {x∈space N. x i ∈ A} ∈ sets
N
 shows sets (Π_M i ∈ I. M i) ⊆ sets N
 unfolding sets_PiM_single space[symmetric]
 by (intro sets.sigma_sets_subset subsetI) (auto intro: sets)

lemma sets_PiM_cong[measurable_cong]:
 assumes I = J ⋀i. i ∈ J ⟹ sets (M i) = sets (N i) shows sets (PiM I M) =
sets (PiM J N)
 using assms sets_eq_imp_space_eq[OF assms(2)] by (simp add: sets_PiM_single
cong: PiE_cong conj_cong)

lemma sets_PiM_I:
 assumes finite J J ⊆ I ∀i∈J. E i ∈ sets (M i)
 shows prod_emb I M J (Π_E j∈J. E j) ∈ sets (Π_M i∈I. M i)
proof cases
 assume J = {}
 then have prod_emb I M J (Π_E j∈J. E j) = (Π_E j∈I. space (M j))
   by (auto simp: prod_emb_def)
 then show ?thesis
   by (auto simp add: sets_PiM intro!: sigma_sets_top)
next
 assume J ≠ {} with assms show ?thesis
   by (force simp add: sets_PiM prod_algebra_def)
qed

proposition measurable_PiM:
 assumes space: f ∈ space N → (Π_E i∈I. space (M i))
 assumes sets: ⋀X J. J ≠ {} ∨ I = {} ⟹ finite J ⟹ J ⊆ I ⟹ (⋀i. i ∈ J
⟹ X i ∈ sets (M i)) ⟹
   f −‘ prod_emb I M J (Pi_E J X) ∩ space N ∈ sets N
 shows f ∈ measurable N (PiM I M)
 using sets_PiM prod_algebra_sets_into_space space
proof (rule measurable_sigma_sets)
 fix A assume A ∈ prod_algebra I M
 from prod_algebraE[OF this] obtain J X where
   A = prod_emb I M J (Pi_E J X)
   finite J
```

$J \neq \{\} \vee I = \{\}$
$J \subseteq I$
$\bigwedge i.\ i \in J \Longrightarrow X\ i \in sets\ (M\ i)$
by *auto*
with *sets*[*of J X*] **show** $f\ -`\ A \cap space\ N \in sets\ N$ **by** *auto*
qed

lemma *measurable_PiM_Collect*:
assumes *space*: $f \in space\ N \to (\Pi_E\ i \in I.\ space\ (M\ i))$
assumes *sets*: $\bigwedge X\ J.\ J \neq \{\} \vee I = \{\} \Longrightarrow finite\ J \Longrightarrow J \subseteq I \Longrightarrow (\bigwedge i.\ i \in J \Longrightarrow X\ i \in sets\ (M\ i)) \Longrightarrow \{\omega \in space\ N.\ \forall i \in J.\ f\ \omega\ i \in X\ i\} \in sets\ N$
shows $f \in measurable\ N\ (PiM\ I\ M)$
using *sets_PiM prod_algebra_sets_into_space space*
proof (*rule measurable_sigma_sets*)
fix *A* **assume** $A \in prod\_algebra\ I\ M$
from *prod_algebraE*[*OF this*] **obtain** *J X*
where *X*:
$A = prod\_emb\ I\ M\ J\ (Pi_E\ J\ X)$
$finite\ J$
$J \neq \{\} \vee I = \{\}$
$J \subseteq I$
$\bigwedge i.\ i \in J \Longrightarrow X\ i \in sets\ (M\ i)$
by *auto*
then have $f\ -`\ A \cap space\ N = \{\omega \in space\ N.\ \forall i \in J.\ f\ \omega\ i \in X\ i\}$
using *space* **by** (*auto simp*: *prod_emb_def del*: *PiE_I*)
also have $\ldots \in sets\ N$ **using** *X*(*3*,*2*,*4*,*5*) **by** (*rule sets*)
finally show $f\ -`\ A \cap space\ N \in sets\ N$ **.**
qed

lemma *measurable_PiM_single*:
assumes *space*: $f \in space\ N \to (\Pi_E\ i \in I.\ space\ (M\ i))$
assumes *sets*: $\bigwedge A\ i.\ i \in I \Longrightarrow A \in sets\ (M\ i) \Longrightarrow \{\omega \in space\ N.\ f\ \omega\ i \in A\} \in sets\ N$
shows $f \in measurable\ N\ (PiM\ I\ M)$
using *sets_PiM_single*
proof (*rule measurable_sigma_sets*)
fix *A* **assume** $A \in \{\{f \in \Pi_E\ i \in I.\ space\ (M\ i).\ f\ i \in A\}\ |i\ A.\ i \in I \wedge A \in sets\ (M\ i)\}$
then obtain *B i* **where** $A = \{f \in \Pi_E\ i \in I.\ space\ (M\ i).\ f\ i \in B\}$ **and** *B*: $i \in I\ B \in sets\ (M\ i)$
by *auto*
with *space* **have** $f\ -`\ A \cap space\ N = \{\omega \in space\ N.\ f\ \omega\ i \in B\}$ **by** *auto*
also have $\ldots \in sets\ N$ **using** *B* **by** (*rule sets*)
finally show $f\ -`\ A \cap space\ N \in sets\ N$ **.**
qed (*auto simp*: *space*)

lemma *measurable_PiM_single'*:
assumes *f*: $\bigwedge i.\ i \in I \Longrightarrow f\ i \in measurable\ N\ (M\ i)$

and $(\lambda\omega\ i.\ f\ i\ \omega) \in space\ N \to (\Pi_E\ i{\in}I.\ space\ (M\ i))$
shows $(\lambda\omega\ i.\ f\ i\ \omega) \in measurable\ N\ (Pi_M\ I\ M)$
proof (*rule measurable_PiM_single*)
fix $A\ i$ **assume** A: $i \in I\ A \in sets\ (M\ i)$
then have $\{\omega \in space\ N.\ f\ i\ \omega \in A\} = f\ i\ -`\ A \cap space\ N$
by *auto*
then show $\{\omega \in space\ N.\ f\ i\ \omega \in A\} \in sets\ N$
using $A\ f$ **by** (*auto intro*!: *measurable_sets*)
qed *fact*

lemma *sets_PiM_I_finite*[*measurable*]:
assumes *finite I* **and** *sets*: $(\bigwedge i.\ i \in I \Longrightarrow E\ i \in sets\ (M\ i))$
shows $(\Pi_E\ j{\in}I.\ E\ j) \in sets\ (\Pi_M\ i{\in}I.\ M\ i)$
using *sets_PiM_I*[*of I I E M*] *sets.sets_into_space*[*OF sets*] ‹*finite I*› *sets* **by** *auto*

lemma *measurable_component_singleton*[*measurable* (*raw*)]:
assumes $i \in I$ **shows** $(\lambda x.\ x\ i) \in measurable\ (Pi_M\ I\ M)\ (M\ i)$
proof (*unfold measurable_def*, *intro CollectI conjI ballI*)
fix A **assume** $A \in sets\ (M\ i)$
then have $(\lambda x.\ x\ i)\ -`\ A \cap space\ (Pi_M\ I\ M) = prod\_emb\ I\ M\ \{i\}\ (\Pi_E\ j{\in}\{i\}.\ A)$
using *sets.sets_into_space* ‹$i \in I$›
by (*fastforce dest*: *Pi_mem simp*: *prod_emb_def space_PiM split*: *if_split_asm*)
then show $(\lambda x.\ x\ i)\ -`\ A \cap space\ (Pi_M\ I\ M) \in sets\ (Pi_M\ I\ M)$
using ‹$A \in sets\ (M\ i)$› ‹$i \in I$› **by** (*auto intro*!: *sets_PiM_I*)
qed (*insert* ‹$i \in I$›, *auto simp*: *space_PiM*)

lemma *measurable_component_singleton'*[*measurable_dest*]:
assumes f: $f \in measurable\ N\ (Pi_M\ I\ M)$
assumes g: $g \in measurable\ L\ N$
assumes i: $i \in I$
shows $(\lambda x.\ (f\ (g\ x))\ i) \in measurable\ L\ (M\ i)$
using *measurable_compose*[*OF measurable_compose*[*OF g f*] *measurable_component_singleton*, *OF i*] **.**

lemma *measurable_PiM_component_rev*:
$i \in I \Longrightarrow f \in measurable\ (M\ i)\ N \Longrightarrow (\lambda x.\ f\ (x\ i)) \in measurable\ (PiM\ I\ M)\ N$
by *simp*

lemma *measurable_case_nat*[*measurable* (*raw*)]:
assumes [*measurable* (*raw*)]: $i = 0 \Longrightarrow f \in measurable\ M\ N$
$\bigwedge j.\ i = Suc\ j \Longrightarrow (\lambda x.\ g\ x\ j) \in measurable\ M\ N$
shows $(\lambda x.\ case\_nat\ (f\ x)\ (g\ x)\ i) \in measurable\ M\ N$
by (*cases i*) *simp_all*

lemma *measurable_case_nat'*[*measurable* (*raw*)]:
assumes *fg*[*measurable*]: $f \in measurable\ N\ M\ g \in measurable\ N\ (\Pi_M\ i{\in}UNIV.\ M)$

shows $(\lambda x.\ case\_nat\ (f\ x)\ (g\ x)) \in measurable\ N\ (\Pi_M\ i \in UNIV.\ M)$
using *fg*[*THEN measurable_space*]
by (*auto intro*!: *measurable_PiM_single'* *simp add*: *space_PiM PiE_iff split*: *nat.split*)

lemma *measurable_add_dim*[*measurable*]:
$(\lambda(f,\ y).\ f(i := y)) \in measurable\ (Pi_M\ I\ M \bigotimes_M M\ i)\ (Pi_M\ (insert\ i\ I)\ M)$
(**is** *?f* $\in$ *measurable ?P ?I*)
proof (*rule measurable_PiM_single*)
fix *j A* **assume** *j*: $j \in insert\ i\ I$ **and** *A*: $A \in sets\ (M\ j)$
have $\{\omega \in space\ ?P.\ (\lambda(f,\ x).\ fun\_upd\ f\ i\ x)\ \omega\ j \in A\} =$
(*if* $j = i$ *then space* $(Pi_M\ I\ M) \times A$ *else* $((\lambda x.\ x\ j) \circ fst) -` A \cap space\ ?P$)
using *sets.sets_into_space*[*OF A*] **by** (*auto simp add*: *space_pair_measure space_PiM*)
also have $\ldots \in sets\ ?P$
using *A j*
by (*auto intro*!: *measurable_sets*[*OF measurable_comp*, *OF _ measurable_component_singleton*])
finally show $\{\omega \in space\ ?P.\ case\_prod\ (\lambda f.\ fun\_upd\ f\ i)\ \omega\ j \in A\} \in sets\ ?P$ **.**
qed (*auto simp*: *space_pair_measure space_PiM PiE_def*)

proposition *measurable_fun_upd*:
assumes *I*: $I = J \cup \{i\}$
assumes *f*[*measurable*]: $f \in measurable\ N\ (PiM\ J\ M)$
assumes *h*[*measurable*]: $h \in measurable\ N\ (M\ i)$
shows $(\lambda x.\ (f\ x)\ (i := h\ x)) \in measurable\ N\ (PiM\ I\ M)$
proof (*intro measurable_PiM_single'*)
fix *j* **assume** $j \in I$ **then show** $(\lambda\omega.\ ((f\ \omega)(i := h\ \omega))\ j) \in measurable\ N\ (M\ j)$
unfolding *I* **by** (*cases* $j = i$) *auto*
next
show $(\lambda x.\ (f\ x)(i := h\ x)) \in space\ N \to (\Pi_E\ i \in I.\ space\ (M\ i))$
using *I f*[*THEN measurable_space*] *h*[*THEN measurable_space*]
by (*auto simp*: *space_PiM PiE_iff extensional_def*)
qed

lemma *measurable_component_update*:
$x \in space\ (Pi_M\ I\ M) \Longrightarrow i \notin I \Longrightarrow (\lambda v.\ x(i := v)) \in measurable\ (M\ i)\ (Pi_M\ (insert\ i\ I)\ M)$
by *simp*

lemma *measurable_merge*[*measurable*]:
$merge\ I\ J \in measurable\ (Pi_M\ I\ M \bigotimes_M Pi_M\ J\ M)\ (Pi_M\ (I \cup J)\ M)$
(**is** *?f* $\in$ *measurable ?P ?U*)
proof (*rule measurable_PiM_single*)
fix *i A* **assume** *A*: $A \in sets\ (M\ i)\ i \in I \cup J$
then have $\{\omega \in space\ ?P.\ merge\ I\ J\ \omega\ i \in A\} =$
(*if* $i \in I$ *then* $((\lambda x.\ x\ i) \circ fst) -` A \cap space\ ?P$ *else* $((\lambda x.\ x\ i) \circ snd) -` A \cap space\ ?P$)
by (*auto simp*: *merge_def*)
also have $\ldots \in sets\ ?P$

using A
by (*auto intro*!: *measurable_sets*[*OF measurable_comp*, *OF _ measurable_component_singleton*])
finally show $\{\omega \in space\ ?P.\ merge\ I\ J\ \omega\ i \in A\} \in sets\ ?P$ **.**
qed (*auto simp*: *space_pair_measure space_PiM PiE_iff merge_def extensional_def*)

lemma *measurable_restrict*[*measurable* (*raw*)]:
assumes X: $\bigwedge i.\ i \in I \Longrightarrow X\ i \in measurable\ N\ (M\ i)$
shows $(\lambda x.\ \lambda i{\in}I.\ X\ i\ x) \in measurable\ N\ (Pi_M\ I\ M)$
proof (*rule measurable_PiM_single*)
fix $A\ i$ **assume** A: $i \in I\ A \in sets\ (M\ i)$
then have $\{\omega \in space\ N.\ (\lambda i{\in}I.\ X\ i\ \omega)\ i \in A\} = X\ i\ -`\ A \cap space\ N$
by *auto*
then show $\{\omega \in space\ N.\ (\lambda i{\in}I.\ X\ i\ \omega)\ i \in A\} \in sets\ N$
using $A\ X$ **by** (*auto intro*!: *measurable_sets*)
qed (*insert X*, *auto simp add*: *PiE_def dest*: *measurable_space*)

lemma *measurable_abs_UNIV*:
$(\bigwedge n.\ (\lambda\omega.\ f\ n\ \omega) \in measurable\ M\ (N\ n)) \Longrightarrow (\lambda\omega\ n.\ f\ n\ \omega) \in measurable\ M\ (PiM\ UNIV\ N)$
by (*intro measurable_PiM_single*) (*auto dest*: *measurable_space*)

lemma *measurable_restrict_subset*: $J \subseteq L \Longrightarrow (\lambda f.\ restrict\ f\ J) \in measurable\ (Pi_M\ L\ M)\ (Pi_M\ J\ M)$
by (*intro measurable_restrict measurable_component_singleton*) *auto*

lemma *measurable_restrict_subset'*:
assumes $J \subseteq L\ \bigwedge x.\ x \in J \Longrightarrow sets\ (M\ x) = sets\ (N\ x)$
shows $(\lambda f.\ restrict\ f\ J) \in measurable\ (Pi_M\ L\ M)\ (Pi_M\ J\ N)$
proof−
from *assms*(*1*) **have** $(\lambda f.\ restrict\ f\ J) \in measurable\ (Pi_M\ L\ M)\ (Pi_M\ J\ M)$
by (*rule measurable_restrict_subset*)
also from *assms*(*2*) **have** $measurable\ (Pi_M\ L\ M)\ (Pi_M\ J\ M) = measurable\ (Pi_M\ L\ M)\ (Pi_M\ J\ N)$
by (*intro sets_PiM_cong measurable_cong_sets*) *simp_all*
finally show *?thesis* **.**
qed

lemma *measurable_prod_emb*[*intro*, *simp*]:
$J \subseteq L \Longrightarrow X \in sets\ (Pi_M\ J\ M) \Longrightarrow prod\_emb\ L\ M\ J\ X \in sets\ (Pi_M\ L\ M)$
unfolding *prod_emb_def space_PiM*[*symmetric*]
by (*auto intro*!: *measurable_sets measurable_restrict measurable_component_singleton*)

lemma *merge_in_prod_emb*:
assumes $y \in space\ (PiM\ I\ M)\ x \in X$ **and** X: $X \in sets\ (Pi_M\ J\ M)$ **and** $J \subseteq I$
shows $merge\ J\ I\ (x, y) \in prod\_emb\ I\ M\ J\ X$
using *assms sets.sets_into_space*[*OF X*]
by (*simp add*: *merge_def prod_emb_def subset_eq space_PiM PiE_def extensional_restrict Pi_iff*
cong: *if_cong restrict_cong*)

(*simp add*: *extensional_def*)

lemma *prod_emb_eq_emptyD*:
assumes *J*: $J \subseteq I$ **and** *ne*: *space* (*PiM I M*) $\neq \{\}$ **and** *X*: $X \in$ *sets* (Pi_M *J M*)
and $*$: *prod_emb I M J X* $= \{\}$
shows $X = \{\}$
proof *safe*
fix x **assume** $x \in X$
obtain ω **where** $\omega \in$ *space* (*PiM I M*)
using *ne* **by** *blast*
from *merge_in_prod_emb*[*OF this* ‹$x \in X$› *X J*] $*$ **show** $x \in \{\}$ **by** *auto*
qed

lemma *sets_in_Pi_aux*:
finite I $\Longrightarrow$ ($\bigwedge j.\ j \in I \Longrightarrow \{x \in$ *space* (*M j*). $x \in F\ j\} \in$ *sets* (*M j*)) $\Longrightarrow$
$\{x \in$ *space* (*PiM I M*). $x \in$ *Pi I F*$\} \in$ *sets* (*PiM I M*)
by (*simp add*: *subset_eq Pi_iff*)

lemma *sets_in_Pi*[*measurable* (*raw*)]:
finite I $\Longrightarrow$ $f \in$ *measurable N* (*PiM I M*) $\Longrightarrow$
($\bigwedge j.\ j \in I \Longrightarrow \{x \in$ *space* (*M j*). $x \in F\ j\} \in$ *sets* (*M j*)) $\Longrightarrow$
Measurable.pred N ($\lambda x.\ f\ x \in$ *Pi I F*)
unfolding *pred_def*
by (*rule measurable_sets_Collect*[*of f N PiM I M*, *OF* _ *sets_in_Pi_aux*]) *auto*

lemma *sets_in_extensional_aux*:
$\{x \in$ *space* (*PiM I M*). $x \in$ *extensional I*$\} \in$ *sets* (*PiM I M*)
proof −
have $\{x \in$ *space* (*PiM I M*). $x \in$ *extensional I*$\}$ = *space* (*PiM I M*)
by (*auto simp add*: *extensional_def space_PiM*)
then show *?thesis* **by** *simp*
qed

lemma *sets_in_extensional*[*measurable* (*raw*)]:
$f \in$ *measurable N* (*PiM I M*) $\Longrightarrow$ *Measurable.pred N* ($\lambda x.\ f\ x \in$ *extensional I*)
unfolding *pred_def*
by (*rule measurable_sets_Collect*[*of f N PiM I M*, *OF* _ *sets_in_extensional_aux*])
auto

lemma *sets_PiM_I_countable*:
assumes *I*: *countable I* **and** *E*: $\bigwedge i.\ i \in I \Longrightarrow E\ i \in$ *sets* (*M i*) **shows** Pi_E *I E*
$\in$ *sets* (Pi_M *I M*)
proof *cases*
assume $I \neq \{\}$
then have Pi_E *I E* = ($\bigcap i \in I.$ *prod_emb I M* $\{i\}$ (Pi_E $\{i\}$ *E*))
using *E*[*THEN sets.sets_into_space*] **by** (*auto simp*: *PiE_iff prod_emb_def fun_eq_iff*)
also have ... $\in$ *sets* (*PiM I M*)
using *I* ‹$I \neq \{\}$› **by** (*safe intro*!: *sets.countable_INT'* *measurable_prod_emb*

sets_PiM_I_finite E)
finally show *?thesis* **.**
qed (*simp add*: *sets_PiM_empty*)

lemma *sets_PiM_D_countable*:
assumes *A*: $A \in PiM\ I\ M$
shows $\exists J{\subseteq}I.\ \exists X{\in}PiM\ J\ M.\ countable\ J \wedge A = prod\_emb\ I\ M\ J\ X$
using *A*[*unfolded sets_PiM_single*]
proof *induction*
case (*Basic A*)
then obtain *i X* **where** ∗: $i \in I\ X \in sets\ (M\ i)$ **and** $A = \{f \in \Pi_E\ i{\in}I.\ space\ (M\ i).\ f\ i \in X\}$
by *auto*
then have *A*: $A = prod\_emb\ I\ M\ \{i\}\ (\Pi_E\ \_{\in}\{i\}.\ X)$
by (*auto simp*: *prod_emb_def*)
then show *?case*
by (*intro exI*[*of* _ $\{i\}$] *conjI bexI*[*of* _ $\Pi_E\ \_{\in}\{i\}.\ X$])
(*auto intro*: *countable_finite* ∗ *sets_PiM_I_finite*)
next
case *Empty* **then show** *?case*
by (*intro exI*[*of* _ {}] *conjI bexI*[*of* _ {}]) *auto*
next
case (*Compl A*)
then obtain *J X* **where** $J \subseteq I\ X \in sets\ (Pi_M\ J\ M)\ countable\ J\ A = prod\_emb\ I\ M\ J\ X$
by *auto*
then show *?case*
by (*intro exI*[*of* _ *J*] *bexI*[*of* _ *space* (*PiM J M*) − *X*] *conjI*)
(*auto simp add*: *space_PiM prod_emb_PiE intro*!: *sets_PiM_I_countable*)
next
case (*Union K*)
obtain *J X* **where** *J*: $\bigwedge i.\ J\ i \subseteq I\ \bigwedge i.\ countable\ (J\ i)$ **and** *X*: $\bigwedge i.\ X\ i \in sets\ (Pi_M\ (J\ i)\ M)$
and *K*: $\bigwedge i.\ K\ i = prod\_emb\ I\ M\ (J\ i)\ (X\ i)$
by (*metis Union.IH*)
show *?case*
proof (*intro exI*[*of* _ $\bigcup i.\ J\ i$] *bexI*[*of* _ $\bigcup i.$ *prod_emb* $(\bigcup i.\ J\ i)\ M\ (J\ i)\ (X\ i)$] *conjI*)
show $(\bigcup i.\ J\ i) \subseteq I$ *countable* $(\bigcup i.\ J\ i)$ **using** *J* **by** *auto*
with *J* **show** $\bigcup (K\ `\ UNIV) = prod\_emb\ I\ M\ (\bigcup i.\ J\ i)\ (\bigcup i.\ prod\_emb\ (\bigcup i.\ J\ i)\ M\ (J\ i)\ (X\ i))$
by (*simp add*: *K*[*abs_def*] *SUP_upper*)
qed(*auto intro*: *X*)
qed

proposition *measure_eqI_PiM_finite*:
assumes [*simp*]: *finite I sets P* = *PiM I M sets Q* = *PiM I M*
assumes *eq*: $\bigwedge A.\ (\bigwedge i.\ i \in I \Longrightarrow A\ i \in sets\ (M\ i)) \Longrightarrow P\ (Pi_E\ I\ A) = Q\ (Pi_E\ I\ A)$

assumes *A*: *range A* ⊆ *prod_algebra I M* (⋃ *i*. *A i*) = *space* (*PiM I M*) ⋀*i*::*nat*. *P* (*A i*) ≠ ∞
shows *P* = *Q*
proof (*rule measure_eqI_generator_eq*[*OF Int_stable_prod_algebra prod_algebra_sets_into_space*])
show *range A* ⊆ *prod_algebra I M* (⋃ *i*. *A i*) = (Π$_E$ *i*∈*I*. *space* (*M i*)) ⋀*i*. *P* (*A i*) ≠ ∞
unfolding *space_PiM*[*symmetric*] **by** *fact*+
fix *X* **assume** *X* ∈ *prod_algebra I M*
then obtain *J E* **where** *X*: *X* = *prod_emb I M J* (Π$_E$ *j*∈*J*. *E j*)
and *J*: *finite J J* ⊆ *I* ⋀*j*. *j* ∈ *J* ⟹ *E j* ∈ *sets* (*M j*)
by (*force elim*!: *prod_algebraE*)
then show *emeasure P X* = *emeasure Q X*
unfolding *X* **by** (*subst* (*1 2*) *prod_emb_Pi*) (*auto simp*: *eq*)
qed (*simp_all add*: *sets_PiM*)

proposition *measure_eqI_PiM_infinite*:
assumes [*simp*]: *sets P* = *PiM I M sets Q* = *PiM I M*
assumes *eq*: ⋀*A J*. *finite J* ⟹ *J* ⊆ *I* ⟹ (⋀*i*. *i* ∈ *J* ⟹ *A i* ∈ *sets* (*M i*)) ⟹
P (*prod_emb I M J* (*Pi*$_E$ *J A*)) = *Q* (*prod_emb I M J* (*Pi*$_E$ *J A*))
assumes *A*: *finite_measure P*
shows *P* = *Q*
proof (*rule measure_eqI_generator_eq*[*OF Int_stable_prod_algebra prod_algebra_sets_into_space*])
interpret *finite_measure P* **by** *fact*
define *i* **where** *i* = (*SOME i*. *i* ∈ *I*)
have *i*: *I* ≠ {} ⟹ *i* ∈ *I*
unfolding *i_def* **by** (*rule someI_ex*) *auto*
define *A* **where** *A n* =
(*if I* = {} *then prod_emb I M* {} (Π$_E$ *i*∈{}. {}) *else prod_emb I M* {*i*} (Π$_E$ *i*∈{*i*}. *space* (*M i*)))
for *n* :: *nat*
then show *range A* ⊆ *prod_algebra I M*
using *prod_algebraI*[*of* {} *I* λ*i*. *space* (*M i*) *M*] **by** (*auto intro*!: *prod_algebraI i*)
have ⋀*i*. *A i* = *space* (*PiM I M*)
by (*auto simp*: *prod_emb_def space_PiM PiE_iff A_def i ex_in_conv*[*symmetric*] *exI*)
then show (⋃ *i*. *A i*) = (Π$_E$ *i*∈*I*. *space* (*M i*)) ⋀*i*. *emeasure P* (*A i*) ≠ ∞
by (*auto simp*: *space_PiM*)
next
fix *X* **assume** *X*: *X* ∈ *prod_algebra I M*
then obtain *J E* **where** *X*: *X* = *prod_emb I M J* (Π$_E$ *j*∈*J*. *E j*)
and *J*: *finite J J* ⊆ *I* ⋀*j*. *j* ∈ *J* ⟹ *E j* ∈ *sets* (*M j*)
by (*force elim*!: *prod_algebraE*)
then show *emeasure P X* = *emeasure Q X*
by (*auto intro*!: *eq*)
qed (*auto simp*: *sets_PiM*)

locale *product_sigma_finite* =

fixes $M :: 'i \Rightarrow 'a\ measure$
assumes *sigma_finite_measures*: $\bigwedge i.\ sigma\_finite\_measure\ (M\ i)$

sublocale *product_sigma_finite* $\subseteq$ M*?*: *sigma_finite_measure* $M\ i$ **for** i
by (*rule sigma_finite_measures*)

locale *finite_product_sigma_finite* = *product_sigma_finite* M **for** $M :: 'i \Rightarrow 'a$ *measure* +
fixes $I :: 'i\ set$
assumes *finite_index*: *finite* I

proposition (**in** *finite_product_sigma_finite*) *sigma_finite_pairs*:
$\exists F::'i \Rightarrow nat \Rightarrow 'a\ set.$
$(\forall i{\in}I.\ range\ (F\ i) \subseteq sets\ (M\ i)) \wedge$
$(\forall k.\ \forall i{\in}I.\ emeasure\ (M\ i)\ (F\ i\ k) \neq \infty) \wedge incseq\ (\lambda k.\ \Pi_E\ i{\in}I.\ F\ i\ k) \wedge$
$(\bigcup k.\ \Pi_E\ i{\in}I.\ F\ i\ k) = space\ (PiM\ I\ M)$
proof −
have $\forall i::'i.\ \exists F::nat \Rightarrow 'a\ set.\ range\ F \subseteq sets\ (M\ i) \wedge incseq\ F \wedge (\bigcup i.\ F\ i) = space\ (M\ i) \wedge (\forall k.\ emeasure\ (M\ i)\ (F\ k) \neq \infty)$
using *M.sigma_finite_incseq* **by** *metis*
then obtain $F :: 'i \Rightarrow nat \Rightarrow 'a\ set$
where $\forall x.\ range\ (F\ x) \subseteq sets\ (M\ x) \wedge incseq\ (F\ x) \wedge \bigcup\ (range\ (F\ x)) = space\ (M\ x) \wedge (\forall k.\ emeasure\ (M\ x)\ (F\ x\ k) \neq \infty)$
by *metis*
then have F: $\bigwedge i.\ range\ (F\ i) \subseteq sets\ (M\ i)\ \bigwedge i.\ incseq\ (F\ i)\ \bigwedge i.\ (\bigcup j.\ F\ i\ j) = space\ (M\ i)\ \bigwedge i\ k.\ emeasure\ (M\ i)\ (F\ i\ k) \neq \infty$
by *auto*
let *?F* $= \lambda k.\ \Pi_E\ i{\in}I.\ F\ i\ k$
note *space_PiM*[*simp*]
show *?thesis*
proof (*intro exI*[*of* _ F] *conjI allI incseq_SucI set_eqI iffI ballI*)
fix i **show** $range\ (F\ i) \subseteq sets\ (M\ i)$ **by** *fact*
next
fix $i\ k$ **show** $emeasure\ (M\ i)\ (F\ i\ k) \neq \infty$ **by** *fact*
next
fix x **assume** $x \in (\bigcup i.$ *?F* $i)$ **with** $F(1)$ **show** $x \in space\ (PiM\ I\ M)$
by (*auto simp*: *PiE_def dest*!: *sets.sets_into_space*)
next
fix f **assume** $f \in space\ (PiM\ I\ M)$
with *Pi_UN*[*OF finite_index*, *of* $\lambda k\ i.\ F\ i\ k$] F
show $f \in (\bigcup i.$ *?F* $i)$ **by** (*auto simp*: *incseq_def PiE_def*)
next
fix i **show** *?F* $i \subseteq$ *?F* $(Suc\ i)$
using ‹$\bigwedge i.\ incseq\ (F\ i)$›[*THEN incseq_SucD*] **by** *auto*
qed
qed

lemma *emeasure_PiM_empty*[*simp*]: $emeasure\ (PiM\ \{\}\ M)\ \{\lambda\_.\ undefined\} = 1$
proof −

let *?μ* = λ*A*. *if A* = {} *then 0 else* (*1*::*ennreal*)
have *emeasure* (*Pi*$_M$ {} *M*) (*prod_emb* {} *M* {} (Π$_E$ *i*∈{}. {})) = *1*
proof (*subst emeasure_extend_measure_Pair*[*OF PiM_def*])
show *positive* (*PiM* {} *M*) *?μ*
by (*auto simp*: *positive_def*)
show *countably_additive* (*PiM* {} *M*) *?μ*
by (*rule sets.countably_additiveI_finite*)
(*auto simp*: *additive_def positive_def sets_PiM_empty space_PiM_empty intro*!:)
qed (*auto simp*: *prod_emb_def*)
also have (*prod_emb* {} *M* {} (Π$_E$ *i*∈{}. {})) = {λ_. *undefined*}
by (*auto simp*: *prod_emb_def*)
finally show *?thesis*
by *simp*
qed

lemma *PiM_empty*: *PiM* {} *M* = *count_space* {λ_. *undefined*}
by (*rule measure_eqI*) (*auto simp add*: *sets_PiM_empty*)

lemma (**in** *product_sigma_finite*) *emeasure_PiM*:
finite I ⟹ (⋀*i*. *i*∈*I* ⟹ *A i* ∈ *sets* (*M i*)) ⟹ *emeasure* (*PiM I M*) (*Pi*$_E$ *I A*) = (∏ *i*∈*I*. *emeasure* (*M i*) (*A i*))
proof (*induct I arbitrary*: *A rule*: *finite_induct*)
case (*insert i I*)
interpret *finite_product_sigma_finite M I* **by** *standard fact*
have *finite* (*insert i I*) **using** ‹*finite I*› **by** *auto*
interpret *I*′: *finite_product_sigma_finite M insert i I* **by** *standard fact*
let *?h* = (λ(*f*, *y*). *f*(*i* := *y*))

let *?P* = *distr* (*Pi*$_M$ *I M* ⨂$_M$ *M i*) (*Pi*$_M$ (*insert i I*) *M*) *?h*
let *?μ* = *emeasure ?P*
let *?I* = {*j* ∈ *insert i I*. *emeasure* (*M j*) (*space* (*M j*)) ≠ *1*}
let *?f* = λ*J E j*. *if j* ∈ *J then emeasure* (*M j*) (*E j*) *else emeasure* (*M j*) (*space* (*M j*))

have *emeasure* (*Pi*$_M$ (*insert i I*) *M*) (*prod_emb* (*insert i I*) *M* (*insert i I*) (*Pi*$_E$ (*insert i I*) *A*)) =
(∏ *i*∈*insert i I*. *emeasure* (*M i*) (*A i*))
proof (*subst emeasure_extend_measure_Pair*[*OF PiM_def*])
fix *J E* **assume** (*J* ≠ {} ∨ *insert i I* = {}) ∧ *finite J* ∧ *J* ⊆ *insert i I* ∧ *E* ∈ (Π *j*∈*J*. *sets* (*M j*))
then have *J*: *J* ≠ {} *finite J J* ⊆ *insert i I* **and** *E*: ∀*j*∈*J*. *E j* ∈ *sets* (*M j*) **by** *auto*
let *?p* = *prod_emb* (*insert i I*) *M J* (*Pi*$_E$ *J E*)
let *?p*′ = *prod_emb I M* (*J* − {*i*}) (Π$_E$ *j*∈*J*−{*i*}. *E j*)
have *?μ ?p* =
emeasure (*Pi*$_M$ *I M* ⨂$_M$ (*M i*)) (*?h* −‘ *?p* ∩ *space* (*Pi*$_M$ *I M* ⨂$_M$ *M i*))
by (*intro emeasure_distr measurable_add_dim sets_PiM_I*) *fact*+
also have *?h* −‘ *?p* ∩ *space* (*Pi*$_M$ *I M* ⨂$_M$ *M i*) = *?p*′ × (*if i* ∈ *J then E i*

else space (*M i*))
using *J E*[*rule_format*, *THEN sets.sets_into_space*]
by (*force simp*: *space_pair_measure space_PiM prod_emb_iff PiE_def Pi_iff split*: *if_split_asm*)
also have *emeasure* (*Pi*$_M$ *I M* ⨂$_M$ (*M i*)) (*?p*′ × (*if i* ∈ *J then E i else space* (*M i*))) =
emeasure (*Pi*$_M$ *I M*) *?p*′ ∗ *emeasure* (*M i*) (*if i* ∈ *J then* (*E i*) *else space* (*M i*))
using *J E* **by** (*intro M.emeasure_pair_measure_Times sets_PiM_I*) *auto*
also have *?p*′ = (Π$_E$ *j*∈*I*. *if j* ∈ *J*−{*i*} *then E j else space* (*M j*))
using *J E*[*rule_format*, *THEN sets.sets_into_space*]
by (*auto simp*: *prod_emb_iff PiE_def Pi_iff split*: *if_split_asm*) *blast*+
also have *emeasure* (*Pi*$_M$ *I M*) (Π$_E$ *j*∈*I*. *if j* ∈ *J*−{*i*} *then E j else space* (*M j*)) =
(∏ *j*∈*I*. *if j* ∈ *J*−{*i*} *then emeasure* (*M j*) (*E j*) *else emeasure* (*M j*) (*space* (*M j*)))
using *E* **by** (*subst insert*) (*auto intro*!: *prod.cong*)
also have (∏*j*∈*I*. *if j* ∈ *J* − {*i*} *then emeasure* (*M j*) (*E j*) *else emeasure* (*M j*) (*space* (*M j*))) ∗
emeasure (*M i*) (*if i* ∈ *J then E i else space* (*M i*)) = (∏*j*∈*insert i I*. *?f J E j*)
using *insert* **by** (*auto simp*: *mult.commute intro*!: *arg_cong2*[**where** *f*=(∗)] *prod.cong*)
also have ... = (∏*j*∈*J* ∪ *?I*. *?f J E j*)
using *insert*(*1*,*2*) *J E* **by** (*intro prod.mono_neutral_right*) *auto*
finally show *?μ ?p* = ... **.**

show *prod_emb* (*insert i I*) *M J* (*Pi*$_E$ *J E*) ∈ *Pow* (Π$_E$ *i*∈*insert i I*. *space* (*M i*))
using *J E*[*rule_format*, *THEN sets.sets_into_space*] **by** (*auto simp*: *prod_emb_iff PiE_def*)
next
show *positive* (*sets* (*Pi*$_M$ (*insert i I*) *M*)) *?μ countably_additive* (*sets* (*Pi*$_M$ (*insert i I*) *M*)) *?μ*
using *emeasure_positive*[*of ?P*] *emeasure_countably_additive*[*of ?P*] **by** *simp_all*
next
show (*insert i I* ≠ {} ∨ *insert i I* = {}) ∧ *finite* (*insert i I*) ∧
insert i I ⊆ *insert i I* ∧ *A* ∈ (Π *j*∈*insert i I*. *sets* (*M j*))
using *insert* **by** *auto*
qed (*auto intro*!: *prod.cong*)
with *insert* **show** *?case*
by (*subst* (*asm*) *prod_emb_PiE_same_index*) (*auto intro*!: *sets.sets_into_space*)
qed *simp*

lemma (**in** *product_sigma_finite*) *PiM_eqI*:
assumes *I*[*simp*]: *finite I* **and** *P*: *sets P* = *PiM I M*
assumes *eq*: ⋀*A*. (⋀*i*. *i* ∈ *I* ⟹ *A i* ∈ *sets* (*M i*)) ⟹ *P* (*Pi*$_E$ *I A*) = (∏ *i*∈*I*. *emeasure* (*M i*) (*A i*))

shows $P = PiM\ I\ M$
proof −
interpret *finite_product_sigma_finite M I*
by *standard fact*
from *sigma_finite_pairs* **obtain** C **where** C:
$\forall i{\in}I.\ range\ (C\ i) \subseteq sets\ (M\ i)\ \forall k.\ \forall i{\in}I.\ emeasure\ (M\ i)\ (C\ i\ k) \neq \infty$
$incseq\ (\lambda k.\ \Pi_E\ i{\in}I.\ C\ i\ k)\ (\bigcup k.\ \Pi_E\ i{\in}I.\ C\ i\ k) = space\ (Pi_M\ I\ M)$
by *auto*
show *?thesis*
proof (*rule measure_eqI_PiM_finite*[*OF I refl P, symmetric*])
show $(\bigwedge i.\ i \in I \Longrightarrow A\ i \in sets\ (M\ i)) \Longrightarrow (Pi_M\ I\ M)\ (Pi_E\ I\ A) = P\ (Pi_E\ I\ A)$ **for** A
by (*simp add*: *eq emeasure_PiM*)
define A **where** $A\ n = (\Pi_E\ i{\in}I.\ C\ i\ n)$ **for** n
with C **show** $range\ A \subseteq prod\_algebra\ I\ M\ \bigwedge i.\ emeasure\ (Pi_M\ I\ M)\ (A\ i) \neq \infty\ (\bigcup i.\ A\ i) = space\ (PiM\ I\ M)$
by (*auto intro*!: *prod_algebraI_finite simp*: *emeasure_PiM subset_eq ennreal_prod_eq_top*)
qed
qed

lemma (**in** *product_sigma_finite*) *sigma_finite*:
assumes *finite I*
shows *sigma_finite_measure* (*PiM I M*)
proof
interpret *finite_product_sigma_finite M I* **by** *standard fact*

obtain F **where** F: $\bigwedge j.\ countable\ (F\ j)\ \bigwedge j\ f.\ f \in F\ j \Longrightarrow f \in sets\ (M\ j)$
$\bigwedge j\ f.\ f \in F\ j \Longrightarrow emeasure\ (M\ j)\ f \neq \infty$ **and**
in_space: $\bigwedge j.\ space\ (M\ j) = \bigcup (F\ j)$
using *sigma_finite_countable* **by** (*metis subset_eq*)
moreover have $(\bigcup (Pi_E\ I\ `\ Pi_E\ I\ F)) = space\ (Pi_M\ I\ M)$
using *in_space* **by** (*auto simp*: *space_PiM PiE_iff intro*!: *PiE_choice*[*THEN iffD2*])
ultimately show $\exists A.\ countable\ A \wedge A \subseteq sets\ (Pi_M\ I\ M) \wedge \bigcup A = space\ (Pi_M\ I\ M) \wedge (\forall a{\in}A.\ emeasure\ (Pi_M\ I\ M)\ a \neq \infty)$
by (*intro exI*[*of* _ $Pi_E\ I\ `\ Pi_E\ I\ F$])
(*auto intro*!: *countable_PiE sets_PiM_I_finite*
simp: *PiE_iff emeasure_PiM finite_index ennreal_prod_eq_top*)
qed

sublocale *finite_product_sigma_finite* ⊆ *sigma_finite_measure* $Pi_M\ I\ M$
using *sigma_finite*[*OF finite_index*] **.**

lemma (**in** *finite_product_sigma_finite*) *measure_times*:
$(\bigwedge i.\ i \in I \Longrightarrow A\ i \in sets\ (M\ i)) \Longrightarrow emeasure\ (Pi_M\ I\ M)\ (Pi_E\ I\ A) = (\prod i{\in}I.\ emeasure\ (M\ i)\ (A\ i))$
using *emeasure_PiM*[*OF finite_index*] **by** *auto*

lemma (**in** *product_sigma_finite*) *nn_integral_empty*:
  $0 \leq f\ (\lambda k.\ undefined) \Longrightarrow integral^N\ (Pi_M\ \{\}\ M)\ f = f\ (\lambda k.\ undefined)$
  by (*simp add*: *PiM_empty nn_integral_count_space_finite max.absorb2*)

lemma (**in** *product_sigma_finite*) *distr_merge*:
  assumes *IJ*[*simp*]: $I \cap J = \{\}$ **and** *fin*: *finite I finite J*
  shows $distr\ (Pi_M\ I\ M \bigotimes_M Pi_M\ J\ M)\ (Pi_M\ (I \cup J)\ M)\ (merge\ I\ J) = Pi_M\ (I \cup J)\ M$
    (**is** *?D* = *?P*)
proof (*rule PiM_eqI*)
  interpret *I*: *finite_product_sigma_finite M I* **by** *standard fact*
  interpret *J*: *finite_product_sigma_finite M J* **by** *standard fact*
  fix *A* **assume** *A*: $\bigwedge i.\ i \in I \cup J \Longrightarrow A\ i \in sets\ (M\ i)$
  have ∗: $(merge\ I\ J\ -`\ Pi_E\ (I \cup J)\ A \cap space\ (Pi_M\ I\ M \bigotimes_M Pi_M\ J\ M)) = Pi_E\ I\ A \times Pi_E\ J\ A$
    using *A*[*THEN sets.sets_into_space*] **by** (*auto simp*: *space_PiM space_pair_measure*)
  from *A fin* **show** $emeasure\ (distr\ (Pi_M\ I\ M \bigotimes_M Pi_M\ J\ M)\ (Pi_M\ (I \cup J)\ M)\ (merge\ I\ J))\ (Pi_E\ (I \cup J)\ A) =$
      $(\prod i \in I \cup J.\ emeasure\ (M\ i)\ (A\ i))$
    by (*subst emeasure_distr*)
      (*auto simp*: ∗ *J.emeasure_pair_measure_Times I.measure_times J.measure_times prod.union_disjoint*)
qed (*insert fin*, *simp_all*)

proposition (**in** *product_sigma_finite*) *product_nn_integral_fold*:
  assumes *IJ*: $I \cap J = \{\}$ *finite I finite J*
  and *f*[*measurable*]: $f \in borel\_measurable\ (Pi_M\ (I \cup J)\ M)$
  shows $integral^N\ (Pi_M\ (I \cup J)\ M)\ f =$
    $(\int^+ x.\ (\int^+ y.\ f\ (merge\ I\ J\ (x, y))\ \partial(Pi_M\ J\ M))\ \partial(Pi_M\ I\ M))$
proof −
  interpret *I*: *finite_product_sigma_finite M I* **by** *standard fact*
  interpret *J*: *finite_product_sigma_finite M J* **by** *standard fact*
  interpret *P*: *pair_sigma_finite* $Pi_M\ I\ M\ Pi_M\ J\ M$ **by** *standard*
  have *P_borel*: $(\lambda x.\ f\ (merge\ I\ J\ x)) \in borel\_measurable\ (Pi_M\ I\ M \bigotimes_M Pi_M\ J\ M)$
    using *measurable_comp*[*OF measurable_merge f*] **by** (*simp add*: *comp_def*)
  show *?thesis*
    apply (*subst distr_merge*[*OF IJ*, *symmetric*])
    apply (*subst nn_integral_distr*[*OF measurable_merge*])
    apply *measurable* []
    apply (*subst J.nn_integral_fst*[*symmetric*, *OF P_borel*])
    apply *simp*
    done
qed

lemma (**in** *product_sigma_finite*) *distr_singleton*:
  $distr\ (Pi_M\ \{i\}\ M)\ (M\ i)\ (\lambda x.\ x\ i) = M\ i$ (**is** *?D* = _)
proof (*intro measure_eqI*[*symmetric*])
  interpret *I*: *finite_product_sigma_finite M* $\{i\}$ **by** *standard simp*

```
  fix A assume A: A ∈ sets (M i)
  then have (λx. x i) -` A ∩ space (Pi_M {i} M) = (Π_E i∈{i}. A)
    using sets.sets_into_space by (auto simp: space_PiM)
  then show emeasure (M i) A = emeasure ?D A
    using A I.measure_times[of λ_. A]
    by (simp add: emeasure_distr measurable_component_singleton)
qed simp

lemma (in product_sigma_finite) product_nn_integral_singleton:
  assumes f: f ∈ borel_measurable (M i)
  shows integral^N (Pi_M {i} M) (λx. f (x i)) = integral^N (M i) f
proof -
  interpret I: finite_product_sigma_finite M {i} by standard simp
  from f show ?thesis
    apply (subst distr_singleton[symmetric])
    apply (subst nn_integral_distr[OF measurable_component_singleton])
    apply simp_all
    done
qed

proposition (in product_sigma_finite) product_nn_integral_insert:
  assumes I[simp]: finite I i ∉ I
    and f: f ∈ borel_measurable (Pi_M (insert i I) M)
  shows integral^N (Pi_M (insert i I) M) f = (∫+ x. (∫+ y. f (x(i := y)) ∂(M i))
∂(Pi_M I M))
proof -
  interpret I: finite_product_sigma_finite M I by standard auto
  interpret i: finite_product_sigma_finite M {i} by standard auto
  have IJ: I ∩ {i} = {} and insert: I ∪ {i} = insert i I
    using f by auto
  show ?thesis
   unfolding product_nn_integral_fold[OF IJ, unfolded insert, OF I(1) i.finite_index
f]
  proof (rule nn_integral_cong, subst product_nn_integral_singleton[symmetric])
    fix x assume x: x ∈ space (Pi_M I M)
    let ?f = λy. f (x(i := y))
    show ?f ∈ borel_measurable (M i)
      using measurable_comp[OF measurable_component_update f, OF x ‹i ∉ I›]
      unfolding comp_def .
    show (∫+ y. f (merge I {i} (x, y)) ∂Pi_M {i} M) = (∫+ y. f (x(i := y i))
∂Pi_M {i} M)
      using x
      by (auto intro!: nn_integral_cong arg_cong[where f=f]
            simp add: space_PiM extensional_def PiE_def)
  qed
qed

lemma (in product_sigma_finite) product_nn_integral_insert_rev:
  assumes I[simp]: finite I i ∉ I
```

and [*measurable*]: $f \in$ *borel_measurable* (Pi_M (*insert i I*) *M*)
shows *integral*^{N} (Pi_M (*insert i I*) *M*) *f* = ($\int^+$ *y*. ($\int^+$ *x*. *f* (*x*(*i* := *y*)) ∂(Pi_M *I M*)) ∂(*M i*))
apply (*subst product_nn_integral_insert*[*OF assms*])
apply (*rule pair_sigma_finite.Fubini'*)
apply *intro_locales* []
apply (*rule sigma_finite*[*OF I*(*1*)])
apply *measurable*
done

lemma (**in** *product_sigma_finite*) *product_nn_integral_prod*:
assumes *finite I* ⋀*i*. *i* ∈ *I* ⟹ *f i* ∈ *borel_measurable* (*M i*)
shows ($\int^+$ *x*. ($\prod$ *i*∈*I*. *f i* (*x i*)) ∂Pi_M *I M*) = ($\prod$ *i*∈*I*. *integral*^{N} (*M i*) (*f i*))
using *assms* **proof** (*induction I*)
case (*insert i I*)
note *insert.prems*[*measurable*]
note ‹*finite I*›[*intro*, *simp*]
interpret *I*: *finite_product_sigma_finite M I* **by** *standard auto*
have ∗: ⋀*x y*. ($\prod$ *j*∈*I*. *f j* (*if j* = *i then y else x j*)) = ($\prod$ *j*∈*I*. *f j* (*x j*))
using *insert* **by** (*auto intro*!: *prod.cong*)
have *prod*: ⋀*J*. *J* ⊆ *insert i I* ⟹ (λ*x*. ($\prod$ *i*∈*J*. *f i* (*x i*))) ∈ *borel_measurable* (Pi_M *J M*)
using *sets.sets_into_space insert*
by (*intro borel_measurable_prod_ennreal*
measurable_comp[*OF measurable_component_singleton*, *unfolded comp_def*])
auto
then show *?case*
apply (*simp add*: *product_nn_integral_insert*[*OF insert*(*1*,*2*)])
apply (*simp add*: *insert*(*2*−) ∗ *nn_integral_multc*)
apply (*subst nn_integral_cmult*)
apply (*auto simp add*: *insert*(*2*−))
done
qed (*simp add*: *space_PiM*)

proposition (**in** *product_sigma_finite*) *product_nn_integral_pair*:
assumes [*measurable*]: *case_prod f* ∈ *borel_measurable* (*M x* ⨂$_M$ *M y*)
assumes *xy*: *x* ≠ *y*
shows ($\int^+\sigma$. *f* (*σ x*) (*σ y*) ∂*PiM* {*x*, *y*} *M*) = ($\int^+z$. *f* (*fst z*) (*snd z*) ∂(*M x* ⨂$_M$ *M y*))
proof −
interpret *psm*: *pair_sigma_finite M x M y*
unfolding *pair_sigma_finite_def* **using** *sigma_finite_measures* **by** *simp_all*
have {*x*, *y*} = {*y*, *x*} **by** *auto*
also have ($\int^+\sigma$. *f* (*σ x*) (*σ y*) ∂*PiM* {*y*, *x*} *M*) = ($\int^+y$. $\int^+\sigma$. *f* (*σ x*) *y* ∂*PiM* {*x*} *M* ∂*M y*)
using *xy* **by** (*subst product_nn_integral_insert_rev*) *simp_all*
also have ... = ($\int^+y$. $\int^+x$. *f x y* ∂*M x* ∂*M y*)
by (*intro nn_integral_cong*, *subst product_nn_integral_singleton*) *simp_all*

also have ... = $(\int^+ z.\ f\ (fst\ z)\ (snd\ z)\ \partial(M\ x \bigotimes_M M\ y))$
by (*subst psm.nn_integral_snd*[*symmetric*]) *simp_all*
finally show *?thesis* **.**
qed

lemma (**in** *product_sigma_finite*) *distr_component*:
$distr\ (M\ i)\ (Pi_M\ \{i\}\ M)\ (\lambda x.\ \lambda i{\in}\{i\}.\ x) = Pi_M\ \{i\}\ M$ (**is** *?D* = *?P*)
proof (*intro PiM_eqI*)
fix *A* **assume** *A*: $\bigwedge ia.\ ia \in \{i\} \Longrightarrow A\ ia \in sets\ (M\ ia)$
then have $(\lambda x.\ \lambda i{\in}\{i\}.\ x) -` Pi_E\ \{i\}\ A \cap space\ (M\ i) = A\ i$
by (*fastforce dest*: *sets.sets_into_space*)
with *A* **show** $emeasure\ (distr\ (M\ i)\ (Pi_M\ \{i\}\ M)\ (\lambda x.\ \lambda i{\in}\{i\}.\ x))\ (Pi_E\ \{i\}\ A) = (\prod i{\in}\{i\}.\ emeasure\ (M\ i)\ (A\ i))$
by (*subst emeasure_distr*) (*auto intro*!: *sets_PiM_I_finite measurable_restrict*)
qed *simp_all*

lemma (**in** *product_sigma_finite*)
assumes *IJ*: $I \cap J = \{\}$ *finite I finite J* **and** *A*: $A \in sets\ (Pi_M\ (I \cup J)\ M)$
shows *emeasure_fold_integral*:
$emeasure\ (Pi_M\ (I \cup J)\ M)\ A = (\int^+ x.\ emeasure\ (Pi_M\ J\ M)\ ((\lambda y.\ merge\ I\ J\ (x,\ y)) -` A \cap space\ (Pi_M\ J\ M))\ \partial Pi_M\ I\ M)$ (**is** *?I*)
and *emeasure_fold_measurable*:
$(\lambda x.\ emeasure\ (Pi_M\ J\ M)\ ((\lambda y.\ merge\ I\ J\ (x,\ y)) -` A \cap space\ (Pi_M\ J\ M))) \in borel_measurable\ (Pi_M\ I\ M)$ (**is** *?B*)
proof −
interpret *I*: *finite_product_sigma_finite M I* **by** *standard fact*
interpret *J*: *finite_product_sigma_finite M J* **by** *standard fact*
interpret *IJ*: *pair_sigma_finite* $Pi_M\ I\ M\ Pi_M\ J\ M$ **..**
have *merge*: $merge\ I\ J -` A \cap space\ (Pi_M\ I\ M \bigotimes_M Pi_M\ J\ M) \in sets\ (Pi_M\ I\ M \bigotimes_M Pi_M\ J\ M)$
by (*intro measurable_sets*[*OF* _ *A*] *measurable_merge assms*)

show *?I*
apply (*subst distr_merge*[*symmetric*, *OF IJ*])
apply (*subst emeasure_distr*[*OF measurable_merge A*])
apply (*subst J.emeasure_pair_measure_alt*[*OF merge*])
apply (*auto intro*!: *nn_integral_cong arg_cong2*[**where** *f*=*emeasure*] *simp*: *space_pair_measure*)
done

show *?B*
using *IJ.measurable_emeasure_Pair1*[*OF merge*]
by (*simp add*: *vimage_comp comp_def space_pair_measure cong*: *measurable_cong*)
qed

lemma *sets_Collect_single*:
$i \in I \Longrightarrow A \in sets\ (M\ i) \Longrightarrow \{\ x \in space\ (Pi_M\ I\ M).\ x\ i \in A\ \} \in sets\ (Pi_M\ I\ M)$

by *simp*

lemma *pair_measure_eq_distr_PiM*:
fixes *M1* :: *'a measure* **and** *M2* :: *'a measure*
assumes *sigma_finite_measure M1 sigma_finite_measure M2*
shows (*M1* $\bigotimes_M$ *M2*) = *distr* (*Pi*$_M$ *UNIV* (*case_bool M1 M2*)) (*M1* $\bigotimes_M$ *M2*) (λ*x*. (*x True*, *x False*))
(**is** *?P* = *?D*)
proof (*rule pair_measure_eqI*[*OF assms*])
interpret *B*: *product_sigma_finite case_bool M1 M2*
unfolding *product_sigma_finite_def* **using** *assms* **by** (*auto split*: *bool.split*)
let *?B* = *Pi*$_M$ *UNIV* (*case_bool M1 M2*)

have [*simp*]: *fst* ∘ (λ*x*. (*x True*, *x False*)) = (λ*x*. *x True*) *snd* ∘ (λ*x*. (*x True*, *x False*)) = (λ*x*. *x False*)
by *auto*
fix *A B* **assume** *A*: *A* ∈ *sets M1* **and** *B*: *B* ∈ *sets M2*
have *emeasure M1 A* ∗ *emeasure M2 B* = (∏ *i*∈*UNIV*. *emeasure* (*case_bool M1 M2 i*) (*case_bool A B i*))
by (*simp add*: *UNIV_bool ac_simps*)
also have ... = *emeasure ?B* (*Pi*$_E$ *UNIV* (*case_bool A B*))
using *A B* **by** (*subst B.emeasure_PiM*) (*auto split*: *bool.split*)
also have *Pi*$_E$ *UNIV* (*case_bool A B*) = (λ*x*. (*x True*, *x False*)) −‘ (*A* × *B*) ∩ *space ?B*
using *A*[*THEN sets.sets_into_space*] *B*[*THEN sets.sets_into_space*]
by (*auto simp*: *PiE_iff all_bool_eq space_PiM split*: *bool.split*)
finally show *emeasure M1 A* ∗ *emeasure M2 B* = *emeasure ?D* (*A* × *B*)
using *A B*
measurable_component_singleton[*of True UNIV case_bool M1 M2*]
measurable_component_singleton[*of False UNIV case_bool M1 M2*]
by (*subst emeasure_distr*) (*auto simp*: *measurable_pair_iff*)
qed *simp*

lemma *infprod_in_sets*[*intro*]:
fixes *E* :: *nat* ⇒ *'a set* **assumes** *E*: ⋀*i*. *E i* ∈ *sets* (*M i*)
shows *Pi UNIV E* ∈ *sets* (Π$_M$ *i*∈*UNIV*::*nat set*. *M i*)
proof −
have *Pi UNIV E* = (⋂ *i*. *prod_emb UNIV M* {..*i*} (Π$_E$ *j*∈{..*i*}. *E j*))
using *E E*[*THEN sets.sets_into_space*]
by (*auto simp*: *prod_emb_def Pi_iff extensional_def*)
with *E* **show** *?thesis* **by** *auto*
qed

6.7.3 Measurability

There are two natural sigma-algebras on a product space: the borel sigma algebra, generated by open sets in the product, and the product sigma algebra, countably generated by products of measurable sets along finitely many coordinates. The second one is defined and studied in `Finite_Product_Measure.thy`.

These sigma-algebra share a lot of natural properties (measurability of coordinates, for instance), but there is a fundamental difference: open sets are generated by arbitrary unions, not only countable ones, so typically many open sets will not be measurable with respect to the product sigma algebra (while all sets in the product sigma algebra are borel). The two sigma algebras coincide only when everything is countable (i.e., the product is countable, and the borel sigma algebra in the factor is countably generated).

In this paragraph, we develop basic measurability properties for the borel sigma algebra, and compare it with the product sigma algebra as explained above.

lemma *measurable_product_coordinates* [*measurable* (*raw*)]:
  $(\lambda x.\ x\ i) \in$ *measurable borel borel*
by (*rule borel_measurable_continuous_onI*[*OF continuous_on_product_coordinates*])

lemma *measurable_product_then_coordinatewise*:
  fixes f::$'a \Rightarrow 'b \Rightarrow$ ($'c$::*topological_space*)
  assumes [*measurable*]: $f \in$ *borel_measurable M*
  shows $(\lambda x.\ f\ x\ i) \in$ *borel_measurable M*
proof −
  have $(\lambda x.\ f\ x\ i) = (\lambda y.\ y\ i)\ o\ f$
    unfolding *comp_def* **by** *auto*
  then show *?thesis* **by** *simp*
qed

To compare the Borel sigma algebra with the product sigma algebra, we give a presentation of the product sigma algebra that is more similar to the one we used above for the product topology.

lemma *sets_PiM_finite*:
  sets $(Pi_M\ I\ M) =$ *sigma_sets* $(\Pi_E\ i{\in}I.\ space\ (M\ i))$
      $\{(\Pi_E\ i{\in}I.\ X\ i)\ |X.\ (\forall i.\ X\ i \in sets\ (M\ i)) \wedge finite\ \{i.\ X\ i \neq space\ (M\ i)\}\}$
proof
  have $\{(\Pi_E\ i{\in}I.\ X\ i)\ |X.\ (\forall i.\ X\ i \in sets\ (M\ i)) \wedge finite\ \{i.\ X\ i \neq space\ (M\ i)\}\}$
$\subseteq$ *sets* $(Pi_M\ I\ M)$
  proof (*auto*)
    fix X **assume** H: $\forall i.\ X\ i \in sets\ (M\ i)$ *finite* $\{i.\ X\ i \neq space\ (M\ i)\}$
    then have $*$: $X\ i \in sets\ (M\ i)$ **for** i **by** *simp*
    define J **where** $J = \{i \in I.\ X\ i \neq space\ (M\ i)\}$
    have *finite* J $J \subseteq I$ **unfolding** *J_def* **using** H **by** *auto*
    define Y **where** $Y = (\Pi_E\ j{\in}J.\ X\ j)$
    have *prod_emb I M J Y* $\in$ *sets* $(Pi_M\ I\ M)$
      unfolding *Y_def* **apply** (*rule sets_PiM_I*) **using** ‹*finite J*› ‹$J \subseteq I$› $*$ **by**
auto
    moreover have *prod_emb I M J Y* $= (\Pi_E\ i{\in}I.\ X\ i)$
      unfolding *prod_emb_def Y_def J_def* **using** H *sets.sets_into_space*[*OF* $*$]
      by (*auto simp add*: *PiE_iff*, *blast*)
    ultimately show $Pi_E\ I\ X \in sets\ (Pi_M\ I\ M)$ **by** *simp*
  qed

```
  then show sigma_sets (Π_E i∈I. space (M i)) {(Π_E i∈I. X i) |X. (∀ i. X i ∈
sets (M i)) ∧ finite {i. X i ≠ space (M i)}}
              ⊆ sets (Pi_M I M)
    by (metis (mono_tags, lifting) sets.sigma_sets_subset' sets.top space_PiM)

  have *: ∃ X. {f. (∀ i∈I. f i ∈ space (M i)) ∧ f ∈ extensional I ∧ f i ∈ A} = Pi_E
I X ∧
                (∀ i. X i ∈ sets (M i)) ∧ finite {i. X i ≠ space (M i)}
    if i ∈ I A ∈ sets (M i) for i A
  proof −
    define X where X = (λj. if j = i then A else space (M j))
    have {f. (∀ i∈I. f i ∈ space (M i)) ∧ f ∈ extensional I ∧ f i ∈ A} = Pi_E I X
      unfolding X_def using sets.sets_into_space[OF ‹A ∈ sets (M i)›] ‹i ∈ I›
      by (auto simp add: PiE_iff extensional_def, metis subsetCE, metis)
    moreover have X j ∈ sets (M j) for j
      unfolding X_def using ‹A ∈ sets (M i)› by auto
    moreover have finite {j. X j ≠ space (M j)}
      unfolding X_def by simp
    ultimately show ?thesis by auto
  qed
  show sets (Pi_M I M) ⊆ sigma_sets (Π_E i∈I. space (M i)) {(Π_E i∈I. X i) |X.
(∀ i. X i ∈ sets (M i)) ∧ finite {i. X i ≠ space (M i)}}
    unfolding sets_PiM_single
    apply (rule sigma_sets_mono')
    apply (auto simp add: PiE_iff *)
    done
qed

lemma sets_PiM_subset_borel:
  sets (Pi_M UNIV (λ_. borel)) ⊆ sets borel
proof −
  have *: Pi_E UNIV X ∈ sets borel if [measurable]: ⋀i. X i ∈ sets borel finite {i.
X i ≠ UNIV} for X::'a ⇒ 'b set
  proof −
    define I where I = {i. X i ≠ UNIV}
    have finite I unfolding I_def using that by simp
    have Pi_E UNIV X = (⋂ i∈I. (λx. x i)−‘(X i) ∩ space borel) ∩ space borel
      unfolding I_def by auto
    also have ... ∈ sets borel
      using that ‹finite I› by measurable
    finally show ?thesis by simp
  qed
  then have {(Π_E i∈UNIV. X i) |X::('a ⇒ 'b set). (∀ i. X i ∈ sets borel) ∧ finite
{i. X i ≠ space borel}} ⊆ sets borel
    by auto
  then show ?thesis unfolding sets_PiM_finite space_borel
    by (simp add: * sets.sigma_sets_subset')
qed
```

proposition *sets_PiM_equal_borel*:
sets (*Pi*$_M$ *UNIV* (λ*i*::(*'a*::*countable*). *borel*::(*'b*::*second_countable_topology measure*))) = *sets borel*
proof
obtain *K*::(*'a* ⇒ *'b*) *set set* **where** *K*: *topological_basis K countable K*
⋀*k*. *k* ∈ *K* ⟹ ∃ *X*. (*k* = *Pi*$_E$ *UNIV X*) ∧ (∀ *i*. *open* (*X i*)) ∧ *finite* {*i*. *X i* ≠ *UNIV*}
using *product_topology_countable_basis* **by** *fast*
have ∗: *k* ∈ *sets* (*Pi*$_M$ *UNIV* (λ_. *borel*)) **if** *k* ∈ *K* **for** *k*
proof −
obtain *X* **where** *H*: *k* = *Pi*$_E$ *UNIV X* ⋀*i*. *open* (*X i*) *finite* {*i*. *X i* ≠ *UNIV*}
using *K*(*3*)[*OF* ‹*k* ∈ *K*›] **by** *blast*
show *?thesis* **unfolding** *H*(*1*) *sets_PiM_finite space_borel* **using** *borel_open*[*OF H*(*2*)] *H*(*3*) **by** *auto*
qed
have ∗∗: *U* ∈ *sets* (*Pi*$_M$ *UNIV* (λ_. *borel*)) **if** *open U* **for** *U*::(*'a* ⇒ *'b*) *set*
proof −
obtain *B* **where** *B* ⊆ *K U* = (⋃ *B*)
using ‹*open U*› ‹*topological_basis K*› **by** (*metis topological_basis_def*)
have *countable B* **using** ‹*B* ⊆ *K*› ‹*countable K*› *countable_subset* **by** *blast*
moreover have *k* ∈ *sets* (*Pi*$_M$ *UNIV* (λ_. *borel*)) **if** *k* ∈ *B* **for** *k*
using ‹*B* ⊆ *K*› ∗ *that* **by** *auto*
ultimately show *?thesis* **unfolding** ‹*U* = (⋃ *B*)› **by** *auto*
qed
have *sigma_sets UNIV* (*Collect open*) ⊆ *sets* (*Pi*$_M$ *UNIV* (λ*i*::*'a*. (*borel*::(*'b measure*))))
apply (*rule sets.sigma_sets_subset'*) **using** ∗∗ **by** *auto*
then show *sets* (*borel*::(*'a* ⇒ *'b*) *measure*) ⊆ *sets* (*Pi*$_M$ *UNIV* (λ_. *borel*))
unfolding *borel_def* **by** *auto*
qed (*simp add*: *sets_PiM_subset_borel*)

lemma *measurable_coordinatewise_then_product*:
fixes *f*::*'a* ⇒ (*'b*::*countable*) ⇒ (*'c*::*second_countable_topology*)
assumes [*measurable*]: ⋀*i*. (λ*x*. *f x i*) ∈ *borel_measurable M*
shows *f* ∈ *borel_measurable M*
proof −
have *f* ∈ *measurable M* (*Pi*$_M$ *UNIV* (λ_. *borel*))
by (*rule measurable_PiM_single'*, *auto simp add*: *assms*)
then show *?thesis* **using** *sets_PiM_equal_borel measurable_cong_sets* **by** *blast*
qed

end

6.8 Caratheodory Extension Theorem

theory *Caratheodory*
imports *Measure_Space*
begin

Originally from the Hurd/Coble measure theory development, translated by Lawrence Paulson.

lemma *suminf_ennreal_2dimen*:
fixes *f*:: *nat* × *nat* ⇒ *ennreal*
assumes ⋀*m*. *g m* = (∑ *n*. *f* (*m*,*n*))
shows (∑ *i*. *f* (*prod_decode i*)) = *suminf g*
proof −
have *g_def*: *g* = (λ*m*. (∑ *n*. *f* (*m*,*n*)))
using *assms* **by** (*simp add*: *fun_eq_iff*)
have *reindex*: ⋀*B*. (∑ *x*∈*B*. *f* (*prod_decode x*)) = *sum f* (*prod_decode* ‘ *B*)
by (*simp add*: *sum.reindex*[*OF inj_prod_decode*] *comp_def*)
have (*SUP n*. ∑ *i*<*n*. *f* (*prod_decode i*)) = (*SUP p* ∈ *UNIV* × *UNIV*. ∑ *i*<*fst p*. ∑ *n*<*snd p*. *f* (*i*, *n*))
proof (*intro SUP_eq*; *clarsimp simp*: *sum.cartesian_product reindex*)
fix *n*
let *?M* = λ*f*. *Suc* (*Max* (*f* ‘ *prod_decode* ‘ {..<*n*}))
{ fix *a b x* **assume** *x* < *n* **and** [*symmetric*]: (*a*, *b*) = *prod_decode x*
then have *a* < *?M fst b* < *?M snd*
by (*auto intro*!: *Max_ge le_imp_less_Suc image_eqI*) **}**
then have *sum f* (*prod_decode* ‘ {..<*n*}) ≤ *sum f* ({..<*?M fst*} × {..<*?M snd*})
by (*auto intro*!: *sum_mono2*)
then show ∃ *a b*. *sum f* (*prod_decode* ‘ {..<*n*}) ≤ *sum f* ({..<*a*} × {..<*b*}) **by** *auto*
next
fix *a b*
let *?M* = *prod_decode* ‘ {..<*Suc* (*Max* (*prod_encode* ‘ ({..<*a*} × {..<*b*})))}
{ fix *a′ b′* **assume** *a′* < *a b′* < *b* **then have** (*a′*, *b′*) ∈ *?M*
by (*auto intro*!: *Max_ge le_imp_less_Suc image_eqI*[**where** *x*=*prod_encode* (*a′*, *b′*)]) **}**
then have *sum f* ({..<*a*} × {..<*b*}) ≤ *sum f ?M*
by (*auto intro*!: *sum_mono2*)
then show ∃ *n*. *sum f* ({..<*a*} × {..<*b*}) ≤ *sum f* (*prod_decode* ‘ {..<*n*})
by *auto*
qed
also have ... = (*SUP p*. ∑ *i*<*p*. ∑ *n*. *f* (*i*, *n*))
unfolding *suminf_sum*[*OF summableI*, *symmetric*]
by (*simp add*: *suminf_eq_SUP SUP_pair sum.swap*[*of* _ {..< *fst* _}])
finally show *?thesis* **unfolding** *g_def*
by (*simp add*: *suminf_eq_SUP*)
qed

6.8.1 Characterizations of Measures

definition *outer_measure_space* **where**
outer_measure_space M f ⟷ *positive M f* ∧ *increasing M f* ∧ *countably_subadditive M f*

Lambda Systems

definition *lambda_system* :: *'a set* ⇒ *'a set set* ⇒ (*'a set* ⇒ *ennreal*) ⇒ *'a set set*
where
lambda_system Ω *M f* = {*l* ∈ *M*. ∀ *x* ∈ *M*. *f* (*l* ∩ *x*) + *f* ((Ω − *l*) ∩ *x*) = *f x*}

lemma (**in** *algebra*) *lambda_system_eq*:
lambda_system Ω *M f* = {*l* ∈ *M*. ∀ *x* ∈ *M*. *f* (*x* ∩ *l*) + *f* (*x* − *l*) = *f x*}
proof −
have [*simp*]: ⋀*l x*. *l* ∈ *M* ⟹ *x* ∈ *M* ⟹ (Ω − *l*) ∩ *x* = *x* − *l*
by (*metis Int_Diff Int_absorb1 Int_commute sets_into_space*)
show *?thesis*
by (*auto simp add*: *lambda_system_def*) (*metis Int_commute*)+
qed

lemma (**in** *algebra*) *lambda_system_empty*: *positive M f* ⟹ {} ∈ *lambda_system* Ω *M f*
by (*auto simp add*: *positive_def lambda_system_eq*)

lemma *lambda_system_sets*: *x* ∈ *lambda_system* Ω *M f* ⟹ *x* ∈ *M*
by (*simp add*: *lambda_system_def*)

lemma (**in** *algebra*) *lambda_system_Compl*:
fixes *f*:: *'a set* ⇒ *ennreal*
assumes *x*: *x* ∈ *lambda_system* Ω *M f*
shows Ω − *x* ∈ *lambda_system* Ω *M f*
proof −
have *x* ⊆ Ω
by (*metis sets_into_space lambda_system_sets x*)
hence Ω − (Ω − *x*) = *x*
by (*metis double_diff equalityE*)
with *x* **show** *?thesis*
by (*force simp add*: *lambda_system_def ac_simps*)
qed

lemma (**in** *algebra*) *lambda_system_Int*:
fixes *f*:: *'a set* ⇒ *ennreal*
assumes *xl*: *x* ∈ *lambda_system* Ω *M f* **and** *yl*: *y* ∈ *lambda_system* Ω *M f*
shows *x* ∩ *y* ∈ *lambda_system* Ω *M f*
proof −
from *xl yl* **show** *?thesis*
proof (*auto simp add*: *positive_def lambda_system_eq Int*)
fix *u*
assume *x*: *x* ∈ *M* **and** *y*: *y* ∈ *M* **and** *u*: *u* ∈ *M*
and *fx*: ∀ *z*∈*M*. *f* (*z* ∩ *x*) + *f* (*z* − *x*) = *f z*
and *fy*: ∀ *z*∈*M*. *f* (*z* ∩ *y*) + *f* (*z* − *y*) = *f z*
have *u* − *x* ∩ *y* ∈ *M*
by (*metis Diff Diff_Int Un u x y*)
moreover
have (*u* − (*x* ∩ *y*)) ∩ *y* = *u* ∩ *y* − *x* **by** *blast*

```
    moreover
    have u − x ∩ y − y = u − y by blast
    ultimately
    have ey: f (u − x ∩ y) = f (u ∩ y − x) + f (u − y) using fy
      by force
    have f (u ∩ (x ∩ y)) + f (u − x ∩ y)
          = (f (u ∩ (x ∩ y)) + f (u ∩ y − x)) + f (u − y)
      by (simp add: ey ac_simps)
    also have ... =  (f ((u ∩ y) ∩ x) + f (u ∩ y − x)) + f (u − y)
      by (simp add: Int_ac)
    also have ... = f (u ∩ y) + f (u − y)
      using fx [THEN bspec, of u ∩ y] Int y u
      by force
    also have ... = f u
      by (metis fy u)
    finally show f (u ∩ (x ∩ y)) + f (u − x ∩ y) = f u .
  qed
qed

lemma (in algebra) lambda_system_Un:
  fixes f:: 'a set ⇒ ennreal
  assumes xl: x ∈ lambda_system Ω M f and yl: y ∈ lambda_system Ω M f
  shows x ∪ y ∈ lambda_system Ω M f
proof −
  have (Ω − x) ∩ (Ω − y) ∈ M
    by (metis Diff_Un Un compl_sets lambda_system_sets xl yl)
  moreover
  have x ∪ y = Ω − ((Ω − x) ∩ (Ω − y))
    by auto (metis subsetD lambda_system_sets sets_into_space xl yl)+
  ultimately show ?thesis
    by (metis lambda_system_Compl lambda_system_Int xl yl)
qed

lemma (in algebra) lambda_system_algebra:
  positive M f ⟹ algebra Ω (lambda_system Ω M f)
  apply (auto simp add: algebra_iff_Un)
  apply (metis lambda_system_sets subsetD sets_into_space)
  apply (metis lambda_system_empty)
  apply (metis lambda_system_Compl)
  apply (metis lambda_system_Un)
  done

lemma (in algebra) lambda_system_strong_additive:
  assumes z: z ∈ M and disj: x ∩ y = {}
     and xl: x ∈ lambda_system Ω M f and yl: y ∈ lambda_system Ω M f
  shows f (z ∩ (x ∪ y)) = f (z ∩ x) + f (z ∩ y)
proof −
  have z ∩ x = (z ∩ (x ∪ y)) ∩ x using disj by blast
  moreover
```

```
  have z ∩ y = (z ∩ (x ∪ y)) − x using disj by blast
  moreover
  have (z ∩ (x ∪ y)) ∈ M
    by (metis Int Un lambda_system_sets xl yl z)
  ultimately show ?thesis using xl yl
    by (simp add: lambda_system_eq)
qed

lemma (in algebra) lambda_system_additive: additive (lambda_system Ω M f) f
proof (auto simp add: additive_def)
  fix x and y
  assume disj: x ∩ y = {}
     and xl: x ∈ lambda_system Ω M f and yl: y ∈ lambda_system Ω M f
  hence  x ∈ M y ∈ M by (blast intro: lambda_system_sets)+
  thus f (x ∪ y) = f x + f y
    using lambda_system_strong_additive [OF top disj xl yl]
    by (simp add: Un)
qed

lemma lambda_system_increasing: increasing M f ⟹ increasing (lambda_system
Ω M f) f
  by (simp add: increasing_def lambda_system_def)

lemma lambda_system_positive: positive M f ⟹ positive (lambda_system Ω M
f) f
  by (simp add: positive_def lambda_system_def)

lemma (in algebra) lambda_system_strong_sum:
  fixes A:: nat ⇒ 'a set and f :: 'a set ⇒ ennreal
  assumes f: positive M f and a: a ∈ M
      and A: range A ⊆ lambda_system Ω M f
      and disj: disjoint_family A
  shows  (∑ i = 0..<n. f (a ∩A i)) = f (a ∩ (⋃ i∈{0..<n}. A i))
proof (induct n)
  case 0 show ?case using f by (simp add: positive_def)
next
  case (Suc n)
  have 2: A n ∩ ⋃ (A ` {0..<n}) = {} using disj
    by (force simp add: disjoint_family_on_def neq_iff)
  have 3: A n ∈ lambda_system Ω M f using A
    by blast
  interpret l: algebra Ω lambda_system Ω M f
    using f by (rule lambda_system_algebra)
  have 4: ⋃ (A ` {0..<n}) ∈ lambda_system Ω M f
    using A l.UNION_in_sets by simp
  from Suc.hyps show ?case
    by (simp add: atLeastLessThanSuc lambda_system_strong_additive [OF a 2 3
4])
qed
```

proposition (**in** *sigma_algebra*) *lambda_system_caratheodory*:
assumes *oms*: *outer_measure_space M f*
and *A*: *range A* ⊆ *lambda_system* Ω *M f*
and *disj*: *disjoint_family A*
shows (⋃ *i*. *A i*) ∈ *lambda_system* Ω *M f* ∧ (∑ *i*. *f* (*A i*)) = *f* (⋃ *i*. *A i*)
proof −
have *pos*: *positive M f* **and** *inc*: *increasing M f*
and *csa*: *countably_subadditive M f*
by (*metis oms outer_measure_space_def*)+
have *sa*: *subadditive M f*
by (*metis countably_subadditive_subadditive csa pos*)
have *A'*: ⋀*S*. *A*‘*S* ⊆ (*lambda_system* Ω *M f*) **using** *A*
by *auto*
interpret *ls*: *algebra* Ω *lambda_system* Ω *M f*
using *pos* **by** (*rule lambda_system_algebra*)
have *A''*: *range A* ⊆ *M*
by (*metis A image_subset_iff lambda_system_sets*)

have *U_in*: (⋃ *i*. *A i*) ∈ *M*
by (*metis A'' countable_UN*)
have *U_eq*: *f* (⋃ *i*. *A i*) = (∑ *i*. *f* (*A i*))
proof (*rule antisym*)
show *f* (⋃ *i*. *A i*) ≤ (∑ *i*. *f* (*A i*))
using *csa*[*unfolded countably_subadditive_def*] *A'' disj U_in* **by** *auto*
have *dis*: ⋀*N*. *disjoint_family_on A* {..<*N*} **by** (*intro disjoint_family_on_mono*[*OF* _ *disj*]) *auto*
show (∑ *i*. *f* (*A i*)) ≤ *f* (⋃ *i*. *A i*)
using *ls.additive_sum* [*OF lambda_system_positive*[*OF pos*] *lambda_system_additive* _ *A' dis*] *A''*
by (*intro suminf_le_const*[*OF summableI*]) (*auto intro*!: *increasingD*[*OF inc*] *countable_UN*)
qed
have *f* (*a* ∩ (⋃ *i*. *A i*)) + *f* (*a* − (⋃ *i*. *A i*)) = *f a*
if *a* [*iff*]: *a* ∈ *M* **for** *a*
proof (*rule antisym*)
have *range* (λ*i*. *a* ∩ *A i*) ⊆ *M* **using** *A''*
by *blast*
moreover
have *disjoint_family* (λ*i*. *a* ∩ *A i*) **using** *disj*
by (*auto simp add*: *disjoint_family_on_def*)
moreover
have *a* ∩ (⋃ *i*. *A i*) ∈ *M*
by (*metis Int U_in a*)
ultimately
have *f* (*a* ∩ (⋃ *i*. *A i*)) ≤ (∑ *i*. *f* (*a* ∩ *A i*))
using *csa*[*unfolded countably_subadditive_def*, *rule_format*, *of* (λ*i*. *a* ∩ *A i*)]
by (*simp add*: *o_def*)
hence *f* (*a* ∩ (⋃ *i*. *A i*)) + *f* (*a* − (⋃ *i*. *A i*)) ≤ (∑ *i*. *f* (*a* ∩ *A i*)) + *f* (*a* −

$(\bigcup i.\ A\ i))$
by (*rule add_right_mono*)
also have $\ldots \leq f\ a$
proof (*intro ennreal_suminf_bound_add*)
fix n
have *UNION_in*: $(\bigcup i{\in}\{0..{<}n\}.\ A\ i) \in M$
by (*metis A'' UNION_in_sets*)
have *le_fa*: $f\ (\bigcup\ (A\ `\ \{0..{<}n\}) \cap a) \leq f\ a$ **using** A''
by (*blast intro*: *increasingD* [*OF inc*] A'' *UNION_in_sets*)
have *ls*: $(\bigcup i{\in}\{0..{<}n\}.\ A\ i) \in$ *lambda_system* $\Omega\ M\ f$
using *ls.UNION_in_sets* **by** (*simp add*: *A*)
hence *eq_fa*: $f\ a = f\ (a \cap (\bigcup i{\in}\{0..{<}n\}.\ A\ i)) + f\ (a - (\bigcup i{\in}\{0..{<}n\}.\ A\ i))$
by (*simp add*: *lambda_system_eq UNION_in*)
have $f\ (a - (\bigcup i.\ A\ i)) \leq f\ (a - (\bigcup i{\in}\{0..{<}n\}.\ A\ i))$
by (*blast intro*: *increasingD* [*OF inc*] *UNION_in U_in*)
thus $(\sum i{<}n.\ f\ (a \cap A\ i)) + f\ (a - (\bigcup i.\ A\ i)) \leq f\ a$
by (*simp add*: *lambda_system_strong_sum pos A disj eq_fa add_left_mono atLeast0LessThan*[*symmetric*])
qed
finally show $f\ (a \cap (\bigcup i.\ A\ i)) + f\ (a - (\bigcup i.\ A\ i)) \leq f\ a$
by *simp*
next
have $f\ a \leq f\ (a \cap (\bigcup i.\ A\ i) \cup (a - (\bigcup i.\ A\ i)))$
by (*blast intro*: *increasingD* [*OF inc*] *U_in*)
also have $\ldots \leq\ f\ (a \cap (\bigcup i.\ A\ i)) + f\ (a - (\bigcup i.\ A\ i))$
by (*blast intro*: *subadditiveD* [*OF sa*] *U_in*)
finally show $f\ a \leq f\ (a \cap (\bigcup i.\ A\ i)) + f\ (a - (\bigcup i.\ A\ i))$ **.**
qed
thus *?thesis*
by (*simp add*: *lambda_system_eq sums_iff U_eq U_in*)
qed

proposition (**in** *sigma_algebra*) *caratheodory_lemma*:
assumes *oms*: *outer_measure_space M f*
defines $L \equiv$ *lambda_system* $\Omega\ M\ f$
shows *measure_space* $\Omega\ L\ f$
proof −
have *pos*: *positive M f*
by (*metis oms outer_measure_space_def*)
have *alg*: *algebra* $\Omega\ L$
using *lambda_system_algebra* [*of f*, *OF pos*]
by (*simp add*: *algebra_iff_Un L_def*)
then
have *sigma_algebra* $\Omega\ L$
using *lambda_system_caratheodory* [*OF oms*]
by (*simp add*: *sigma_algebra_disjoint_iff L_def*)
moreover
have *countably_additive L f positive L f*

using *pos lambda_system_caratheodory* [*OF oms*]
by (*auto simp add*: *lambda_system_sets L_def countably_additive_def positive_def*)
ultimately
show *?thesis*
using *pos* **by** (*simp add*: *measure_space_def*)
qed

definition *outer_measure* :: *'a set set* ⇒ (*'a set* ⇒ *ennreal*) ⇒ *'a set* ⇒ *ennreal*
where
outer_measure M f X =
(*INF A*∈{*A*. *range A* ⊆ *M* ∧ *disjoint_family A* ∧ *X* ⊆ (⋃ *i*. *A i*)}. ∑ *i*. *f* (*A i*))

lemma (**in** *ring_of_sets*) *outer_measure_agrees*:
assumes *posf*: *positive M f* **and** *ca*: *countably_additive M f* **and** *s*: *s* ∈ *M*
shows *outer_measure M f s* = *f s*
unfolding *outer_measure_def*
proof (*safe intro*!: *antisym INF_greatest*)
fix *A* :: *nat* ⇒ *'a set* **assume** *A*: *range A* ⊆ *M* **and** *dA*: *disjoint_family A* **and** *sA*: *s* ⊆ (⋃ *x*. *A x*)
have *inc*: *increasing M f*
by (*metis additive_increasing ca countably_additive_additive posf*)
have *f s* = *f* (⋃ *i*. *A i* ∩ *s*)
using *sA* **by** (*auto simp*: *Int_absorb1*)
also have ... = (∑ *i*. *f* (*A i* ∩ *s*))
using *sA dA A s*
by (*intro ca*[*unfolded countably_additive_def*, *rule_format*, *symmetric*])
(*auto simp*: *Int_absorb1 disjoint_family_on_def*)
also have ... ≤ (∑ *i*. *f* (*A i*))
using *A s* **by** (*auto intro*!: *suminf_le increasingD*[*OF inc*])
finally show *f s* ≤ (∑ *i*. *f* (*A i*)) **.**
next
have (∑ *i*. *f* (*if i* = *0 then s else* {})) ≤ *f s*
using *positiveD1*[*OF posf*] **by** (*subst suminf_finite*[*of* {*0*}]) *auto*
with *s* **show** (*INF A*∈{*A*. *range A* ⊆ *M* ∧ *disjoint_family A* ∧ *s* ⊆ ⋃(*A* ‘ *UNIV*)}. ∑ *i*. *f* (*A i*)) ≤ *f s*
by (*intro INF_lower2*[*of* λ*i*. *if i* = *0 then s else* {}])
(*auto simp*: *disjoint_family_on_def*)
qed

lemma *outer_measure_empty*:
positive M f ⟹ {} ∈ *M* ⟹ *outer_measure M f* {} = *0*
unfolding *outer_measure_def*
by (*intro antisym INF_lower2*[*of* λ_. {}]) (*auto simp*: *disjoint_family_on_def positive_def*)

lemma (**in** *ring_of_sets*) *positive_outer_measure*:
assumes *positive M f* **shows** *positive* (*Pow* Ω) (*outer_measure M f*)

unfolding *positive_def* **by** (*auto simp*: *assms outer_measure_empty*)

lemma (**in** *ring_of_sets*) *increasing_outer_measure*: *increasing* (*Pow* Ω) (*outer_measure M f*)
by (*force simp*: *increasing_def outer_measure_def intro*!: *INF_greatest intro*: *INF_lower*)

lemma (**in** *ring_of_sets*) *outer_measure_le*:
assumes *pos*: *positive M f* **and** *inc*: *increasing M f* **and** *A*: *range A* ⊆ *M* **and** *X*: *X* ⊆ (⋃ *i*. *A i*)
shows *outer_measure M f X* ≤ (∑ *i*. *f* (*A i*))
unfolding *outer_measure_def*
proof (*safe intro*!: *INF_lower2*[*of disjointed A*] *del*: *subsetI*)
show *dA*: *range* (*disjointed A*) ⊆ *M*
by (*auto intro*!: *A range_disjointed_sets*)
have ∀ *n*. *f* (*disjointed A n*) ≤ *f* (*A n*)
by (*metis increasingD* [*OF inc*] *UNIV_I dA image_subset_iff disjointed_subset A*)
then show (∑ *i*. *f* (*disjointed A i*)) ≤ (∑ *i*. *f* (*A i*))
by (*blast intro*!: *suminf_le*)
qed (*auto simp*: *X UN_disjointed_eq disjoint_family_disjointed*)

lemma (**in** *ring_of_sets*) *outer_measure_close*:
outer_measure M f X < *e* ⟹ ∃ *A*. *range A* ⊆ *M* ∧ *disjoint_family A* ∧ *X* ⊆ (⋃ *i*. *A i*) ∧ (∑ *i*. *f* (*A i*)) < *e*
unfolding *outer_measure_def INF_less_iff* **by** *auto*

lemma (**in** *ring_of_sets*) *countably_subadditive_outer_measure*:
assumes *posf*: *positive M f* **and** *inc*: *increasing M f*
shows *countably_subadditive* (*Pow* Ω) (*outer_measure M f*)
proof (*simp add*: *countably_subadditive_def*, *safe*)
fix *A* :: *nat* ⇒ _ **assume** *A*: *range A* ⊆ *Pow* (Ω) **and** *sb*: (⋃ *i*. *A i*) ⊆ Ω
let *?O* = *outer_measure M f*
show *?O* (⋃ *i*. *A i*) ≤ (∑ *n*. *?O* (*A n*))
proof (*rule ennreal_le_epsilon*)
fix *b* **and** *e* :: *real* **assume** *0* < *e* (∑ *n*. *outer_measure M f* (*A n*)) < *top*
then have ∗: ⋀*n*. *outer_measure M f* (*A n*) < *outer_measure M f* (*A n*) + *e* ∗ (*1*/*2*) ^ *Suc n*
by (*auto simp add*: *less_top dest*!: *ennreal_suminf_lessD*)
obtain *B*
where *B*: ⋀*n*. *range* (*B n*) ⊆ *M*
and *sbB*: ⋀*n*. *A n* ⊆ (⋃ *i*. *B n i*)
and *Ble*: ⋀*n*. (∑ *i*. *f* (*B n i*)) ≤ *?O* (*A n*) + *e* ∗ (*1*/*2*) ^ (*Suc n*)
by (*metis less_imp_le outer_measure_close*[*OF* ∗])

define *C* **where** *C* = *case_prod B o prod_decode*
from *B* **have** *B_in_M*: ⋀*i j*. *B i j* ∈ *M*
by (*rule range_subsetD*)
then have *C*: *range C* ⊆ *M*

```
    by (auto simp add: C_def split_def)
  have A_C: (⋃ i. A i) ⊆ (⋃ i. C i)
  using sbB by (auto simp add: C_def subset_eq) (metis prod.case prod_encode_inverse)

  have ?O (⋃ i. A i) ≤ ?O (⋃ i. C i)
    using A_C A C by (intro increasing_outer_measure[THEN increasingD])
(auto dest!: sets_into_space)
  also have ... ≤ (∑ i. f (C i))
    using C by (intro outer_measure_le[OF posf inc]) auto
  also have ... = (∑ n. ∑ i. f (B n i))
  using B_in_M unfolding C_def comp_def by (intro suminf_ennreal_2dimen)
auto
  also have ... ≤ (∑ n. ?O (A n) + e * (1/2) ^ Suc n)
    using B_in_M by (intro suminf_le suminf_nonneg allI Ble) auto
  also have ... = (∑ n. ?O (A n)) + (∑ n. ennreal e * ennreal ((1/2) ^ Suc n))
    using ‹0 < e› by (subst suminf_add[symmetric])
                     (auto simp del: ennreal_suminf_cmult simp add: en-
nreal_mult[symmetric])
  also have ... = (∑ n. ?O (A n)) + e
    unfolding ennreal_suminf_cmult
    by (subst suminf_ennreal_eq[OF zero_le_power power_half_series]) auto
  finally show ?O (⋃ i. A i) ≤ (∑ n. ?O (A n)) + e .
 qed
qed

lemma (in ring_of_sets) outer_measure_space_outer_measure:
 positive M f ⟹ increasing M f ⟹ outer_measure_space (Pow Ω) (outer_measure
M f)
 by (simp add: outer_measure_space_def
  positive_outer_measure increasing_outer_measure countably_subadditive_outer_measure)

lemma (in ring_of_sets) algebra_subset_lambda_system:
 assumes posf: positive M f and inc: increasing M f
    and add: additive M f
 shows M ⊆ lambda_system Ω (Pow Ω) (outer_measure M f)
proof (auto dest: sets_into_space
          simp add: algebra.lambda_system_eq [OF algebra_Pow])
 fix x s assume x: x ∈ M and s: s ⊆ Ω
 have [simp]: ⋀x. x ∈ M ⟹ s ∩ (Ω - x) = s - x using s
   by blast
 have outer_measure M f (s ∩ x) + outer_measure M f (s - x) ≤ outer_measure
M f s
   unfolding outer_measure_def[of M f s]
 proof (safe intro!: INF_greatest)
   fix A :: nat ⇒ 'a set assume A: disjoint_family A range A ⊆ M s ⊆ (⋃ i. A
i)
   have outer_measure M f (s ∩ x) ≤ (∑ i. f (A i ∩ x))
     unfolding outer_measure_def
   proof (safe intro!: INF_lower2[of λi. A i ∩ x])
```

from *A*(*1*) **show** *disjoint_family* (λ*i*. *A i* ∩ *x*)
by (*rule disjoint_family_on_bisimulation*) *auto*
qed (*insert x A*, *auto*)
moreover
have *outer_measure M f* (*s* − *x*) ≤ (∑ *i*. *f* (*A i* − *x*))
unfolding *outer_measure_def*
proof (*safe intro*!: *INF_lower2*[*of* λ*i*. *A i* − *x*])
from *A*(*1*) **show** *disjoint_family* (λ*i*. *A i* − *x*)
by (*rule disjoint_family_on_bisimulation*) *auto*
qed (*insert x A*, *auto*)
ultimately have *outer_measure M f* (*s* ∩ *x*) + *outer_measure M f* (*s* − *x*) ≤
(∑ *i*. *f* (*A i* ∩ *x*)) + (∑ *i*. *f* (*A i* − *x*)) **by** (*rule add_mono*)
also have ... = (∑ *i*. *f* (*A i* ∩ *x*) + *f* (*A i* − *x*))
using *A*(*2*) *x posf* **by** (*subst suminf_add*) (*auto simp*: *positive_def*)
also have ... = (∑ *i*. *f* (*A i*))
using *A x*
by (*subst add*[*THEN additiveD*, *symmetric*])
(*auto intro*!: *arg_cong*[**where** *f*=*suminf*] *arg_cong*[**where** *f*=*f*])
finally show *outer_measure M f* (*s* ∩ *x*) + *outer_measure M f* (*s* − *x*) ≤
(∑ *i*. *f* (*A i*)) **.**
qed
moreover
have *outer_measure M f s* ≤ *outer_measure M f* (*s* ∩ *x*) + *outer_measure M f*
(*s* − *x*)
proof −
have *outer_measure M f s* = *outer_measure M f* ((*s* ∩ *x*) ∪ (*s* − *x*))
by (*metis Un_Diff_Int Un_commute*)
also have ... ≤ *outer_measure M f* (*s* ∩ *x*) + *outer_measure M f* (*s* − *x*)
apply (*rule subadditiveD*)
apply (*rule ring_of_sets.countably_subadditive_subadditive* [*OF ring_of_sets_Pow*])
apply (*simp add*: *positive_def outer_measure_empty*[*OF posf*])
apply (*rule countably_subadditive_outer_measure*)
using *s* **by** (*auto intro*!: *posf inc*)
finally show *?thesis* **.**
qed
ultimately
show *outer_measure M f* (*s* ∩ *x*) + *outer_measure M f* (*s* − *x*) = *outer_measure*
M f s
by (*rule order_antisym*)
qed

lemma *measure_down*: *measure_space* Ω *N* μ ⟹ *sigma_algebra* Ω *M* ⟹ *M* ⊆
N ⟹ *measure_space* Ω *M* μ
by (*auto simp add*: *measure_space_def positive_def countably_additive_def subset_eq*)

6.8.2 Caratheodory's theorem

theorem (**in** *ring_of_sets*) *caratheodory'*:

assumes *posf*: *positive M f* **and** *ca*: *countably_additive M f*
shows $\exists \mu :: 'a\ set \Rightarrow ennreal.\ (\forall s \in M.\ \mu\ s = f\ s) \wedge measure\_space\ \Omega\ (sigma\_sets\ \Omega\ M)\ \mu$
proof −
have *inc*: *increasing M f*
by (*metis additive_increasing ca countably_additive_additive posf*)
let *?O* = *outer_measure M f*
define *ls* **where** *ls* = *lambda_system* Ω (*Pow* Ω) *?O*
have *mls*: *measure_space* Ω *ls ?O*
using *sigma_algebra.caratheodory_lemma*
[*OF sigma_algebra_Pow outer_measure_space_outer_measure* [*OF posf inc*]]
by (*simp add*: *ls_def*)
hence *sls*: *sigma_algebra* Ω *ls*
by (*simp add*: *measure_space_def*)
have *M* ⊆ *ls*
by (*simp add*: *ls_def*)
(*metis ca posf inc countably_additive_additive algebra_subset_lambda_system*)
hence *sgs_sb*: *sigma_sets* (Ω) (*M*) ⊆ *ls*
using *sigma_algebra.sigma_sets_subset* [*OF sls*, *of M*]
by *simp*
have *measure_space* Ω (*sigma_sets* Ω *M*) *?O*
by (*rule measure_down* [*OF mls*], *rule sigma_algebra_sigma_sets*)
(*simp_all add*: *sgs_sb space_closed*)
thus *?thesis* **using** *outer_measure_agrees* [*OF posf ca*]
by (*intro exI*[*of* _ *?O*]) *auto*
qed

lemma (**in** *ring_of_sets*) *caratheodory_empty_continuous*:
assumes *f*: *positive M f additive M f* **and** *fin*: $\bigwedge A.\ A \in M \Longrightarrow f\ A \neq \infty$
assumes *cont*: $\bigwedge A.\ range\ A \subseteq M \Longrightarrow decseq\ A \Longrightarrow (\bigcap i.\ A\ i) = \{\} \Longrightarrow (\lambda i.\ f\ (A\ i)) \longrightarrow 0$
shows $\exists \mu :: 'a\ set \Rightarrow ennreal.\ (\forall s \in M.\ \mu\ s = f\ s) \wedge measure\_space\ \Omega\ (sigma\_sets\ \Omega\ M)\ \mu$
proof (*intro caratheodory' empty_continuous_imp_countably_additive f*)
show $\forall A \in M.\ f\ A \neq \infty$ **using** *fin* **by** *auto*
qed (*rule cont*)

6.8.3 Volumes

definition *volume* :: *'a set set* ⇒ (*'a set* ⇒ *ennreal*) ⇒ *bool* **where**
volume M f ⟷
(*f* {} = *0*) ∧ (∀ *a*∈*M*. *0* ≤ *f a*) ∧
(∀ *C*⊆*M*. *disjoint C* ⟶ *finite C* ⟶ ⋃ *C* ∈ *M* ⟶ *f* (⋃ *C*) = (∑ *c*∈*C*. *f c*))

lemma *volumeI*:
assumes *f* {} = *0*
assumes $\bigwedge a.\ a \in M \Longrightarrow 0 \leq f\ a$
assumes $\bigwedge C.\ C \subseteq M \Longrightarrow disjoint\ C \Longrightarrow finite\ C \Longrightarrow \bigcup C \in M \Longrightarrow f\ (\bigcup C)$

$= (\sum c{\in}C.\ f\ c)$
shows *volume M f*
using *assms* **by** (*auto simp*: *volume_def*)

lemma *volume_positive*:
volume M f $\Longrightarrow$ *a* $\in$ *M* $\Longrightarrow$ *0* $\leq$ *f a*
by (*auto simp*: *volume_def*)

lemma *volume_empty*:
volume M f $\Longrightarrow$ *f* {} = *0*
by (*auto simp*: *volume_def*)

proposition *volume_finite_additive*:
assumes *volume M f*
assumes *A*: $\bigwedge i.\ i \in I \Longrightarrow A\ i \in M$ *disjoint_family_on A I finite I* $\bigcup (A\ `\ I) \in M$
shows *f* $(\bigcup (A\ `\ I)) = (\sum i{\in}I.\ f\ (A\ i))$
proof −
have *A`I* $\subseteq$ *M disjoint* (*A`I*) *finite* (*A`I*) $\bigcup (A`I) \in M$
using *A* **by** (*auto simp*: *disjoint_family_on_disjoint_image*)
with ‹*volume M f*› **have** *f* $(\bigcup (A`I)) = (\sum a{\in}A`I.\ f\ a)$
unfolding *volume_def* **by** *blast*
also have ... $= (\sum i{\in}I.\ f\ (A\ i))$
proof (*subst sum.reindex_nontrivial*)
fix *i j* **assume** *i* $\in$ *I j* $\in$ *I i* $\neq$ *j A i* = *A j*
with ‹*disjoint_family_on A I*› **have** *A i* = {}
by (*auto simp*: *disjoint_family_on_def*)
then show *f* (*A i*) = *0*
using *volume_empty*[*OF* ‹*volume M f*›] **by** *simp*
qed (*auto intro*: ‹*finite I*›)
finally show *f* $(\bigcup (A\ `\ I)) = (\sum i{\in}I.\ f\ (A\ i))$
by *simp*
qed

lemma (**in** *ring_of_sets*) *volume_additiveI*:
assumes *pos*: $\bigwedge a.\ a \in M \Longrightarrow 0 \leq \mu\ a$
assumes [*simp*]: $\mu\ \{\} = 0$
assumes *add*: $\bigwedge a\ b.\ a \in M \Longrightarrow b \in M \Longrightarrow a \cap b = \{\} \Longrightarrow \mu\ (a \cup b) = \mu\ a + \mu\ b$
shows *volume M* μ
proof (*unfold volume_def*, *safe*)
fix *C* **assume** *finite C C* $\subseteq$ *M disjoint C*
then show $\mu\ (\bigcup C) = sum\ \mu\ C$
proof (*induct C*)
case (*insert c C*)
from *insert*(*1*,*2*,*4*,*5*) **have** $\mu\ (\bigcup (insert\ c\ C)) = \mu\ c + \mu\ (\bigcup C)$
by (*auto intro*!: *add simp*: *disjoint_def*)
with *insert* **show** *?case*
by (*simp add*: *disjoint_def*)

```
  qed simp
qed fact+

proposition (in semiring_of_sets) extend_volume:
  assumes volume M μ
  shows ∃μ'. volume generated_ring μ' ∧ (∀a∈M. μ' a = μ a)
proof −
  let ?R = generated_ring
  have ∀a∈?R. ∃m. ∃C⊆M. a = ⋃C ∧ finite C ∧ disjoint C ∧ m = (∑c∈C. μ c)
    by (auto simp: generated_ring_def)
  from bchoice[OF this] obtain μ'
    where μ'_spec: ∀x∈generated_ring. ∃C⊆M. x = ⋃ C ∧ finite C ∧ disjoint C ∧ μ' x = sum μ C
    by blast
  { fix C assume C: C ⊆ M finite C disjoint C
    fix D assume D: D ⊆ M finite D disjoint D
    assume ⋃C = ⋃D
    have (∑d∈D. μ d) = (∑d∈D. ∑c∈C. μ (c ∩ d))
    proof (intro sum.cong refl)
      fix d assume d ∈ D
      have Un_eq_d: (⋃c∈C. c ∩ d) = d
        using ‹d ∈ D› ‹⋃C = ⋃D› by auto
      moreover have μ (⋃c∈C. c ∩ d) = (∑c∈C. μ (c ∩ d))
      proof (rule volume_finite_additive)
        { fix c assume c ∈ C then show c ∩ d ∈ M
            using C D ‹d ∈ D› by auto }
        show (⋃a∈C. a ∩ d) ∈ M
          unfolding Un_eq_d using ‹d ∈ D› D by auto
        show disjoint_family_on (λa. a ∩ d) C
          using ‹disjoint C› by (auto simp: disjoint_family_on_def disjoint_def)
      qed fact+
      ultimately show μ d = (∑c∈C. μ (c ∩ d)) by simp
    qed }
  note split_sum = this

  { fix C assume C: C ⊆ M finite C disjoint C
    fix D assume D: D ⊆ M finite D disjoint D
    assume ⋃C = ⋃D
    with split_sum[OF C D] split_sum[OF D C]
    have (∑d∈D. μ d) = (∑c∈C. μ c)
      by (simp, subst sum.swap, simp add: ac_simps) }
  note sum_eq = this

  { fix C assume C: C ⊆ M finite C disjoint C
    then have ⋃C ∈ ?R by (auto simp: generated_ring_def)
    with μ'_spec[THEN bspec, of ⋃C]
    obtain D where
      D: D ⊆ M finite D disjoint D ⋃C = ⋃D and μ' (⋃C) = (∑d∈D. μ d)
```

```
    by auto
  with sum_eq[OF C D] have μ' (⋃ C) = (∑ c∈C. μ c) by simp }
 note μ' = this

 show ?thesis
proof (intro exI conjI ring_of_sets.volume_additiveI[OF generating_ring] ballI)
  fix a assume a ∈ M with μ'[of {a}] show μ' a = μ a
    by (simp add: disjoint_def)
 next
  fix a assume a ∈ ?R
  then obtain Ca where Ca: finite Ca disjoint Ca Ca ⊆ M a = ⋃ Ca ..
  with μ'[of Ca] ‹volume M μ›[THEN volume_positive]
  show 0 ≤ μ' a
    by (auto intro!: sum_nonneg)
 next
  show μ' {} = 0 using μ'[of {}] by auto
 next
  fix a b assume a ∈ ?R b ∈ ?R
  then obtain Ca Cb
    where Ca: finite Ca disjoint Ca Ca ⊆ M a = ⋃ Ca
      and Cb: finite Cb disjoint Cb Cb ⊆ M b = ⋃ Cb
    by (meson generated_ringE)
  assume a ∩ b = {}
  with Ca Cb have Ca ∩ Cb ⊆ {{}} by auto
  then have C_Int_cases: Ca ∩ Cb = {{}} ∨ Ca ∩ Cb = {} by auto

  from ‹a ∩ b = {}› have μ' (⋃(Ca ∪ Cb)) = (∑ c∈Ca ∪ Cb. μ c)
    using Ca Cb by (intro μ') (auto intro!: disjoint_union)
  also have ... = (∑ c∈Ca ∪ Cb. μ c) + (∑ c∈Ca ∩ Cb. μ c)
      using C_Int_cases volume_empty[OF ‹volume M μ›] by (elim disjE)
simp_all
  also have ... = (∑ c∈Ca. μ c) + (∑ c∈Cb. μ c)
    using Ca Cb by (simp add: sum.union_inter)
  also have ... = μ' a + μ' b
    using Ca Cb by (simp add: μ')
  finally show μ' (a ∪ b) = μ' a + μ' b
    using Ca Cb by simp
 qed
qed
```

Caratheodory on semirings

```
theorem (in semiring_of_sets) caratheodory:
 assumes pos: positive M μ and ca: countably_additive M μ
  shows ∃μ' :: 'a set ⇒ ennreal. (∀ s ∈ M. μ' s = μ s) ∧ measure_space Ω
(sigma_sets Ω M) μ'
proof −
 have volume M μ
 proof (rule volumeI)
```

```
    { fix a assume a ∈ M then show 0 ≤ μ a
        using pos unfolding positive_def by auto }
    note p = this

    fix C assume sets_C: C ⊆ M ⋃C ∈ M and disjoint C finite C
    have ∃F'. bij_betw F' {..<card C} C
      by (rule finite_same_card_bij[OF _ ‹finite C›]) auto
    then obtain F' where bij_betw F' {..<card C} C ..
    then have F': C = F' ` {..< card C} inj_on F' {..< card C}
      by (auto simp: bij_betw_def)
    { fix i j assume *: i < card C j < card C i ≠ j
      with F' have F' i ∈ C F' j ∈ C F' i ≠ F' j
        unfolding inj_on_def by auto
      with ‹disjoint C›[THEN disjointD]
      have F' i ∩ F' j = {}
        by auto }
    note F'_disj = this
    define F where F i = (if i < card C then F' i else {}) for i
    then have disjoint_family F
      using F'_disj by (auto simp: disjoint_family_on_def)
    moreover from F' have (⋃i. F i) = ⋃C
      by (auto simp add: F_def split: if_split_asm cong del: SUP_cong)
    moreover have sets_F: ⋀i. F i ∈ M
      using F' sets_C by (auto simp: F_def)
    moreover note sets_C
    ultimately have μ (⋃C) = (∑i. μ (F i))
      using ca[unfolded countably_additive_def, THEN spec, of F] by auto
    also have ... = (∑i<card C. μ (F' i))
    proof -
      have (λi. if i ∈ {..< card C} then μ (F' i) else 0) sums (∑i<card C. μ (F'
i))
        by (rule sums_If_finite_set) auto
      also have (λi. if i ∈ {..< card C} then μ (F' i) else 0) = (λi. μ (F i))
        using pos by (auto simp: positive_def F_def)
      finally show (∑i. μ (F i)) = (∑i<card C. μ (F' i))
        by (simp add: sums_iff)
    qed
    also have ... = (∑c∈C. μ c)
      using F'(2) by (subst (2) F') (simp add: sum.reindex)
    finally show μ (⋃C) = (∑c∈C. μ c) .
  next
    show μ {} = 0
      using ‹positive M μ› by (rule positiveD1)
  qed
  from extend_volume[OF this] obtain μ_r where
    V: volume generated_ring μ_r ⋀a. a ∈ M ⟹ μ a = μ_r a
    by auto

  interpret G: ring_of_sets Ω generated_ring
```

by (*rule generating_ring*)

have *pos*: *positive generated_ring* μ_r
using *V* **unfolding** *positive_def* **by** (*auto simp*: *positive_def intro*!: *volume_positive volume_empty*)

have *countably_additive generated_ring* μ_r
proof (*rule countably_additiveI*)
fix $A' :: nat \Rightarrow 'a\ set$
assume A': *range* $A' \subseteq$ *generated_ring disjoint_family* A'
and *Un_A*: $(\bigcup i.\ A'\ i) \in$ *generated_ring*

obtain C' **where** C': *finite* C' *disjoint* C' $C' \subseteq M$ $\bigcup$ (*range* A') $= \bigcup\ C'$
using *generated_ringE*[*OF Un_A*] **by** *auto*

{ **fix** c **assume** $c \in C'$
moreover define A **where** [*abs_def*]: $A\ i = A'\ i \cap c$ **for** i
ultimately have A: *range* $A \subseteq$ *generated_ring disjoint_family* A
and *Un_A*: $(\bigcup i.\ A\ i) \in$ *generated_ring*
using A' C'
by (*auto intro*!: *G.Int G.finite_Union intro*: *generated_ringI_Basic simp*: *disjoint_family_on_def*)
from A C' ‹$c \in C'$› **have** *UN_eq*: $(\bigcup i.\ A\ i) = c$
by (*auto simp*: *A_def*)

have $\forall i{::}nat.\ \exists f{::}nat \Rightarrow 'a\ set.\ \mu_r\ (A\ i) = (\sum j.\ \mu_r\ (f\ j)) \wedge$ *disjoint_family* $f \wedge \bigcup$(*range* f) $= A\ i \wedge (\forall j.\ f\ j \in M)$
(**is** $\forall i.\ ?P\ i$)
proof
fix i
from A **have** *Ai*: $A\ i \in$ *generated_ring* **by** *auto*
from *generated_ringE*[*OF this*] **obtain** C
where C: *finite* C *disjoint* C $C \subseteq M$ $A\ i = \bigcup\ C$ **by** *auto*
have $\exists F'.$ *bij_betw* F' {..<*card* C} C
by (*rule finite_same_card_bij*[*OF _* ‹*finite* C›]) *auto*
then obtain F **where** F: *bij_betw* F {..<*card* C} C **..**
define f **where** [*abs_def*]: $f\ i = (if\ i < card\ C\ then\ F\ i\ else\ \{\})$ **for** i
then have f: *bij_betw* f {..< *card* C} C
by (*intro bij_betw_cong*[*THEN iffD1*, *OF _ F*]) *auto*
with C **have** $\forall j.\ f\ j \in M$
by (*auto simp*: *Pi_iff f_def dest*!: *bij_betw_imp_funcset*)
moreover
from f C **have** *d_f*: *disjoint_family_on* f {..<*card* C}
by (*intro disjoint_image_disjoint_family_on*) (*auto simp*: *bij_betw_def*)
then have *disjoint_family* f
by (*auto simp*: *disjoint_family_on_def f_def*)
moreover
have *Ai_eq*: $A\ i = (\bigcup x{<}card\ C.\ f\ x)$
using f C *Ai* **unfolding** *bij_betw_def* **by** *auto*

```
      then have ⋃(range f) = A i
        using f by (auto simp add: f_def)
      moreover
      { have (∑j. μ_r (f j)) = (∑j. if j ∈ {..< card C} then μ_r (f j) else 0)
            using volume_empty[OF V(1)] by (auto intro!: arg_cong[where
f=suminf] simp: f_def)
        also have ... = (∑j<card C. μ_r (f j))
          by (rule sums_If_finite_set[THEN sums_unique, symmetric]) simp
        also have ... = μ_r (A i)
          using C f[THEN bij_betw_imp_funcset] unfolding Ai_eq
          by (intro volume_finite_additive[OF V(1) _ d_f, symmetric])
            (auto simp: Pi_iff Ai_eq intro: generated_ringI_Basic)
        finally have μ_r (A i) = (∑j. μ_r (f j)) .. }
      ultimately show ?P i
        by blast
    qed
    from choice[OF this] obtain f
      where f: ∀x. μ_r (A x) = (∑j. μ_r (f x j)) ∧ disjoint_family (f x) ∧ ⋃
(range (f x)) = A x ∧ (∀j. f x j ∈ M)
      ..
    then have UN_f_eq: (⋃i. case_prod f (prod_decode i)) = (⋃i. A i)
     unfolding UN_extend_simps surj_prod_decode by (auto simp: set_eq_iff)

    have d: disjoint_family (λi. case_prod f (prod_decode i))
      unfolding disjoint_family_on_def
    proof (intro ballI impI)
      fix m n :: nat assume m ≠ n
      then have neq: prod_decode m ≠ prod_decode n
        using inj_prod_decode[of UNIV] by (auto simp: inj_on_def)
      show case_prod f (prod_decode m) ∩ case_prod f (prod_decode n) = {}
      proof cases
        assume fst (prod_decode m) = fst (prod_decode n)
        then show ?thesis
          using neq f by (fastforce simp: disjoint_family_on_def)
      next
        assume neq: fst (prod_decode m) ≠ fst (prod_decode n)
        have case_prod f (prod_decode m) ⊆ A (fst (prod_decode m))
          case_prod f (prod_decode n) ⊆ A (fst (prod_decode n))
          using f[THEN spec, of fst (prod_decode m)]
          using f[THEN spec, of fst (prod_decode n)]
          by (auto simp: set_eq_iff)
        with f A neq show ?thesis
          by (fastforce simp: disjoint_family_on_def subset_eq set_eq_iff)
      qed
    qed
    from f have (∑n. μ_r (A n)) = (∑n. μ_r (case_prod f (prod_decode n)))
      by (intro suminf_ennreal_2dimen[symmetric] generated_ringI_Basic)
       (auto split: prod.split)
    also have ... = (∑n. μ (case_prod f (prod_decode n)))
```

```
      using f V(2) by (auto intro!: arg_cong[where f=suminf] split: prod.split)
    also have ... = μ (⋃i. case_prod f (prod_decode i))
      using f ‹c ∈ C'› C'
      by (intro ca[unfolded countably_additive_def, rule_format])
        (auto split: prod.split simp: UN_f_eq d UN_eq)
    finally have (∑n. μ_r (A' n ∩ c)) = μ c
      using UN_f_eq UN_eq by (simp add: A_def) }
  note eq = this

  have (∑n. μ_r (A' n)) = (∑n. ∑c∈C'. μ_r (A' n ∩ c))
    using C' A'
    by (subst volume_finite_additive[symmetric, OF V(1)])
      (auto simp: disjoint_def disjoint_family_on_def
          intro!: G.Int G.finite_Union arg_cong[where f=λX. suminf (λi. μ_r
(X i))] ext
          intro: generated_ringI_Basic)
  also have ... = (∑c∈C'. ∑n. μ_r (A' n ∩ c))
    using C' A'
   by (intro suminf_sum G.Int G.finite_Union) (auto intro: generated_ringI_Basic)
  also have ... = (∑c∈C'. μ_r c)
    using eq V C' by (auto intro!: sum.cong)
  also have ... = μ_r (⋃C')
    using C' Un_A
    by (subst volume_finite_additive[symmetric, OF V(1)])
      (auto simp: disjoint_family_on_def disjoint_def
          intro: generated_ringI_Basic)
  finally show (∑n. μ_r (A' n)) = μ_r (⋃i. A' i)
    using C' by simp
 qed
  obtain μ' where (∀s∈generated_ring. μ' s = μ_r s) ∧ measure_space Ω
(sigma_sets Ω generated_ring) μ'
   using G.caratheodory'[OF pos ‹countably_additive generated_ring μ_r›] by
auto
 with V show ?thesis
   unfolding sigma_sets_generated_ring_eq
   by (intro exI[of _ μ']) (auto intro: generated_ringI_Basic)
qed

lemma extend_measure_caratheodory:
 fixes G :: 'i ⇒ 'a set
 assumes M: M = extend_measure Ω I G μ
 assumes i ∈ I
 assumes semiring_of_sets Ω (G ‘ I)
 assumes empty: ⋀i. i ∈ I ⟹ G i = {} ⟹ μ i = 0
 assumes inj: ⋀i j. i ∈ I ⟹ j ∈ I ⟹ G i = G j ⟹ μ i = μ j
 assumes nonneg: ⋀i. i ∈ I ⟹ 0 ≤ μ i
 assumes add: ⋀A::nat ⇒ 'i. ⋀j. A ∈ UNIV → I ⟹ j ∈ I ⟹ disjoint_family
(G ∘ A) ⟹
   (⋃i. G (A i)) = G j ⟹ (∑n. μ (A n)) = μ j
```

shows *emeasure M (G i) = μ i*

proof −
interpret *semiring_of_sets Ω G ‘ I*
by *fact*
have *∀ g∈G‘I. ∃ i∈I. g = G i*
by *auto*
then obtain *sel* **where** *sel*: *⋀g. g ∈ G ‘ I ⟹ sel g ∈ I ⋀g. g ∈ G ‘ I ⟹ G (sel g) = g*
by *metis*

have *∃ μ'. (∀ s∈G ‘ I. μ' s = μ (sel s)) ∧ measure_space Ω (sigma_sets Ω (G ‘ I)) μ'*
proof (*rule caratheodory*)
show *positive (G ‘ I) (λs. μ (sel s))*
by (*auto simp*: *positive_def intro*!: *empty sel nonneg*)
show *countably_additive (G ‘ I) (λs. μ (sel s))*
proof (*rule countably_additiveI*)
fix *A :: nat ⇒ 'a set* **assume** *range A ⊆ G ‘ I disjoint_family A (⋃i. A i) ∈ G ‘ I*
then show *(∑ i. μ (sel (A i))) = μ (sel (⋃ i. A i))*
by (*intro add*) (*auto simp*: *sel image_subset_iff_funcset comp_def Pi_iff intro*!: *sel*)
qed
qed
then obtain *μ'* **where** *μ'*: *∀ s∈G ‘ I. μ' s = μ (sel s) measure_space Ω (sigma_sets Ω (G ‘ I)) μ'*
by *metis*

show *?thesis*
proof (*rule emeasure_extend_measure*[*OF M*])
{ fix *i* **assume** *i ∈ I* **then show** *μ' (G i) = μ i*
using *μ'* **by** (*auto intro*!: *inj sel*) **}**
show *G ‘ I ⊆ Pow Ω*
by (*rule space_closed*)
then show *positive (sets M) μ' countably_additive (sets M) μ'*
using *μ'* **by** (*simp_all add*: *M sets_extend_measure measure_space_def*)
qed *fact*
qed

proposition *extend_measure_caratheodory_pair*:
fixes *G :: 'i ⇒ 'j ⇒ 'a set*
assumes *M*: *M = extend_measure Ω {(a, b). P a b} (λ(a, b). G a b) (λ(a, b). μ a b)*
assumes *P i j*
assumes *semiring*: *semiring_of_sets Ω {G a b | a b. P a b}*
assumes *empty*: *⋀i j. P i j ⟹ G i j = {} ⟹ μ i j = 0*
assumes *inj*: *⋀i j k l. P i j ⟹ P k l ⟹ G i j = G k l ⟹ μ i j = μ k l*
assumes *nonneg*: *⋀i j. P i j ⟹ 0 ≤ μ i j*

assumes *add*: $\bigwedge A{::}nat \Rightarrow 'i.\ \bigwedge B{::}nat \Rightarrow 'j.\ \bigwedge j\ k.$
$(\bigwedge n.\ P\ (A\ n)\ (B\ n)) \Longrightarrow P\ j\ k \Longrightarrow disjoint\_family\ (\lambda n.\ G\ (A\ n)\ (B\ n)) \Longrightarrow$
$(\bigcup i.\ G\ (A\ i)\ (B\ i)) = G\ j\ k \Longrightarrow (\sum n.\ \mu\ (A\ n)\ (B\ n)) = \mu\ j\ k$
shows *emeasure M* $(G\ i\ j) = \mu\ i\ j$
proof −
have *emeasure M* $((\lambda(a,\ b).\ G\ a\ b)\ (i,\ j)) = (\lambda(a,\ b).\ \mu\ a\ b)\ (i,\ j)$
proof (*rule extend_measure_caratheodory*[*OF M*])
show *semiring_of_sets* Ω $((\lambda(a,\ b).\ G\ a\ b)\ `\ \{(a,\ b).\ P\ a\ b\})$
using *semiring* **by** (*simp add*: *image_def conj_commute*)
next
fix $A :: nat \Rightarrow ('i \times 'j)$ **and** j **assume** $A \in UNIV \rightarrow \{(a,\ b).\ P\ a\ b\}\ j \in \{(a,\ b).\ P\ a\ b\}$
disjoint_family $((\lambda(a,\ b).\ G\ a\ b) \circ A)$
$(\bigcup i.\ case\ A\ i\ of\ (a,\ b) \Rightarrow G\ a\ b) = (case\ j\ of\ (a,\ b) \Rightarrow G\ a\ b)$
then show $(\sum n.\ case\ A\ n\ of\ (a,\ b) \Rightarrow \mu\ a\ b) = (case\ j\ of\ (a,\ b) \Rightarrow \mu\ a\ b)$
using *add*[*of* $\lambda i.\ fst\ (A\ i)\ \lambda i.\ snd\ (A\ i)\ fst\ j\ snd\ j$]
by (*simp add*: *split_beta' comp_def Pi_iff*)
qed (*auto split*: *prod.splits intro*: *assms*)
then show *?thesis* **by** *simp*
qed

end

6.9 Bochner Integration for Vector-Valued Functions

theory *Bochner_Integration*
imports *Finite_Product_Measure*
begin

In the following development of the Bochner integral we use second countable topologies instead of separable spaces. A second countable topology is also separable.

proposition *borel_measurable_implies_sequence_metric*:
fixes $f :: 'a \Rightarrow 'b :: \{metric\_space,\ second\_countable\_topology\}$
assumes [*measurable*]: $f \in borel\_measurable\ M$
shows $\exists F.\ (\forall i.\ simple\_function\ M\ (F\ i)) \wedge (\forall x{\in}space\ M.\ (\lambda i.\ F\ i\ x) \longrightarrow f\ x) \wedge$
$(\forall i.\ \forall x{\in}space\ M.\ dist\ (F\ i\ x)\ z \leq 2 * dist\ (f\ x)\ z)$
proof −
obtain $D :: 'b\ set$ **where** *countable D* **and** *D*: $\bigwedge X.\ open\ X \Longrightarrow X \neq \{\} \Longrightarrow \exists d{\in}D.\ d \in X$
by (*erule countable_dense_setE*)

define *e* **where** $e = from\_nat\_into\ D$
{ fix *n x*
obtain *d* **where** $d \in D$ **and** *d*: $d \in ball\ x\ (1\ /\ Suc\ n)$
using *D*[*of ball x* $(1\ /\ Suc\ n)$] **by** *auto*

```
    from ‹d ∈ D› D[of UNIV] ‹countable D› obtain i where d = e i
      unfolding e_def by (auto dest: from_nat_into_surj)
    with d have ∃ i. dist x (e i) < 1 / Suc n
      by auto }
  note e = this

  define A where [abs_def]: A m n =
    {x∈space M. dist (f x) (e n) < 1 / (Suc m) ∧ 1 / (Suc m) ≤ dist (f x) z} for
m n
  define B where [abs_def]: B m = disjointed (A m) for m

  define m where [abs_def]: m N x = Max {m. m ≤ N ∧ x ∈ (⋃ n≤N. B m n)}
for N x
  define F where [abs_def]: F N x =
    (if (∃ m≤N. x ∈ (⋃ n≤N. B m n)) ∧ (∃ n≤N. x ∈ B (m N x) n)
     then e (LEAST n. x ∈ B (m N x) n) else z) for N x

  have B_imp_A[intro, simp]: ⋀x m n. x ∈ B m n ⟹ x ∈ A m n
    using disjointed_subset[of A m for m] unfolding B_def by auto

  { fix m
    have ⋀n. A m n ∈ sets M
      by (auto simp: A_def)
    then have ⋀n. B m n ∈ sets M
      using sets.range_disjointed_sets[of A m M] by (auto simp: B_def) }
  note this[measurable]

  { fix N i x assume ∃ m≤N. x ∈ (⋃ n≤N. B m n)
    then have m N x ∈ {m::nat. m ≤ N ∧ x ∈ (⋃ n≤N. B m n)}
      unfolding m_def by (intro Max_in) auto
    then have m N x ≤ N ∃ n≤N. x ∈ B (m N x) n
      by auto }
  note m = this

  { fix j N i x assume j ≤ N i ≤ N x ∈ B j i
    then have j ≤ m N x
      unfolding m_def by (intro Max_ge) auto }
  note m_upper = this

  show ?thesis
    unfolding simple_function_def
  proof (safe intro!: exI[of _ F])
    have [measurable]: ⋀i. F i ∈ borel_measurable M
      unfolding F_def m_def by measurable
    show ⋀x i. F i -` {x} ∩ space M ∈ sets M
      by measurable

    { fix i
      { fix n x assume x ∈ B (m i x) n
```

```
      then have (LEAST n. x ∈ B (m i x) n) ≤ n
        by (intro Least_le)
      also assume n ≤ i
      finally have (LEAST n. x ∈ B (m i x) n) ≤ i . }
    then have F i ‘ space M ⊆ {z} ∪ e ‘ {.. i}
      by (auto simp: F_def)
    then show finite (F i ‘ space M)
      by (rule finite_subset) auto }

  { fix N i n x assume i ≤ N n ≤ N x ∈ B i n
    then have 1: ∃ m≤N. x ∈ (⋃ n≤N. B m n) by auto
    from m[OF this] obtain n where n: m N x ≤ N n ≤ N x ∈ B (m N x) n
by auto
    moreover
    define L where L = (LEAST n. x ∈ B (m N x) n)
    have dist (f x) (e L) < 1 / Suc (m N x)
    proof −
      have x ∈ B (m N x) L
        using n(3) unfolding L_def by (rule LeastI)
      then have x ∈ A (m N x) L
        by auto
      then show ?thesis
        unfolding A_def by simp
    qed
    ultimately have dist (f x) (F N x) < 1 / Suc (m N x)
      by (auto simp: F_def L_def) }
  note ∗ = this

  fix x assume x ∈ space M
  show (λi. F i x) ⟶ f x
  proof (cases f x = z)
    case True
    then have ⋀i n. x ∉ A i n
      unfolding A_def by auto
    then show ?thesis
      by (metis B_imp_A F_def LIMSEQ_def True dist_self)
  next
    case False
    show ?thesis
    proof (rule tendstoI)
      fix e :: real assume 0 < e
      with ‹f x ≠ z› obtain n where 1 / Suc n < e 1 / Suc n < dist (f x) z
        by (metis dist_nz order_less_trans neq_iff nat_approx_posE)
      with ‹x∈space M› ‹f x ≠ z› have x ∈ (⋃ i. B n i)
        unfolding A_def B_def UN_disjointed_eq using e by auto
      then obtain i where i: x ∈ B n i by auto

      show eventually (λi. dist (F i x) (f x) < e) sequentially
        using eventually_ge_at_top[of max n i]
```

```
      proof eventually_elim
        fix j assume j: max n i ≤ j
        with i have dist (f x) (F j x) < 1 / Suc (m j x)
          by (intro *[OF _ _ i]) auto
        also have ... ≤ 1 / Suc n
          using j m_upper[OF _ _ i]
          by (auto simp: field_simps)
        also note ‹1 / Suc n < e›
        finally show dist (F j x) (f x) < e
          by (simp add: less_imp_le dist_commute)
      qed
    qed
   qed
   fix i
   { fix n m assume x ∈ A n m
    then have dist (e m) (f x) + dist (f x) z ≤ 2 * dist (f x) z
      unfolding A_def by (auto simp: dist_commute)
    also have dist (e m) z ≤ dist (e m) (f x) + dist (f x) z
      by (rule dist_triangle)
    finally (xtrans) have dist (e m) z ≤ 2 * dist (f x) z . }
   then show dist (F i x) z ≤ 2 * dist (f x) z
    unfolding F_def
   by (smt (verit, ccfv_threshold) LeastI2 B_imp_A dist_eq_0_iff zero_le_dist)
  qed
qed

lemma
  fixes f :: 'a ⇒ 'b::semiring_1 assumes finite A
   shows sum_mult_indicator[simp]: (∑ x ∈ A. f x * indicator (B x) (g x)) =
(∑ x∈{x∈A. g x ∈ B x}. f x)
    and sum_indicator_mult[simp]: (∑ x ∈ A. indicator (B x) (g x) * f x) =
(∑ x∈{x∈A. g x ∈ B x}. f x)
  unfolding indicator_def
  using assms by (auto intro!: sum.mono_neutral_cong_right split: if_split_asm)

lemma borel_measurable_induct_real[consumes 2, case_names set mult add seq]:
  fixes P :: ('a ⇒ real) ⇒ bool
  assumes u: u ∈ borel_measurable M ⋀x. 0 ≤ u x
  assumes set: ⋀A. A ∈ sets M ⟹ P (indicator A)
  assumes mult: ⋀u c. 0 ≤ c ⟹ u ∈ borel_measurable M ⟹ (⋀x. 0 ≤ u x)
⟹ P u ⟹ P (λx. c * u x)
  assumes add: ⋀u v. u ∈ borel_measurable M ⟹ (⋀x. 0 ≤ u x) ⟹ P u ⟹
v ∈ borel_measurable M ⟹ (⋀x. 0 ≤ v x) ⟹ (⋀x. x ∈ space M ⟹ u x = 0
∨ v x = 0) ⟹ P v ⟹ P (λx. v x + u x)
  assumes seq: ⋀U. (⋀i. U i ∈ borel_measurable M) ⟹ (⋀i x. 0 ≤ U i x) ⟹
(⋀i. P (U i)) ⟹ incseq U ⟹ (⋀x. x ∈ space M ⟹ (λi. U i x) ⟶ u x)
⟹ P u
  shows P u
proof −
```

```
  have (λx. ennreal (u x)) ∈ borel_measurable M using u by auto
  from borel_measurable_implies_simple_function_sequence'[OF this]
  obtain U where U: ⋀i. simple_function M (U i) incseq U ⋀i x. U i x < top
and
    sup: ⋀x. (SUP i. U i x) = ennreal (u x)
    by blast

  define U' where [abs_def]: U' i x = indicator (space M) x * enn2real (U i x)
for i x
  then have U'_sf[measurable]: ⋀i. simple_function M (U' i)
    using U by (auto intro!: simple_function_compose1[where g=enn2real])

  show P u
  proof (rule seq)
    show U': U' i ∈ borel_measurable M ⋀x. 0 ≤ U' i x for i
      using U'_sf by (auto simp: U'_def borel_measurable_simple_function)
    show incseq U'
      using U(2,3)
      by (auto simp: incseq_def le_fun_def image_iff eq_commute U'_def indica-
tor_def enn2real_mono)

    fix x assume x: x ∈ space M
    have (λi. U i x) ⟶ (SUP i. U i x)
      using U(2) by (intro LIMSEQ_SUP) (auto simp: incseq_def le_fun_def)
    moreover have (λi. U i x) = (λi. ennreal (U' i x))
      using x U(3) by (auto simp: fun_eq_iff U'_def image_iff eq_commute)
    moreover have (SUP i. U i x) = ennreal (u x)
      using sup u(2) by (simp add: max_def)
    ultimately show (λi. U' i x) ⟶ u x
      using u U' by simp
  next
    fix i
    have U' i ‘ space M ⊆ enn2real ‘ (U i ‘ space M) finite (U i ‘ space M)
      unfolding U'_def using U(1) by (auto dest: simple_functionD)
    then have fin: finite (U' i ‘ space M)
      by (metis finite_subset finite_imageI)
    moreover have ⋀z. {y. U' i z = y ∧ y ∈ U' i ‘ space M ∧ z ∈ space M} =
(if z ∈ space M then {U' i z} else {})
      by auto
    ultimately have U': (λz. ∑ y∈U' i‘space M. y * indicator {x∈space M. U' i
x = y} z) = U' i
      by (simp add: U'_def fun_eq_iff)
    have ⋀x. x ∈ U' i ‘ space M ⟹ 0 ≤ x
      by (auto simp: U'_def)
    with fin have P (λz. ∑ y∈U' i‘space M. y * indicator {x∈space M. U' i x =
y} z)
    proof induct
      case empty from set[of {}] show ?case
        by (simp add: indicator_def[abs_def])
```

```
    next
      case (insert x F)
      from insert.prems have nonneg: x ≥ 0 ⋀y. y ∈ F ⟹ y ≥ 0
        by simp_all
      hence ∗: P (λxa. x ∗ indicat_real {x' ∈ space M. U' i x' = x} xa)
        by (intro mult set) auto
      have P (λz. x ∗ indicat_real {x' ∈ space M. U' i x' = x} z +
                  (∑ y∈F. y ∗ indicat_real {x ∈ space M. U' i x = y} z))
        using insert(1−3)
        by (intro add ∗ sum_nonneg mult_nonneg_nonneg)
           (auto simp: nonneg indicator_def of_bool_def sum_nonneg_eq_0_iff)
      thus ?case
        using insert.hyps by (subst sum.insert) auto
    qed
    with U' show P (U' i) by simp
  qed
qed

lemma scaleR_cong_right:
  fixes x :: 'a :: real_vector
  shows (x ≠ 0 ⟹ r = p) ⟹ r ∗R x = p ∗R x
  by auto

inductive simple_bochner_integrable :: 'a measure ⇒ ('a ⇒ 'b::real_vector) ⇒
bool for M f where
  simple_function M f ⟹ emeasure M {y∈space M. f y ≠ 0} ≠ ∞ ⟹
    simple_bochner_integrable M f

lemma simple_bochner_integrable_compose2:
  assumes p_0: p 0 0 = 0
  shows simple_bochner_integrable M f ⟹ simple_bochner_integrable M g ⟹
    simple_bochner_integrable M (λx. p (f x) (g x))
proof (safe intro!: simple_bochner_integrable.intros elim!: simple_bochner_integrable.cases
del: notI)
  assume sf: simple_function M f simple_function M g
  then show simple_function M (λx. p (f x) (g x))
    by (rule simple_function_compose2)

  from sf have [measurable]:
      f ∈ measurable M (count_space UNIV)
      g ∈ measurable M (count_space UNIV)
    by (auto intro: measurable_simple_function)

  assume fin: emeasure M {y ∈ space M. f y ≠ 0} ≠ ∞ emeasure M {y ∈ space
M. g y ≠ 0} ≠ ∞

  have emeasure M {x∈space M. p (f x) (g x) ≠ 0} ≤
      emeasure M ({x∈space M. f x ≠ 0} ∪ {x∈space M. g x ≠ 0})
    by (intro emeasure_mono) (auto simp: p_0)
```

also have ... ≤ *emeasure M* {*x*∈*space M*. *f x* ≠ *0*} + *emeasure M* {*x*∈*space M*. *g x* ≠ *0*}
by (*intro emeasure_subadditive*) *auto*
finally show *emeasure M* {*y* ∈ *space M*. *p* (*f y*) (*g y*) ≠ *0*} ≠ ∞
using *fin* **by** (*auto simp*: *top_unique*)
qed

lemma *simple_function_finite_support*:
assumes *f*: *simple_function M f* **and** *fin*: (∫⁺*x*. *f x* ∂*M*) < ∞ **and** *nn*: ⋀*x*. *0* ≤ *f x*
shows *emeasure M* {*x*∈*space M*. *f x* ≠ *0*} ≠ ∞
proof *cases*
from *f* **have** *meas*[*measurable*]: *f* ∈ *borel_measurable M*
by (*rule borel_measurable_simple_function*)

assume *non_empty*: ∃*x*∈*space M*. *f x* ≠ *0*

define *m* **where** *m* = *Min* (*f*‘*space M* − {*0*})
have *m* ∈ *f*‘*space M* − {*0*}
unfolding *m_def* **using** *f non_empty* **by** (*intro Min_in*) (*auto simp*: *simple_function_def*)
then have *m*: *0* < *m*
using *nn* **by** (*auto simp*: *less_le*)

from *m* **have** *m* ∗ *emeasure M* {*x*∈*space M*. *0* ≠ *f x*} =
(∫⁺*x*. *m* ∗ *indicator* {*x*∈*space M*. *0* ≠ *f x*} *x* ∂*M*)
using *f* **by** (*intro nn_integral_cmult_indicator*[*symmetric*]) *auto*
also have ... ≤ (∫⁺*x*. *f x* ∂*M*)
using *AE_space*
proof (*intro nn_integral_mono_AE*, *eventually_elim*)
fix *x* **assume** *x* ∈ *space M*
with *nn* **show** *m* ∗ *indicator* {*x* ∈ *space M*. *0* ≠ *f x*} *x* ≤ *f x*
using *f* **by** (*auto split*: *split_indicator simp*: *simple_function_def m_def*)
qed
also note ‹... < ∞›
finally show *?thesis*
using *m* **by** (*auto simp*: *ennreal_mult_less_top*)
next
assume ¬ (∃*x*∈*space M*. *f x* ≠ *0*)
with *nn* **have** ∗: {*x*∈*space M*. *f x* ≠ *0*} = {}
by *auto*
show *?thesis* **unfolding** ∗ **by** *simp*
qed

lemma *simple_bochner_integrableI_bounded*:
assumes *f*: *simple_function M f* **and** *fin*: (∫⁺*x*. *norm* (*f x*) ∂*M*) < ∞
shows *simple_bochner_integrable M f*
proof
have *emeasure M* {*y* ∈ *space M*. *ennreal* (*norm* (*f y*)) ≠ *0*} ≠ ∞

using *simple_function_finite_support simple_function_compose1 f fin* **by** *force*
then show *emeasure M* $\{y \in space\ M.\ f\ y \neq 0\} \neq \infty$ **by** *simp*
qed *fact*

definition *simple_bochner_integral* :: *'a measure* $\Rightarrow$ (*'a* $\Rightarrow$ *'b*::*real_vector*) $\Rightarrow$ *'b*
where
simple_bochner_integral M f = ($\sum$ *y*∈*f*‘*space M*. *measure M* {*x*∈*space M*. *f x* = *y*} $*_R$ *y*)

proposition *simple_bochner_integral_partition*:
assumes *f*: *simple_bochner_integrable M f* **and** *g*: *simple_function M g*
assumes *sub*: $\bigwedge x\ y.\ x \in space\ M \Longrightarrow y \in space\ M \Longrightarrow g\ x = g\ y \Longrightarrow f\ x = f\ y$
assumes *v*: $\bigwedge x.\ x \in space\ M \Longrightarrow f\ x = v\ (g\ x)$
shows *simple_bochner_integral M f* = ($\sum$ *y*∈*g* ‘ *space M*. *measure M* {*x*∈*space M*. *g x* = *y*} $*_R$ *v y*)
(**is** *_* = *?r*)
proof −
from *f g* **have** [*simp*]: *finite* (*f*‘*space M*) *finite* (*g*‘*space M*)
by (*auto simp*: *simple_function_def elim*: *simple_bochner_integrable.cases*)

from *f* **have** [*measurable*]: *f* ∈ *measurable M* (*count_space UNIV*)
by (*auto intro*: *measurable_simple_function elim*: *simple_bochner_integrable.cases*)

from *g* **have** [*measurable*]: *g* ∈ *measurable M* (*count_space UNIV*)
by (*auto intro*: *measurable_simple_function elim*: *simple_bochner_integrable.cases*)

{ fix *y* **assume** *y* ∈ *space M*
then have *f* ‘ *space M* ∩ {*i*. ∃ *x*∈*space M*. *i* = *f x* ∧ *g y* = *g x*} = {*v* (*g y*)}
by (*auto cong*: *sub simp*: *v*[*symmetric*]) **}**
note *eq* = *this*

have *simple_bochner_integral M f* =
($\sum$ *y*∈*f*‘*space M*. ($\sum$ *z*∈*g*‘*space M*.
if ∃ *x*∈*space M*. *y* = *f x* ∧ *z* = *g x then measure M* {*x*∈*space M*. *g x* = *z*} *else 0*) $*_R$ *y*)
unfolding *simple_bochner_integral_def*
proof (*safe intro*!: *sum.cong scaleR_cong_right*)
fix *y* **assume** *y*: *y* ∈ *space M f y* ≠ *0*
have [*simp*]: *g* ‘ *space M* ∩ {*z*. ∃ *x*∈*space M*. *f y* = *f x* ∧ *z* = *g x*} =
{*z*. ∃ *x*∈*space M*. *f y* = *f x* ∧ *z* = *g x*}
by *auto*
have *eq*:{*x* ∈ *space M*. *f x* = *f y*} =
($\bigcup$ *i*∈{*z*. ∃ *x*∈*space M*. *f y* = *f x* ∧ *z* = *g x*}. {*x* ∈ *space M*. *g x* = *i*})
by (*auto simp*: *eq_commute cong*: *sub rev_conj_cong*)
have *finite* (*g*‘*space M*) **by** *simp*
then have *finite* {*z*. ∃ *x*∈*space M*. *f y* = *f x* ∧ *z* = *g x*}
by (*rule rev_finite_subset*) *auto*
moreover
{ fix *x* **assume** *x* ∈ *space M f x* = *f y*

```
    then have x ∈ space M f x ≠ 0
      using y by auto
    then have emeasure M {y ∈ space M. g y = g x} ≤ emeasure M {y ∈ space
M. f y ≠ 0}
      by (auto intro!: emeasure_mono cong: sub)
    then have emeasure M {xa ∈ space M. g xa = g x} < ∞
      using f by (auto simp: simple_bochner_integrable.simps less_top) }
  ultimately
  show measure M {x ∈ space M. f x = f y} =
    (∑ z∈g ‘ space M. if ∃x∈space M. f y = f x ∧ z = g x then measure M {x
∈ space M. g x = z} else 0)
    apply (simp add: sum.If_cases eq)
    apply (subst measure_finite_Union[symmetric])
    apply (auto simp: disjoint_family_on_def less_top)
    done
 qed
 also have ... = (∑ y∈f‘space M. (∑ z∈g‘space M.
    if ∃x∈space M. y = f x ∧ z = g x then measure M {x∈space M. g x = z} *R
y else 0))
  by (auto intro!: sum.cong simp: scaleR_sum_left)
 also have ... = ?r
  by (subst sum.swap)
    (auto intro!: sum.cong simp: sum.If_cases scaleR_sum_right[symmetric] eq)
 finally show simple_bochner_integral M f = ?r .
qed

lemma simple_bochner_integral_add:
 assumes f: simple_bochner_integrable M f and g: simple_bochner_integrable
M g
 shows simple_bochner_integral M (λx. f x + g x) =
   simple_bochner_integral M f + simple_bochner_integral M g
proof −
 from f g have simple_bochner_integral M (λx. f x + g x) =
   (∑ y∈(λx. (f x, g x)) ‘ space M. measure M {x ∈ space M. (f x, g x) = y} *R
(fst y + snd y))
   by (intro simple_bochner_integral_partition)
    (auto simp: simple_bochner_integrable_compose2 elim: simple_bochner_integrable.cases)
 moreover from f g have simple_bochner_integral M f =
   (∑ y∈(λx. (f x, g x)) ‘ space M. measure M {x ∈ space M. (f x, g x) = y} *R
fst y)
   by (intro simple_bochner_integral_partition)
    (auto simp: simple_bochner_integrable_compose2 elim: simple_bochner_integrable.cases)
 moreover from f g have simple_bochner_integral M g =
   (∑ y∈(λx. (f x, g x)) ‘ space M. measure M {x ∈ space M. (f x, g x) = y} *R
snd y)
   by (intro simple_bochner_integral_partition)
    (auto simp: simple_bochner_integrable_compose2 elim: simple_bochner_integrable.cases)
 ultimately show ?thesis
   by (simp add: sum.distrib[symmetric] scaleR_add_right)
```

qed

lemma *simple_bochner_integral_linear*:
assumes *linear f*
assumes *g*: *simple_bochner_integrable M g*
shows *simple_bochner_integral M* (λ*x*. *f* (*g x*)) = *f* (*simple_bochner_integral M g*)
proof −
interpret *linear f* **by** *fact*
from *g* **have** *simple_bochner_integral M* (λ*x*. *f* (*g x*)) =
(∑ *y*∈*g* ‘ *space M*. *measure M* {*x* ∈ *space M*. *g x* = *y*} ∗$_R$ *f y*)
by (*intro simple_bochner_integral_partition*)
(*auto simp*: *simple_bochner_integrable_compose2*[**where** *p*=λ*x y*. *f x*]
elim: *simple_bochner_integrable.cases*)
also have ... = *f* (*simple_bochner_integral M g*)
by (*simp add*: *simple_bochner_integral_def sum scale*)
finally show *?thesis* **.**
qed

lemma *simple_bochner_integral_minus*:
assumes *f*: *simple_bochner_integrable M f*
shows *simple_bochner_integral M* (λ*x*. − *f x*) = − *simple_bochner_integral M f*
proof −
from *linear_uminus f* **show** *?thesis*
by (*rule simple_bochner_integral_linear*)
qed

lemma *simple_bochner_integral_diff*:
assumes *f*: *simple_bochner_integrable M f* **and** *g*: *simple_bochner_integrable M g*
shows *simple_bochner_integral M* (λ*x*. *f x* − *g x*) =
simple_bochner_integral M f − *simple_bochner_integral M g*
unfolding *diff_conv_add_uminus* **using** *f g*
by (*subst simple_bochner_integral_add*)
(*auto simp*: *simple_bochner_integral_minus simple_bochner_integrable_compose2*[**where** *p*=λ*x y*. − *y*])

lemma *simple_bochner_integral_norm_bound*:
assumes *f*: *simple_bochner_integrable M f*
shows *norm* (*simple_bochner_integral M f*) ≤ *simple_bochner_integral M* (λ*x*. *norm* (*f x*))
proof −
have *norm* (*simple_bochner_integral M f*) ≤
(∑ *y*∈*f* ‘ *space M*. *norm* (*measure M* {*x* ∈ *space M*. *f x* = *y*} ∗$_R$ *y*))
unfolding *simple_bochner_integral_def* **by** (*rule norm_sum*)
also have ... = (∑ *y*∈*f* ‘ *space M*. *measure M* {*x* ∈ *space M*. *f x* = *y*} ∗$_R$ *norm y*)
by *simp*

```
  also have ... = simple_bochner_integral M (λx. norm (f x))
    using f
    by (intro simple_bochner_integral_partition[symmetric])
     (auto intro: f simple_bochner_integrable_compose2 elim: simple_bochner_integrable.cases)
  finally show ?thesis .
qed

lemma simple_bochner_integral_nonneg[simp]:
  fixes f :: 'a ⇒ real
  shows (⋀x. 0 ≤ f x) ⟹ 0 ≤ simple_bochner_integral M f
  by (force simp: simple_bochner_integral_def intro: sum_nonneg)

lemma simple_bochner_integral_eq_nn_integral:
  assumes f: simple_bochner_integrable M f ⋀x. 0 ≤ f x
  shows simple_bochner_integral M f = (∫⁺x. f x ∂M)
proof −
  have ennreal_cong_mult: (x ≠ 0 ⟹ y = z) ⟹ ennreal x * y = ennreal x * z
for x y z
      by fastforce

  have [measurable]: f ∈ borel_measurable M
   by (meson borel_measurable_simple_function f(1) simple_bochner_integrable.cases)

  { fix y assume y: y ∈ space M f y ≠ 0
    have ennreal (measure M {x ∈ space M. f x = f y}) = emeasure M {x ∈ space
M. f x = f y}
    proof (rule emeasure_eq_ennreal_measure[symmetric])
      have emeasure M {x ∈ space M. f x = f y} ≤ emeasure M {x ∈ space M. f
x ≠ 0}
        using y by (intro emeasure_mono) auto
      with f show emeasure M {x ∈ space M. f x = f y} ≠ top
        by (auto simp: simple_bochner_integrable.simps top_unique)
    qed
    moreover have {x ∈ space M. f x = f y} = (λx. ennreal (f x)) −‘ {ennreal (f
y)} ∩ space M
      using f by auto
    ultimately have ennreal (measure M {x ∈ space M. f x = f y}) =
        emeasure M ((λx. ennreal (f x)) −‘ {ennreal (f y)} ∩ space M) by simp }
  with f have simple_bochner_integral M f = (∫ˢx. f x ∂M)
    unfolding simple_integral_def
   by (subst simple_bochner_integral_partition[OF f(1), where g=λx. ennreal (f
x) and v=enn2real])
     (auto intro: f simple_function_compose1 elim: simple_bochner_integrable.cases
           intro!: sum.cong ennreal_cong_mult
           simp: ac_simps ennreal_mult
           simp flip: sum_ennreal)
  also have ... = (∫⁺x. f x ∂M)
    using f
      by (metis nn_integral_eq_simple_integral simple_bochner_integrable.cases
```

simple_function_compose1)
finally show *?thesis* **.**
qed

lemma *simple_bochner_integral_bounded*:
fixes *f* :: *'a* ⇒ *'b*::{*real_normed_vector*, *second_countable_topology*}
assumes *f*[*measurable*]: *f* ∈ *borel_measurable M*
assumes *s*: *simple_bochner_integrable M s* **and** *t*: *simple_bochner_integrable M t*
shows *ennreal* (*norm* (*simple_bochner_integral M s* − *simple_bochner_integral M t*)) ≤
(∫⁺ *x*. *norm* (*f x* − *s x*) ∂*M*) + (∫⁺ *x*. *norm* (*f x* − *t x*) ∂*M*)
(**is** *ennreal* (*norm* (*?s* − *?t*)) ≤ *?S* + *?T*)
proof −
have [*measurable*]: *s* ∈ *borel_measurable M t* ∈ *borel_measurable M*
using *s t* **by** (*auto intro*: *borel_measurable_simple_function elim*: *simple_bochner_integrable.cases*)

have *ennreal* (*norm* (*?s* − *?t*)) = *norm* (*simple_bochner_integral M* (λ*x*. *s x* − *t x*))
using *s t* **by** (*subst simple_bochner_integral_diff*) *auto*
also have ... ≤ *simple_bochner_integral M* (λ*x*. *norm* (*s x* − *t x*))
using *simple_bochner_integrable_compose2*[*of* (−) *M s t*] *s t*
by (*auto intro*!: *simple_bochner_integral_norm_bound*)
also have ... = (∫⁺*x*. *norm* (*s x* − *t x*) ∂*M*)
using *simple_bochner_integrable_compose2*[*of* λ*x y*. *norm* (*x* − *y*) *M s t*] *s t*
by (*auto intro*!: *simple_bochner_integral_eq_nn_integral*)
also have ... ≤ (∫⁺*x*. *ennreal* (*norm* (*f x* − *s x*)) + *ennreal* (*norm* (*f x* − *t x*)) ∂*M*)
proof −
have ⋀*x*. *x* ∈ *space M* ⟹
norm (*s x* − *t x*) ≤ *norm* (*f x* − *s x*) + *norm* (*f x* − *t x*)
by (*metis dual_order.refl norm_diff_triangle_le norm_minus_commute*)
then show *?thesis*
by (*metis* (*mono_tags*, *lifting*) *ennreal_leI ennreal_plus nn_integral_mono norm_ge_zero*)
qed
also have ... = *?S* + *?T*
by (*rule nn_integral_add*) *auto*
finally show *?thesis* **.**
qed

inductive *has_bochner_integral* :: *'a measure* ⇒ (*'a* ⇒ *'b*) ⇒ *'b*::{*real_normed_vector*, *second_countable_topology*} ⇒ *bool*
for *M f x* **where**
f ∈ *borel_measurable M* ⟹
(⋀*i*. *simple_bochner_integrable M* (*s i*)) ⟹
(λ*i*. ∫⁺*x*. *norm* (*f x* − *s i x*) ∂*M*) ⟶ *0* ⟹
(λ*i*. *simple_bochner_integral M* (*s i*)) ⟶ *x* ⟹
has_bochner_integral M f x

lemma *has_bochner_integral_cong*:
assumes $M = N$ $\bigwedge x.\ x \in space\ N \Longrightarrow f\ x = g\ x\ x = y$
shows *has_bochner_integral M f x* $\longleftrightarrow$ *has_bochner_integral N g y*
unfolding *has_bochner_integral.simps assms*(*1*,*3*)
using *assms*(*2*) **by** (*simp cong*: *measurable_cong_simp nn_integral_cong_simp*)

lemma *has_bochner_integral_cong_AE*:
f $\in$ *borel_measurable M* $\Longrightarrow$ *g* $\in$ *borel_measurable M* $\Longrightarrow$ (*AE x in M. f x = g x*) $\Longrightarrow$
has_bochner_integral M f x $\longleftrightarrow$ *has_bochner_integral M g x*
unfolding *has_bochner_integral.simps*
by (*intro arg_cong*[**where** *f*=*Ex*] *ext conj_cong rev_conj_cong refl arg_cong*[**where** *f*=$\lambda x.\ x \longrightarrow 0$]
nn_integral_cong_AE)
auto

lemma *borel_measurable_has_bochner_integral*:
has_bochner_integral M f x $\Longrightarrow$ *f* $\in$ *borel_measurable M*
by (*rule has_bochner_integral.cases*)

lemma *borel_measurable_has_bochner_integral'*[*measurable_dest*]:
has_bochner_integral M f x $\Longrightarrow$ *g* $\in$ *measurable N M* $\Longrightarrow$ $(\lambda x.\ f\ (g\ x))$ $\in$ *borel_measurable N*
using *borel_measurable_has_bochner_integral*[*measurable*] **by** *measurable*

lemma *has_bochner_integral_simple_bochner_integrable*:
simple_bochner_integrable M f $\Longrightarrow$ *has_bochner_integral M f* (*simple_bochner_integral M f*)
by (*rule has_bochner_integral.intros*[**where** *s*=λ_. *f*])
(*auto intro*: *borel_measurable_simple_function*
elim: *simple_bochner_integrable.cases*
simp flip: *zero_ennreal_def*)

lemma *has_bochner_integral_real_indicator*:
assumes [*measurable*]: *A* $\in$ *sets M* **and** *A*: *emeasure M A* $< \infty$
shows *has_bochner_integral M* (*indicator A*) (*measure M A*)
proof −
have *sbi*: *simple_bochner_integrable M* (*indicator A*::*'a* $\Rightarrow$ *real*)
proof
have $\{y \in space\ M.\ (indicator\ A\ y{::}real) \neq 0\} = A$
using *sets.sets_into_space*[*OF* ‹*A*∈*sets M*›] **by** (*auto split*: *split_indicator*)
then show *emeasure M* $\{y \in space\ M.\ (indicator\ A\ y{::}real) \neq 0\} \neq \infty$
using *A* **by** *auto*
qed (*rule simple_function_indicator assms*)+
moreover have *simple_bochner_integral M* (*indicator A*) = *measure M A*
using *simple_bochner_integral_eq_nn_integral*[*OF sbi*] *A*
by (*simp add*: *ennreal_indicator emeasure_eq_ennreal_measure*)
ultimately show *?thesis*

by (*metis has_bochner_integral_simple_bochner_integrable*)
qed

lemma *has_bochner_integral_add*[*intro*]:
has_bochner_integral M f x ⟹ *has_bochner_integral M g y* ⟹
has_bochner_integral M (λ*x*. *f x* + *g x*) (*x* + *y*)
proof (*safe intro*!: *has_bochner_integral.intros elim*!: *has_bochner_integral.cases*)
fix *sf sg*
assume *f_sf*: (λ*i*. ∫⁺ *x*. *norm* (*f x* − *sf i x*) ∂*M*) ⟶ 0
assume *g_sg*: (λ*i*. ∫⁺ *x*. *norm* (*g x* − *sg i x*) ∂*M*) ⟶ 0

assume *sf*: ∀ *i*. *simple_bochner_integrable M* (*sf i*)
and *sg*: ∀ *i*. *simple_bochner_integrable M* (*sg i*)
then have [*measurable*]: ⋀*i*. *sf i* ∈ *borel_measurable M* ⋀*i*. *sg i* ∈ *borel_measurable M*
by (*auto intro*: *borel_measurable_simple_function elim*: *simple_bochner_integrable.cases*)
assume [*measurable*]: *f* ∈ *borel_measurable M g* ∈ *borel_measurable M*

show ⋀*i*. *simple_bochner_integrable M* (λ*x*. *sf i x* + *sg i x*)
using *sf sg* **by** (*simp add*: *simple_bochner_integrable_compose2*)

show (λ*i*. ∫⁺ *x*. (*norm* (*f x* + *g x* − (*sf i x* + *sg i x*))) ∂*M*) ⟶ 0
(**is** *?f* ⟶ 0)
proof (*rule tendsto_sandwich*)
show *eventually* (λ*n*. 0 ≤ *?f n*) *sequentially* (λ_. 0) ⟶ 0
by *auto*
show *eventually* (λ*i*. *?f i* ≤ (∫⁺ *x*. (*norm* (*f x* − *sf i x*)) ∂*M*) + ∫⁺ *x*. (*norm* (*g x* − *sg i x*)) ∂*M*) *sequentially*
(**is** *eventually* (λ*i*. *?f i* ≤ *?g i*) *sequentially*)
proof (*intro always_eventually allI*)
fix *i* **have** *?f i* ≤ (∫⁺ *x*. (*norm* (*f x* − *sf i x*)) + *ennreal* (*norm* (*g x* − *sg i x*)) ∂*M*)
by (*auto intro*!: *nn_integral_mono norm_diff_triangle_ineq*
simp flip: *ennreal_plus*)
also have ... = *?g i*
by (*intro nn_integral_add*) *auto*
finally show *?f i* ≤ *?g i* **.**
qed
show *?g* ⟶ 0
using *tendsto_add*[*OF f_sf g_sg*] **by** *simp*
qed
qed (*auto simp*: *simple_bochner_integral_add tendsto_add*)

lemma *has_bochner_integral_bounded_linear*:
assumes *bounded_linear T*
shows *has_bochner_integral M f x* ⟹ *has_bochner_integral M* (λ*x*. *T* (*f x*)) (*T x*)
proof (*safe intro*!: *has_bochner_integral.intros elim*!: *has_bochner_integral.cases*)
interpret *T*: *bounded_linear T* **by** *fact*

```
  have [measurable]: T ∈ borel_measurable borel
   by (intro borel_measurable_continuous_onI T.continuous_on continuous_on_id)
  assume [measurable]: f ∈ borel_measurable M
  then show (λx. T (f x)) ∈ borel_measurable M
    by auto

  fix s assume f_s: (λi. ∫⁺ x. norm (f x − s i x) ∂M) ⟶ 0
  assume s: ∀ i. simple_bochner_integrable M (s i)
  then show ⋀i. simple_bochner_integrable M (λx. T (s i x))
    by (auto intro: simple_bochner_integrable_compose2 T.zero)

  have [measurable]: ⋀i. s i ∈ borel_measurable M
   using s by (auto intro: borel_measurable_simple_function elim: simple_bochner_integrable.cases)

  obtain K where K: K > 0 ⋀x i. norm (T (f x) − T (s i x)) ≤ norm (f x − s
i x) * K
    using T.pos_bounded by (auto simp: T.diff[symmetric])

  show (λi. ∫⁺ x. norm (T (f x) − T (s i x)) ∂M) ⟶ 0
    (is ?f ⟶ 0)
  proof (rule tendsto_sandwich)
    show eventually (λn. 0 ≤ ?f n) sequentially (λ_. 0) ⟶ 0
      by auto

    show eventually (λi. ?f i ≤ K * (∫⁺ x. norm (f x − s i x) ∂M)) sequentially
      (is eventually (λi. ?f i ≤ ?g i) sequentially)
    proof (intro always_eventually allI)
      fix i have ?f i ≤ (∫⁺ x. ennreal K * norm (f x − s i x) ∂M)
      using K by (intro nn_integral_mono) (auto simp: ac_simps ennreal_mult[symmetric])
      also have ... = ?g i
        using K by (intro nn_integral_cmult) auto
      finally show ?f i ≤ ?g i .
    qed
    show ?g ⟶ 0
      using ennreal_tendsto_cmult[OF _ f_s] by simp
  qed

  assume (λi. simple_bochner_integral M (s i)) ⟶ x
  with s show (λi. simple_bochner_integral M (λx. T (s i x))) ⟶ T x
   by (auto intro!: T.tendsto simp: simple_bochner_integral_linear T.linear_axioms)
qed

lemma has_bochner_integral_zero[intro]: has_bochner_integral M (λx. 0) 0
  by (auto intro!: has_bochner_integral.intros[where s=λ_ _. 0]
          simp: zero_ennreal_def[symmetric] simple_bochner_integrable.simps
               simple_bochner_integral_def image_constant_conv)

lemma has_bochner_integral_scaleR_left[intro]:
  (c ≠ 0 ⟹ has_bochner_integral M f x) ⟹ has_bochner_integral M (λx. f x
```

$*_R$ c) (x $*_R$ c)
by (*cases* $c = 0$) (*auto simp*: *has_bochner_integral_bounded_linear*[*OF bounded_linear_scaleR_left*])

lemma *has_bochner_integral_scaleR_right*[*intro*]:
($c \neq 0 \Longrightarrow$ *has_bochner_integral* M f x) $\Longrightarrow$ *has_bochner_integral* M ($\lambda x.$ c $*_R$
f x) (c $*_R$ x)
by (*cases* $c = 0$) (*auto simp*: *has_bochner_integral_bounded_linear*[*OF bounded_linear_scaleR_right*])

lemma *has_bochner_integral_mult_left*[*intro*]:
fixes c :: _::{*real_normed_algebra*,*second_countable_topology*}
shows ($c \neq 0 \Longrightarrow$ *has_bochner_integral* M f x) $\Longrightarrow$ *has_bochner_integral* M
($\lambda x.$ f x $*$ c) (x $*$ c)
by (*cases* $c = 0$) (*auto simp*: *has_bochner_integral_bounded_linear*[*OF bounded_linear_mult_left*])

lemma *has_bochner_integral_mult_right*[*intro*]:
fixes c :: _::{*real_normed_algebra*,*second_countable_topology*}
shows ($c \neq 0 \Longrightarrow$ *has_bochner_integral* M f x) $\Longrightarrow$ *has_bochner_integral* M
($\lambda x.$ c $*$ f x) (c $*$ x)
by (*cases* $c = 0$) (*auto simp*: *has_bochner_integral_bounded_linear*[*OF bounded_linear_mult_right*])

lemmas *has_bochner_integral_divide* =
has_bochner_integral_bounded_linear[*OF bounded_linear_divide*]

lemma *has_bochner_integral_divide_zero*[*intro*]:
fixes c :: _::{*real_normed_field*, *field*, *second_countable_topology*}
shows ($c \neq 0 \Longrightarrow$ *has_bochner_integral* M f x) $\Longrightarrow$ *has_bochner_integral* M
($\lambda x.$ f x / c) (x / c)
using *has_bochner_integral_divide* **by** (*cases* $c = 0$) *auto*

lemma *has_bochner_integral_inner_left*[*intro*]:
($c \neq 0 \Longrightarrow$ *has_bochner_integral* M f x) $\Longrightarrow$ *has_bochner_integral* M ($\lambda x.$ f x $\cdot$
c) (x $\cdot$ c)
by (*cases* $c = 0$) (*auto simp*: *has_bochner_integral_bounded_linear*[*OF bounded_linear_inner_left*])

lemma *has_bochner_integral_inner_right*[*intro*]:
($c \neq 0 \Longrightarrow$ *has_bochner_integral* M f x) $\Longrightarrow$ *has_bochner_integral* M ($\lambda x.$ c $\cdot$ f
x) (c $\cdot$ x)
by (*cases* $c = 0$) (*auto simp*: *has_bochner_integral_bounded_linear*[*OF bounded_linear_inner_right*])

lemmas *has_bochner_integral_minus* =
has_bochner_integral_bounded_linear[*OF bounded_linear_minus*[*OF bounded_linear_ident*]]
lemmas *has_bochner_integral_Re* =
has_bochner_integral_bounded_linear[*OF bounded_linear_Re*]
lemmas *has_bochner_integral_Im* =
has_bochner_integral_bounded_linear[*OF bounded_linear_Im*]
lemmas *has_bochner_integral_cnj* =
has_bochner_integral_bounded_linear[*OF bounded_linear_cnj*]
lemmas *has_bochner_integral_of_real* =
has_bochner_integral_bounded_linear[*OF bounded_linear_of_real*]

lemmas *has_bochner_integral_fst* =
has_bochner_integral_bounded_linear[*OF bounded_linear_fst*]
lemmas *has_bochner_integral_snd* =
has_bochner_integral_bounded_linear[*OF bounded_linear_snd*]

lemma *has_bochner_integral_indicator*:
A ∈ *sets M* ⟹ *emeasure M A* < ∞ ⟹
has_bochner_integral M (λ*x*. *indicator A x* ∗$_R$ *c*) (*measure M A* ∗$_R$ *c*)
by (*intro has_bochner_integral_scaleR_left has_bochner_integral_real_indicator*)

lemma *has_bochner_integral_diff*:
has_bochner_integral M f x ⟹ *has_bochner_integral M g y* ⟹
has_bochner_integral M (λ*x*. *f x* − *g x*) (*x* − *y*)
unfolding *diff_conv_add_uminus*
by (*intro has_bochner_integral_add has_bochner_integral_minus*)

lemma *has_bochner_integral_sum*:
(⋀*i*. *i* ∈ *I* ⟹ *has_bochner_integral M* (*f i*) (*x i*)) ⟹
has_bochner_integral M (λ*x*. ∑ *i*∈*I*. *f i x*) (∑ *i*∈*I*. *x i*)
by (*induct I rule*: *infinite_finite_induct*) *auto*

proposition *has_bochner_integral_implies_finite_norm*:
has_bochner_integral M f x ⟹ (∫$^+$*x*. *norm* (*f x*) ∂*M*) < ∞
proof (*elim has_bochner_integral.cases*)
fix *s v*
assume [*measurable*]: *f* ∈ *borel_measurable M* **and** *s*: ⋀*i*. *simple_bochner_integrable M* (*s i*) **and**
lim_0: (λ*i*. ∫$^+$ *x*. *ennreal* (*norm* (*f x* − *s i x*)) ∂*M*) ⟶ *0*
from *order_tendstoD*[*OF lim_0*, *of* ∞]
obtain *i* **where** *f_s_fin*: (∫$^+$ *x*. *ennreal* (*norm* (*f x* − *s i x*)) ∂*M*) < ∞
by (*auto simp*: *eventually_sequentially*)

have [*measurable*]: ⋀*i*. *s i* ∈ *borel_measurable M*
using *s* **by** (*auto intro*: *borel_measurable_simple_function elim*: *simple_bochner_integrable.cases*)

define *m* **where** *m* = (*if space M* = {} *then 0 else Max* ((λ*x*. *norm* (*s i x*)) ‘*space M*))
have *finite* (*s i* ‘ *space M*)
using *s* **by** (*auto simp*: *simple_function_def simple_bochner_integrable.simps*)
then have *finite* (*norm* ‘ *s i* ‘ *space M*)
by (*rule finite_imageI*)
then have ⋀*x*. *x* ∈ *space M* ⟹ *norm* (*s i x*) ≤ *m* *0* ≤ *m*
by (*auto simp*: *m_def image_comp comp_def Max_ge_iff*)
then have (∫$^+$*x*. *norm* (*s i x*) ∂*M*) ≤ (∫$^+$*x*. *ennreal m* ∗ *indicator* {*x*∈*space M*. *s i x* ≠ *0*} *x* ∂*M*)
by (*auto split*: *split_indicator intro*!: *Max_ge nn_integral_mono simp*:)
also have ... < ∞
using *s* **by** (*subst nn_integral_cmult_indicator*) (*auto simp*: ‹*0* ≤ *m*› *simple_bochner_integrable.simps ennreal_mult_less_top less_top*)

```
  finally have s_fin: (∫⁺x. norm (s i x) ∂M) < ∞ .

  have (∫⁺ x. norm (f x) ∂M) ≤ (∫⁺ x. ennreal (norm (f x − s i x)) + ennreal
(norm (s i x)) ∂M)
    by (auto intro!: nn_integral_mono simp flip: ennreal_plus)
       (metis add.commute norm_triangle_sub)
  also have ... = (∫⁺x. norm (f x − s i x) ∂M) + (∫⁺x. norm (s i x) ∂M)
    by (rule nn_integral_add) auto
  also have ... < ∞
    using s_fin f_s_fin by auto
  finally show (∫⁺ x. ennreal (norm (f x)) ∂M) < ∞ .
qed

proposition has_bochner_integral_norm_bound:
  assumes i: has_bochner_integral M f x
  shows norm x ≤ (∫⁺x. norm (f x) ∂M)
using assms proof
  fix s assume
    x: (λi. simple_bochner_integral M (s i)) ⟶ x (is ?s ⟶ x) and
    s[simp]: ⋀i. simple_bochner_integrable M (s i) and
    lim: (λi. ∫⁺ x. ennreal (norm (f x − s i x)) ∂M) ⟶ 0 and
    f[measurable]: f ∈ borel_measurable M

  have [measurable]: ⋀i. s i ∈ borel_measurable M
   using s by (auto simp: simple_bochner_integrable.simps intro: borel_measurable_simple_function)

  show norm x ≤ (∫⁺x. norm (f x) ∂M)
  proof (rule LIMSEQ_le)
    show (λi. ennreal (norm (?s i))) ⟶ norm x
      using x by (auto simp: tendsto_ennreal_iff intro: tendsto_intros)
    show ∃N. ∀n≥N. norm (?s n) ≤ (∫⁺x. norm (f x − s n x) ∂M) + (∫⁺x.
norm (f x) ∂M)
      (is ∃N. ∀n≥N. _ ≤ ?t n)
    proof (intro exI allI impI)
      fix n
      have ennreal (norm (?s n)) ≤ simple_bochner_integral M (λx. norm (s n x))
        by (auto intro!: simple_bochner_integral_norm_bound)
      also have ... = (∫⁺x. norm (s n x) ∂M)
        by (intro simple_bochner_integral_eq_nn_integral)
           (auto intro: s simple_bochner_integrable_compose2)
      also have ... ≤ (∫⁺x. ennreal (norm (f x − s n x)) + norm (f x) ∂M)
        by (auto intro!: nn_integral_mono simp flip: ennreal_plus)
           (metis add.commute norm_minus_commute norm_triangle_sub)
      also have ... = ?t n
        by (rule nn_integral_add) auto
      finally show norm (?s n) ≤ ?t n .
    qed
    have ?t ⟶ 0 + (∫⁺ x. ennreal (norm (f x)) ∂M)
      using has_bochner_integral_implies_finite_norm[OF i]
```

```
    by (intro tendsto_add tendsto_const lim)
   then show ?t ⟶ ∫⁺ x. ennreal (norm (f x)) ∂M
    by simp
  qed
qed

lemma has_bochner_integral_eq:
  has_bochner_integral M f x ⟹ has_bochner_integral M f y ⟹ x = y
proof (elim has_bochner_integral.cases)
  assume f[measurable]: f ∈ borel_measurable M

  fix s t
  assume (λi. ∫⁺ x. norm (f x − s i x) ∂M) ⟶ 0 (is ?S ⟶ 0)
  assume (λi. ∫⁺ x. norm (f x − t i x) ∂M) ⟶ 0 (is ?T ⟶ 0)
  assume s: ⋀i. simple_bochner_integrable M (s i)
  assume t: ⋀i. simple_bochner_integrable M (t i)

  have [measurable]: ⋀i. s i ∈ borel_measurable M ⋀i. t i ∈ borel_measurable M
   using s t by (auto intro: borel_measurable_simple_function elim: simple_bochner_integrable.cases)

  let ?s = λi. simple_bochner_integral M (s i)
  let ?t = λi. simple_bochner_integral M (t i)
  assume ?s ⟶ x ?t ⟶ y
  then have (λi. norm (?s i − ?t i)) ⟶ norm (x − y)
    by (intro tendsto_intros)
  moreover
  have (λi. ennreal (norm (?s i − ?t i))) ⟶ ennreal 0
  proof (rule tendsto_sandwich)
    show eventually (λi. 0 ≤ ennreal (norm (?s i − ?t i))) sequentially (λ_. 0)
⟶ ennreal 0
      by auto

    show eventually (λi. norm (?s i − ?t i) ≤ ?S i + ?T i) sequentially
      by (intro always_eventually allI simple_bochner_integral_bounded s t f)
    show (λi. ?S i + ?T i) ⟶ ennreal 0
      using tendsto_add[OF ‹?S ⟶ 0› ‹?T ⟶ 0›] by simp
  qed
  then have (λi. norm (?s i − ?t i)) ⟶ 0
    by (simp flip: ennreal_0)
  ultimately have norm (x − y) = 0
    by (rule LIMSEQ_unique)
  then show x = y by simp
qed

lemma has_bochner_integralI_AE:
  assumes f: has_bochner_integral M f x
    and g: g ∈ borel_measurable M
    and ae: AE x in M. f x = g x
  shows has_bochner_integral M g x
```

using *f*
proof (*safe intro*!: *has_bochner_integral.intros elim*!: *has_bochner_integral.cases*)
fix *s* **assume** (λ*i*. ∫⁺ *x*. *ennreal* (*norm* (*f x* − *s i x*)) ∂*M*) ⟶ *0*
also have (λ*i*. ∫⁺ *x*. *ennreal* (*norm* (*f x* − *s i x*)) ∂*M*) = (λ*i*. ∫⁺ *x*. *ennreal* (*norm* (*g x* − *s i x*)) ∂*M*)
using *ae*
by (*intro ext nn_integral_cong_AE*, *eventually_elim*) *simp*
finally show (λ*i*. ∫⁺ *x*. *ennreal* (*norm* (*g x* − *s i x*)) ∂*M*) ⟶ *0* .
qed (*auto intro*: *g*)

lemma *has_bochner_integral_eq_AE*:
assumes *has_bochner_integral M f x* **and** *has_bochner_integral M g y*
and *AE x in M*. *f x* = *g x*
shows *x* = *y*
by (*meson assms has_bochner_integral.simps has_bochner_integral_cong_AE has_bochner_integral_eq*)

lemma *simple_bochner_integrable_restrict_space*:
fixes *f* :: _ ⇒ 'b::*real_normed_vector*
assumes Ω: Ω ∩ *space M* ∈ *sets M*
shows *simple_bochner_integrable* (*restrict_space M* Ω) *f* ⟷
simple_bochner_integrable M (λ*x*. *indicator* Ω *x* ∗*R f x*)
by (*simp add*: *simple_bochner_integrable.simps space_restrict_space*
simple_function_restrict_space[*OF* Ω] *emeasure_restrict_space*[*OF* Ω] *Collect_restrict*
indicator_eq_0_iff conj_left_commute)

lemma *simple_bochner_integral_restrict_space*:
fixes *f* :: _ ⇒ 'b::*real_normed_vector*
assumes Ω: Ω ∩ *space M* ∈ *sets M*
assumes *f*: *simple_bochner_integrable* (*restrict_space M* Ω) *f*
shows *simple_bochner_integral* (*restrict_space M* Ω) *f* =
simple_bochner_integral M (λ*x*. *indicator* Ω *x* ∗*R f x*)
proof −
have *finite* ((λ*x*. *indicator* Ω *x* ∗*R f x*)‘*space M*)
using *f simple_bochner_integrable_restrict_space*[*OF* Ω, *of f*]
by (*simp add*: *simple_bochner_integrable.simps simple_function_def*)
then show *?thesis*
by (*auto simp*: *space_restrict_space measure_restrict_space*[*OF* Ω(*1*)] *le_infI2*
simple_bochner_integral_def Collect_restrict
split: *split_indicator split_indicator_asm*
intro!: *sum.mono_neutral_cong_left arg_cong2*[**where** *f*=*measure*])
qed

context
notes [[*inductive_internals*]]
begin

inductive *integrable* **for** *M f* **where**

has_bochner_integral M f x ⟹ integrable M f

end

definition *lebesgue_integral* (*integral*^{L}) **where**
integral^L *M f* = (*if* ∃ *x*. *has_bochner_integral M f x then THE x*. *has_bochner_integral M f x else 0*)

syntax
_lebesgue_integral :: *pttrn* ⇒ *real* ⇒ *'a measure* ⇒ *real* (∫((*2* _./ _)/ ∂_) [*60*,*61*] *110*)

translations
∫ *x*. *f* ∂*M* == *CONST lebesgue_integral M* (λ*x*. *f*)

syntax
_ascii_lebesgue_integral :: *pttrn* ⇒ *'a measure* ⇒ *real* ⇒ *real* ((*3LINT* (*1*_)/|(_)./ _) [*0*,*110*,*60*] *60*)

translations
LINT x|*M*. *f* == *CONST lebesgue_integral M* (λ*x*. *f*)

lemma *has_bochner_integral_integral_eq*: *has_bochner_integral M f x ⟹ integral*^{L} *M f* = *x*
by (*metis the_equality has_bochner_integral_eq lebesgue_integral_def*)

lemma *has_bochner_integral_integrable*:
integrable M f ⟹ has_bochner_integral M f (*integral*^{L} *M f*)
by (*auto simp*: *has_bochner_integral_integral_eq integrable.simps*)

lemma *has_bochner_integral_iff*:
has_bochner_integral M f x ⟷ *integrable M f* ∧ *integral*^{L} *M f* = *x*
by (*metis has_bochner_integral_integrable has_bochner_integral_integral_eq integrable.intros*)

lemma *simple_bochner_integrable_eq_integral*:
simple_bochner_integrable M f ⟹ simple_bochner_integral M f = *integral*^{L} *M f*
using *has_bochner_integral_simple_bochner_integrable*[*of M f*]
by (*simp add*: *has_bochner_integral_integral_eq*)

lemma *not_integrable_integral_eq*: ¬ *integrable M f ⟹ integral*^{L} *M f* = *0*
unfolding *integrable.simps lebesgue_integral_def* **by** (*auto intro*!: *arg_cong*[**where** *f*=*The*])

lemma *integral_eq_cases*:
integrable M f ⟷ *integrable N g* ⟹
(*integrable M f ⟹ integrable N g ⟹ integral*^{L} *M f* = *integral*^{L} *N g*) ⟹
integral^L *M f* = *integral*^{L} *N g*

by (*metis not_integrable_integral_eq*)

lemma *borel_measurable_integrable*[*measurable_dest*]: *integrable M f* $\Longrightarrow$ *f* $\in$ *borel_measurable M*
by (*auto elim*: *integrable.cases has_bochner_integral.cases*)

lemma *borel_measurable_integrable'*[*measurable_dest*]:
integrable M f $\Longrightarrow$ *g* $\in$ *measurable N M* $\Longrightarrow$ $(\lambda x.\ f\ (g\ x))$ $\in$ *borel_measurable N*
using *borel_measurable_integrable*[*measurable*] **by** *measurable*

lemma *integrable_cong*:
M = *N* $\Longrightarrow$ $(\bigwedge x.\ x \in$ *space N* $\Longrightarrow$ *f x* = *g x*$)$ $\Longrightarrow$ *integrable M f* $\longleftrightarrow$ *integrable N g*
by (*simp cong*: *has_bochner_integral_cong add*: *integrable.simps*)

lemma *integrable_cong_AE*:
f $\in$ *borel_measurable M* $\Longrightarrow$ *g* $\in$ *borel_measurable M* $\Longrightarrow$ *AE x in M*. *f x* = *g x* $\Longrightarrow$
integrable M f $\longleftrightarrow$ *integrable M g*
unfolding *integrable.simps*
by (*intro has_bochner_integral_cong_AE arg_cong*[**where** *f*=*Ex*] *ext*)

lemma *integrable_cong_AE_imp*:
integrable M g $\Longrightarrow$ *f* $\in$ *borel_measurable M* $\Longrightarrow$ (*AE x in M*. *g x* = *f x*) $\Longrightarrow$ *integrable M f*
using *integrable_cong_AE*[*of f M g*] **by** (*auto simp*: *eq_commute*)

lemma *integral_cong*:
M = *N* $\Longrightarrow$ $(\bigwedge x.\ x \in$ *space N* $\Longrightarrow$ *f x* = *g x*$)$ $\Longrightarrow$ $integral^L$ *M f* = $integral^L$ *N g*
by (*simp cong*: *has_bochner_integral_cong cong del*: *if_weak_cong add*: *lebesgue_integral_def*)

lemma *integral_cong_AE*:
f $\in$ *borel_measurable M* $\Longrightarrow$ *g* $\in$ *borel_measurable M* $\Longrightarrow$ *AE x in M*. *f x* = *g x* $\Longrightarrow$
$integral^L$ *M f* = $integral^L$ *M g*
unfolding *lebesgue_integral_def*
by (*rule arg_cong*[**where** *x*=*has_bochner_integral M f*]) (*intro has_bochner_integral_cong_AE ext*)

lemma *integrable_add*[*simp*, *intro*]: *integrable M f* $\Longrightarrow$ *integrable M g* $\Longrightarrow$ *integrable M* $(\lambda x.\ f\ x + g\ x)$
by (*auto simp*: *integrable.simps*)

lemma *integrable_zero*[*simp*, *intro*]: *integrable M* $(\lambda x.\ 0)$
by (*metis has_bochner_integral_zero integrable.simps*)

lemma *integrable_sum*[*simp*, *intro*]: $(\bigwedge i.\ i \in I \Longrightarrow$ *integrable M* $(f\ i)) \Longrightarrow$ *integrable M* $(\lambda x.\ \sum i{\in}I.\ f\ i\ x)$
by (*metis has_bochner_integral_sum integrable.simps*)

lemma *integrable_indicator*[*simp*, *intro*]: $A \in sets\ M \Longrightarrow emeasure\ M\ A < \infty \Longrightarrow$
integrable M ($\lambda x.$ *indicator A x* $*_R$ *c*)
by (*metis has_bochner_integral_indicator integrable.simps*)

lemma *integrable_real_indicator*[*simp*, *intro*]: $A \in sets\ M \Longrightarrow emeasure\ M\ A < \infty \Longrightarrow$
integrable M (*indicator A* :: $'a \Rightarrow real$)
by (*metis has_bochner_integral_real_indicator integrable.simps*)

lemma *integrable_diff*[*simp*, *intro*]: *integrable M f* $\Longrightarrow$ *integrable M g* $\Longrightarrow$ *integrable M* ($\lambda x.$ *f x* $-$ *g x*)
by (*auto simp*: *integrable.simps intro*: *has_bochner_integral_diff*)

lemma *integrable_bounded_linear*: *bounded_linear T* $\Longrightarrow$ *integrable M f* $\Longrightarrow$ *integrable M* ($\lambda x.$ *T* (*f x*))
by (*auto simp*: *integrable.simps intro*: *has_bochner_integral_bounded_linear*)

lemma *integrable_scaleR_left*[*simp*, *intro*]: (*c* $\neq$ *0* $\Longrightarrow$ *integrable M f*) $\Longrightarrow$ *integrable M* ($\lambda x.$ *f x* $*_R$ *c*)
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable_scaleR_right*[*simp*, *intro*]: (*c* $\neq$ *0* $\Longrightarrow$ *integrable M f*) $\Longrightarrow$ *integrable M* ($\lambda x.$ *c* $*_R$ *f x*)
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable_mult_left*[*simp*, *intro*]:
fixes *c* :: _::{*real_normed_algebra*,*second_countable_topology*}
shows (*c* $\neq$ *0* $\Longrightarrow$ *integrable M f*) $\Longrightarrow$ *integrable M* ($\lambda x.$ *f x* $*$ *c*)
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable_mult_right*[*simp*, *intro*]:
fixes *c* :: _::{*real_normed_algebra*,*second_countable_topology*}
shows (*c* $\neq$ *0* $\Longrightarrow$ *integrable M f*) $\Longrightarrow$ *integrable M* ($\lambda x.$ *c* $*$ *f x*)
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable_divide_zero*[*simp*, *intro*]:
fixes *c* :: _::{*real_normed_field*, *field*, *second_countable_topology*}
shows (*c* $\neq$ *0* $\Longrightarrow$ *integrable M f*) $\Longrightarrow$ *integrable M* ($\lambda x.$ *f x* / *c*)
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable_inner_left*[*simp*, *intro*]:
(*c* $\neq$ *0* $\Longrightarrow$ *integrable M f*) $\Longrightarrow$ *integrable M* ($\lambda x.$ *f x* $\cdot$ *c*)
unfolding *integrable.simps* **by** *fastforce*

lemma *integrable_inner_right*[*simp*, *intro*]:
(*c* $\neq$ *0* $\Longrightarrow$ *integrable M f*) $\Longrightarrow$ *integrable M* ($\lambda x.$ *c* $\cdot$ *f x*)
unfolding *integrable.simps* **by** *fastforce*

lemmas *integrable_minus*[*simp*, *intro*] =
integrable_bounded_linear[*OF bounded_linear_minus*[*OF bounded_linear_ident*]]
lemmas *integrable_divide*[*simp*, *intro*] =
integrable_bounded_linear[*OF bounded_linear_divide*]
lemmas *integrable_Re*[*simp*, *intro*] =
integrable_bounded_linear[*OF bounded_linear_Re*]
lemmas *integrable_Im*[*simp*, *intro*] =
integrable_bounded_linear[*OF bounded_linear_Im*]
lemmas *integrable_cnj*[*simp*, *intro*] =
integrable_bounded_linear[*OF bounded_linear_cnj*]
lemmas *integrable_of_real*[*simp*, *intro*] =
integrable_bounded_linear[*OF bounded_linear_of_real*]
lemmas *integrable_fst*[*simp*, *intro*] =
integrable_bounded_linear[*OF bounded_linear_fst*]
lemmas *integrable_snd*[*simp*, *intro*] =
integrable_bounded_linear[*OF bounded_linear_snd*]

lemma *integral_zero*[*simp*]: $integral^L\ M\ (\lambda x.\ 0) = 0$
by (*intro has_bochner_integral_integral_eq has_bochner_integral_zero*)

lemma *integral_add*[*simp*]: *integrable M f* $\Longrightarrow$ *integrable M g* $\Longrightarrow$
$integral^L\ M\ (\lambda x.\ f\ x + g\ x) = integral^L\ M\ f + integral^L\ M\ g$
by (*intro has_bochner_integral_integral_eq has_bochner_integral_add has_bochner_integral_integrab*

lemma *integral_diff*[*simp*]: *integrable M f* $\Longrightarrow$ *integrable M g* $\Longrightarrow$
$integral^L\ M\ (\lambda x.\ f\ x - g\ x) = integral^L\ M\ f - integral^L\ M\ g$
by (*intro has_bochner_integral_integral_eq has_bochner_integral_diff has_bochner_integral_integrabl*

lemma *integral_sum*: $(\bigwedge i.\ i \in I \Longrightarrow integrable\ M\ (f\ i)) \Longrightarrow$
$integral^L\ M\ (\lambda x.\ \sum i{\in}I.\ f\ i\ x) = (\sum i{\in}I.\ integral^L\ M\ (f\ i))$
by (*intro has_bochner_integral_integral_eq has_bochner_integral_sum has_bochner_integral_integrab*

lemma *integral_sum'*[*simp*]: $(\bigwedge i.\ i \in I =simp=> integrable\ M\ (f\ i)) \Longrightarrow$
$integral^L\ M\ (\lambda x.\ \sum i{\in}I.\ f\ i\ x) = (\sum i{\in}I.\ integral^L\ M\ (f\ i))$
unfolding *simp_implies_def* **by** (*rule integral_sum*)

lemma *integral_bounded_linear*: *bounded_linear T* $\Longrightarrow$ *integrable M f* $\Longrightarrow$
$integral^L\ M\ (\lambda x.\ T\ (f\ x)) = T\ (integral^L\ M\ f)$
by (*metis has_bochner_integral_bounded_linear has_bochner_integral_integrable*
has_bochner_integral_integral_eq)

lemma *integral_bounded_linear'*:
assumes *T*: *bounded_linear T* **and** *T'*: *bounded_linear T'*
assumes $*$: $\neg\ (\forall x.\ T\ x = 0) \Longrightarrow (\forall x.\ T'\ (T\ x) = x)$
shows $integral^L\ M\ (\lambda x.\ T\ (f\ x)) = T\ (integral^L\ M\ f)$
proof *cases*
assume $(\forall x.\ T\ x = 0)$ **then show** *?thesis*
by *simp*
next

assume $**$: $\neg$ $(\forall x.\ T\ x = 0)$
show *?thesis*
proof *cases*
assume *integrable M f* **with** *T* **show** *?thesis*
by (*rule integral_bounded_linear*)
next
assume *not*: $\neg$ *integrable M f*
moreover have $\neg$ *integrable M* $(\lambda x.\ T\ (f\ x))$
by (*metis* (*full_types*) $*$ $**$ T' *integrable_bounded_linear integrable_cong not*)
ultimately show *?thesis*
using *T* **by** (*simp add*: *not_integrable_integral_eq linear_simps*)
qed
qed

lemma *integral_scaleR_left*[*simp*]: $(c \neq 0 \Longrightarrow$ *integrable M f*$) \Longrightarrow (\int\ x.\ f\ x *_R$
$c\ \partial M) =$ *integral*^{L} *M f* $*_R$ *c*
by (*intro has_bochner_integral_integral_eq has_bochner_integral_integrable has_bochner_integral_scaleR_left*)

lemma *integral_scaleR_right*[*simp*]: $(\int\ x.\ c *_R f\ x\ \partial M) = c *_R$ *integral*^{L} *M f*
by (*rule integral_bounded_linear'*[*OF bounded_linear_scaleR_right bounded_linear_scaleR_right*[*of*
1 / c]]) *simp*

lemma *integral_mult_left*[*simp*]:
fixes *c* :: _::{*real_normed_algebra,second_countable_topology*}
shows $(c \neq 0 \Longrightarrow$ *integrable M f*$) \Longrightarrow (\int\ x.\ f\ x * c\ \partial M) =$ *integral*^{L} *M f* $* c$
by (*intro has_bochner_integral_integral_eq has_bochner_integral_integrable has_bochner_integral_mult_left*)

lemma *integral_mult_right*[*simp*]:
fixes *c* :: _::{*real_normed_algebra,second_countable_topology*}
shows $(c \neq 0 \Longrightarrow$ *integrable M f*$) \Longrightarrow (\int\ x.\ c * f\ x\ \partial M) = c *$ *integral*^{L} *M f*
by (*intro has_bochner_integral_integral_eq has_bochner_integral_integrable has_bochner_integral_mult_right*)

lemma *integral_mult_left_zero*[*simp*]:
fixes *c* :: _::{*real_normed_field,second_countable_topology*}
shows $(\int\ x.\ f\ x * c\ \partial M) =$ *integral*^{L} *M f* $* c$
by (*rule integral_bounded_linear'*[*OF bounded_linear_mult_left bounded_linear_mult_left*[*of*
1 / c]]) *simp*

lemma *integral_mult_right_zero*[*simp*]:
fixes *c* :: _::{*real_normed_field,second_countable_topology*}
shows $(\int\ x.\ c * f\ x\ \partial M) = c *$ *integral*^{L} *M f*
by (*rule integral_bounded_linear'*[*OF bounded_linear_mult_right bounded_linear_mult_right*[*of*
1 / c]]) *simp*

lemma *integral_inner_left*[*simp*]: $(c \neq 0 \Longrightarrow$ *integrable M f*$) \Longrightarrow (\int\ x.\ f\ x \cdot c$
$\partial M) =$ *integral*^{L} *M f* $\cdot\ c$
by (*intro has_bochner_integral_integral_eq has_bochner_integral_integrable has_bochner_integral_inner_left*)

lemma *integral_inner_right*[*simp*]: $(c \neq 0 \Longrightarrow$ *integrable M f*$) \Longrightarrow (\int\ x.\ c \cdot f\ x$

$\partial M) = c \cdot \mathit{integral}^L\ M\ f$
by (*intro has_bochner_integral_integral_eq has_bochner_integral_integrable has_bochner_integral_in*

lemma *integral_divide_zero*[*simp*]:
fixes $c :: \_::\{\mathit{real\_normed\_field}, \mathit{field}, \mathit{second\_countable\_topology}\}$
shows $\mathit{integral}^L\ M\ (\lambda x.\ f\ x\ /\ c) = \mathit{integral}^L\ M\ f\ /\ c$
by (*rule integral_bounded_linear'*[*OF bounded_linear_divide bounded_linear_mult_left*[*of c*]]) *simp*

lemma *integral_minus*[*simp*]: $\mathit{integral}^L\ M\ (\lambda x.\ -\ f\ x) = -\ \mathit{integral}^L\ M\ f$
by (*rule integral_bounded_linear'*[*OF bounded_linear_minus*[*OF bounded_linear_ident*] *bounded_linear_minus*[*OF bounded_linear_ident*]]) *simp*

lemma *integral_complex_of_real*[*simp*]: $\mathit{integral}^L\ M\ (\lambda x.\ \mathit{complex\_of\_real}\ (f\ x))$
$= \mathit{of\_real}\ (\mathit{integral}^L\ M\ f)$
by (*rule integral_bounded_linear'*[*OF bounded_linear_of_real bounded_linear_Re*]) *simp*

lemma *integral_cnj*[*simp*]: $\mathit{integral}^L\ M\ (\lambda x.\ \mathit{cnj}\ (f\ x)) = \mathit{cnj}\ (\mathit{integral}^L\ M\ f)$
by (*rule integral_bounded_linear'*[*OF bounded_linear_cnj bounded_linear_cnj*]) *simp*

lemmas *integral_divide*[*simp*] =
integral_bounded_linear[*OF bounded_linear_divide*]
lemmas *integral_Re*[*simp*] =
integral_bounded_linear[*OF bounded_linear_Re*]
lemmas *integral_Im*[*simp*] =
integral_bounded_linear[*OF bounded_linear_Im*]
lemmas *integral_of_real*[*simp*] =
integral_bounded_linear[*OF bounded_linear_of_real*]
lemmas *integral_fst*[*simp*] =
integral_bounded_linear[*OF bounded_linear_fst*]
lemmas *integral_snd*[*simp*] =
integral_bounded_linear[*OF bounded_linear_snd*]

lemma *integral_norm_bound_ennreal*:
$\mathit{integrable}\ M\ f \Longrightarrow \mathit{norm}\ (\mathit{integral}^L\ M\ f) \leq (\int^+ x.\ \mathit{norm}\ (f\ x)\ \partial M)$
by (*metis has_bochner_integral_integrable has_bochner_integral_norm_bound*)

lemma *integrableI_sequence*:
fixes $f :: 'a \Rightarrow 'b::\{\mathit{banach}, \mathit{second\_countable\_topology}\}$
assumes f[*measurable*]: $f \in \mathit{borel\_measurable}\ M$
assumes s: $\bigwedge i.\ \mathit{simple\_bochner\_integrable}\ M\ (s\ i)$
assumes *lim*: $(\lambda i.\ \int^+ x.\ \mathit{norm}\ (f\ x - s\ i\ x)\ \partial M) \longrightarrow 0$ (**is** $?S \longrightarrow 0$)
shows $\mathit{integrable}\ M\ f$
proof −
let $?s = \lambda n.\ \mathit{simple\_bochner\_integral}\ M\ (s\ n)$

have $\exists x.\ ?s \longrightarrow x$

```
    unfolding convergent_eq_Cauchy
  proof (rule metric_CauchyI)
    fix e :: real assume 0 < e
    then have 0 < ennreal (e / 2) by auto
    from order_tendstoD(2)[OF lim this]
    obtain M where M: ⋀n. M ≤ n ⟹ ?S n < e / 2
      by (auto simp: eventually_sequentially)
    show ∃M. ∀m≥M. ∀n≥M. dist (?s m) (?s n) < e
    proof (intro exI allI impI)
      fix m n assume m: M ≤ m and n: M ≤ n
      have ?S n ≠ ∞
        using M[OF n] by auto
      have norm (?s n − ?s m) ≤ ?S n + ?S m
        by (intro simple_bochner_integral_bounded s f)
      also have ... < ennreal (e / 2) + e / 2
        by (intro add_strict_mono M n m)
      also have ... = e using ‹0<e› by (simp flip: ennreal_plus)
      finally show dist (?s n) (?s m) < e
        using ‹0<e› by (simp add: dist_norm ennreal_less_iff)
    qed
  qed
  then obtain x where ?s ⟶ x ..
  show ?thesis
    by (rule, rule) fact+
qed

proposition nn_integral_dominated_convergence_norm:
  fixes u' :: _ ⇒ _::{real_normed_vector, second_countable_topology}
  assumes [measurable]:
      ⋀i. u i ∈ borel_measurable M u' ∈ borel_measurable M w ∈ borel_measurable
M
    and bound: ⋀j. AE x in M. norm (u j x) ≤ w x
    and w: (∫⁺x. w x ∂M) < ∞
    and u': AE x in M. (λi. u i x) ⟶ u' x
  shows (λi. (∫⁺x. norm (u' x − u i x) ∂M)) ⟶ 0
proof −
  have AE x in M. ∀j. norm (u j x) ≤ w x
    unfolding AE_all_countable by rule fact
  with u' have bnd: AE x in M. ∀j. norm (u' x − u j x) ≤ 2 * w x
  proof (eventually_elim, intro allI)
    fix i x assume (λi. u i x) ⟶ u' x ∀j. norm (u j x) ≤ w x ∀j. norm (u j
x) ≤ w x
    then have norm (u' x) ≤ w x norm (u i x) ≤ w x
      by (auto intro: LIMSEQ_le_const2 tendsto_norm)
    then have norm (u' x) + norm (u i x) ≤ 2 * w x
      by simp
    also have norm (u' x − u i x) ≤ norm (u' x) + norm (u i x)
      by (rule norm_triangle_ineq4)
    finally (xtrans) show norm (u' x − u i x) ≤ 2 * w x .
```

```
  qed
  have w_nonneg: AE x in M. 0 ≤ w x
    using bound[of 0] by (auto intro: order_trans[OF norm_ge_zero])

  have (λi. (∫⁺x. norm (u' x − u i x) ∂M)) ⟶ (∫⁺x. 0 ∂M)
  proof (rule nn_integral_dominated_convergence)
    show (∫⁺x. 2 * w x ∂M) < ∞
      by (rule nn_integral_mult_bounded_inf[OF _ w, of 2]) (insert w_nonneg,
auto simp: ennreal_mult )
    show AE x in M. (λi. ennreal (norm (u' x − u i x))) ⟶ 0
      using u'
    proof eventually_elim
      fix x assume (λi. u i x) ⟶ u' x
      from tendsto_diff[OF tendsto_const[of u' x] this]
      show (λi. ennreal (norm (u' x − u i x))) ⟶ 0
        by (simp add: tendsto_norm_zero_iff flip: ennreal_0)
    qed
  qed (use bnd w_nonneg in auto)
  then show ?thesis by simp
qed

proposition integrableI_bounded:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}
  assumes f[measurable]: f ∈ borel_measurable M and fin: (∫⁺x. norm (f x) ∂M)
< ∞
  shows integrable M f
proof −
  from borel_measurable_implies_sequence_metric[OF f, of 0] obtain s where
    s: ⋀i. simple_function M (s i) and
    pointwise: ⋀x. x ∈ space M ⟹ (λi. s i x) ⟶ f x and
    bound: ⋀i x. x ∈ space M ⟹ norm (s i x) ≤ 2 * norm (f x)
    by simp metis

  show ?thesis
  proof (rule integrableI_sequence)
    { fix i
      have (∫⁺x. norm (s i x) ∂M) ≤ (∫⁺x. ennreal (2 * norm (f x)) ∂M)
        by (intro nn_integral_mono) (simp add: bound)
      also have ... = 2 * (∫⁺x. ennreal (norm (f x)) ∂M)
        by (simp add: ennreal_mult nn_integral_cmult)
      also have ... < top
        using fin by (simp add: ennreal_mult_less_top)
      finally have (∫⁺x. norm (s i x) ∂M) < ∞
        by simp }
    note fin_s = this

    show ⋀i. simple_bochner_integrable M (s i)
      by (rule simple_bochner_integrableI_bounded) fact+
```

```
    show (λi. ∫⁺ x. ennreal (norm (f x − s i x)) ∂M) ⟶ 0
    proof (rule nn_integral_dominated_convergence_norm)
      show ⋀j. AE x in M. norm (s j x) ≤ 2 * norm (f x)
        using bound by auto
      show ⋀i. s i ∈ borel_measurable M (λx. 2 * norm (f x)) ∈ borel_measurable
M
        using s by (auto intro: borel_measurable_simple_function)
      show (∫⁺ x. ennreal (2 * norm (f x)) ∂M) < ∞
      using fin by (simp add: nn_integral_cmult ennreal_mult ennreal_mult_less_top)
      show AE x in M. (λi. s i x) ⟶ f x
        using pointwise by auto
    qed fact
  qed fact
qed

lemma integrableI_bounded_set:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}
  assumes [measurable]: A ∈ sets M f ∈ borel_measurable M
  assumes finite: emeasure M A < ∞
    and bnd: AE x in M. x ∈ A ⟶ norm (f x) ≤ B
    and null: AE x in M. x ∉ A ⟶ f x = 0
  shows integrable M f
proof (rule integrableI_bounded)
  { fix x :: 'b have norm x ≤ B ⟹ 0 ≤ B
      using norm_ge_zero[of x] by arith }
  with bnd null have (∫⁺ x. ennreal (norm (f x)) ∂M) ≤ (∫⁺ x. ennreal (max 0
B) * indicator A x ∂M)
    by (intro nn_integral_mono_AE) (auto split: split_indicator split_max)
  also have ... < ∞
   using finite by (subst nn_integral_cmult_indicator) (auto simp: ennreal_mult_less_top)
  finally show (∫⁺ x. ennreal (norm (f x)) ∂M) < ∞ .
qed simp

lemma integrableI_bounded_set_indicator:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}
  shows A ∈ sets M ⟹ f ∈ borel_measurable M ⟹
    emeasure M A < ∞ ⟹ (AE x in M. x ∈ A ⟶ norm (f x) ≤ B) ⟹
    integrable M (λx. indicator A x *ᵣ f x)
  by (rule integrableI_bounded_set[where A=A]) auto

lemma integrableI_nonneg:
  fixes f :: 'a ⇒ real
  assumes f ∈ borel_measurable M AE x in M. 0 ≤ f x (∫⁺x. f x ∂M) < ∞
  shows integrable M f
proof −
  have (∫⁺x. norm (f x) ∂M) = (∫⁺x. f x ∂M)
    using assms by (intro nn_integral_cong_AE) auto
  then show ?thesis
    using assms by (intro integrableI_bounded) auto
```

qed

lemma *integrable_iff_bounded*:
fixes $f :: 'a \Rightarrow 'b::\{banach, second\_countable\_topology\}$
shows *integrable M f* $\longleftrightarrow$ $f \in$ *borel_measurable M* $\wedge$ $(\int^+ x.\ norm\ (f\ x)\ \partial M) < \infty$
using *integrableI_bounded*[*of f M*] *has_bochner_integral_implies_finite_norm*[*of M f*]
unfolding *integrable.simps has_bochner_integral.simps*[*abs_def*] **by** *auto*

lemma (**in** *finite_measure*) *square_integrable_imp_integrable*:
fixes $f :: 'a \Rightarrow 'b :: \{second\_countable\_topology, banach, real\_normed\_div\_algebra\}$
assumes [*measurable*]: $f \in$ *borel_measurable M*
assumes *integrable M* $(\lambda x.\ f\ x\ \widehat{}\ 2)$
shows *integrable M f*
proof −
have *less_top*: *emeasure M* (*space M*) $<$ *top*
using *finite_emeasure_space* **by** (*meson top.not_eq_extremum*)
have *nn_integral M* $(\lambda x.\ norm\ (f\ x))\ \widehat{}\ 2 \leq$
nn_integral M $(\lambda x.\ norm\ (f\ x)\ \widehat{}\ 2) *$ *emeasure M* (*space M*)
using *Cauchy_Schwarz_nn_integral*[*of* $\lambda x.\ norm\ (f\ x)$ *M* $\lambda\_.\ 1$]
by (*simp add*: *ennreal_power*)
also have $\ldots < \infty$
using *assms*(*2*) *less_top*
by (*subst* (*asm*) *integrable_iff_bounded*) (*auto simp*: *norm_power ennreal_mult_less_top*)
finally have *nn_integral M* $(\lambda x.\ norm\ (f\ x)) < \infty$
by (*simp add*: *power_less_top_ennreal*)
thus *?thesis*
by (*simp add*: *integrable_iff_bounded*)
qed

lemma *integrable_bound*:
fixes $f :: 'a \Rightarrow 'b::\{banach, second\_countable\_topology\}$
and $g :: 'a \Rightarrow 'c::\{banach, second\_countable\_topology\}$
shows *integrable M f* $\Longrightarrow$ $g \in$ *borel_measurable M* $\Longrightarrow$ $(AE\ x\ in\ M.\ norm\ (g\ x) \leq norm\ (f\ x)) \Longrightarrow$
integrable M g
unfolding *integrable_iff_bounded*
by (*smt* (*verit*) *AE_cong ennreal_le_iff nn_integral_mono_AE norm_ge_zero order_less_subst2 linorder_not_le order_less_le_trans*)

lemma *integrable_mult_indicator*:
fixes $f :: 'a \Rightarrow 'b::\{banach, second\_countable\_topology\}$
shows $A \in$ *sets M* $\Longrightarrow$ *integrable M f* $\Longrightarrow$ *integrable M* $(\lambda x.\ indicator\ A\ x *_R f\ x)$
by (*rule integrable_bound*[*of M f*]) (*auto split*: *split_indicator*)

lemma *integrable_real_mult_indicator*:

fixes $f :: 'a \Rightarrow real$
shows $A \in sets\ M \Longrightarrow integrable\ M\ f \Longrightarrow integrable\ M\ (\lambda x.\ f\ x * indicator\ A\ x)$
using *integrable_mult_indicator*[*of A M f*] **by** (*simp add*: *mult_ac*)

lemma *integrable_abs*[*simp*, *intro*]:
fixes $f :: 'a \Rightarrow real$
assumes [*measurable*]: *integrable M f* **shows** *integrable M* $(\lambda x.\ |f\ x|)$
using *assms* **by** (*rule integrable_bound*) *auto*

lemma *integrable_norm*[*simp*, *intro*]:
fixes $f :: 'a \Rightarrow 'b::\{banach, second\_countable\_topology\}$
assumes [*measurable*]: *integrable M f* **shows** *integrable M* $(\lambda x.\ norm\ (f\ x))$
using *assms* **by** (*rule integrable_bound*) *auto*

lemma *integrable_norm_cancel*:
fixes $f :: 'a \Rightarrow 'b::\{banach, second\_countable\_topology\}$
assumes [*measurable*]: *integrable M* $(\lambda x.\ norm\ (f\ x))$ $f \in borel\_measurable\ M$
shows *integrable M f*
using *assms* **by** (*rule integrable_bound*) *auto*

lemma *integrable_norm_iff*:
fixes $f :: 'a \Rightarrow 'b::\{banach, second\_countable\_topology\}$
shows $f \in borel\_measurable\ M \Longrightarrow integrable\ M\ (\lambda x.\ norm\ (f\ x)) \longleftrightarrow integrable\ M\ f$
by (*auto intro*: *integrable_norm_cancel*)

lemma *integrable_abs_cancel*:
fixes $f :: 'a \Rightarrow real$
assumes [*measurable*]: *integrable M* $(\lambda x.\ |f\ x|)$ $f \in borel\_measurable\ M$ **shows** *integrable M f*
using *assms* **by** (*rule integrable_bound*) *auto*

lemma *integrable_abs_iff*:
fixes $f :: 'a \Rightarrow real$
shows $f \in borel\_measurable\ M \Longrightarrow integrable\ M\ (\lambda x.\ |f\ x|) \longleftrightarrow integrable\ M\ f$
by (*auto intro*: *integrable_abs_cancel*)

lemma *integrable_max*[*simp*, *intro*]:
fixes $f :: 'a \Rightarrow real$
assumes *fg*[*measurable*]: *integrable M f integrable M g*
shows *integrable M* $(\lambda x.\ max\ (f\ x)\ (g\ x))$
using *integrable_add*[*OF integrable_norm*[*OF fg*(*1*)] *integrable_norm*[*OF fg*(*2*)]]
by (*rule integrable_bound*) *auto*

lemma *integrable_min*[*simp*, *intro*]:
fixes $f :: 'a \Rightarrow real$
assumes *fg*[*measurable*]: *integrable M f integrable M g*
shows *integrable M* $(\lambda x.\ min\ (f\ x)\ (g\ x))$
using *integrable_add*[*OF integrable_norm*[*OF fg*(*1*)] *integrable_norm*[*OF fg*(*2*)]]

by (*rule integrable_bound*) *auto*

lemma *integral_minus_iff*[*simp*]:
integrable M (λ*x*. − *f x* ::′*a*::{*banach*, *second_countable_topology*}) ⟷ *integrable M f*
unfolding *integrable_iff_bounded*
by (*auto*)

lemma *integrable_indicator_iff*:
integrable M (*indicator A*::_ ⇒ *real*) ⟷ *A* ∩ *space M* ∈ *sets M* ∧ *emeasure M* (*A* ∩ *space M*) < ∞
by (*simp add*: *integrable_iff_bounded borel_measurable_indicator_iff ennreal_indicator nn_integral_indicator′*
cong: *conj_cong*)

lemma *integral_indicator*[*simp*]: *integral*^{L} *M* (*indicator A*) = *measure M* (*A* ∩ *space M*)
proof *cases*
assume ∗: *A* ∩ *space M* ∈ *sets M* ∧ *emeasure M* (*A* ∩ *space M*) < ∞
have *integral*^{L} *M* (*indicator A*) = *integral*^{L} *M* (*indicator* (*A* ∩ *space M*))
by (*metis* (*no_types*, *lifting*) *Int_iff indicator_simps integral_cong*)
also have . . . = *measure M* (*A* ∩ *space M*)
using ∗ **by** (*intro has_bochner_integral_integral_eq has_bochner_integral_real_indicator*) *auto*
finally show *?thesis* **.**
next
assume ∗: ¬ (*A* ∩ *space M* ∈ *sets M* ∧ *emeasure M* (*A* ∩ *space M*) < ∞)
have *integral*^{L} *M* (*indicator A*) = *integral*^{L} *M* (*indicator* (*A* ∩ *space M*) :: _ ⇒ *real*)
by (*intro integral_cong*) (*auto split*: *split_indicator*)
also have . . . = *0*
using ∗ **by** (*subst not_integrable_integral_eq*) (*auto simp*: *integrable_indicator_iff*)
also have . . . = *measure M* (*A* ∩ *space M*)
using ∗ **by** (*auto simp*: *measure_def emeasure_notin_sets not_less top_unique*)
finally show *?thesis* **.**
qed

lemma *integrable_discrete_difference_aux*:
fixes *f* :: ′*a* ⇒ ′*b*::{*banach*, *second_countable_topology*}
assumes *f*: *integrable M f* **and** *X*: *countable X*
assumes *null*: ⋀*x*. *x* ∈ *X* ⟹ *emeasure M* {*x*} = *0*
assumes *sets*: ⋀*x*. *x* ∈ *X* ⟹ {*x*} ∈ *sets M*
assumes *eq*: ⋀*x*. *x* ∈ *space M* ⟹ *x* ∉ *X* ⟹ *f x* = *g x*
shows *integrable M g*
unfolding *integrable_iff_bounded*
proof
have *fmeas*: *f* ∈ *borel_measurable M* (∫$^+$ *x*. *ennreal* (*norm* (*f x*)) ∂*M*) < ∞
using *f integrable_iff_bounded* **by** *auto*
then show *g* ∈ *borel_measurable M*

using *measurable_discrete_difference*[**where** *X*=*X*]
by (*smt* (*verit*) *UNIV_I X eq sets space_borel*)
have *AE x in M*. *x* ∉ *X*
using *AE_discrete_difference X null sets* **by** *blast*
with *fmeas* **show** (∫⁺ *x*. *ennreal* (*norm* (*g x*)) ∂*M*) < ∞
by (*metis* (*mono_tags*, *lifting*) *AE_I2 AE_mp eq nn_integral_cong_AE*)
qed

lemma *integrable_discrete_difference*:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second_countable_topology*}
assumes *X*: *countable X*
assumes *null*: ⋀*x*. *x* ∈ *X* ⟹ *emeasure M* {*x*} = *0*
assumes *sets*: ⋀*x*. *x* ∈ *X* ⟹ {*x*} ∈ *sets M*
assumes *eq*: ⋀*x*. *x* ∈ *space M* ⟹ *x* ∉ *X* ⟹ *f x* = *g x*
shows *integrable M f* ⟷ *integrable M g*
by (*metis X eq integrable_discrete_difference_aux null sets*)

lemma *integral_discrete_difference*:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second_countable_topology*}
assumes *X*: *countable X*
assumes *null*: ⋀*x*. *x* ∈ *X* ⟹ *emeasure M* {*x*} = *0*
assumes *sets*: ⋀*x*. *x* ∈ *X* ⟹ {*x*} ∈ *sets M*
assumes *eq*: ⋀*x*. *x* ∈ *space M* ⟹ *x* ∉ *X* ⟹ *f x* = *g x*
shows *integral*^{L} *M f* = *integral*^{L} *M g*
proof (*rule integral_eq_cases*)
show *eq*: *integrable M f* ⟷ *integrable M g*
by (*rule integrable_discrete_difference*[**where** *X*=*X*]) *fact*+

assume *f*: *integrable M f*
show *integral*^{L} *M f* = *integral*^{L} *M g*
proof (*rule integral_cong_AE*)
show *f* ∈ *borel_measurable M g* ∈ *borel_measurable M*
using *f eq* **by** (*auto intro*: *borel_measurable_integrable*)
have *AE x in M*. *x* ∉ *X*
by (*rule AE_discrete_difference*) *fact*+
with *AE_space* **show** *AE x in M*. *f x* = *g x*
by *eventually_elim fact*
qed
qed

lemma *has_bochner_integral_discrete_difference*:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second_countable_topology*}
assumes *countable X*
assumes ⋀*x*. *x* ∈ *X* ⟹ *emeasure M* {*x*} = *0*
assumes ⋀*x*. *x* ∈ *X* ⟹ {*x*} ∈ *sets M*
assumes ⋀*x*. *x* ∈ *space M* ⟹ *x* ∉ *X* ⟹ *f x* = *g x*
shows *has_bochner_integral M f x* ⟷ *has_bochner_integral M g x*
by (*metis assms has_bochner_integral_iff integrable_discrete_difference integral_discrete_difference*)

lemma
fixes $f :: 'a \Rightarrow 'b::\{banach, second\_countable\_topology\}$ **and** $w :: 'a \Rightarrow real$
assumes $f \in borel\_measurable\ M$ $\bigwedge i.\ s\ i \in borel\_measurable\ M$ *integrable* $M\ w$
assumes *lim*: $AE\ x\ in\ M.\ (\lambda i.\ s\ i\ x) \longrightarrow f\ x$
assumes *bound*: $\bigwedge i.\ AE\ x\ in\ M.\ norm\ (s\ i\ x) \leq w\ x$
shows *integrable_dominated_convergence*: *integrable* $M\ f$
and *integrable_dominated_convergence2*: $\bigwedge i.$ *integrable* $M\ (s\ i)$
and *integral_dominated_convergence*: $(\lambda i.\ integral^L\ M\ (s\ i)) \longrightarrow integral^L\ M\ f$
proof −
have *w_nonneg*: $AE\ x\ in\ M.\ 0 \leq w\ x$
using *bound*[*of 0*] **by** *eventually_elim* (*auto intro*: *norm_ge_zero order_trans*)
then have $(\int^+ x.\ w\ x\ \partial M) = (\int^+ x.\ norm\ (w\ x)\ \partial M)$
by (*intro nn_integral_cong_AE*) *auto*
with ‹*integrable* $M\ w$› **have** *w*: $w \in borel\_measurable\ M$ $(\int^+ x.\ w\ x\ \partial M) < \infty$
unfolding *integrable_iff_bounded* **by** *auto*

show *int_s*: $\bigwedge i.$ *integrable* $M\ (s\ i)$
unfolding *integrable_iff_bounded*
proof
fix *i*
have $(\int^+ x.\ ennreal\ (norm\ (s\ i\ x))\ \partial M) \leq (\int^+ x.\ w\ x\ \partial M)$
using *bound*[*of i*] *w_nonneg* **by** (*intro nn_integral_mono_AE*) *auto*
with *w* **show** $(\int^+ x.\ ennreal\ (norm\ (s\ i\ x))\ \partial M) < \infty$ **by** *auto*
qed *fact*

have *all_bound*: $AE\ x\ in\ M.\ \forall i.\ norm\ (s\ i\ x) \leq w\ x$
using *bound* **unfolding** *AE_all_countable* **by** *auto*

show *int_f*: *integrable* $M\ f$
unfolding *integrable_iff_bounded*
proof
have $(\int^+ x.\ ennreal\ (norm\ (f\ x))\ \partial M) \leq (\int^+ x.\ w\ x\ \partial M)$
using *all_bound lim w_nonneg*
proof (*intro nn_integral_mono_AE*, *eventually_elim*)
fix *x* **assume** $\forall i.\ norm\ (s\ i\ x) \leq w\ x$ $(\lambda i.\ s\ i\ x) \longrightarrow f\ x\ 0 \leq w\ x$
then show *ennreal* $(norm\ (f\ x)) \leq ennreal\ (w\ x)$
by (*intro LIMSEQ_le_const2*[**where** $X=\lambda i.\ ennreal\ (norm\ (s\ i\ x))$]) (*auto intro*: *tendsto_intros*)
qed
with *w* **show** $(\int^+ x.\ ennreal\ (norm\ (f\ x))\ \partial M) < \infty$ **by** *auto*
qed *fact*

have $(\lambda n.\ ennreal\ (norm\ (integral^L\ M\ (s\ n) - integral^L\ M\ f))) \longrightarrow ennreal\ 0$ (**is** *?d* $\longrightarrow$ *ennreal 0*)
proof (*rule tendsto_sandwich*)
show *eventually* $(\lambda n.\ ennreal\ 0 \leq ?d\ n)$ *sequentially* $(\lambda\_.\ ennreal\ 0) \longrightarrow ennreal\ 0$ **by** *auto*

```
    show eventually (λn. ?d n ≤ (∫⁺x. norm (s n x − f x) ∂M)) sequentially
    proof (intro always_eventually allI)
      fix n
      have ?d n = norm (integral^L M (λx. s n x − f x))
        using int_f int_s by simp
      also have ... ≤ (∫⁺x. norm (s n x − f x) ∂M)
        by (intro int_f int_s integrable_diff integral_norm_bound_ennreal)
      finally show ?d n ≤ (∫⁺x. norm (s n x − f x) ∂M) .
    qed
    show (λn. ∫⁺x. norm (s n x − f x) ∂M) ⟶ ennreal 0
      unfolding ennreal_0
      apply (subst norm_minus_commute)
    proof (rule nn_integral_dominated_convergence_norm[where w=w])
      show ⋀n. s n ∈ borel_measurable M
        using int_s unfolding integrable_iff_bounded by auto
    qed fact+
  qed
  then have (λn. integral^L M (s n) − integral^L M f) ⟶ 0
    by (simp add: tendsto_norm_zero_iff del: ennreal_0)
  from tendsto_add[OF this tendsto_const[of integral^L M f]]
  show (λi. integral^L M (s i)) ⟶ integral^L M f by simp
qed

context
  fixes s :: real ⇒ 'a ⇒ 'b::{banach, second_countable_topology} and w :: 'a ⇒
real
    and f :: 'a ⇒ 'b and M
  assumes f ∈ borel_measurable M ⋀t. s t ∈ borel_measurable M integrable M w
  assumes lim: AE x in M. ((λi. s i x) ⟶ f x) at_top
  assumes bound: ∀_F i in at_top. AE x in M. norm (s i x) ≤ w x
begin

lemma integral_dominated_convergence_at_top: ((λt. integral^L M (s t)) ⟶
integral^L M f) at_top
proof (rule tendsto_at_topI_sequentially)
  fix X :: nat ⇒ real assume X: filterlim X at_top sequentially
  from filterlim_iff[THEN iffD1, OF this, rule_format, OF bound]
  obtain N where w: ⋀n. N ≤ n ⟹ AE x in M. norm (s (X n) x) ≤ w x
    by (auto simp: eventually_sequentially)

  show (λn. integral^L M (s (X n))) ⟶ integral^L M f
  proof (rule LIMSEQ_offset, rule integral_dominated_convergence)
    show AE x in M. norm (s (X (n + N)) x) ≤ w x for n
      by (rule w) auto
    show AE x in M. (λn. s (X (n + N)) x) ⟶ f x
      using lim
    proof eventually_elim
      fix x assume ((λi. s i x) ⟶ f x) at_top
      then show (λn. s (X (n + N)) x) ⟶ f x
```

```
      by (intro LIMSEQ_ignore_initial_segment filterlim_compose[OF _ X])
    qed
  qed fact+
qed

lemma integrable_dominated_convergence_at_top: integrable M f
proof −
  from bound obtain N where w: ⋀n. N ≤ n ⟹ AE x in M. norm (s n x) ≤
w x
    by (auto simp: eventually_at_top_linorder)
  show ?thesis
  proof (rule integrable_dominated_convergence)
    show AE x in M. norm (s (N + i) x) ≤ w x for i :: nat
      by (intro w) auto
    show AE x in M. (λi. s (N + real i) x) ⟶ f x
      using lim
    proof eventually_elim
      fix x assume ((λi. s i x) ⟶ f x) at_top
      then show (λn. s (N + n) x) ⟶ f x
        by (rule filterlim_compose)
           (auto intro!: filterlim_tendsto_add_at_top filterlim_real_sequentially)
    qed
  qed fact+
qed

end

lemma integrable_mult_left_iff [simp]:
  fixes f :: 'a ⇒ real
  shows integrable M (λx. c ∗ f x) ⟷ c = 0 ∨ integrable M f
  using integrable_mult_left[of c M f] integrable_mult_left[of 1 / c M λx. c ∗ f
x]
  by (cases c = 0) auto

lemma integrable_mult_right_iff [simp]:
  fixes f :: 'a ⇒ real
  shows integrable M (λx. f x ∗ c) ⟷ c = 0 ∨ integrable M f
  using integrable_mult_left_iff [of M c f] by (simp add: mult.commute)

lemma integrableI_nn_integral_finite:
  assumes [measurable]: f ∈ borel_measurable M
    and nonneg: AE x in M. 0 ≤ f x
    and finite: (∫+x. f x ∂M) = ennreal x
  shows integrable M f
proof (rule integrableI_bounded)
  have (∫+ x. ennreal (norm (f x)) ∂M) = (∫+ x. ennreal (f x) ∂M)
    using nonneg by (intro nn_integral_cong_AE) auto
  with finite show (∫+ x. ennreal (norm (f x)) ∂M) < ∞
    by auto
```

qed *simp*

lemma *integral_nonneg_AE*:
fixes $f :: 'a \Rightarrow real$
assumes *nonneg*: $AE\ x\ in\ M.\ 0 \leq f\ x$
shows $0 \leq integral^L\ M\ f$
proof *cases*
assume *f*: *integrable M f*
then have [*measurable*]: $f \in M \rightarrow_M borel$
by *auto*
have $(\lambda x.\ max\ 0\ (f\ x)) \in M \rightarrow_M borel\ \bigwedge x.\ 0 \leq max\ 0\ (f\ x)\ integrable\ M\ (\lambda x.\ max\ 0\ (f\ x))$
using *f* **by** *auto*
from *this* **have** $0 \leq integral^L\ M\ (\lambda x.\ max\ 0\ (f\ x))$
proof (*induction rule*: *borel_measurable_induct_real*)
case (*add f g*)
then have *integrable M f integrable M g*
by (*auto intro*!: *integrable_bound*[*OF add.prems*])
with *add* **show** *?case*
by (*simp add*: *nn_integral_add*)
next
case (*seq U*)
show *?case*
proof (*rule LIMSEQ_le_const*)
have *U_le*: $x \in space\ M \Longrightarrow U\ i\ x \leq max\ 0\ (f\ x)$ **for** *x i*
using *seq* **by** (*intro incseq_le*) (*auto simp*: *incseq_def le_fun_def*)
with *seq nonneg* **show** $(\lambda i.\ integral^L\ M\ (U\ i)) \longrightarrow LINT\ x|M.\ max\ 0\ (f\ x)$
by (*intro integral_dominated_convergence*)
(*simp_all*, *fastforce*)
have *integrable M* (*U i*) **for** *i*
using *seq.prems* **by** (*rule integrable_bound*) (*insert U_le seq*, *auto*)
with *seq* **show** $\exists N.\ \forall n \geq N.\ 0 \leq integral^L\ M\ (U\ n)$
by *auto*
qed
qed (*auto*)
also have $\ldots = integral^L\ M\ f$
using *nonneg* **by** (*auto intro*!: *integral_cong_AE*)
finally show *?thesis* **.**
qed (*simp add*: *not_integrable_integral_eq*)

lemma *integral_nonneg*[*simp*]:
fixes $f :: 'a \Rightarrow real$
shows $(\bigwedge x.\ x \in space\ M \Longrightarrow 0 \leq f\ x) \Longrightarrow 0 \leq integral^L\ M\ f$
by (*intro integral_nonneg_AE*) *auto*

proposition *nn_integral_eq_integral*:
assumes *f*: *integrable M f*
assumes *nonneg*: $AE\ x\ in\ M.\ 0 \leq f\ x$

```
  shows (∫⁺ x. f x ∂M) = integral^L M f
proof −
  { fix f :: 'a ⇒ real assume f: f ∈ borel_measurable M ⋀x. 0 ≤ f x integrable
M f
    then have (∫⁺ x. f x ∂M) = integral^L M f
    proof (induct rule: borel_measurable_induct_real)
      case (set A) then show ?case
      by (simp add: integrable_indicator_iff ennreal_indicator emeasure_eq_ennreal_measure)
    next
      case (mult f c) then show ?case
        by (auto simp: nn_integral_cmult ennreal_mult integral_nonneg_AE)
    next
      case (add g f)
      then have integrable M f integrable M g
        by (auto intro!: integrable_bound[OF add.prems])
      with add show ?case
        by (simp add: nn_integral_add integral_nonneg_AE)
    next
      case (seq U)
      show ?case
      proof (rule LIMSEQ_unique)
        have U_le_f: x ∈ space M ⟹ U i x ≤ f x for x i
          using seq by (intro incseq_le) (auto simp: incseq_def le_fun_def)
        have int_U: ⋀i. integrable M (U i)
          using seq f U_le_f by (intro integrable_bound[OF f(3)]) auto
        from U_le_f seq have (λi. integral^L M (U i)) ⟶ integral^L M f
          by (intro integral_dominated_convergence) auto
        then show (λi. ennreal (integral^L M (U i))) ⟶ ennreal (integral^L M
f)
          using seq f int_U by (simp add: f integral_nonneg_AE)
        have (λi. ∫⁺ x. U i x ∂M) ⟶ ∫⁺ x. f x ∂M
          using seq U_le_f f
          by (intro nn_integral_dominated_convergence[where w=f]) (auto simp:
integrable_iff_bounded)
        then show (λi. ∫ x. U i x ∂M) ⟶ ∫⁺x. f x ∂M
          using seq int_U by simp
      qed
    qed }
  from this[of λx. max 0 (f x)] assms have (∫⁺ x. max 0 (f x) ∂M) = integral^L
M (λx. max 0 (f x))
    by simp
  also have ... = integral^L M f
    using assms by (auto intro!: integral_cong_AE simp: integral_nonneg_AE)
  also have (∫⁺ x. max 0 (f x) ∂M) = (∫⁺ x. f x ∂M)
    using assms by (auto intro!: nn_integral_cong_AE simp: max_def)
  finally show ?thesis .
qed

lemma nn_integral_eq_integrable:
```

assumes $f \in M \to_M$ *borel AE x in M.* $0 \le f\ x$ **and** $0 \le x$
shows $(\int^+ x.\ f\ x\ \partial M) = ennreal\ x \longleftrightarrow (integrable\ M\ f \wedge integral^L\ M\ f = x)$
by (*metis assms ennreal_inj integrableI_nn_integral_finite integral_nonneg_AE nn_integral_eq_integral*)

lemma
fixes $f :: \_ \Rightarrow \_ \Rightarrow 'a :: \{banach, second\_countable\_topology\}$
assumes *integrable*[*measurable*]: $\bigwedge i.\ integrable\ M\ (f\ i)$
and *summable*: *AE x in M. summable* $(\lambda i.\ norm\ (f\ i\ x))$
and *sums*: *summable* $(\lambda i.\ (\int x.\ norm\ (f\ i\ x)\ \partial M))$
shows *integrable_suminf*: *integrable M* $(\lambda x.\ (\sum i.\ f\ i\ x))$ (**is** *integrable M ?S*)
and *sums_integral*: $(\lambda i.\ integral^L\ M\ (f\ i))$ *sums* $(\int x.\ (\sum i.\ f\ i\ x)\ \partial M)$ (**is** *?f sums ?x*)
and *integral_suminf*: $(\int x.\ (\sum i.\ f\ i\ x)\ \partial M) = (\sum i.\ integral^L\ M\ (f\ i))$
and *summable_integral*: *summable* $(\lambda i.\ integral^L\ M\ (f\ i))$
proof −
have *1*: *integrable M* $(\lambda x.\ \sum i.\ norm\ (f\ i\ x))$
proof (*rule integrableI_bounded*)
have $\bigwedge x.\ summable\ (\lambda i.\ norm\ (f\ i\ x)) \Longrightarrow$
$ennreal\ (norm\ (\sum i.\ norm\ (f\ i\ x))) = (\sum i.\ ennreal\ (norm\ (f\ i\ x)))$
by (*simp add*: *suminf_nonneg ennreal_suminf_neq_top*)
then have $(\int^+ x.\ ennreal\ (norm\ (\sum i.\ norm\ (f\ i\ x)))\ \partial M) = (\int^+ x.\ (\sum i.\ ennreal\ (norm\ (f\ i\ x)))\ \partial M)$
using *local.summable* **by** (*intro nn_integral_cong_AE*) *auto*
also have $\ldots = (\sum i.\ \int^+ x.\ norm\ (f\ i\ x)\ \partial M)$
by (*intro nn_integral_suminf*) *auto*
also have $\ldots = (\sum i.\ ennreal\ (\int x.\ norm\ (f\ i\ x)\ \partial M))$
by (*intro arg_cong*[**where** *f=suminf*] *ext nn_integral_eq_integral integrable_norm integrable*) *auto*
finally show $(\int^+ x.\ ennreal\ (norm\ (\sum i.\ norm\ (f\ i\ x)))\ \partial M) < \infty$
by (*simp add*: *sums ennreal_suminf_neq_top less_top*[*symmetric*] *integral_nonneg_AE*)
qed *simp*

have *2*: *AE x in M.* $(\lambda n.\ \sum i{<}n.\ f\ i\ x) \longrightarrow (\sum i.\ f\ i\ x)$
using *summable* **by** *eventually_elim* (*auto intro*: *summable_LIMSEQ summable_norm_cancel*)

have *3*: $\bigwedge j.$ *AE x in M.* $norm\ (\sum i{<}j.\ f\ i\ x) \le (\sum i.\ norm\ (f\ i\ x))$
using *summable*
proof *eventually_elim*
fix *j x* **assume** [*simp*]: *summable* $(\lambda i.\ norm\ (f\ i\ x))$
have $norm\ (\sum i{<}j.\ f\ i\ x) \le (\sum i{<}j.\ norm\ (f\ i\ x))$ **by** (*rule norm_sum*)
also have $\ldots \le (\sum i.\ norm\ (f\ i\ x))$
using *sum_le_suminf*[*of* $\lambda i.\ norm\ (f\ i\ x)$] **unfolding** *sums_iff* **by** *auto*
finally show $norm\ (\sum i{<}j.\ f\ i\ x) \le (\sum i.\ norm\ (f\ i\ x))$ **by** *simp*
qed

note *ibl* = *integrable_dominated_convergence*[*OF _ _ 1 2 3*]
note *int* = *integral_dominated_convergence*[*OF _ _ 1 2 3*]

show *integrable M ?S*
by (*rule ibl*) *measurable*

show *?f sums ?x* **unfolding** *sums_def*
using *int* **by** (*simp add*: *integrable*)
then show *?x = suminf ?f summable ?f*
unfolding *sums_iff* **by** *auto*
qed

proposition *integral_norm_bound* [*simp*]:
fixes *f* :: *_* ⇒ *'a* :: {*banach*, *second_countable_topology*}
shows *norm* (*integral*^{L} *M f*) ≤ (∫ *x*. *norm* (*f x*) ∂*M*)
proof (*cases integrable M f*)
case *True* **then show** *?thesis*
using *nn_integral_eq_integral*[*of M* λ*x*. *norm* (*f x*)] *integral_norm_bound_ennreal*[*of M f*]
by (*simp add*: *integral_nonneg_AE*)
next
case *False*
then have *norm* (*integral*^{L} *M f*) = *0* **by** (*simp add*: *not_integrable_integral_eq*)
moreover have (∫ *x*. *norm* (*f x*) ∂*M*) ≥ *0* **by** *auto*
ultimately show *?thesis* **by** *simp*
qed

proposition *integral_abs_bound* [*simp*]:
fixes *f* :: *'a* ⇒ *real* **shows** *abs* (∫ *x*. *f x* ∂*M*) ≤ (∫ *x*. |*f x*| ∂*M*)
using *integral_norm_bound*[*of M f*] **by** *auto*

lemma *integral_eq_nn_integral*:
assumes *f* ∈ *borel_measurable M*
assumes *AE x in M*. *0* ≤ *f x*
shows *integral*^{L} *M f* = *enn2real* (∫$^+$ *x*. *ennreal* (*f x*) ∂*M*)
by (*metis assms enn2real_ennreal enn2real_top infinity_ennreal_def integrableI_nonneg integral_nonneg_AE*
less_top nn_integral_eq_integral not_integrable_integral_eq)

lemma *enn2real_nn_integral_eq_integral*:
assumes *AE x in M*. *f x* = *ennreal* (*g x*) **and** *nn*: *AE x in M*. *0* ≤ *g x*
and *g* ∈ *M* →$_M$ *borel*
shows *enn2real* (∫^+x. *f x* ∂*M*) = (∫ *x*. *g x* ∂*M*)
by (*metis assms integral_eq_nn_integral nn nn_integral_cong_AE*)

lemma *has_bochner_integral_nn_integral*:
assumes *f* ∈ *borel_measurable M AE x in M*. *0* ≤ *f x 0* ≤ *x*
assumes (∫^+x. *f x* ∂*M*) = *ennreal x*
shows *has_bochner_integral M f x*
by (*metis assms has_bochner_integral_iff nn_integral_eq_integrable*)

lemma *integrableI_simple_bochner_integrable*:
fixes $f :: 'a \Rightarrow 'b::\{banach, second\_countable\_topology\}$
shows *simple_bochner_integrable M f* $\Longrightarrow$ *integrable M f*
using *has_bochner_integral_simple_bochner_integrable integrable.intros* **by** *blast*

proposition *integrable_induct*[*consumes 1*, *case_names base add lim*, *induct pred*: *integrable*]:
fixes $f :: 'a \Rightarrow 'b::\{banach, second\_countable\_topology\}$
assumes *integrable M f*
assumes *base*: $\bigwedge A\ c.\ A \in sets\ M \Longrightarrow emeasure\ M\ A < \infty \Longrightarrow P\ (\lambda x.\ indicator\ A\ x *_R c)$
assumes *add*: $\bigwedge f\ g.$ *integrable M f* $\Longrightarrow P\ f \Longrightarrow$ *integrable M g* $\Longrightarrow P\ g \Longrightarrow P\ (\lambda x.\ f\ x + g\ x)$
assumes *lim*: $\bigwedge f\ s.\ (\bigwedge i.$ *integrable M* $(s\ i)) \Longrightarrow (\bigwedge i.\ P\ (s\ i)) \Longrightarrow$
$(\bigwedge x.\ x \in space\ M \Longrightarrow (\lambda i.\ s\ i\ x) \longrightarrow f\ x) \Longrightarrow$
$(\bigwedge i\ x.\ x \in space\ M \Longrightarrow norm\ (s\ i\ x) \leq 2 * norm\ (f\ x)) \Longrightarrow$ *integrable M f* $\Longrightarrow P\ f$
shows $P\ f$
proof −
from ‹*integrable M f*› **have** *f*: $f \in borel\_measurable\ M\ (\int^+ x.\ norm\ (f\ x)\ \partial M) < \infty$
unfolding *integrable_iff_bounded* **by** *auto*
from *borel_measurable_implies_sequence_metric*[*OF f*(*1*)]
obtain *s* **where** *s*: $\bigwedge i.$ *simple_function M* $(s\ i)\ \bigwedge x.\ x \in space\ M \Longrightarrow (\lambda i.\ s\ i\ x) \longrightarrow f\ x$
$\bigwedge i\ x.\ x \in space\ M \Longrightarrow norm\ (s\ i\ x) \leq 2 * norm\ (f\ x)$
unfolding *norm_conv_dist* **by** *metis*

{ fix *f A*
have [*simp*]: $P\ (\lambda x.\ 0)$
using *base*[*of* {} *undefined*] **by** *simp*
have $(\bigwedge i::'b.\ i \in A \Longrightarrow$ *integrable M* $(f\ i::'a \Rightarrow 'b)) \Longrightarrow$
$(\bigwedge i.\ i \in A \Longrightarrow P\ (f\ i)) \Longrightarrow P\ (\lambda x.\ \sum i \in A.\ f\ i\ x)$
by (*induct A rule*: *infinite_finite_induct*) (*auto intro*!: *add*) **}**
note *sum* = *this*

define s' **where** [*abs_def*]: $s'\ i\ z =$ *indicator* (*space M*) $z *_R s\ i\ z$ **for** *i z*
then have s'_eq_s: $\bigwedge i\ x.\ x \in space\ M \Longrightarrow s'\ i\ x = s\ i\ x$
by *simp*

have *sf*[*measurable*]: $\bigwedge i.$ *simple_function M* $(s'\ i)$
unfolding s'_def **using** *s*(*1*)
by (*intro simple_function_compose2*[**where** $h=(*_R)$] *simple_function_indicator*) *auto*

{ fix *i*
have $\bigwedge z.\ \{y.\ s'\ i\ z = y \wedge y \in s'\ i\ `\ space\ M \wedge y \neq 0 \wedge z \in space\ M\} =$
(*if* $z \in space\ M \wedge s'\ i\ z \neq 0$ *then* $\{s'\ i\ z\}$ *else* {})
by (*auto simp*: s'_def *split*: *split_indicator*)

```
    then have ⋀z. s' i = (λz. ∑ y∈s' i`space M - {0}. indicator {x∈space M. s'
i x = y} z *R y)
      using sf by (auto simp: fun_eq_iff simple_function_def s'_def) }
  note s'_eq = this

  show P f
  proof (rule lim)
    fix i

    have (∫+x. norm (s' i x) ∂M) ≤ (∫+x. ennreal (2 * norm (f x)) ∂M)
      using s by (intro nn_integral_mono) (auto simp: s'_eq_s)
    also have ... < ∞
     using f by (simp add: nn_integral_cmult ennreal_mult_less_top ennreal_mult)
    finally have sbi: simple_bochner_integrable M (s' i)
      using sf by (intro simple_bochner_integrableI_bounded) auto
    then show integrable M (s' i)
      by (rule integrableI_simple_bochner_integrable)

    { fix x assumex ∈ space M s' i x ≠ 0
      then have emeasure M {y ∈ space M. s' i y = s' i x} ≤ emeasure M {y ∈
space M. s' i y ≠ 0}
        by (intro emeasure_mono) auto
      also have ... < ∞
        using sbi by (auto elim: simple_bochner_integrable.cases simp: less_top)
      finally have emeasure M {y ∈ space M. s' i y = s' i x} ≠ ∞ by simp }
    then show P (s' i)
      by (subst s'_eq) (auto intro!: sum base simp: less_top)

    fix x assume x ∈ space M with s show (λi. s' i x) ⟶ f x
      by (simp add: s'_eq_s)
    show norm (s' i x) ≤ 2 * norm (f x)
      using ‹x ∈ space M› s by (simp add: s'_eq_s)
  qed fact
qed

lemma integral_eq_zero_AE:
  (AE x in M. f x = 0) ⟹ integral^L M f = 0
  using integral_cong_AE[of f M λ_. 0]
  by (cases integrable M f) (simp_all add: not_integrable_integral_eq)

lemma integral_nonneg_eq_0_iff_AE:
  fixes f :: _ ⇒ real
  assumes f[measurable]: integrable M f and nonneg: AE x in M. 0 ≤ f x
  shows integral^L M f = 0 ⟷ (AE x in M. f x = 0)
proof
  assume integral^L M f = 0
  then have integral^N M f = 0
    using nn_integral_eq_integral[OF f nonneg] by simp
  then have AE x in M. ennreal (f x) ≤ 0
```

by (*simp add*: *nn_integral_0_iff_AE*)
with *nonneg* **show** *AE x in M*. *f x* = *0*
by *auto*
qed (*auto simp*: *integral_eq_zero_AE*)

lemma *integral_mono_AE*:
fixes *f* :: *'a* ⇒ *real*
assumes *integrable M f integrable M g AE x in M*. *f x* ≤ *g x*
shows *integral*^{L} *M f* ≤ *integral*^{L} *M g*
proof −
have *0* ≤ *integral*^{L} *M* (λ*x*. *g x* − *f x*)
using *assms* **by** (*intro integral_nonneg_AE integrable_diff assms*) *auto*
also have ... = *integral*^{L} *M g* − *integral*^{L} *M f*
by (*intro integral_diff assms*)
finally show *?thesis* **by** *simp*
qed

lemma *integral_mono*:
fixes *f* :: *'a* ⇒ *real*
shows *integrable M f* ⟹ *integrable M g* ⟹ (⋀*x*. *x* ∈ *space M* ⟹ *f x* ≤ *g x*)
⟹
integral^L *M f* ≤ *integral*^{L} *M g*
by (*intro integral_mono_AE*) *auto*

lemma *integral_norm_bound_integral*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::{*banach*,*second_countable_topology*}
assumes *integrable M f integrable M g* ⋀*x*. *x* ∈ *space M* ⟹ *norm*(*f x*) ≤ *g x*
shows *norm* (∫ *x*. *f x* ∂*M*) ≤ (∫ *x*. *g x* ∂*M*)
by (*smt* (*verit*, *best*) *assms integrable_norm integral_mono integral_norm_bound*)

lemma *integral_abs_bound_integral*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *real*
assumes *integrable M f integrable M g* ⋀*x*. *x* ∈ *space M* ⟹ |*f x*| ≤ *g x*
shows |∫ *x*. *f x* ∂*M*| ≤ (∫ *x*. *g x* ∂*M*)
by (*metis integral_norm_bound_integral assms real_norm_def*)

The next two statements are useful to bound Lebesgue integrals, as they avoid one integrability assumption. The price to pay is that the upper function has to be nonnegative, but this is often true and easy to check in computations.

lemma *integral_mono_AE′*:
fixes *f*::_ ⇒ *real*
assumes *integrable M f AE x in M*. *g x* ≤ *f x AE x in M*. *0* ≤ *f x*
shows (∫ *x*. *g x* ∂*M*) ≤ (∫ *x*. *f x* ∂*M*)
by (*metis assms integral_mono_AE integral_nonneg_AE not_integrable_integral_eq*)

lemma *integral_mono′*:
fixes *f*::_ ⇒ *real*
assumes *integrable M f* ⋀*x*. *x* ∈ *space M* ⟹ *g x* ≤ *f x* ⋀*x*. *x* ∈ *space M* ⟹

$0 \leq f\ x$
shows $(\int x.\ g\ x\ \partial M) \leq (\int x.\ f\ x\ \partial M)$
by (*rule integral_mono_AE'*, *insert assms*, *auto*)

lemma (**in** *finite_measure*) *integrable_measure*:
assumes *I*: *disjoint_family_on X I countable I*
shows *integrable* (*count_space I*) ($\lambda i.$ *measure M* (*X i*))
proof −
have $(\int^+ i.$ *measure M* (*X i*) *∂count_space I*$) = (\int^+ i.$ *measure M* (*if X i ∈ sets M then X i else* {}) *∂count_space I*)
by (*auto intro*!: *nn_integral_cong measure_notin_sets*)
also have ... = *measure M* ($\bigcup i \in I.$ *if X i ∈ sets M then X i else* {})
using *I* **unfolding** *emeasure_eq_measure*[*symmetric*]
by (*subst emeasure_UN_countable*) (*auto simp*: *disjoint_family_on_def*)
finally show *?thesis*
by (*auto intro*!: *integrableI_bounded*)
qed

lemma *nn_integral_nonneg_infinite*:
fixes *f*::*'a* ⇒ *real*
assumes *f ∈ borel_measurable M* ¬ *integrable M f AE x in M. f x ≥ 0*
shows $(\int^+ x.$ *f x ∂M*$) = \infty$
using *assms integrableI_nonneg less_top* **by** *auto*

lemma *integral_real_bounded*:
assumes *0 ≤ r integral*^{N} *M f ≤ ennreal r*
shows *integral*^{L} *M f ≤ r*
proof *cases*
assume [*simp*]: *integrable M f*

have *integral*^{L} *M* ($\lambda x.$ *max 0* (*f x*)) = *integral*^{N} *M* ($\lambda x.$ *max 0* (*f x*))
by (*intro nn_integral_eq_integral*[*symmetric*]) *auto*
also have ... = *integral*^{N} *M f*
by (*intro nn_integral_cong*) (*simp add*: *max_def ennreal_neg*)
also have ... ≤ *r*
by *fact*
finally have *integral*^{L} *M* ($\lambda x.$ *max 0* (*f x*)) ≤ *r*
using ‹*0 ≤ r*› **by** *simp*

moreover have *integral*^{L} *M f ≤ integral*^{L} *M* ($\lambda x.$ *max 0* (*f x*))
by (*rule integral_mono_AE*) *auto*
ultimately show *?thesis*
by *simp*
next
assume ¬ *integrable M f* **then show** *?thesis*
using ‹*0 ≤ r*› **by** (*simp add*: *not_integrable_integral_eq*)
qed

lemma *integrable_MIN*:

fixes f:: _ ⇒ _ ⇒ *real*
shows ⟦ *finite* I; $I \neq \{\}$; $\bigwedge i.\ i \in I \Longrightarrow$ *integrable* M $(f\ i)$ ⟧
$\Longrightarrow$ *integrable* M $(\lambda x.\ MIN\ i{\in}I.\ f\ i\ x)$
by (*induct rule*: *finite_ne_induct*) *simp+*

lemma *integrable_MAX*:
fixes f:: _ ⇒ _ ⇒ *real*
shows ⟦ *finite* I; $I \neq \{\}$; $\bigwedge i.\ i \in I \Longrightarrow$ *integrable* M $(f\ i)$ ⟧
$\Longrightarrow$ *integrable* M $(\lambda x.\ MAX\ i{\in}I.\ f\ i\ x)$
by (*induct rule*: *finite_ne_induct*) *simp+*

theorem *integral_Markov_inequality*:
assumes [*measurable*]: *integrable* M u **and** *AE* x *in* M. $0 \leq u\ x$ $0 < (c{::}real)$
shows $(emeasure\ M)\ \{x{\in}space\ M.\ u\ x \geq c\} \leq (1/c) * (\int x.\ u\ x\ \partial M)$
proof −
have $(\int^+ x.\ ennreal(u\ x) * indicator\ (space\ M)\ x\ \partial M) \leq (\int^+ x.\ u\ x\ \partial M)$
by (*rule nn_integral_mono_AE*, *auto simp*: ‹$c{>}0$› *less_eq_real_def*)
also have ... $= (\int x.\ u\ x\ \partial M)$
by (*rule nn_integral_eq_integral*, *auto simp*: *assms*)
finally have ∗: $(\int^+ x.\ ennreal(u\ x) * indicator\ (space\ M)\ x\ \partial M) \leq (\int x.\ u\ x\ \partial M)$
by *simp*

have $\{x \in space\ M.\ u\ x \geq c\} = \{x \in space\ M.\ ennreal(1/c) * u\ x \geq 1\} \cap (space\ M)$
using ‹$c{>}0$› **by** (*auto simp*: *ennreal_mult'*[*symmetric*])
then have $emeasure\ M\ \{x \in space\ M.\ u\ x \geq c\} = emeasure\ M\ (\{x \in space\ M.\ ennreal(1/c) * u\ x \geq 1\})$
by *simp*
also have ... $\leq ennreal(1/c) * (\int^+ x.\ ennreal(u\ x) * indicator\ (space\ M)\ x\ \partial M)$
by (*rule nn_integral_Markov_inequality*) (*auto simp*: *assms*)
also have ... $\leq ennreal(1/c) * (\int x.\ u\ x\ \partial M)$
by (*rule mult_left_mono*) (*use* ∗ ‹$c > 0$› **in** *auto*)
finally show *?thesis*
using ‹$0{<}c$› **by** (*simp add*: *ennreal_mult'*[*symmetric*])
qed

theorem *integral_Markov_inequality_measure*:
assumes [*measurable*]: *integrable* M u **and** $A \in sets\ M$ **and** *AE* x *in* M. $0 \leq u\ x\ 0 < (c{::}real)$
shows $measure\ M\ \{x{\in}space\ M.\ u\ x \geq c\} \leq (\int x.\ u\ x\ \partial M)\ /\ c$
proof −
have *le*: $emeasure\ M\ \{x{\in}space\ M.\ u\ x \geq c\} \leq ennreal\ ((1/c) * (\int x.\ u\ x\ \partial M))$
by (*rule integral_Markov_inequality*) (*use assms* **in** *auto*)
also have ... $< top$
by *auto*
finally have $ennreal\ (measure\ M\ \{x{\in}space\ M.\ u\ x \geq c\}) = emeasure\ M\ \{x{\in}space\ M.\ u\ x \geq c\}$

```
    by (intro emeasure_eq_ennreal_measure [symmetric]) auto
  also note le
  finally show ?thesis
    by (simp add: assms divide_nonneg_pos integral_nonneg_AE)
qed

theorem (in finite_measure) second_moment_method:
  assumes [measurable]: f ∈ M →_M borel
  assumes integrable M (λx. f x ^ 2)
  defines μ ≡ lebesgue_integral M f
  assumes a > 0
  shows measure M {x∈space M. |f x| ≥ a} ≤ lebesgue_integral M (λx. f x ^ 2)
/ a^2
proof −
  have integrable: integrable M f
    using assms by (blast dest: square_integrable_imp_integrable)
  have {x∈space M. |f x| ≥ a} = {x∈space M. f x ^ 2 ≥ a^2}
    using ‹a > 0› by (simp flip: abs_le_square_iff)
  hence measure M {x∈space M. |f x| ≥ a} = measure M {x∈space M. f x ^ 2 ≥
a^2}
    by simp
  also have ... ≤ lebesgue_integral M (λx. f x ^ 2) / a^2
    using assms by (intro integral_Markov_inequality_measure) auto
  finally show ?thesis .
qed

lemma integral_ineq_eq_0_then_AE:
  fixes f::_ ⇒ real
  assumes AE x in M. f x ≤ g x integrable M f integrable M g
        (∫ x. f x ∂M) = (∫ x. g x ∂M)
  shows AE x in M. f x = g x
proof −
  define h where h = (λx. g x − f x)
  have AE x in M. h x = 0
    apply (subst integral_nonneg_eq_0_iff_AE[symmetric])
    unfolding h_def using assms by auto
  then show ?thesis unfolding h_def by auto
qed

lemma not_AE_zero_int_E:
  fixes f::'a ⇒ real
  assumes AE x in M. f x ≥ 0 (∫ x. f x ∂M) > 0
     and [measurable]: f ∈ borel_measurable M
  shows ∃ A e. A ∈ sets M ∧ e>0 ∧ emeasure M A > 0 ∧ (∀ x ∈ A. f x ≥ e)
proof (rule not_AE_zero_E, auto simp: assms)
  assume ∗: AE x in M. f x = 0
  have (∫ x. f x ∂M) = (∫ x. 0 ∂M)
    by (rule integral_cong_AE, auto simp: ∗)
  then have (∫ x. f x ∂M) = 0 by simp
```

then show *False* **using** *assms*(*2*) **by** *simp*
qed

proposition *tendsto_L1_int*:
fixes *u* :: _ ⇒ _ ⇒ *'b*::{*banach*, *second_countable_topology*}
assumes [*measurable*]: ⋀*n*. *integrable M* (*u n*) *integrable M f*
and ((λ*n*. (∫⁺*x*. *norm*(*u n x* − *f x*) ∂*M*)) ⟶ *0*) *F*
shows ((λ*n*. (∫ *x*. *u n x* ∂*M*)) ⟶ (∫ *x*. *f x* ∂*M*)) *F*
proof −
have ((λ*n*. *norm*((∫ *x*. *u n x* ∂*M*) − (∫ *x*. *f x* ∂*M*))) ⟶ (*0*::*ennreal*)) *F*
proof (*rule tendsto_sandwich*[*of* λ_. *0*, **where** *?h* = λ*n*. (∫⁺*x*. *norm*(*u n x* − *f x*) ∂*M*)], *auto simp*: *assms*)
{
fix *n*
have (∫ *x*. *u n x* ∂*M*) − (∫ *x*. *f x* ∂*M*) = (∫ *x*. *u n x* − *f x* ∂*M*)
by (*simp add*: *assms*)
then have *norm*((∫ *x*. *u n x* ∂*M*) − (∫ *x*. *f x* ∂*M*)) = *norm* (∫ *x*. *u n x* − *f x* ∂*M*)
by *auto*
also have ... ≤ (∫ *x*. *norm*(*u n x* − *f x*) ∂*M*)
by (*rule integral_norm_bound*)
finally have *ennreal*(*norm*((∫ *x*. *u n x* ∂*M*) − (∫ *x*. *f x* ∂*M*))) ≤ (∫ *x*. *norm*(*u n x* − *f x*) ∂*M*)
by *simp*
also have ... = (∫⁺*x*. *norm*(*u n x* − *f x*) ∂*M*)
by (*simp add*: *assms nn_integral_eq_integral*)
finally have *norm*((∫ *x*. *u n x* ∂*M*) − (∫ *x*. *f x* ∂*M*)) ≤ (∫⁺*x*. *norm*(*u n x* − *f x*) ∂*M*)
by *simp*
}
then show *eventually* (λ*n*. *norm*((∫ *x*. *u n x* ∂*M*) − (∫ *x*. *f x* ∂*M*)) ≤ (∫⁺*x*. *norm*(*u n x* − *f x*) ∂*M*)) *F*
by *auto*
qed
then have ((λ*n*. *norm*((∫ *x*. *u n x* ∂*M*) − (∫ *x*. *f x* ∂*M*))) ⟶ *0*) *F*
by (*simp flip*: *ennreal_0*)
then have ((λ*n*. ((∫ *x*. *u n x* ∂*M*) − (∫ *x*. *f x* ∂*M*))) ⟶ *0*) *F*
using *tendsto_norm_zero_iff* **by** *blast*
then show *?thesis*
using *Lim_null* **by** *auto*
qed

The next lemma asserts that, if a sequence of functions converges in L^1, then it admits a subsequence that converges almost everywhere.

proposition *tendsto_L1_AE_subseq*:
fixes *u* :: *nat* ⇒ *'a* ⇒ *'b*::{*banach*, *second_countable_topology*}
assumes [*measurable*]: ⋀*n*. *integrable M* (*u n*)
and (λ*n*. (∫ *x*. *norm*(*u n x*) ∂*M*)) ⟶ *0*
shows ∃ *r*::*nat*⇒*nat*. *strict_mono r* ∧ (*AE x in M*. (λ*n*. *u* (*r n*) *x*) ⟶ *0*)

```
proof −
  {
    fix k
    have eventually (λn. (∫ x. norm(u n x) ∂M) < (1/2)^k) sequentially
      using order_tendstoD(2)[OF assms(2)] by auto
    with eventually_elim2[OF eventually_gt_at_top[of k] this]
    have ∃ n>k. (∫ x. norm(u n x) ∂M) < (1/2)^k
      by (metis eventually_False_sequentially)
  }
  then have ∃ r. ∀ n. True ∧ (r (Suc n) > r n ∧ (∫ x. norm(u (r (Suc n)) x) ∂M)
< (1/2)^(r n))
    by (intro dependent_nat_choice, auto)
  then obtain r0 where r0: strict_mono r0 ⋀n. (∫ x. norm(u (r0 (Suc n)) x)
∂M) < (1/2)^(r0 n)
    by (auto simp: strict_mono_Suc_iff)
  define r where r = (λn. r0(n+1))
 have strict_mono r unfolding r_def using r0(1) by (simp add: strict_mono_Suc_iff)
  have I: (∫⁺x. norm(u (r n) x) ∂M) < ennreal((1/2)^n) for n
  proof −
    have r0 n ≥ n using ‹strict_mono r0› by (simp add: seq_suble)
     have (1/2::real)^(r0 n) ≤ (1/2)^n by (rule power_decreasing[OF ‹r0 n ≥
n›], auto)
    then have (∫ x. norm(u (r0 (Suc n)) x) ∂M) < (1/2)^n
      using r0(2) less_le_trans by blast
    then have (∫ x. norm(u (r n) x) ∂M) < (1/2)^n
      unfolding r_def by auto
    moreover have (∫⁺x. norm(u (r n) x) ∂M) = (∫ x. norm(u (r n) x) ∂M)
     by (rule nn_integral_eq_integral, auto simp: integrable_norm[OF assms(1)[of
r n]])
    ultimately show ?thesis by (auto intro: ennreal_lessI)
  qed

  have AE x in M. limsup (λn. ennreal (norm(u (r n) x))) ≤ 0
  proof (rule AE_upper_bound_inf_ennreal)
    fix e::real assume e > 0
    define A where A = (λn. {x ∈ space M. norm(u (r n) x) ≥ e})
    have A_meas [measurable]: ⋀n. A n ∈ sets M unfolding A_def by auto
    have A_bound: emeasure M (A n) < (1/e) * ennreal((1/2)^n) for n
    proof −
      have ∗: indicator (A n) x ≤ (1/e) * ennreal(norm(u (r n) x)) for x
      using ‹0 < e› by (cases x ∈ A n) (auto simp: A_def ennreal_mult[symmetric])
      have emeasure M (A n) = (∫⁺x. indicator (A n) x ∂M) by auto
      also have ... ≤ (∫⁺x. (1/e) * ennreal(norm(u (r n) x)) ∂M)
        by (meson ∗ nn_integral_mono)
      also have ... = (1/e) * (∫⁺x. norm(u (r n) x) ∂M)
        using ‹e > 0› by (force simp add: intro: nn_integral_cmult)
      also have ... < (1/e) * ennreal((1/2)^n)
        using I[of n] ‹e > 0› by (intro ennreal_mult_strict_left_mono) auto
      finally show ?thesis by simp
```

```
    qed
    have A_fin: emeasure M (A n) < ∞ for n
      using ‹e > 0› A_bound[of n]
      by (auto simp: ennreal_mult less_top[symmetric])

    have A_sum: summable (λn. measure M (A n))
    proof (rule summable_comparison_test')
      have summable (λn. (1/(2::real))^n)
        by (simp add: summable_geometric)
     then show summable (λn. (1/e) * (1/2)^n) using summable_mult by blast
      fix n::nat assume n ≥ 0
      have norm(measure M (A n)) = measure M (A n) by simp
       also have ... = enn2real(emeasure M (A n)) unfolding measure_def by
simp
      also have ... < enn2real((1/e) * (1/2)^n)
        using A_bound[of n] ‹emeasure M (A n) < ∞› ‹0 < e›
          by (auto simp: emeasure_eq_ennreal_measure ennreal_mult[symmetric]
ennreal_less_iff)
      also have ... = (1/e) * (1/2)^n
        using ‹0<e› by auto
      finally show norm(measure M (A n)) ≤ (1/e) * (1/2)^n by simp
    qed

    have AE x in M. eventually (λn. x ∈ space M − A n) sequentially
      by (rule borel_cantelli_AE1[OF A_meas A_fin A_sum])
    moreover
    {
      fix x assume eventually (λn. x ∈ space M − A n) sequentially
      moreover have norm(u (r n) x) ≤ ennreal e if x ∈ space M − A n for n
        using that unfolding A_def by (auto intro: ennreal_leI)
      ultimately have eventually (λn. norm(u (r n) x) ≤ ennreal e) sequentially
        by (simp add: eventually_mono)
      then have limsup (λn. ennreal (norm(u (r n) x))) ≤ e
        by (simp add: Limsup_bounded)
    }
    ultimately show AE x in M. limsup (λn. ennreal (norm(u (r n) x))) ≤ 0 +
ennreal e
      by auto
  qed
  moreover
  {
    fix x assume x: limsup (λn. ennreal (norm(u (r n) x))) ≤ 0
    moreover have liminf (λn. ennreal (norm(u (r n) x))) ≤ 0
      by (metis x Liminf_le_Limsup le_zero_eq sequentially_bot)
    ultimately have (λn. ennreal (norm(u (r n) x))) ⟶ 0
     using tendsto_Limsup[of sequentially λn. ennreal (norm(u (r n) x))] by auto
    then have (λn. norm(u (r n) x)) ⟶ 0
      by (simp flip: ennreal_0)
    then have (λn. u (r n) x) ⟶ 0
```

```
    by (simp add: tendsto_norm_zero_iff)
  }
  ultimately have AE x in M. (λn. u (r n) x) ⟶ 0 by auto
  then show ?thesis using ‹strict_mono r› by auto
qed
```

6.9.1 Restricted measure spaces

```
lemma integrable_restrict_space:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}
  assumes Ω[simp]: Ω ∩ space M ∈ sets M
  shows integrable (restrict_space M Ω) f ⟷ integrable M (λx. indicator Ω x
*R f x)
  unfolding integrable_iff_bounded
    borel_measurable_restrict_space_iff[OF Ω]
    nn_integral_restrict_space[OF Ω]
  by (simp add: ac_simps ennreal_indicator ennreal_mult)

lemma integral_restrict_space:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}
  assumes Ω[simp]: Ω ∩ space M ∈ sets M
  shows integral^L (restrict_space M Ω) f = integral^L M (λx. indicator Ω x *R f
x)
proof (rule integral_eq_cases)
  assume integrable (restrict_space M Ω) f
  then show ?thesis
  proof induct
    case (base A c) then show ?case
      by (simp add: indicator_inter_arith[symmetric] sets_restrict_space_iff
                    emeasure_restrict_space Int_absorb1 measure_restrict_space)
  next
    case (add g f) then show ?case
      by (simp add: scaleR_add_right integrable_restrict_space)
  next
    case (lim f s)
    show ?case
    proof (rule LIMSEQ_unique)
    show (λi. integral^L (restrict_space M Ω) (s i)) ⟶ integral^L (restrict_space
M Ω) f
        using lim by (intro integral_dominated_convergence[where w=λx. 2 *
norm (f x)]) simp_all

      show (λi. integral^L (restrict_space M Ω) (s i)) ⟶ (∫ x. indicator Ω x
*R f x ∂M)
        unfolding lim
        using lim
      by (intro integral_dominated_convergence[where w=λx. 2 * norm (indicator
Ω x *R f x)])
             (auto simp: space_restrict_space integrable_restrict_space simp del:
```

norm_scaleR
split: *split_indicator*)
qed
qed
qed (*simp add*: *integrable_restrict_space*)

lemma *integral_empty*:
assumes *space M* = {}
shows *integral*^{L} *M f* = *0*
by (*metis AE_I2 assms empty_iff integral_eq_zero_AE*)

6.9.2 Measure spaces with an associated density

lemma *integrable_density*:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second_countable_topology*} **and** *g* :: *'a* ⇒ *real*
assumes [*measurable*]: *f* ∈ *borel_measurable M g* ∈ *borel_measurable M*
and *nn*: *AE x in M*. *0* ≤ *g x*
shows *integrable* (*density M g*) *f* ⟷ *integrable M* (λ*x*. *g x* ∗$_R$ *f x*)
unfolding *integrable_iff_bounded* **using** *nn*
apply (*simp add*: *nn_integral_density less_top*[*symmetric*])
apply (*intro arg_cong2*[**where** *f*=(=)] *refl nn_integral_cong_AE*)
apply (*auto simp*: *ennreal_mult*)
done

lemma *integral_density*:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second_countable_topology*} **and** *g* :: *'a* ⇒ *real*
assumes *f*: *f* ∈ *borel_measurable M*
and *g*[*measurable*]: *g* ∈ *borel_measurable M AE x in M*. *0* ≤ *g x*
shows *integral*^{L} (*density M g*) *f* = *integral*^{L} *M* (λ*x*. *g x* ∗$_R$ *f x*)
proof (*rule integral_eq_cases*)
assume *integrable* (*density M g*) *f*
then show *?thesis*
proof *induct*
case (*base A c*)
then have [*measurable*]: *A* ∈ *sets M* **by** *auto*

have *int*: *integrable M* (λ*x*. *g x* ∗ *indicator A x*)
using *g base integrable_density*[*of indicator A* :: *'a* ⇒ *real M g*] **by** *simp*
then have *integral*^{L} *M* (λ*x*. *g x* ∗ *indicator A x*) = (∫$^+$ *x*. *ennreal* (*g x* ∗ *indicator A x*) ∂*M*)
using *g* **by** (*subst nn_integral_eq_integral*) *auto*
also have ... = (∫$^+$ *x*. *ennreal* (*g x*) ∗ *indicator A x* ∂*M*)
by (*intro nn_integral_cong*) (*auto split*: *split_indicator*)
also have ... = *emeasure* (*density M g*) *A*
by (*rule emeasure_density*[*symmetric*]) *auto*
also have ... = *ennreal* (*measure* (*density M g*) *A*)
using *base* **by** (*auto intro*: *emeasure_eq_ennreal_measure*)
also have ... = *integral*^{L} (*density M g*) (*indicator A*)
using *base* **by** *simp*

```
    finally show ?case
      using base g
      apply (simp add: int integral_nonneg_AE)
      apply (subst (asm) ennreal_inj)
      apply (auto intro!: integral_nonneg_AE)
      done
  next
    case (add f h)
    then have [measurable]: f ∈ borel_measurable M h ∈ borel_measurable M
      by (auto dest!: borel_measurable_integrable)
    from add g show ?case
      by (simp add: scaleR_add_right integrable_density)
  next
    case (lim f s)
    have [measurable]: f ∈ borel_measurable M ⋀i. s i ∈ borel_measurable M
      using lim(1,5)[THEN borel_measurable_integrable] by auto

    show ?case
    proof (rule LIMSEQ_unique)
      show (λi. integral^L M (λx. g x *_R s i x)) ⟶ integral^L M (λx. g x *_R f
x)
      proof (rule integral_dominated_convergence)
        show integrable M (λx. 2 * norm (g x *_R f x))
         by (intro integrable_mult_right integrable_norm integrable_density[THEN
iffD1] lim g) auto
        show AE x in M. (λi. g x *_R s i x) ⟶ g x *_R f x
          using lim(3) by (auto intro!: tendsto_scaleR AE_I2[of M])
        show ⋀i. AE x in M. norm (g x *_R s i x) ≤ 2 * norm (g x *_R f x)
            using lim(4) g by (auto intro!: AE_I2[of M] mult_left_mono simp:
field_simps)
      qed auto
      show (λi. integral^L M (λx. g x *_R s i x)) ⟶ integral^L (density M g) f
        unfolding lim(2)[symmetric]
        by (rule integral_dominated_convergence[where w=λx. 2 * norm (f x)])
           (use lim in auto)
    qed
  qed
qed (simp add: f g integrable_density)

lemma
  fixes g :: 'a ⇒ real
  assumes f ∈ borel_measurable M AE x in M. 0 ≤ f x g ∈ borel_measurable M
  shows integral_real_density: integral^L (density M f) g = (∫ x. f x * g x ∂M)
    and integrable_real_density: integrable (density M f) g ⟷ integrable M (λx.
f x * g x)
  using assms integral_density[of g M f] integrable_density[of g M f] by auto

lemma has_bochner_integral_density:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology} and g :: 'a ⇒ real
```

shows $f \in$ *borel_measurable* $M \Longrightarrow g \in$ *borel_measurable* $M \Longrightarrow$ (*AE* x *in* M. $0 \leq g\ x) \Longrightarrow$
has_bochner_integral M ($\lambda x.\ g\ x *_R f\ x$) $x \Longrightarrow$ *has_bochner_integral* (*density* $M\ g$) $f\ x$
by (*simp add*: *has_bochner_integral_iff integrable_density integral_density*)

6.9.3 Distributions

lemma *integrable_distr_eq*:
fixes $f :: 'a \Rightarrow 'b$::{*banach*, *second_countable_topology*}
assumes [*measurable*]: $g \in$ *measurable* $M\ N\ f \in$ *borel_measurable* N
shows *integrable* (*distr* $M\ N\ g$) $f \longleftrightarrow$ *integrable* M ($\lambda x.\ f\ (g\ x)$)
unfolding *integrable_iff_bounded* **by** (*simp_all add*: *nn_integral_distr*)

lemma *integrable_distr*:
fixes $f :: 'a \Rightarrow 'b$::{*banach*, *second_countable_topology*}
shows $T \in$ *measurable* $M\ M' \Longrightarrow$ *integrable* (*distr* $M\ M'\ T$) $f \Longrightarrow$ *integrable* M ($\lambda x.\ f\ (T\ x)$)
by (*metis integrable_distr_eq integrable_iff_bounded measurable_distr_eq1*)

lemma *integral_distr*:
fixes $f :: 'a \Rightarrow 'b$::{*banach*, *second_countable_topology*}
assumes g[*measurable*]: $g \in$ *measurable* $M\ N$ **and** f: $f \in$ *borel_measurable* N
shows $integral^L$ (*distr* $M\ N\ g$) $f = integral^L\ M$ ($\lambda x.\ f\ (g\ x)$)
proof (*rule integral_eq_cases*)
assume *integrable* (*distr* $M\ N\ g$) f
then show *?thesis*
proof *induct*
case (*base* $A\ c$)
then have [*measurable*]: $A \in$ *sets* N **by** *auto*
from *base* **have** *int*: *integrable* (*distr* $M\ N\ g$) ($\lambda a.$ *indicator* $A\ a *_R c$)
by (*intro integrable_indicator*)

have $integral^L$ (*distr* $M\ N\ g$) ($\lambda a.$ *indicator* $A\ a *_R c$) = *measure* (*distr* $M\ N\ g$) $A *_R c$
using *base* **by** *auto*
also have ... = *measure* M ($g -` A \cap$ *space* M) $*_R$ c
by (*subst measure_distr*) *auto*
also have ... = $integral^L\ M$ ($\lambda a.$ *indicator* ($g -` A \cap$ *space* M) $a *_R c$)
using *base* **by** (*auto simp*: *emeasure_distr*)
also have ... = $integral^L\ M$ ($\lambda a.$ *indicator* A ($g\ a$) $*_R$ c)
using *int base* **by** (*intro integral_cong_AE*) (*auto simp*: *emeasure_distr split*: *split_indicator*)
finally show *?case* **.**
next
case (*add* $f\ h$)
then have [*measurable*]: $f \in$ *borel_measurable* $N\ h \in$ *borel_measurable* N
by (*auto dest*!: *borel_measurable_integrable*)
from *add* g **show** *?case*

```
    by (simp add: scaleR_add_right integrable_distr_eq)
  next
    case (lim f s)
    have [measurable]: f ∈ borel_measurable N ⋀i. s i ∈ borel_measurable N
      using lim(1,5)[THEN borel_measurable_integrable] by auto

    show ?case
    proof (rule LIMSEQ_unique)
      show (λi. integral^L M (λx. s i (g x))) ⟶ integral^L M (λx. f (g x))
      proof (rule integral_dominated_convergence)
        show integrable M (λx. 2 * norm (f (g x)))
          using lim by (auto simp: integrable_distr_eq)
        show AE x in M. (λi. s i (g x)) ⟶ f (g x)
          using lim(3) g[THEN measurable_space] by auto
        show ⋀i. AE x in M. norm (s i (g x)) ≤ 2 * norm (f (g x))
          using lim(4) g[THEN measurable_space] by auto
      qed auto
      show (λi. integral^L M (λx. s i (g x))) ⟶ integral^L (distr M N g) f
        unfolding lim(2)[symmetric]
        by (rule integral_dominated_convergence[where w=λx. 2 * norm (f x)])
           (use lim in auto)
    qed
  qed
qed (simp add: f g integrable_distr_eq)

lemma has_bochner_integral_distr:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}
  shows f ∈ borel_measurable N ⟹ g ∈ measurable M N ⟹
    has_bochner_integral M (λx. f (g x)) x ⟹ has_bochner_integral (distr M N
g) f x
  by (simp add: has_bochner_integral_iff integrable_distr_eq integral_distr)
```

6.9.4 Lebesgue integration on *count_space*

```
lemma integrable_count_space:
  fixes f :: 'a ⇒ 'b::{banach,second_countable_topology}
  shows finite X ⟹ integrable (count_space X) f
  by (auto simp: nn_integral_count_space integrable_iff_bounded)

lemma measure_count_space[simp]:
  B ⊆ A ⟹ finite B ⟹ measure (count_space A) B = card B
  unfolding measure_def by (subst emeasure_count_space ) auto

lemma lebesgue_integral_count_space_finite_support:
  assumes f: finite {a∈A. f a ≠ 0}
  shows (∫ x. f x ∂count_space A) = (∑ a | a ∈ A ∧ f a ≠ 0. f a)
proof −
  have eq: ⋀x. x ∈ A ⟹ (∑ a | x = a ∧ a ∈ A ∧ f a ≠ 0. f a) = (∑ x∈{x}. f x)
    by (intro sum.mono_neutral_cong_left) auto
```

have $(\int x.\ f\ x\ \partial count\_space\ A) = (\int x.\ (\sum a \mid a \in A \wedge f\ a \neq 0.\ indicator\ \{a\}\ x *_R f\ a)\ \partial count\_space\ A)$
by (*intro integral_cong refl*) (*simp add*: *f eq*)
also have $\ldots = (\sum a \mid a \in A \wedge f\ a \neq 0.\ measure\ (count\_space\ A)\ \{a\} *_R f\ a)$
by (*subst integral_sum*) (*auto intro*!: *sum.cong*)
finally show *?thesis*
by *auto*
qed

lemma *lebesgue_integral_count_space_finite*: *finite A* $\Longrightarrow (\int x.\ f\ x\ \partial count\_space\ A) = (\sum a{\in}A.\ f\ a)$
by (*subst lebesgue_integral_count_space_finite_support*)
(*auto intro*!: *sum.mono_neutral_cong_left*)

lemma *integrable_count_space_nat_iff*:
fixes *f* :: *nat* ⇒ _::{*banach*,*second_countable_topology*}
shows *integrable* (*count_space UNIV*) *f* ⟷ *summable* (λ*x*. *norm* (*f x*))
by (*auto simp*: *integrable_iff_bounded nn_integral_count_space_nat ennreal_suminf_neq_top*
intro: *summable_suminf_not_top*)

lemma *sums_integral_count_space_nat*:
fixes *f* :: *nat* ⇒ _::{*banach*,*second_countable_topology*}
assumes ∗: *integrable* (*count_space UNIV*) *f*
shows *f sums* (*integral*^{L} (*count_space UNIV*) *f*)
proof −
let *?f* = λ*n i*. *indicator* {*n*} *i* ∗$_R$ *f i*
have *f′*: ⋀*n i*. *?f n i* = *indicator* {*n*} *i* ∗$_R$ *f n*
by (*auto simp*: *fun_eq_iff split*: *split_indicator*)

have $(\lambda i.\ \int n.\ ?f\ i\ n\ \partial count\_space\ UNIV)\ sums\ \int\ n.\ (\sum i.\ ?f\ i\ n)\ \partial count\_space\ UNIV$
proof (*rule sums_integral*)
show ⋀*i*. *integrable* (*count_space UNIV*) (*?f i*)
using ∗ **by** (*intro integrable_mult_indicator*) *auto*
show *AE n in count_space UNIV*. *summable* (λ*i*. *norm* (*?f i n*))
using *summable_finite*[*of* {*n*} λ*i*. *norm* (*?f i n*) **for** *n*] **by** *simp*
show *summable* (λ*i*. ∫ *n*. *norm* (*?f i n*) *∂count_space UNIV*)
using ∗ **by** (*subst f′*) (*simp add*: *integrable_count_space_nat_iff*)
qed
also have $(\int\ n.\ (\sum i.\ ?f\ i\ n)\ \partial count\_space\ UNIV) = (\int n.\ f\ n\ \partial count\_space\ UNIV)$
using *suminf_finite*[*of* {*n*} λ*i*. *?f i n* **for** *n*] **by** (*auto intro*!: *integral_cong*)
also have (λ*i*. ∫ *n*. *?f i n ∂count_space UNIV*) = *f*
by (*subst f′*) *simp*
finally show *?thesis* **.**
qed

lemma *integral_count_space_nat*:

fixes $f :: nat \Rightarrow \_::\{banach,second\_countable\_topology\}$
shows $integrable\ (count\_space\ UNIV)\ f \Longrightarrow integral^L\ (count\_space\ UNIV)\ f = (\sum x.\ f\ x)$
using *sums_integral_count_space_nat sums_unique* **by** *auto*

lemma *integrable_bij_count_space*:
fixes $f :: 'a \Rightarrow 'b::\{banach,\ second\_countable\_topology\}$
assumes *g*: *bij_betw g A B*
shows $integrable\ (count\_space\ A)\ (\lambda x.\ f\ (g\ x)) \longleftrightarrow integrable\ (count\_space\ B)\ f$
unfolding *integrable_iff_bounded* **by** (*subst nn_integral_bij_count_space*[*OF g*]) *auto*

lemma *integral_bij_count_space*:
fixes $f :: 'a \Rightarrow 'b::\{banach,\ second\_countable\_topology\}$
assumes *g*: *bij_betw g A B*
shows $integral^L\ (count\_space\ A)\ (\lambda x.\ f\ (g\ x)) = integral^L\ (count\_space\ B)\ f$
using *g*[*THEN bij_betw_imp_funcset*] *distr_bij_count_space* [*OF g*]
by (*metis borel_measurable_count_space integral_distr measurable_count_space_eq1 space_count_space*)

lemma *has_bochner_integral_count_space_nat*:
fixes $f :: nat \Rightarrow \_::\{banach,second\_countable\_topology\}$
shows $has\_bochner\_integral\ (count\_space\ UNIV)\ f\ x \Longrightarrow f\ sums\ x$
unfolding *has_bochner_integral_iff* **by** (*auto intro*!: *sums_integral_count_space_nat*)

6.9.5 Point measure

lemma *lebesgue_integral_point_measure_finite*:
fixes $g :: 'a \Rightarrow 'b::\{banach,\ second\_countable\_topology\}$
shows $finite\ A \Longrightarrow (\bigwedge a.\ a \in A \Longrightarrow 0 \le f\ a) \Longrightarrow$
$integral^L\ (point\_measure\ A\ f)\ g = (\sum a{\in}A.\ f\ a *_R\ g\ a)$
by (*simp add*: *lebesgue_integral_count_space_finite AE_count_space integral_density point_measure_def*)

proposition *integrable_point_measure_finite*:
fixes $g :: 'a \Rightarrow 'b::\{banach,\ second\_countable\_topology\}$ **and** $f :: 'a \Rightarrow real$
assumes *finite A*
shows $integrable\ (point\_measure\ A\ f)\ g$
proof −
have $integrable\ (density\ (count\_space\ A)\ (\lambda x.\ ennreal\ (max\ 0\ (f\ x))))\ g$
using *assms*
by (*simp add*: *integrable_count_space integrable_density*)
then show *?thesis*
by (*smt* (*verit*) *AE_I2 borel_measurable_count_space density_cong ennreal_neg point_measure_def*)
qed

6.9.6 Lebesgue integration on *null_measure*

lemma *has_bochner_integral_null_measure_iff*[*iff*]:
has_bochner_integral (*null_measure M*) *f* 0 ⟷ *f* ∈ *borel_measurable M*
by (*auto simp*: *has_bochner_integral.simps simple_bochner_integral_def*[*abs_def*]
intro!: *exI*[*of* _ λ*n x*. 0] *simple_bochner_integrable.intros*)

lemma *integrable_null_measure_iff*[*iff*]: *integrable* (*null_measure M*) *f* ⟷ *f* ∈ *borel_measurable M*
by (*auto simp*: *integrable.simps*)

lemma *integral_null_measure*[*simp*]: *integral*^{L} (*null_measure M*) *f* = 0
using *integral_norm_bound_ennreal not_integrable_integral_eq* **by** *fastforce*

6.9.7 Legacy lemmas for the real-valued Lebesgue integral

theorem *real_lebesgue_integral_def*:
assumes *f*[*measurable*]: *integrable M f*
shows *integral*^{L} *M f* = *enn2real* (∫$^+$*x*. *f x* ∂*M*) − *enn2real* (∫$^+$*x*. *ennreal* (− *f x*) ∂*M*)
proof −
have *integral*^{L} *M f* = *integral*^{L} *M* (λ*x*. *max* 0 (*f x*) − *max* 0 (− *f x*))
by (*auto intro*!: *arg_cong*[**where** *f*=*integral*^{L} *M*])
also have ... = *integral*^{L} *M* (λ*x*. *max* 0 (*f x*)) − *integral*^{L} *M* (λ*x*. *max* 0 (− *f x*))
by (*intro integral_diff integrable_max integrable_minus integrable_zero f*)
also have *integral*^{L} *M* (λ*x*. *max* 0 (*f x*)) = *enn2real* (∫$^+$*x*. *ennreal* (*f x*) ∂*M*)
by (*subst integral_eq_nn_integral*) (*auto intro*!: *arg_cong*[**where** *f*=*enn2real*] *nn_integral_cong simp*: *max_def ennreal_neg*)
also have *integral*^{L} *M* (λ*x*. *max* 0 (− *f x*)) = *enn2real* (∫$^+$*x*. *ennreal* (− *f x*) ∂*M*)
by (*subst integral_eq_nn_integral*) (*auto intro*!: *arg_cong*[**where** *f*=*enn2real*] *nn_integral_cong simp*: *max_def ennreal_neg*)
finally show *?thesis* **.**
qed

theorem *real_integrable_def*:
integrable M f ⟷ *f* ∈ *borel_measurable M* ∧
(∫$^+$ *x*. *ennreal* (*f x*) ∂*M*) ≠ ∞ ∧ (∫$^+$ *x*. *ennreal* (− *f x*) ∂*M*) ≠ ∞
unfolding *integrable_iff_bounded*
proof (*safe del*: *notI*)
assume ∗: (∫$^+$ *x*. *ennreal* (*norm* (*f x*)) ∂*M*) < ∞
have (∫$^+$ *x*. *ennreal* (*f x*) ∂*M*) ≤ (∫$^+$ *x*. *ennreal* (*norm* (*f x*)) ∂*M*)
by (*intro nn_integral_mono*) *auto*
also note ∗
finally show (∫$^+$ *x*. *ennreal* (*f x*) ∂*M*) ≠ ∞
by *simp*
have (∫$^+$ *x*. *ennreal* (− *f x*) ∂*M*) ≤ (∫$^+$ *x*. *ennreal* (*norm* (*f x*)) ∂*M*)
by (*intro nn_integral_mono*) *auto*
also note ∗

finally show $(\int^+ x.\ ennreal\ (-\ f\ x)\ \partial M) \neq \infty$
by *simp*
next
assume [*measurable*]: $f \in borel\_measurable\ M$
assume *fin*: $(\int^+ x.\ ennreal\ (f\ x)\ \partial M) \neq \infty$ $(\int^+ x.\ ennreal\ (-\ f\ x)\ \partial M) \neq \infty$
have $(\int^+ x.\ norm\ (f\ x)\ \partial M) = (\int^+ x.\ ennreal\ (f\ x) + ennreal\ (-\ f\ x)\ \partial M)$
by (*intro nn_integral_cong*) (*auto simp*: *abs_real_def ennreal_neg*)
also have... $= (\int^+ x.\ ennreal\ (f\ x)\ \partial M) + (\int^+ x.\ ennreal\ (-\ f\ x)\ \partial M)$
by (*intro nn_integral_add*) *auto*
also have ... $< \infty$
using *fin* **by** (*auto simp*: *less_top*)
finally show $(\int^+ x.\ norm\ (f\ x)\ \partial M) < \infty$ **.**
qed

lemma *integrableD*[*dest*]:
assumes *integrable M f*
shows $f \in borel\_measurable\ M$ $(\int^+ x.\ ennreal\ (f\ x)\ \partial M) \neq \infty$ $(\int^+ x.\ ennreal\ (-\ f\ x)\ \partial M) \neq \infty$
using *assms* **unfolding** *real_integrable_def* **by** *auto*

lemma *integrableE*:
assumes *integrable M f*
obtains *r q* **where** $0 \leq r$ $0 \leq q$
$(\int^+x.\ ennreal\ (f\ x)\partial M) = ennreal\ r$
$(\int^+x.\ ennreal\ (-f\ x)\partial M) = ennreal\ q$
$f \in borel\_measurable\ M$ $integral^L\ M\ f = r - q$
using *assms* **unfolding** *real_integrable_def real_lebesgue_integral_def*[*OF assms*]
by (*cases rule*: *ennreal2_cases*[*of* $(\int^+x.\ ennreal\ (-f\ x)\partial M)$ $(\int^+x.\ ennreal\ (f\ x)\partial M)$]) *auto*

lemma *integral_monotone_convergence_nonneg*:
fixes *f* :: $nat \Rightarrow 'a \Rightarrow real$
assumes *i*: $\bigwedge i.\ integrable\ M\ (f\ i)$ **and** *mono*: $AE\ x\ in\ M.\ mono\ (\lambda n.\ f\ n\ x)$
and *pos*: $\bigwedge i.\ AE\ x\ in\ M.\ 0 \leq f\ i\ x$
and *lim*: $AE\ x\ in\ M.\ (\lambda i.\ f\ i\ x) \longrightarrow u\ x$
and *ilim*: $(\lambda i.\ integral^L\ M\ (f\ i)) \longrightarrow x$
and *u*: $u \in borel\_measurable\ M$
shows *integrable M u*
and $integral^L\ M\ u = x$
proof −
have *nn*: $AE\ x\ in\ M.\ \forall i.\ 0 \leq f\ i\ x$
using *pos* **unfolding** *AE_all_countable* **by** *auto*
with *lim* **have** *u_nn*: $AE\ x\ in\ M.\ 0 \leq u\ x$
by *eventually_elim* (*auto intro*: *LIMSEQ_le_const*)
have [*simp*]: $0 \leq x$
by (*intro LIMSEQ_le_const*[*OF ilim*] *allI exI impI integral_nonneg_AE pos*)
have $(\int^+ x.\ ennreal\ (u\ x)\ \partial M) = (SUP\ n.\ (\int^+ x.\ ennreal\ (f\ n\ x)\ \partial M))$
proof (*subst nn_integral_monotone_convergence_SUP_AE*[*symmetric*])
fix *i*

```
    from mono nn show AE x in M. ennreal (f i x) ≤ ennreal (f (Suc i) x)
      by eventually_elim (auto simp: mono_def)
    show (λx. ennreal (f i x)) ∈ borel_measurable M
      using i by auto
  next
    show (∫⁺ x. ennreal (u x) ∂M) = ∫⁺ x. (SUP i. ennreal (f i x)) ∂M
    proof (rule nn_integral_cong_AE)
      show AE x in M. ennreal (u x) = (SUP i. ennreal (f i x))
        using lim mono nn u_nn
          by eventually_elim (simp add: LIMSEQ_unique[OF _ LIMSEQ_SUP]
incseq_def)
    qed
  qed
  also have ... = ennreal x
    using mono i nn unfolding nn_integral_eq_integral[OF i pos]
     by (subst LIMSEQ_unique[OF LIMSEQ_SUP]) (auto simp: mono_def inte-
gral_nonneg_AE pos intro!: integral_mono_AE ilim)
  finally have (∫⁺ x. ennreal (u x) ∂M) = ennreal x .
  moreover have (∫⁺ x. ennreal (− u x) ∂M) = 0
    using u u_nn by (subst nn_integral_0_iff_AE) (auto simp: ennreal_neg)
  ultimately show integrable M u integral^L M u = x
    by (auto simp: real_integrable_def real_lebesgue_integral_def u)
qed

lemma
  fixes f :: nat ⇒ 'a ⇒ real
  assumes f: ⋀i. integrable M (f i) and mono: AE x in M. mono (λn. f n x)
  and lim: AE x in M. (λi. f i x) ⟶ u x
  and ilim: (λi. integral^L M (f i)) ⟶ x
  and u: u ∈ borel_measurable M
  shows integrable_monotone_convergence: integrable M u
    and integral_monotone_convergence: integral^L M u = x
    and has_bochner_integral_monotone_convergence: has_bochner_integral M u
x
proof −
  have 1: ⋀i. integrable M (λx. f i x − f 0 x)
    using f by auto
  have 2: AE x in M. mono (λn. f n x − f 0 x)
    using mono by (auto simp: mono_def le_fun_def)
  have 3: ⋀n. AE x in M. 0 ≤ f n x − f 0 x
    using mono by (auto simp: field_simps mono_def le_fun_def)
  have 4: AE x in M. (λi. f i x − f 0 x) ⟶ u x − f 0 x
    using lim by (auto intro!: tendsto_diff)
  have 5: (λi. (∫ x. f i x − f 0 x ∂M)) ⟶ x − integral^L M (f 0)
    using f ilim by (auto intro!: tendsto_diff)
  have 6: (λx. u x − f 0 x) ∈ borel_measurable M
    using f[of 0] u by auto
  note diff = integral_monotone_convergence_nonneg[OF 1 2 3 4 5 6]
  have integrable M (λx. (u x − f 0 x) + f 0 x)
```

```
    using diff(1) f by (rule integrable_add)
  with diff(2) f show integrable M u integral^L M u = x
    by auto
  then show has_bochner_integral M u x
    by (metis has_bochner_integral_integrable)
qed

lemma integral_norm_eq_0_iff:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}
  assumes f[measurable]: integrable M f
  shows (∫ x. norm (f x) ∂M) = 0 ⟷ emeasure M {x∈space M. f x ≠ 0} = 0
proof −
  have (∫⁺x. norm (f x) ∂M) = (∫ x. norm (f x) ∂M)
    using f by (intro nn_integral_eq_integral integrable_norm) auto
  then have (∫ x. norm (f x) ∂M) = 0 ⟷ (∫⁺x. norm (f x) ∂M) = 0
    by simp
  also have ... ⟷ emeasure M {x∈space M. ennreal (norm (f x)) ≠ 0} = 0
    by (intro nn_integral_0_iff) auto
  finally show ?thesis
    by simp
qed

lemma integral_0_iff:
  fixes f :: 'a ⇒ real
  shows integrable M f ⟹ (∫ x. |f x| ∂M) = 0 ⟷ emeasure M {x∈space M. f
x ≠ 0} = 0
  using integral_norm_eq_0_iff[of M f] by simp

lemma (in finite_measure) integrable_const[intro!, simp]: integrable M (λx. a)
  using integrable_indicator[of space M M a] by (simp cong: integrable_cong add:
less_top[symmetric])

lemma lebesgue_integral_const[simp]:
  fixes a :: 'a :: {banach, second_countable_topology}
  shows (∫ x. a ∂M) = measure M (space M) *R a
proof −
  { assume emeasure M (space M) = ∞ a ≠ 0
    then have ?thesis
        by (auto simp: not_integrable_integral_eq ennreal_mult_less_top mea-
sure_def integrable_iff_bounded) }
  moreover
  { assume a = 0 then have ?thesis by simp }
  moreover
  { assume emeasure M (space M) ≠ ∞
    interpret finite_measure M
      proof qed fact
    have (∫ x. a ∂M) = (∫ x. indicator (space M) x *R a ∂M)
      by (intro integral_cong) auto
    also have ... = measure M (space M) *R a
```

```
    by (simp add: less_top[symmetric])
  finally have ?thesis . }
 ultimately show ?thesis by blast
qed

lemma (in finite_measure) integrable_const_bound:
 fixes f :: 'a ⇒ 'b::{banach,second_countable_topology}
 shows AE x in M. norm (f x) ≤ B ⟹ f ∈ borel_measurable M ⟹ integrable
M f
 using integrable_bound[OF integrable_const[of B], of f]
 by (smt (verit, ccfv_SIG) eventually_elim2 real_norm_def)

lemma (in finite_measure) integral_bounded_eq_bound_then_AE:
 assumes AE x in M. f x ≤ (c::real)
   integrable M f (∫ x. f x ∂M) = c * measure M (space M)
 shows AE x in M. f x = c
 using assms  by (intro integral_ineq_eq_0_then_AE) auto

lemma integral_indicator_finite_real:
 fixes f :: 'a ⇒ real
 assumes [simp]: finite A
 assumes [measurable]: ⋀a. a ∈ A ⟹ {a} ∈ sets M
 assumes finite: ⋀a. a ∈ A ⟹ emeasure M {a} < ∞
 shows (∫ x. f x * indicator A x ∂M) = (∑ a∈A. f a * measure M {a})
proof −
 have (∫ x. f x * indicator A x ∂M) = (∫ x. (∑ a∈A. f a * indicator {a} x) ∂M)
 proof (intro integral_cong refl)
   fix x show f x * indicator A x = (∑ a∈A. f a * indicator {a} x)
     by (auto split: split_indicator simp: eq_commute[of x] cong: conj_cong)
 qed
 also have ... = (∑ a∈A. f a * measure M {a})
   using finite by (subst integral_sum) (auto)
 finally show ?thesis .
qed

lemma (in finite_measure) ennreal_integral_real:
 assumes [measurable]: f ∈ borel_measurable M
 assumes ae: AE x in M. f x ≤ ennreal B 0 ≤ B
 shows ennreal (∫ x. enn2real (f x) ∂M) = (∫⁺x. f x ∂M)
proof (subst nn_integral_eq_integral[symmetric])
 show integrable M (λx. enn2real (f x))
  using ae by (intro integrable_const_bound[where B=B]) (auto simp: enn2real_leI)
 show (∫⁺ x. ennreal (enn2real (f x)) ∂M) = integral^N M f
   using ae by (intro nn_integral_cong_AE) (auto simp: le_less_trans[OF _
ennreal_less_top])
qed auto

lemma (in finite_measure) integral_less_AE:
 fixes X Y :: 'a ⇒ real
```

assumes *int*: *integrable M X integrable M Y*
assumes *A*: (*emeasure M*) *A* ≠ *0 A* ∈ *sets M AE x in M*. *x* ∈ *A* ⟶ *X x* ≠ *Y x*
assumes *gt*: *AE x in M*. *X x* ≤ *Y x*
shows *integral*^{L} *M X* < *integral*^{L} *M Y*
proof −
have *integral*^{L} *M X* ≤ *integral*^{L} *M Y*
using *gt int* **by** (*intro integral_mono_AE*) *auto*
moreover
have *integral*^{L} *M X* ≠ *integral*^{L} *M Y*
proof
assume *eq*: *integral*^{L} *M X* = *integral*^{L} *M Y*
have *integral*^{L} *M* (λ*x*. |*Y x* − *X x*|) = *integral*^{L} *M* (λ*x*. *Y x* − *X x*)
using *gt int* **by** (*intro integral_cong_AE*) *auto*
also have ... = *0*
using *eq int* **by** *simp*
finally have (*emeasure M*) {*x* ∈ *space M*. *Y x* − *X x* ≠ *0*} = *0*
using *int* **by** (*simp add*: *integral_0_iff*)
moreover
have (∫$^+$*x*. *indicator A x* ∂*M*) ≤ (∫$^+$*x*. *indicator* {*x* ∈ *space M*. *Y x* − *X x* ≠ *0*} *x* ∂*M*)
using *A* **by** (*intro nn_integral_mono_AE*) *auto*
then have (*emeasure M*) *A* ≤ (*emeasure M*) {*x* ∈ *space M*. *Y x* − *X x* ≠ *0*}
using *int A* **by** (*simp add*: *integrable_def*)
ultimately have *emeasure M A* = *0*
by *simp*
with ‹(*emeasure M*) *A* ≠ *0*› **show** *False* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma (**in** *finite_measure*) *integral_less_AE_space*:
fixes *X Y* :: *'a* ⇒ *real*
assumes *int*: *integrable M X integrable M Y*
assumes *gt*: *AE x in M*. *X x* < *Y x emeasure M* (*space M*) ≠ *0*
shows *integral*^{L} *M X* < *integral*^{L} *M Y*
using *gt* **by** (*intro integral_less_AE*[*OF int*, **where** *A*=*space M*]) *auto*

lemma *tendsto_integral_at_top*:
fixes *f* :: *real* ⇒ *'a*::{*banach*, *second_countable_topology*}
assumes [*measurable_cong*]: *sets M* = *sets borel* **and** *f*[*measurable*]: *integrable M f*
shows ((λ*y*. ∫ *x*. *indicator* {.. *y*} *x* ∗$_R$ *f x* ∂*M*) ⟶ ∫ *x*. *f x* ∂*M*) *at_top*
proof (*rule tendsto_at_topI_sequentially*)
fix *X* :: *nat* ⇒ *real* **assume** *filterlim X at_top sequentially*
show (λ*n*. ∫ *x*. *indicator* {..*X n*} *x* ∗$_R$ *f x* ∂*M*) ⟶ *integral*^{L} *M f*
proof (*rule integral_dominated_convergence*)
show *integrable M* (λ*x*. *norm* (*f x*))
by (*rule integrable_norm*) *fact*
show *AE x in M*. (λ*n*. *indicator* {..*X n*} *x* ∗$_R$ *f x*) ⟶ *f x*

```
    proof
      fix x
      from ‹filterlim X at_top sequentially›
      have eventually (λn. x ≤ X n) sequentially
        unfolding filterlim_at_top_ge[where c=x] by auto
      then show (λn. indicator {..X n} x *R f x) ⟶ f x
          by (intro tendsto_eventually) (auto split: split_indicator elim!: eventu-
ally_mono)
    qed
    fix n show AE x in M. norm (indicator {..X n} x *R f x) ≤ norm (f x)
      by (auto split: split_indicator)
  qed auto
qed

lemma
  fixes f :: real ⇒ real
  assumes M: sets M = sets borel
  assumes nonneg: AE x in M. 0 ≤ f x
  assumes borel: f ∈ borel_measurable borel
  assumes int: ⋀y. integrable M (λx. f x * indicator {.. y} x)
  assumes conv: ((λy. ∫ x. f x * indicator {.. y} x ∂M) ⟶ x) at_top
 shows has_bochner_integral_monotone_convergence_at_top: has_bochner_integral
M f x
    and integrable_monotone_convergence_at_top: integrable M f
    and integral_monotone_convergence_at_top:integral^L M f = x
proof −
  from nonneg have AE x in M. mono (λn::nat. f x * indicator {..real n} x)
    by (auto split: split_indicator intro!: monoI)
  { fix x have eventually (λn. f x * indicator {..real n} x = f x) sequentially
      by (rule eventually_sequentiallyI[of nat ⌈x⌉])
        (auto split: split_indicator simp: nat_le_iff ceiling_le_iff) }
  from filterlim_cong[OF refl refl this]
  have AE x in M. (λi. f x * indicator {..real i} x) ⟶ f x
    by simp
  have (λi. ∫ x. f x * indicator {..real i} x ∂M) ⟶ x
    using conv filterlim_real_sequentially by (rule filterlim_compose)
  have M_measure[simp]: borel_measurable M = borel_measurable borel
    using M by (simp add: sets_eq_imp_space_eq measurable_def)
  have f ∈ borel_measurable M
    using borel by simp
  show has_bochner_integral M f x
    by (rule has_bochner_integral_monotone_convergence) fact+
  then show integrable M f integral^L M f = x
    by (auto simp: _has_bochner_integral_iff)
qed
```

6.9.8 Product measure

lemma (**in** *sigma_finite_measure*) *borel_measurable_lebesgue_integral*[*measurable* (*raw*)]:
fixes $f :: \_ \Rightarrow \_ \Rightarrow \_::\{banach, second\_countable\_topology\}$
assumes [*measurable*]: *case_prod f* ∈ *borel_measurable* ($N \bigotimes_M M$)
shows *Measurable.pred N* (λ*x*. *integrable M* (*f x*))
proof −
have [*simp*]: $\bigwedge x.\ x \in space\ N \Longrightarrow integrable\ M\ (f\ x) \longleftrightarrow (\int^+ y.\ norm\ (f\ x\ y)\ \partial M) < \infty$
unfolding *integrable_iff_bounded* **by** *simp*
show *?thesis*
by (*simp cong*: *measurable_cong*)
qed

lemma (**in** *sigma_finite_measure*) *measurable_measure*[*measurable* (*raw*)]:
$(\bigwedge x.\ x \in space\ N \Longrightarrow A\ x \subseteq space\ M) \Longrightarrow$
$\{x \in space\ (N \bigotimes_M M).\ snd\ x \in A\ (fst\ x)\} \in sets\ (N \bigotimes_M M) \Longrightarrow$
(λ*x*. *measure M* (*A x*)) ∈ *borel_measurable N*
unfolding *measure_def* **by** (*intro measurable_emeasure borel_measurable_enn2real*) *auto*

proposition (**in** *sigma_finite_measure*) *borel_measurable_lebesgue_integral*[*measurable* (*raw*)]:
fixes $f :: \_ \Rightarrow \_ \Rightarrow \_::\{banach, second\_countable\_topology\}$
assumes *f*[*measurable*]: *case_prod f* ∈ *borel_measurable* ($N \bigotimes_M M$)
shows (λ*x*. ∫ *y*. *f x y* ∂*M*) ∈ *borel_measurable N*
proof −
from *borel_measurable_implies_sequence_metric*[*OF f*, *of 0*]
obtain *s* **where** *s*: $\bigwedge i.\ simple\_function\ (N \bigotimes_M M)\ (s\ i)$
and ∀ *x*∈*space* ($N \bigotimes_M M$). (λ*i*. *s i x*) ⟶ (*case x of* (*x*, *y*) ⇒ *f x y*)
and ∀ *i*. ∀ *x*∈*space* ($N \bigotimes_M M$). *dist* (*s i x*) *0* ≤ *2* ∗ *dist* (*case x of* (*x*, *xa*) ⇒ *f x xa*) *0*
by *auto*
then have ∗:
$\bigwedge x\ y.\ x \in space\ N \Longrightarrow y \in space\ M \Longrightarrow (\lambda i.\ s\ i\ (x, y)) \longrightarrow f\ x\ y$
$\bigwedge i\ x\ y.\ x \in space\ N \Longrightarrow y \in space\ M \Longrightarrow norm\ (s\ i\ (x, y)) \leq 2 * norm\ (f\ x\ y)$
by (*auto simp*: *space_pair_measure*)

have [*measurable*]: $\bigwedge i.\ s\ i \in borel\_measurable\ (N \bigotimes_M M)$
by (*rule borel_measurable_simple_function*) *fact*

have $\bigwedge i.\ s\ i \in measurable\ (N \bigotimes_M M)\ (count\_space\ UNIV)$
by (*rule measurable_simple_function*) *fact*

define *f′* **where** [*abs_def*]: *f′ i x* =
(*if integrable M* (*f x*) *then simple_bochner_integral M* (λ*y*. *s i* (*x*, *y*)) *else 0*)
for *i x*

{ fix *i x* **assume** *x* ∈ *space N*

```
    then have simple_bochner_integral M (λy. s i (x, y)) =
      (∑z∈s i ‘ (space N × space M). measure M {y ∈ space M. s i (x, y) = z}
*R z)
      using s[THEN simple_functionD(1)]
      unfolding simple_bochner_integral_def
      by (intro sum.mono_neutral_cong_left)
         (auto simp: eq_commute space_pair_measure image_iff cong: conj_cong)
}
  note eq = this

  show ?thesis
  proof (rule borel_measurable_LIMSEQ_metric)
    fix i show f' i ∈ borel_measurable N
      unfolding f'_def by (simp_all add: eq cong: measurable_cong if_cong)
  next
    fix x assume x: x ∈ space N
    { assume int_f: integrable M (f x)
      have int_2f: integrable M (λy. 2 * norm (f x y))
        by (intro integrable_norm integrable_mult_right int_f)
      have (λi. integral^L M (λy. s i (x, y))) ⟶ integral^L M (f x)
      proof (rule integral_dominated_convergence)
        from int_f show f x ∈ borel_measurable M by auto
        show ⋀i. (λy. s i (x, y)) ∈ borel_measurable M
          using x by simp
        show AE xa in M. (λi. s i (x, xa)) ⟶ f x xa
          using x * by auto
        show ⋀i. AE xa in M. norm (s i (x, xa)) ≤ 2 * norm (f x xa)
          using x * by auto
      qed fact
      moreover
      { fix i
        have simple_bochner_integrable M (λy. s i (x, y))
        proof (rule simple_bochner_integrableI_bounded)
          have (λy. s i (x, y)) ‘ space M ⊆ s i ‘ (space N × space M)
            using x by auto
          then show simple_function M (λy. s i (x, y))
            using simple_functionD(1)[OF s(1), of i] x
            by (intro simple_function_borel_measurable)
               (auto simp: space_pair_measure dest: finite_subset)
          have (∫+ y. ennreal (norm (s i (x, y))) ∂M) ≤ (∫+ y. 2 * norm (f x y)
∂M)
            using x * by (intro nn_integral_mono) auto
          also have (∫+ y. 2 * norm (f x y) ∂M) < ∞
            using int_2f unfolding integrable_iff_bounded by simp
          finally show (∫+ xa. ennreal (norm (s i (x, xa))) ∂M) < ∞ .
        qed
        then have integral^L M (λy. s i (x, y)) = simple_bochner_integral M (λy.
s i (x, y))
          by (rule simple_bochner_integrable_eq_integral[symmetric]) }
```

ultimately have (λi. *simple_bochner_integral M* (λy. *s i* (*x*, *y*))) ⟶ *integral*^{L} *M* (*f x*)
by *simp* **}**
then
show (λi. *f' i x*) ⟶ *integral*^{L} *M* (*f x*)
unfolding *f'_def*
by (*cases integrable M* (*f x*)) (*simp_all add*: *not_integrable_integral_eq*)
qed
qed

lemma (**in** *pair_sigma_finite*) *integrable_product_swap*:
fixes *f* :: _ ⇒ _::{*banach*, *second_countable_topology*}
assumes *integrable* (*M1* ⨂$_M$ *M2*) *f*
shows *integrable* (*M2* ⨂$_M$ *M1*) (λ(*x*,*y*). *f* (*y*,*x*))
by (*smt* (*verit*) *assms distr_pair_swap integrable_cong integrable_distr measurable_pair_swap' prod.case_distrib split_cong*)

lemma (**in** *pair_sigma_finite*) *integrable_product_swap_iff*:
fixes *f* :: _ ⇒ _::{*banach*, *second_countable_topology*}
shows *integrable* (*M2* ⨂$_M$ *M1*) (λ(*x*,*y*). *f* (*y*,*x*)) ⟷ *integrable* (*M1* ⨂$_M$ *M2*) *f*
proof −
interpret *Q*: *pair_sigma_finite M2 M1* **..**
from *Q.integrable_product_swap*[*of* λ(*x*,*y*). *f* (*y*,*x*)] *integrable_product_swap*[*of f*]
show *?thesis* **by** *auto*
qed

lemma (**in** *pair_sigma_finite*) *integral_product_swap*:
fixes *f* :: _ ⇒ _::{*banach*, *second_countable_topology*}
assumes *f*: *f* ∈ *borel_measurable* (*M1* ⨂$_M$ *M2*)
shows ($\int$(*x*,*y*). *f* (*y*,*x*) ∂(*M2* ⨂$_M$ *M1*)) = *integral*^{L} (*M1* ⨂$_M$ *M2*) *f*
by (*smt* (*verit*) *distr_pair_swap f integral_cong integral_distr measurable_pair_swap' prod.case_distrib split_cong*)

theorem (**in** *pair_sigma_finite*) *Fubini_integrable*:
fixes *f* :: _ ⇒ _::{*banach*, *second_countable_topology*}
assumes *f*[*measurable*]: *f* ∈ *borel_measurable* (*M1* ⨂$_M$ *M2*)
and *integ1*: *integrable M1* (λx. $\int$ *y*. *norm* (*f* (*x*, *y*)) ∂*M2*)
and *integ2*: *AE x in M1*. *integrable M2* (λy. *f* (*x*, *y*))
shows *integrable* (*M1* ⨂$_M$ *M2*) *f*
proof (*rule integrableI_bounded*)
have ($\int^+$ *p*. *norm* (*f p*) ∂(*M1* ⨂$_M$ *M2*)) = ($\int^+$ *x*. ($\int^+$ *y*. *norm* (*f* (*x*, *y*)) ∂*M2*) ∂*M1*)
by (*simp add*: *M2.nn_integral_fst* [*symmetric*])
also have ... = ($\int^+$ *x*. |$\int$ *y*. *norm* (*f* (*x*, *y*)) ∂*M2*| ∂*M1*)
apply (*intro nn_integral_cong_AE*)
using *integ2*
proof *eventually_elim*

```
    fix x assume integrable M2 (λy. f (x, y))
    then have f: integrable M2 (λy. norm (f (x, y)))
      by simp
     then have (∫⁺y. ennreal (norm (f (x, y))) ∂M2) = ennreal (LINT y|M2.
norm (f (x, y)))
      by (rule nn_integral_eq_integral) simp
    also have ... = ennreal |LINT y|M2. norm (f (x, y))|
      using f by simp
    finally show (∫⁺y. ennreal (norm (f (x, y))) ∂M2) = ennreal |LINT y|M2.
norm (f (x, y))| .
  qed
  also have ... < ∞
    using integ1 by (simp add: integrable_iff_bounded integral_nonneg_AE)
  finally show (∫⁺ p. norm (f p) ∂(M1 ⨂M M2)) < ∞ .
qed fact

lemma (in pair_sigma_finite) emeasure_pair_measure_finite:
  assumes A: A ∈ sets (M1 ⨂M M2) and finite: emeasure (M1 ⨂M M2) A <
∞
  shows AE x in M1. emeasure M2 {y∈space M2. (x, y) ∈ A} < ∞
proof −
  from M2.emeasure_pair_measure_alt[OF A] finite
  have (∫⁺ x. emeasure M2 (Pair x −‘ A) ∂M1) ≠ ∞
    by simp
  then have AE x in M1. emeasure M2 (Pair x −‘ A) ≠ ∞
   by (rule nn_integral_PInf_AE[rotated]) (intro M2.measurable_emeasure_Pair
A)
  moreover have ⋀x. x ∈ space M1 ⟹ Pair x −‘ A = {y∈space M2. (x, y) ∈
A}
    using sets.sets_into_space[OF A] by (auto simp: space_pair_measure)
  ultimately show ?thesis by (auto simp: less_top)
qed

lemma (in pair_sigma_finite) AE_integrable_fst':
  fixes f :: _ ⇒ _::{banach, second_countable_topology}
  assumes f[measurable]: integrable (M1 ⨂M M2) f
  shows AE x in M1. integrable M2 (λy. f (x, y))
proof −
  have (∫⁺x. (∫⁺y. norm (f (x, y)) ∂M2) ∂M1) = (∫⁺x. norm (f x) ∂(M1 ⨂M
M2))
    by (rule M2.nn_integral_fst) simp
  also have (∫⁺x. norm (f x) ∂(M1 ⨂M M2)) ≠ ∞
    using f unfolding integrable_iff_bounded by simp
  finally have AE x in M1. (∫⁺y. norm (f (x, y)) ∂M2) ≠ ∞
    by (intro nn_integral_PInf_AE M2.borel_measurable_nn_integral )
       (auto simp: measurable_split_conv)
  with AE_space show ?thesis
    by eventually_elim
     (auto simp: integrable_iff_bounded measurable_compose[OF _ borel_measurable_integrable[OF
```

f]] *less_top*)
qed

lemma (**in** *pair_sigma_finite*) *integrable_fst'*:
fixes *f* :: _ ⇒ _::{*banach*, *second_countable_topology*}
assumes *f*[*measurable*]: *integrable* (*M1* ⨂$_M$ *M2*) *f*
shows *integrable M1* (λ*x*. ∫ *y*. *f* (*x*, *y*) ∂*M2*)
unfolding *integrable_iff_bounded*
proof
show (λ*x*. ∫ *y*. *f* (*x*, *y*) ∂*M2*) ∈ *borel_measurable M1*
by (*rule M2.borel_measurable_lebesgue_integral*) *simp*
have (∫$^+$ *x*. *ennreal* (*norm* (∫ *y*. *f* (*x*, *y*) ∂*M2*)) ∂*M1*) ≤ (∫$^+$*x*. (∫$^+$*y*. *norm* (*f* (*x*, *y*)) ∂*M2*) ∂*M1*)
using *AE_integrable_fst'*[*OF f*] **by** (*auto intro*!: *nn_integral_mono_AE integral_norm_bound_ennreal*)
also have (∫$^+$*x*. (∫$^+$*y*. *norm* (*f* (*x*, *y*)) ∂*M2*) ∂*M1*) = (∫$^+$*x*. *norm* (*f x*) ∂(*M1* ⨂$_M$ *M2*))
by (*rule M2.nn_integral_fst*) *simp*
also have (∫$^+$*x*. *norm* (*f x*) ∂(*M1* ⨂$_M$ *M2*)) < ∞
using *f* **unfolding** *integrable_iff_bounded* **by** *simp*
finally show (∫$^+$ *x*. *ennreal* (*norm* (∫ *y*. *f* (*x*, *y*) ∂*M2*)) ∂*M1*) < ∞ .
qed

proposition (**in** *pair_sigma_finite*) *integral_fst'*:
fixes *f* :: _ ⇒ _::{*banach*, *second_countable_topology*}
assumes *f*: *integrable* (*M1* ⨂$_M$ *M2*) *f*
shows (∫ *x*. (∫ *y*. *f* (*x*, *y*) ∂*M2*) ∂*M1*) = *integral*^{L} (*M1* ⨂$_M$ *M2*) *f*
using *f* **proof** *induct*
case (*base A c*)
have *A*[*measurable*]: *A* ∈ *sets* (*M1* ⨂$_M$ *M2*) **by** *fact*

have *eq*: ⋀*x y*. *x* ∈ *space M1* ⟹ *indicator A* (*x*, *y*) = *indicator* {*y*∈*space M2*. (*x*, *y*) ∈ *A*} *y*
using *sets.sets_into_space*[*OF A*] **by** (*auto split*: *split_indicator simp*: *space_pair_measure*)

have *int_A*: *integrable* (*M1* ⨂$_M$ *M2*) (*indicator A* :: _ ⇒ *real*)
using *base* **by** (*rule integrable_real_indicator*)

have (∫ *x*. ∫ *y*. *indicator A* (*x*, *y*) ∗$_R$ *c* ∂*M2* ∂*M1*) = (∫ *x*. *measure M2* {*y*∈*space M2*. (*x*, *y*) ∈ *A*} ∗$_R$ *c* ∂*M1*)
proof (*intro integral_cong_AE*, *simp*, *simp*)
from *AE_integrable_fst'*[*OF int_A*] *AE_space*
show *AE x in M1*. (∫ *y*. *indicator A* (*x*, *y*) ∗$_R$ *c* ∂*M2*) = *measure M2* {*y*∈*space M2*. (*x*, *y*) ∈ *A*} ∗$_R$ *c*
by *eventually_elim* (*simp add*: *eq integrable_indicator_iff*)
qed
also have ... = *measure* (*M1* ⨂$_M$ *M2*) *A* ∗$_R$ *c*
proof (*subst integral_scaleR_left*)
have (∫$^+$*x*. *ennreal* (*measure M2* {*y* ∈ *space M2*. (*x*, *y*) ∈ *A*}) ∂*M1*) =

```
    (∫⁺x. emeasure M2 {y ∈ space M2. (x, y) ∈ A} ∂M1)
     using emeasure_pair_measure_finite[OF base]
   by (intro nn_integral_cong_AE, eventually_elim) (simp add: emeasure_eq_ennreal_measure)
   also have ... = emeasure (M1 ⨂M M2) A
     using sets.sets_into_space[OF A]
     by (subst M2.emeasure_pair_measure_alt)
        (auto intro!: nn_integral_cong arg_cong[where f=emeasure M2] simp:
space_pair_measure)
  finally have *: (∫⁺x. ennreal (measure M2 {y ∈ space M2. (x, y) ∈ A}) ∂M1)
= emeasure (M1 ⨂M M2) A .

    from base * show integrable M1 (λx. measure M2 {y ∈ space M2. (x, y) ∈
A})
     by (simp add: integrable_iff_bounded)
   then have (∫ x. measure M2 {y ∈ space M2. (x, y) ∈ A} ∂M1) =
     (∫⁺x. ennreal (measure M2 {y ∈ space M2. (x, y) ∈ A}) ∂M1)
     by (rule nn_integral_eq_integral[symmetric]) simp
   also note *
    finally show (∫ x. measure M2 {y ∈ space M2. (x, y) ∈ A} ∂M1) *R c =
measure (M1 ⨂M M2) A *R c
     using base by (simp add: emeasure_eq_ennreal_measure)
 qed
 also have ... = (∫ a. indicator A a *R c ∂(M1 ⨂M M2))
   using base by simp
 finally show ?case .
next
 case (add f g)
 then have [measurable]: f ∈ borel_measurable (M1 ⨂M M2) g ∈ borel_measurable
(M1 ⨂M M2)
   by auto
 have AE x in M1. LINT y|M2. f (x, y) + g (x, y) =
            (LINT y|M2. f (x, y)) + (LINT y|M2. g (x, y))
   using AE_integrable_fst'[OF add(1)] AE_integrable_fst'[OF add(3)]
   by eventually_elim simp
 then have (∫ x. ∫ y. f (x, y) + g (x, y) ∂M2 ∂M1) =
   (∫ x. (∫ y. f (x, y) ∂M2) + (∫ y. g (x, y) ∂M2) ∂M1)
   by (intro integral_cong_AE) auto
 also have ... = (∫ x. f x ∂(M1 ⨂M M2)) + (∫ x. g x ∂(M1 ⨂M M2))
   using integrable_fst'[OF add(1)] integrable_fst'[OF add(3)] add(2,4) by simp
 finally show ?case
   using add by simp
next
 case (lim f s)
 then have [measurable]: f ∈ borel_measurable (M1 ⨂M M2) ⋀i. s i ∈ borel_measurable
(M1 ⨂M M2)
   by auto

 show ?case
 proof (rule LIMSEQ_unique)
```

```
  show (λi. integral^L (M1 ⨂_M M2) (s i)) ⟶ integral^L (M1 ⨂_M M2) f
  proof (rule integral_dominated_convergence)
    show integrable (M1 ⨂_M M2) (λx. 2 * norm (f x))
      using lim(5) by auto
  qed (insert lim, auto)
  have (λi. ∫ x. ∫ y. s i (x, y) ∂M2 ∂M1) ⟶ ∫ x. ∫ y. f (x, y) ∂M2 ∂M1
  proof (rule integral_dominated_convergence)
    have AE x in M1. ∀ i. integrable M2 (λy. s i (x, y))
      unfolding AE_all_countable using AE_integrable_fst'[OF lim(1)] ..
    with AE_space AE_integrable_fst'[OF lim(5)]
    show AE x in M1. (λi. ∫ y. s i (x, y) ∂M2) ⟶ ∫ y. f (x, y) ∂M2
    proof eventually_elim
      fix x assume x: x ∈ space M1 and
        s: ∀ i. integrable M2 (λy. s i (x, y)) and f: integrable M2 (λy. f (x, y))
      show (λi. ∫ y. s i (x, y) ∂M2) ⟶ ∫ y. f (x, y) ∂M2
      proof (rule integral_dominated_convergence)
        show integrable M2 (λy. 2 * norm (f (x, y)))
          using f by auto
        show AE xa in M2. (λi. s i (x, xa)) ⟶ f (x, xa)
          using x lim(3) by (auto simp: space_pair_measure)
        show ⋀i. AE xa in M2. norm (s i (x, xa)) ≤ 2 * norm (f (x, xa))
          using x lim(4) by (auto simp: space_pair_measure)
      qed (insert x, measurable)
    qed
    show integrable M1 (λx. (∫ y. 2 * norm (f (x, y)) ∂M2))
      by (intro integrable_mult_right integrable_norm integrable_fst' lim)
    fix i show AE x in M1. norm (∫ y. s i (x, y) ∂M2) ≤ (∫ y. 2 * norm (f
(x, y)) ∂M2)
    using AE_space AE_integrable_fst'[OF lim(1), of i] AE_integrable_fst'[OF
lim(5)]
    proof eventually_elim
      fix x assume x: x ∈ space M1
        and s: integrable M2 (λy. s i (x, y)) and f: integrable M2 (λy. f (x, y))
      from s have norm (∫ y. s i (x, y) ∂M2) ≤ (∫⁺y. norm (s i (x, y)) ∂M2)
        by (rule integral_norm_bound_ennreal)
      also have ... ≤ (∫⁺y. 2 * norm (f (x, y)) ∂M2)
      using x lim by (auto intro!: nn_integral_mono simp: space_pair_measure)
      also have ... = (∫ y. 2 * norm (f (x, y)) ∂M2)
        using f by (intro nn_integral_eq_integral) auto
      finally show norm (∫ y. s i (x, y) ∂M2) ≤ (∫ y. 2 * norm (f (x, y))
∂M2)
        by simp
    qed
  qed simp_all
  then show (λi. integral^L (M1 ⨂_M M2) (s i)) ⟶ ∫ x. ∫ y. f (x, y) ∂M2
∂M1
    using lim by simp
 qed
qed
```

lemma (**in** *pair_sigma_finite*)
fixes $f :: \_ \Rightarrow \_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$
assumes f: *integrable* $(M1 \bigotimes_M M2)$ (*case_prod f*)
shows *AE_integrable_fst*: *AE x in M1. integrable M2* $(\lambda y.\ f\ x\ y)$ (**is** *?AE*)
and *integrable_fst*: *integrable M1* $(\lambda x.\ \int y.\ f\ x\ y\ \partial M2)$ (**is** *?INT*)
and *integral_fst*: $(\int x.\ (\int y.\ f\ x\ y\ \partial M2)\ \partial M1) = integral^L\ (M1 \bigotimes_M M2)\ (\lambda(x, y).\ f\ x\ y)$ (**is** *?EQ*)
using *AE_integrable_fst'*[*OF f*] *integrable_fst'*[*OF f*] *integral_fst'*[*OF f*] **by** *auto*

lemma (**in** *pair_sigma_finite*)
fixes $f :: \_ \Rightarrow \_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$
assumes *f*[*measurable*]: *integrable* $(M1 \bigotimes_M M2)$ (*case_prod f*)
shows *AE_integrable_snd*: *AE y in M2. integrable M1* $(\lambda x.\ f\ x\ y)$ (**is** *?AE*)
and *integrable_snd*: *integrable M2* $(\lambda y.\ \int x.\ f\ x\ y\ \partial M1)$ (**is** *?INT*)
and *integral_snd*: $(\int y.\ (\int x.\ f\ x\ y\ \partial M1)\ \partial M2) = integral^L\ (M1 \bigotimes_M M2)$ $(case\_prod\ f)$ (**is** *?EQ*)
proof −
interpret *Q*: *pair_sigma_finite M2 M1* **..**
have *Q_int*: *integrable* $(M2 \bigotimes_M M1)$ $(\lambda(x, y).\ f\ y\ x)$
using *f* **unfolding** *integrable_product_swap_iff*[*symmetric*] **by** *simp*
show *?AE* **using** *Q.AE_integrable_fst'*[*OF Q_int*] **by** *simp*
show *?INT* **using** *Q.integrable_fst'*[*OF Q_int*] **by** *simp*
show *?EQ* **using** *Q.integral_fst'*[*OF Q_int*]
using *integral_product_swap*[*of case_prod f*] **by** *simp*
qed

proposition (**in** *pair_sigma_finite*) *Fubini_integral*:
fixes $f :: \_ \Rightarrow \_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$
assumes *f*: *integrable* $(M1 \bigotimes_M M2)$ (*case_prod f*)
shows $(\int y.\ (\int x.\ f\ x\ y\ \partial M1)\ \partial M2) = (\int x.\ (\int y.\ f\ x\ y\ \partial M2)\ \partial M1)$
unfolding *integral_snd*[*OF assms*] *integral_fst*[*OF assms*] **..**

lemma (**in** *product_sigma_finite*) *product_integral_singleton*:
fixes $f :: \_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$
shows $f \in borel\_measurable\ (M\ i) \Longrightarrow (\int x.\ f\ (x\ i)\ \partial Pi_M\ \{i\}\ M) = integral^L\ (M\ i)\ f$
by (*metis* (*no_types*) *distr_singleton insert_iff integral_distr measurable_component_singleton*)

lemma (**in** *product_sigma_finite*) *product_integral_fold*:
fixes $f :: \_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$
assumes *IJ*[*simp*]: $I \cap J = \{\}$ **and** *fin*: *finite I finite J*
and *f*: *integrable* $(Pi_M\ (I \cup J)\ M)\ f$
shows $integral^L\ (Pi_M\ (I \cup J)\ M)\ f = (\int x.\ (\int y.\ f\ (merge\ I\ J\ (x, y))\ \partial Pi_M\ J\ M)\ \partial Pi_M\ I\ M)$
proof −
interpret *I*: *finite_product_sigma_finite M I* **by** *standard fact*
interpret *J*: *finite_product_sigma_finite M J* **by** *standard fact*
have *finite* $(I \cup J)$ **using** *fin* **by** *auto*

interpret *IJ*: *finite_product_sigma_finite M I ∪ J* **by** *standard fact*
interpret *P*: *pair_sigma_finite* Pi_M *I M* Pi_M *J M* **..**
let *?M* = *merge I J*
let *?f* = $\lambda x.$ *f* (*?M x*)
from *f* **have** *f_borel*: $f \in$ *borel_measurable* (Pi_M (*I* ∪ *J*) *M*)
by *auto*
have *P_borel*: ($\lambda x.$ *f* (*merge I J x*)) ∈ *borel_measurable* (Pi_M *I M* $\bigotimes_M$ Pi_M *J M*)
using *measurable_comp*[*OF measurable_merge f_borel*] **by** (*simp add*: *comp_def*)
have *f_int*: *integrable* (Pi_M *I M* $\bigotimes_M$ Pi_M *J M*) *?f*
by (*rule integrable_distr*[*OF measurable_merge*]) (*simp add*: *distr_merge*[*OF IJ fin*] *f*)
have *LINT x*|(Pi_M *I M* $\bigotimes_M$ Pi_M *J M*). *f* (*merge I J x*) =
LINT x|Pi_M *I M*. *LINT y*|Pi_M *J M*. *f* (*merge I J* (*x*, *y*))
by (*simp add*: *P.integral_fst'*[*symmetric*, *OF f_int*])
then show *?thesis*
apply (*subst distr_merge*[*symmetric*, *OF IJ fin*])
by (*simp add*: *integral_distr*[*OF measurable_merge f_borel*])
qed

lemma (**in** *product_sigma_finite*) *product_integral_insert*:
fixes *f* :: _ ⇒ _::{*banach*, *second_countable_topology*}
assumes *I*: *finite I i* ∉ *I*
and *f*: *integrable* (Pi_M (*insert i I*) *M*) *f*
shows *integral*^{L} (Pi_M (*insert i I*) *M*) *f* = (∫ *x*. (∫ *y*. *f* (*x*(*i*:=*y*)) ∂*M i*) ∂Pi_M *I M*)
proof −
have *integral*^{L} (Pi_M (*insert i I*) *M*) *f* = *integral*^{L} (Pi_M (*I* ∪ {*i*}) *M*) *f*
by *simp*
also have ... = (∫ *x*. (∫ *y*. *f* (*merge I* {*i*} (*x*,*y*)) ∂Pi_M {*i*} *M*) ∂Pi_M *I M*)
using *f I* **by** (*intro product_integral_fold*) *auto*
also have ... = (∫ *x*. (∫ *y*. *f* (*x*(*i* := *y*)) ∂*M i*) ∂Pi_M *I M*)
proof (*rule integral_cong*[*OF refl*], *subst product_integral_singleton*[*symmetric*])
fix *x* **assume** *x*: *x* ∈ *space* (Pi_M *I M*)
have *f_borel*: *f* ∈ *borel_measurable* (Pi_M (*insert i I*) *M*)
using *f* **by** *auto*
show ($\lambda y.$ *f* (*x*(*i* := *y*))) ∈ *borel_measurable* (*M i*)
using *measurable_comp*[*OF measurable_component_update f_borel*, *OF x* ‹*i* ∉ *I*›]
unfolding *comp_def* **.**
from *x I* **show** (∫ *y*. *f* (*merge I* {*i*} (*x*,*y*)) ∂Pi_M {*i*} *M*) = (∫ *xa*. *f* (*x*(*i* := *xa i*)) ∂Pi_M {*i*} *M*)
by (*auto intro*!: *integral_cong arg_cong*[**where** *f*=*f*] *simp*: *merge_def space_PiM extensional_def PiE_def*)
qed
finally show *?thesis* **.**
qed

lemma (**in** *product_sigma_finite*) *product_integrable_prod*:

fixes $f :: 'i \Rightarrow 'a \Rightarrow \_::\{real\_normed\_field, banach, second\_countable\_topology\}$
assumes [*simp*]: *finite I* **and** *integrable*: $\bigwedge i.\ i \in I \Longrightarrow$ *integrable* $(M\ i)\ (f\ i)$
shows *integrable* $(Pi_M\ I\ M)\ (\lambda x.\ (\prod i \in I.\ f\ i\ (x\ i)))$ (**is** *integrable* _ *?f*)
proof (*unfold integrable_iff_bounded*, *intro conjI*)
interpret *finite_product_sigma_finite M I* **by** *standard fact*

show *?f* $\in$ *borel_measurable* $(Pi_M\ I\ M)$
using *assms* **by** *simp*
have $(\int^+ x.\ ennreal\ (norm\ (\prod i \in I.\ f\ i\ (x\ i)))\ \partial Pi_M\ I\ M) =$
$(\int^+ x.\ (\prod i \in I.\ ennreal\ (norm\ (f\ i\ (x\ i))))\ \partial Pi_M\ I\ M)$
by (*simp add*: *prod_norm prod_ennreal*)
also have $\ldots = (\prod i \in I.\ \int^+ x.\ ennreal\ (norm\ (f\ i\ x))\ \partial M\ i)$
using *assms* **by** (*intro product_nn_integral_prod*) *auto*
also have $\ldots < \infty$
using *integrable* **by** (*simp add*: *less_top*[*symmetric*] *ennreal_prod_eq_top integrable_iff_bounded*)
finally show $(\int^+ x.\ ennreal\ (norm\ (\prod i \in I.\ f\ i\ (x\ i)))\ \partial Pi_M\ I\ M) < \infty$.
qed

lemma (**in** *product_sigma_finite*) *product_integral_prod*:
fixes $f :: 'i \Rightarrow 'a \Rightarrow \_::\{real\_normed\_field, banach, second\_countable\_topology\}$
assumes *finite I* **and** *integrable*: $\bigwedge i.\ i \in I \Longrightarrow$ *integrable* $(M\ i)\ (f\ i)$
shows $(\int x.\ (\prod i \in I.\ f\ i\ (x\ i))\ \partial Pi_M\ I\ M) = (\prod i \in I.\ integral^L\ (M\ i)\ (f\ i))$
using *assms* **proof** *induct*
case *empty*
interpret *finite_measure* $Pi_M\ \{\}\ M$
by *rule* (*simp add*: *space_PiM*)
show *?case* **by** (*simp add*: *space_PiM measure_def*)
next
case (*insert i I*)
then have *iI*: *finite* (*insert i I*) **by** *auto*
then have *prod*: $\bigwedge J.\ J \subseteq insert\ i\ I \Longrightarrow$
integrable $(Pi_M\ J\ M)\ (\lambda x.\ (\prod i \in J.\ f\ i\ (x\ i)))$
by (*intro product_integrable_prod insert*(*4*)) (*auto intro*: *finite_subset*)
interpret *I*: *finite_product_sigma_finite M I* **by** *standard fact*
have $*$: $\bigwedge x\ y.\ (\prod j \in I.\ f\ j\ (if\ j = i\ then\ y\ else\ x\ j)) = (\prod j \in I.\ f\ j\ (x\ j))$
using $\langle i \notin I \rangle$ **by** (*auto intro*!: *prod.cong*)
show *?case*
unfolding *product_integral_insert*[*OF insert*(*1*,*2*) *prod*[*OF subset_refl*]]
by (*simp add*: $*$ *insert prod subset_insertI*)
qed

lemma *integrable_subalgebra*:
fixes $f :: 'a \Rightarrow 'b::\{banach, second\_countable\_topology\}$
assumes *borel*: $f \in$ *borel_measurable N*
and *N*: *sets N* $\subseteq$ *sets M space N* = *space M* $\bigwedge A.\ A \in$ *sets N* $\Longrightarrow$ *emeasure N A* = *emeasure M A*
shows *integrable N f* $\longleftrightarrow$ *integrable M f* (**is** *?P*)
proof −

```
  have f ∈ borel_measurable M
    using assms by (auto simp: measurable_def)
  with assms show ?thesis
    using assms by (auto simp: integrable_iff_bounded nn_integral_subalgebra)
qed

lemma integral_subalgebra:
  fixes f :: 'a ⇒ 'b::{banach, second_countable_topology}
  assumes borel: f ∈ borel_measurable N
  and N: sets N ⊆ sets M space N = space M ⋀A. A ∈ sets N ⟹ emeasure N
A = emeasure M A
  shows integral^L N f = integral^L M f
proof cases
  assume integrable N f
  then show ?thesis
  proof induct
    case base with assms show ?case by (auto simp: subset_eq measure_def)
  next
    case (add f g)
    then have (∫ a. f a + g a ∂N) = integral^L M f + integral^L M g
      by simp
    also have ... = (∫ a. f a + g a ∂M)
      using add integrable_subalgebra[OF _ N, of f] integrable_subalgebra[OF _
N, of g] by simp
    finally show ?case .
  next
    case (lim f s)
    then have M: ⋀i. integrable M (s i) integrable M f
      using integrable_subalgebra[OF _ N, of f] integrable_subalgebra[OF _ N, of
s i for i] by simp_all
    show ?case
    proof (intro LIMSEQ_unique)
      show (λi. integral^L N (s i)) ⟶ integral^L N f
        apply (rule integral_dominated_convergence[where w=λx. 2 * norm (f
x)])
        using lim by auto
      show (λi. integral^L N (s i)) ⟶ integral^L M f
        unfolding lim
        apply (rule integral_dominated_convergence[where w=λx. 2 * norm (f
x)])
        using lim M N by auto
    qed
  qed
qed (simp add: not_integrable_integral_eq integrable_subalgebra[OF assms])

hide_const (open) simple_bochner_integral
hide_const (open) simple_bochner_integrable

end
```

6.10 Complete Measures

theory *Complete_Measure*
imports *Bochner_Integration*
begin

locale *complete_measure* =
fixes M :: *'a measure*
assumes *complete*: $\bigwedge A\ B.\ B \subseteq A \Longrightarrow A \in$ *null_sets* $M \Longrightarrow B \in$ *sets* M

definition
split_completion $M\ A\ p$ = (*if* $A \in$ *sets* M *then* $p = (A, \{\})$ *else*
$\exists N'.\ A =$ *fst* $p \cup$ *snd* $p \wedge$ *fst* $p \cap$ *snd* $p = \{\} \wedge$ *fst* $p \in$ *sets* $M \wedge$ *snd* $p \subseteq N' \wedge N' \in$ *null_sets* M)

definition
main_part $M\ A$ = *fst* (*Eps* (*split_completion* $M\ A$))

definition
null_part $M\ A$ = *snd* (*Eps* (*split_completion* $M\ A$))

definition *completion* :: *'a measure* ⇒ *'a measure* **where**
completion M = *measure_of* (*space* M) { $S \cup N$ |$S\ N\ N'$. $S \in$ *sets* $M \wedge N' \in$ *null_sets* $M \wedge N \subseteq N'$ }
(*emeasure* $M \circ$ *main_part* M)

lemma *completion_into_space*:
{ $S \cup N$ |$S\ N\ N'$. $S \in$ *sets* $M \wedge N' \in$ *null_sets* $M \wedge N \subseteq N'$ } ⊆ *Pow* (*space* M)
using *sets.sets_into_space* **by** *auto*

lemma *space_completion*[*simp*]: *space* (*completion* M) = *space* M
unfolding *completion_def* **using** *space_measure_of*[*OF completion_into_space*]
by *simp*

lemma *completionI*:
assumes $A = S \cup N\ N \subseteq N'\ N' \in$ *null_sets* $M\ S \in$ *sets* M
shows $A \in$ { $S \cup N$ |$S\ N\ N'$. $S \in$ *sets* $M \wedge N' \in$ *null_sets* $M \wedge N \subseteq N'$ }
using *assms* **by** *auto*

lemma *completionE*:
assumes $A \in$ { $S \cup N$ |$S\ N\ N'$. $S \in$ *sets* $M \wedge N' \in$ *null_sets* $M \wedge N \subseteq N'$ }
obtains $S\ N\ N'$ **where** $A = S \cup N\ N \subseteq N'\ N' \in$ *null_sets* $M\ S \in$ *sets* M
using *assms* **by** *auto*

lemma *sigma_algebra_completion*:
sigma_algebra (*space* M) { $S \cup N$ |$S\ N\ N'$. $S \in$ *sets* $M \wedge N' \in$ *null_sets* $M \wedge N \subseteq N'$ }
(**is** *sigma_algebra* _ *?A*)

unfolding *sigma_algebra_iff2*
proof (*intro conjI ballI allI impI*)
show *?A* ⊆ *Pow* (*space M*)
using *sets.sets_into_space* **by** *auto*
next
show {} ∈ *?A* **by** *auto*
next
let *?C* = *space M*
fix *A* **assume** *A* ∈ *?A*
then obtain *S N N′*
where *A* = *S* ∪ *N N* ⊆ *N′ N′* ∈ *null_sets M S* ∈ *sets M*
by (*rule completionE*)
then show *space M* − *A* ∈ *?A*
by (*intro completionI*[*of* _ (*?C* − *S*) ∩ (*?C* − *N′*) (*?C* − *S*) ∩ *N′* ∩ (*?C* − *N*)]) *auto*
next
fix *A* :: *nat* ⇒ *'a set* **assume** *A*: *range A* ⊆ *?A*
then have ∀ *n*. ∃ *S N N′*. *A n* = *S* ∪ *N* ∧ *S* ∈ *sets M* ∧ *N′* ∈ *null_sets M* ∧ *N* ⊆ *N′*
by (*auto simp*: *image_subset_iff*)
then obtain *S N N′* **where** ∀ *x*. *A x* = *S x* ∪ *N x* ∧ *S x* ∈ *sets M* ∧ *N′ x* ∈ *null_sets M* ∧ *N x* ⊆ *N′ x*
by *metis*
then show ⋃(*A* ‘ *UNIV*) ∈ *?A*
using *null_sets_UN*[*of N′*]
by (*intro completionI*[*of* _ ⋃(*S* ‘ *UNIV*) ⋃(*N* ‘ *UNIV*) ⋃(*N′* ‘ *UNIV*)]) *auto*
qed

lemma *sets_completion*:
sets (*completion M*) = { *S* ∪ *N* |*S N N′*. *S* ∈ *sets M* ∧ *N′* ∈ *null_sets M* ∧ *N* ⊆ *N′* }
using *sigma_algebra.sets_measure_of_eq*[*OF sigma_algebra_completion*]
by (*simp add*: *completion_def*)

lemma *sets_completionE*:
assumes *A* ∈ *sets* (*completion M*)
obtains *S N N′* **where** *A* = *S* ∪ *N N* ⊆ *N′ N′* ∈ *null_sets M S* ∈ *sets M*
using *assms* **unfolding** *sets_completion* **by** *auto*

lemma *sets_completionI*:
assumes *A* = *S* ∪ *N N* ⊆ *N′ N′* ∈ *null_sets M S* ∈ *sets M*
shows *A* ∈ *sets* (*completion M*)
using *assms* **unfolding** *sets_completion* **by** *auto*

lemma *sets_completionI_sets*[*intro*, *simp*]:
A ∈ *sets M* ⟹ *A* ∈ *sets* (*completion M*)
unfolding *sets_completion* **by** *force*

lemma *measurable_completion*: *f* ∈ *M* →$_M$ *N* ⟹ *f* ∈ *completion M* →$_M$ *N*

by (*auto simp*: *measurable_def*)

lemma *null_sets_completion*:
assumes $N' \in$ *null_sets M N* $\subseteq N'$ **shows** $N \in$ *sets* (*completion M*)
using *assms* **by** (*intro sets_completionI*[*of N* {} *N N'*]) *auto*

lemma *split_completion*:
assumes $A \in$ *sets* (*completion M*)
shows *split_completion M A* (*main_part M A*, *null_part M A*)
proof *cases*
assume $A \in$ *sets M* **then show** *?thesis*
by (*simp add*: *split_completion_def*[*abs_def*] *main_part_def null_part_def*)
next
assume *nA*: $A \notin$ *sets M*
show *?thesis*
unfolding *main_part_def null_part_def if_not_P*[*OF nA*]
proof (*rule someI2_ex*)
from *assms* **obtain** *S N N'* **where** *A*: $A = S \cup N$ $N \subseteq N'$ $N' \in$ *null_sets M* $S \in$ *sets M*
by (*blast intro*: *sets_completionE*)
let *?P* = (*S*, *N* − *S*)
show $\exists p$. *split_completion M A p*
unfolding *split_completion_def if_not_P*[*OF nA*] **using** *A*
proof (*intro exI conjI*)
show $A =$ *fst ?P* $\cup$ *snd ?P* **using** *A* **by** *auto*
show *snd ?P* $\subseteq N'$ **using** *A* **by** *auto*
qed *auto*
qed *auto*
qed

lemma *sets_restrict_space_subset*:
assumes *S*: $S \in$ *sets* (*completion M*)
shows *sets* (*restrict_space* (*completion M*) *S*) $\subseteq$ *sets* (*completion M*)
by (*metis assms sets.Int_space_eq2 sets_restrict_space_iff subsetI*)

lemma
assumes $S \in$ *sets* (*completion M*)
shows *main_part_sets*[*intro*, *simp*]: *main_part M S* $\in$ *sets M*
and *main_part_null_part_Un*[*simp*]: *main_part M S* $\cup$ *null_part M S* $= S$
and *main_part_null_part_Int*[*simp*]: *main_part M S* $\cap$ *null_part M S* $= \{\}$
using *split_completion*[*OF assms*]
by (*auto simp*: *split_completion_def split*: *if_split_asm*)

lemma *main_part*[*simp*]: $S \in$ *sets M* $\Longrightarrow$ *main_part M S* $= S$
using *split_completion*[*of S M*]
by (*auto simp*: *split_completion_def split*: *if_split_asm*)

lemma *null_part*:
assumes $S \in$ *sets* (*completion M*) **shows** $\exists N$. $N \in$*null_sets M* $\wedge$ *null_part M*

$S \subseteq N$
using *split_completion*[*OF assms*] **by** (*auto simp*: *split_completion_def split*: *if_split_asm*)

lemma *null_part_sets*[*intro*, *simp*]:
assumes $S \in$ *sets* M **shows** *null_part* M $S \in$ *sets* M *emeasure* M (*null_part* M S) $= 0$
proof −
have S: $S \in$ *sets* (*completion* M) **using** *assms* **by** *auto*
have $*$: $S -$ *main_part* M $S \in$ *sets* M **using** *assms* **by** *auto*
from *main_part_null_part_Un*[*OF* S] *main_part_null_part_Int*[*OF* S]
have $S -$ *main_part* M $S =$ *null_part* M S **by** *auto*
with $*$ **show** *sets*: *null_part* M $S \in$ *sets* M **by** *auto*
from *null_part*[*OF* S] **obtain** N **where** $N \in$ *null_sets* $M \wedge$ *null_part* M $S \subseteq N$ **..**
with *emeasure_eq_0*[*of* N _ *null_part* M S] *sets*
show *emeasure* M (*null_part* M S) $= 0$ **by** *auto*
qed

lemma *emeasure_main_part_UN*:
fixes $S :: nat \Rightarrow 'a$ *set*
assumes *range* $S \subseteq$ *sets* (*completion* M)
shows *emeasure* M (*main_part* M ($\bigcup i.$ (S i))) = *emeasure* M ($\bigcup i.$ *main_part* M (S i))
proof −
have S: $\bigwedge i.$ S $i \in$ *sets* (*completion* M) **using** *assms* **by** *auto*
then have *UN*: ($\bigcup i.$ S i) $\in$ *sets* (*completion* M) **by** *auto*
have $\forall i.\ \exists N.\ N \in$ *null_sets* $M \wedge$ *null_part* M (S i) $\subseteq N$
using *null_part*[*OF* S] **by** *auto*
then obtain N **where** N: $\forall x.\ N\ x \in$ *null_sets* $M \wedge$ *null_part* M (S x) $\subseteq N\ x$ **by** *metis*
then have *UN_N*: ($\bigcup i.$ N i) $\in$ *null_sets* M **by** (*intro null_sets_UN*) *auto*
from S **have** ($\bigcup i.$ S i) $\in$ *sets* (*completion* M) **by** *auto*
from *null_part*[*OF this*] **obtain** N' **where** N': $N' \in$ *null_sets* $M \wedge$ *null_part* M ($\bigcup$ (*range* S)) $\subseteq N'$ **..**
let *?N* = ($\bigcup i.$ N i) $\cup N'$
have *null_set*: *?N* $\in$ *null_sets* M **using** N' *UN_N* **by** (*intro null_sets.Un*) *auto*
have *main_part* M ($\bigcup i.$ S i) $\cup$ *?N* = (*main_part* M ($\bigcup i.$ S i) $\cup$ *null_part* M ($\bigcup i.$ S i)) $\cup$ *?N*
using N' **by** *auto*
also have ... = ($\bigcup i.$ *main_part* M (S i) $\cup$ *null_part* M (S i)) $\cup$ *?N*
unfolding *main_part_null_part_Un*[*OF* S] *main_part_null_part_Un*[*OF UN*] **by** *auto*
also have ... = ($\bigcup i.$ *main_part* M (S i)) $\cup$ *?N*
using N **by** *auto*
finally have $*$: *main_part* M ($\bigcup i.$ S i) $\cup$ *?N* = ($\bigcup i.$ *main_part* M (S i)) $\cup$ *?N*
.
have *emeasure* M (*main_part* M ($\bigcup i.$ S i)) = *emeasure* M (*main_part* M ($\bigcup i.$

S i) ∪ *?N*)
using *null_set UN* **by** (*intro emeasure_Un_null_set*[*symmetric*]) *auto*
also have ... = *emeasure M* ((⋃ *i*. *main_part M* (*S i*)) ∪ *?N*)
unfolding ∗ **..**
also have ... = *emeasure M* (⋃ *i*. *main_part M* (*S i*))
using *null_set S* **by** (*intro emeasure_Un_null_set*) *auto*
finally show *?thesis* **.**
qed

lemma *emeasure_completion*[*simp*]:
assumes *S*: *S* ∈ *sets* (*completion M*)
shows *emeasure* (*completion M*) *S* = *emeasure M* (*main_part M S*)
proof (*subst emeasure_measure_of*[*OF completion_def completion_into_space*])
let *?μ* = *emeasure M* ∘ *main_part M*
show *S* ∈ *sets* (*completion M*) *?μ S* = *emeasure M* (*main_part M S*) **using** *S*
by *simp_all*
show *positive* (*sets* (*completion M*)) *?μ*
by (*simp add*: *positive_def*)
show *countably_additive* (*sets* (*completion M*)) *?μ*
proof (*intro countably_additiveI*)
fix *A* :: *nat* ⇒ *'a set* **assume** *A*: *range A* ⊆ *sets* (*completion M*) *disjoint_family*
A
have *disjoint_family* (λ*i*. *main_part M* (*A i*))
proof (*intro disjoint_family_on_bisimulation*[*OF A*(*2*)])
fix *n m* **assume** *A n* ∩ *A m* = {}
then have (*main_part M* (*A n*) ∪ *null_part M* (*A n*)) ∩ (*main_part M* (*A*
m) ∪ *null_part M* (*A m*)) = {}
using *A* **by** (*subst* (*1 2*) *main_part_null_part_Un*) *auto*
then show *main_part M* (*A n*) ∩ *main_part M* (*A m*) = {} **by** *auto*
qed
then have (∑ *n*. *emeasure M* (*main_part M* (*A n*))) = *emeasure M* (⋃ *i*.
main_part M (*A i*))
using *A* **by** (*auto intro*!: *suminf_emeasure*)
then show (∑ *n*. *?μ* (*A n*)) = *?μ* (⋃ (*A* ‘ *UNIV*))
by (*simp add*: *completion_def emeasure_main_part_UN*[*OF A*(*1*)])
qed
qed

lemma *measure_completion*[*simp*]: *S* ∈ *sets M* ⟹ *measure* (*completion M*) *S* =
measure M S
unfolding *measure_def* **by** *auto*

lemma *emeasure_completion_UN*:
range S ⊆ *sets* (*completion M*) ⟹
emeasure (*completion M*) (⋃ *i*::*nat*. (*S i*)) = *emeasure M* (⋃ *i*. *main_part M*
(*S i*))
by (*subst emeasure_completion*) (*auto simp add*: *emeasure_main_part_UN*)

lemma *emeasure_completion_Un*:

```
  assumes S: S ∈ sets (completion M) and T: T ∈ sets (completion M)
  shows emeasure (completion M) (S ∪ T) = emeasure M (main_part M S ∪
main_part M T)
proof (subst emeasure_completion)
  have UN: (⋃i. binary (main_part M S) (main_part M T) i) = (⋃i. main_part
M (binary S T i))
    unfolding binary_def by (auto split: if_split_asm)
  show emeasure M (main_part M (S ∪ T)) = emeasure M (main_part M S ∪
main_part M T)
    using emeasure_main_part_UN[of binary S T M] assms
    by (simp add: range_binary_eq, simp add: Un_range_binary UN)
qed (auto intro: S T)

lemma sets_completionI_sub:
  assumes N: N' ∈ null_sets M N ⊆ N'
  shows N ∈ sets (completion M)
  using assms by (intro sets_completionI[of _ {} N N']) auto

lemma completion_ex_simple_function:
  assumes f: simple_function (completion M) f
  shows ∃f'. simple_function M f' ∧ (AE x in M. f x = f' x)
proof −
  let ?F = λx. f −‘ {x} ∩ space M
  have F: ⋀x. ?F x ∈ sets (completion M) and fin: finite (f‘space M)
    using simple_functionD[OF f] simple_functionD[OF f] by simp_all
  have ∀x. ∃N. N ∈ null_sets M ∧ null_part M (?F x) ⊆ N
    using F null_part by auto
  from choice[OF this] obtain N where
    N: ⋀x. null_part M (?F x) ⊆ N x ⋀x. N x ∈ null_sets M by auto
  let ?N = ⋃x∈f‘space M. N x
  let ?f' = λx. if x ∈ ?N then undefined else f x
  have sets: ?N ∈ null_sets M using N fin by (intro null_sets.finite_UN) auto
  show ?thesis unfolding simple_function_def
  proof (safe intro!: exI[of _ ?f'])
    have ?f' ‘ space M ⊆ f‘space M ∪ {undefined} by auto
    from finite_subset[OF this] simple_functionD(1)[OF f]
    show finite (?f' ‘ space M) by auto
  next
    fix x assume x ∈ space M
    have ?f' −‘ {?f' x} ∩ space M =
      (if x ∈ ?N then ?F undefined ∪ ?N
       else if f x = undefined then ?F (f x) ∪ ?N
       else ?F (f x) − ?N)
    using N(2) sets.sets_into_space by (auto split: if_split_asm simp: null_sets_def)
    moreover { fix y have ?F y ∪ ?N ∈ sets M
      proof cases
        assume y: y ∈ f‘space M
        have ?F y ∪ ?N = (main_part M (?F y) ∪ null_part M (?F y)) ∪ ?N
          using main_part_null_part_Un[OF F] by auto
```

```
      also have ... = main_part M (?F y) ∪ ?N
        using y N by auto
      finally show ?thesis
        using F sets by auto
    next
      assume y ∉ f`space M then have ?F y = {} by auto
      then show ?thesis using sets by auto
    qed }
  moreover {
    have ?F (f x) − ?N = main_part M (?F (f x)) ∪ null_part M (?F (f x)) −
?N
      using main_part_null_part_Un[OF F] by auto
    also have ... = main_part M (?F (f x)) − ?N
      using N ‹x ∈ space M› by auto
    finally have ?F (f x) − ?N ∈ sets M
      using F sets by auto }
  ultimately show ?f' −` {?f' x} ∩ space M ∈ sets M by auto
 next
  show AE x in M. f x = ?f' x
    by (rule AE_I', rule sets) auto
 qed
qed

lemma completion_ex_borel_measurable:
 fixes g :: 'a ⇒ ennreal
 assumes g: g ∈ borel_measurable (completion M)
 shows ∃g'∈borel_measurable M. (AE x in M. g x = g' x)
proof −
 obtain f :: nat ⇒ 'a ⇒ ennreal
   where ∗: ⋀i. simple_function (completion M) (f i)
     and ∗∗: ⋀x. (SUP i. f i x) = g x
  using g[THEN borel_measurable_implies_simple_function_sequence'] by metis
 from ∗[THEN completion_ex_simple_function]
 have ∀ i. ∃f'. simple_function M f' ∧ (AE x in M. f i x = f' x) ..
 then obtain f'
   where sf: ⋀i. simple_function M (f' i)
     and AE: ∀ i. AE x in M. f i x = f' i x
   by metis
 show ?thesis
 proof (intro bexI)
   from AE[unfolded AE_all_countable[symmetric]]
   show AE x in M. g x = (SUP i. f' i x) (is AE x in M. g x = ?f x)
   proof (elim AE_mp, safe intro!: AE_I2)
     fix x assume eq: ∀ i. f i x = f' i x
     have g x = (SUP i. f i x) by (auto simp: ∗∗ split: split_max)
     with eq show g x = ?f x by auto
   qed
   show ?f ∈ borel_measurable M
     using sf[THEN borel_measurable_simple_function] by auto
```

qed
qed

lemma *null_sets_completionI*: $N \in$ *null_sets* $M \Longrightarrow N \in$ *null_sets* (*completion* M)
by (*auto simp*: *null_sets_def*)

lemma *AE_completion*: (*AE* x *in* M. $P\ x$) $\Longrightarrow$ (*AE* x *in completion* M. $P\ x$)
unfolding *eventually_ae_filter* **by** (*auto intro*: *null_sets_completionI*)

lemma *null_sets_completion_iff*: $N \in$ *sets* $M \Longrightarrow N \in$ *null_sets* (*completion* M) $\longleftrightarrow N \in$ *null_sets* M
by (*auto simp*: *null_sets_def*)

lemma *sets_completion_AE*: (*AE* x *in* M. $\neg\ P\ x$) $\Longrightarrow$ *Measurable.pred* (*completion* M) P
unfolding *pred_def sets_completion eventually_ae_filter*
by *auto*

lemma *null_sets_completion_iff2*:
$A \in$ *null_sets* (*completion* M) $\longleftrightarrow$ ($\exists N' \in$*null_sets* M. $A \subseteq N'$)
proof *safe*
assume $A \in$ *null_sets* (*completion* M)
then have A: $A \in$ *sets* (*completion* M) **and** *main_part* $M\ A \in$ *null_sets* M
by (*auto simp*: *null_sets_def*)
moreover obtain N **where** $N \in$ *null_sets* M *null_part* $M\ A \subseteq N$
using *null_part*[*OF* A] **by** *auto*
ultimately show $\exists N' \in$*null_sets* M. $A \subseteq N'$
proof (*intro bexI*)
show $A \subseteq N \cup$ *main_part* $M\ A$
using ‹*null_part* $M\ A \subseteq N$› **by** (*subst main_part_null_part_Un*[*OF* A, *symmetric*]) *auto*
qed *auto*
next
fix N **assume** $N \in$ *null_sets* $M\ A \subseteq N$
then have $A \in$ *sets* (*completion* M) **and** N: $N \in$ *sets* $M\ A \subseteq N$ *emeasure* $M\ N = 0$
by (*auto intro*: *null_sets_completion*)
moreover have *emeasure* (*completion* M) $A = 0$
using N **by** (*intro emeasure_eq_0*[*of* N _ A]) *auto*
ultimately show $A \in$ *null_sets* (*completion* M)
by *auto*
qed

lemma *null_sets_completion_subset*:
$B \subseteq A \Longrightarrow A \in$ *null_sets* (*completion* M) $\Longrightarrow B \in$ *null_sets* (*completion* M)
unfolding *null_sets_completion_iff2* **by** *auto*

interpretation *completion*: *complete_measure completion* M **for** M

proof
show $B \subseteq A \Longrightarrow A \in$ *null_sets* (*completion* M) $\Longrightarrow B \in$ *sets* (*completion* M)
for B A
using *null_sets_completion_subset*[*of* B A M] **by** (*simp add*: *null_sets_def*)
qed

lemma *null_sets_restrict_space*:
$\Omega \in$ *sets* $M \Longrightarrow A \in$ *null_sets* (*restrict_space* M Ω) $\longleftrightarrow A \subseteq \Omega \wedge A \in$ *null_sets* M
by (*auto simp*: *null_sets_def emeasure_restrict_space sets_restrict_space*)

lemma *completion_ex_borel_measurable_real*:
fixes $g :: 'a \Rightarrow$ *real*
assumes *g*: $g \in$ *borel_measurable* (*completion* M)
shows $\exists g' \in$*borel_measurable* M. (*AE* x *in* M. $g\ x = g'\ x$)
proof −
have $(\lambda x.$ *ennreal* $(g\ x)) \in$ *completion* $M \rightarrow_M$ *borel* $(\lambda x.$ *ennreal* $(-\ g\ x)) \in$ *completion* $M \rightarrow_M$ *borel*
using g **by** *auto*
from *this*[*THEN completion_ex_borel_measurable*]
obtain *pf nf* :: $'a \Rightarrow$ *ennreal*
where [*measurable*]: $nf \in M \rightarrow_M$ *borel* $pf \in M \rightarrow_M$ *borel*
and *ae*: *AE* x *in* M. $pf\ x =$ *ennreal* $(g\ x)$ *AE* x *in* M. $nf\ x =$ *ennreal* $(-\ g\ x)$
by (*auto simp*: *eq_commute*)
then have *AE* x *in* M. $pf\ x =$ *ennreal* $(g\ x) \wedge nf\ x =$ *ennreal* $(-\ g\ x)$
by *auto*
then obtain N **where** $N \in$ *null_sets* M $\{x \in$*space* M. $pf\ x \neq$ *ennreal* $(g\ x) \wedge nf\ x \neq$ *ennreal* $(-\ g\ x)\} \subseteq N$
by (*auto elim*!: *AE_E*)
show *?thesis*
proof
let *?F* $= \lambda x.$ *indicator* (*space* $M - N$) $x *$ (*enn2real* $(pf\ x) -$ *enn2real* $(nf\ x)$)
show *?F* $\in M \rightarrow_M$ *borel*
using ‹$N \in$ *null_sets* M› **by** *auto*
show *AE* x *in* M. $g\ x =$ *?F* x
using ‹$N \in$ *null_sets* M›[*THEN AE_not_in*] *ae AE_space*
apply *eventually_elim*
subgoal for x
by (*cases 0*::*real g x rule*: *linorder_le_cases*) (*auto simp*: *ennreal_neg*)
done
qed
qed

lemma *simple_function_completion*: *simple_function* $M\ f \Longrightarrow$ *simple_function* (*completion* M) f
by (*simp add*: *simple_function_def*)

lemma *simple_integral_completion*:
simple_function $M\ f \Longrightarrow$ *simple_integral* (*completion* M) $f =$ *simple_integral* M

f
unfolding *simple_integral_def* **by** *simp*

lemma *nn_integral_completion*: *nn_integral* (*completion M*) *f* = *nn_integral M f*
unfolding *nn_integral_def*
proof (*safe intro*!: *SUP_eq*)
fix *s* **assume** *s*: *simple_function* (*completion M*) *s* **and** *s* ≤ *f*
then obtain *s'* **where** *s'*: *simple_function M s' AE x in M*. *s x* = *s' x*
by (*auto dest*: *completion_ex_simple_function*)
then obtain *N* **where** *N*: *N* ∈ *null_sets M* {*x*∈*space M*. *s x* ≠ *s' x*} ⊆ *N*
by (*auto elim*!: *AE_E*)
then have *ae_N*: *AE x in M*. (*s x* ≠ *s' x* ⟶ *x* ∈ *N*) ∧ *x* ∉ *N*
by (*auto dest*: *AE_not_in*)
define *s''* **where** *s'' x* = (*if x* ∈ *N then 0 else s x*) **for** *x*
then have *ae_s_eq_s''*: *AE x in completion M*. *s x* = *s'' x*
using *s' ae_N* **by** (*intro AE_completion*) *auto*
have *s''*: *simple_function M s''*
proof (*subst simple_function_cong*)
show *t* ∈ *space M* ⟹ *s'' t* = (*if t* ∈ *N then 0 else s' t*) **for** *t*
using *N* **by** (*auto simp*: *s''_def dest*: *sets.sets_into_space*)
show *simple_function M* (λ*t*. *if t* ∈ *N then 0 else s' t*)
unfolding *s''_def*[*abs_def*] **using** *N* **by** (*auto intro*!: *simple_function_If s'*)
qed

show ∃*j*∈{*g*. *simple_function M g* ∧ *g* ≤ *f*}. *integral*^{S} (*completion M*) *s* ≤ *integral*^{S} *M j*
proof (*safe intro*!: *bexI*[*of* _ *s''*])
have *integral*^{S} (*completion M*) *s* = *integral*^{S} (*completion M*) *s''*
by (*intro simple_integral_cong_AE s simple_function_completion s'' ae_s_eq_s''*)
then show *integral*^{S} (*completion M*) *s* ≤ *integral*^{S} *M s''*
using *s''* **by** (*simp add*: *simple_integral_completion*)
from ‹*s* ≤ *f*› **show** *s''* ≤ *f*
unfolding *s''_def le_fun_def* **by** *auto*
qed *fact*
next
fix *s* **assume** *simple_function M s s* ≤ *f*
then show ∃*j*∈{*g*. *simple_function* (*completion M*) *g* ∧ *g* ≤ *f*}. *integral*^{S} *M s* ≤ *integral*^{S} (*completion M*) *j*
by (*intro bexI*[*of* _ *s*]) (*auto simp*: *simple_integral_completion simple_function_completion*)
qed

lemma *integrable_completion*:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second_countable_topology*}
shows *f* ∈ *M* →$_M$ *borel* ⟹ *integrable* (*completion M*) *f* ⟷ *integrable M f*
by (*rule integrable_subalgebra*[*symmetric*]) *auto*

lemma *integral_completion*:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second_countable_topology*}

assumes f: $f \in M \to_M$ *borel* **shows** $integral^L$ (*completion* M) $f = integral^L$ M f
by (*rule integral_subalgebra*[*symmetric*]) (*auto intro*: f)

locale *semifinite_measure* =
fixes M :: $'a$ *measure*
assumes *semifinite*:
$\bigwedge A.\ A \in sets\ M \Longrightarrow emeasure\ M\ A = \infty \Longrightarrow \exists B{\in}sets\ M.\ B \subseteq A \wedge emeasure\ M\ B < \infty$

locale *locally_determined_measure* = *semifinite_measure* +
assumes *locally_determined*:
$\bigwedge A.\ A \subseteq space\ M \Longrightarrow (\bigwedge B.\ B \in sets\ M \Longrightarrow emeasure\ M\ B < \infty \Longrightarrow A \cap B \in sets\ M) \Longrightarrow A \in sets\ M$

locale *cld_measure* =
complete_measure M + *locally_determined_measure* M **for** M :: $'a$ *measure*

definition *outer_measure_of* :: $'a$ *measure* $\Rightarrow$ $'a$ *set* $\Rightarrow$ *ennreal*
where *outer_measure_of* $M\ A = (INF\ B \in \{B{\in}sets\ M.\ A \subseteq B\}.\ emeasure\ M\ B)$

lemma *outer_measure_of_eq*[*simp*]: $A \in sets\ M \Longrightarrow outer\_measure\_of\ M\ A = emeasure\ M\ A$
by (*auto simp*: *outer_measure_of_def intro*!: *INF_eqI emeasure_mono*)

lemma *outer_measure_of_mono*: $A \subseteq B \Longrightarrow outer\_measure\_of\ M\ A \leq outer\_measure\_of\ M\ B$
unfolding *outer_measure_of_def* **by** (*intro INF_superset_mono*) *auto*

lemma *outer_measure_of_attain*:
assumes $A \subseteq space\ M$
shows $\exists E{\in}sets\ M.\ A \subseteq E \wedge outer\_measure\_of\ M\ A = emeasure\ M\ E$
proof −
have *emeasure* M ‘ $\{B \in sets\ M.\ A \subseteq B\} \neq \{\}$
using ‹$A \subseteq space\ M$› **by** *auto*
from *ennreal_Inf_countable_INF*[*OF this*]
obtain f
where f: *range* $f \subseteq$ *emeasure* M ‘ $\{B \in sets\ M.\ A \subseteq B\}$ *decseq* f
and *outer_measure_of* $M\ A = (INF\ i.\ f\ i)$
unfolding *outer_measure_of_def* **by** *auto*
have $\exists E.\ \forall n.\ (E\ n \in sets\ M \wedge A \subseteq E\ n \wedge emeasure\ M\ (E\ n) \leq f\ n) \wedge E\ (Suc\ n) \subseteq E\ n$
proof (*rule dependent_nat_choice*)
show $\exists x.\ x \in sets\ M \wedge A \subseteq x \wedge emeasure\ M\ x \leq f\ 0$
using $f(1)$ **by** (*fastforce simp*: *image_subset_iff image_iff intro*: *eq_refl*[*OF sym*])
next
fix $E\ n$ **assume** $E \in sets\ M \wedge A \subseteq E \wedge emeasure\ M\ E \leq f\ n$

```
    moreover obtain F where F ∈ sets M A ⊆ F f (Suc n) = emeasure M F
      using f(1) by (auto simp: image_subset_iff image_iff)
    ultimately show ∃ y. (y ∈ sets M ∧ A ⊆ y ∧ emeasure M y ≤ f (Suc n)) ∧
y ⊆ E
      by (auto intro!: exI[of _ F ∩ E] emeasure_mono)
  qed
  then obtain E
    where [simp]: ⋀n. E n ∈ sets M
      and ⋀n. A ⊆ E n
      and le_f: ⋀n. emeasure M (E n) ≤ f n
      and decseq E
    by (auto simp: decseq_Suc_iff)
  show ?thesis
  proof cases
    assume fin: ∃ i. emeasure M (E i) < ∞
    show ?thesis
    proof (intro bexI[of _ ⋂ i. E i] conjI)
      show A ⊆ (⋂ i. E i) (⋂ i. E i) ∈ sets M
        using ‹⋀n. A ⊆ E n› by auto

      have  (INF i. emeasure M (E i)) ≤ outer_measure_of M A
        unfolding ‹outer_measure_of M A = (INF n. f n)›
        by (intro INF_superset_mono le_f) auto
      moreover have outer_measure_of M A ≤ (INF i. outer_measure_of M (E
i))
        by (intro INF_greatest outer_measure_of_mono ‹⋀n. A ⊆ E n›)
      ultimately have outer_measure_of M A = (INF i. emeasure M (E i))
        by auto
      also have ... = emeasure M (⋂ i. E i)
       using fin by (intro INF_emeasure_decseq' ‹decseq E›) (auto simp: less_top)
      finally show outer_measure_of M A = emeasure M (⋂ i. E i) .
    qed
  next
    assume ∄ i. emeasure M (E i) < ∞
    then have f n = ∞ for n
      using le_f by (auto simp: not_less top_unique)
    moreover have ∃ E∈sets M. A ⊆ E ∧ f 0 = emeasure M E
      using f by auto
    ultimately show ?thesis
      unfolding ‹outer_measure_of M A = (INF n. f n)› by simp
  qed
qed

lemma SUP_outer_measure_of_incseq:
  assumes A: ⋀n. A n ⊆ space M and incseq A
  shows (SUP n. outer_measure_of M (A n)) = outer_measure_of M (⋃ i. A i)
proof (rule antisym)
  obtain E
    where E: ⋀n. E n ∈ sets M ⋀n. A n ⊆ E n ⋀n. outer_measure_of M (A n)
```

```
= emeasure M (E n)
    using outer_measure_of_attain[OF A] by metis

  define F where F n = (⋂i∈{n ..}. E i) for n
  with E have F: incseq F ⋀n. F n ∈ sets M
    by (auto simp: incseq_def)
  have A n ⊆ F n for n
    using incseqD[OF ‹incseq A›, of n] ‹⋀n. A n ⊆ E n› by (auto simp: F_def)

  have eq: outer_measure_of M (A n) = outer_measure_of M (F n) for n
  proof (intro antisym)
    have outer_measure_of M (F n) ≤ outer_measure_of M (E n)
      by (intro outer_measure_of_mono) (auto simp add: F_def)
    with E show outer_measure_of M (F n) ≤ outer_measure_of M (A n)
      by auto
    show outer_measure_of M (A n) ≤ outer_measure_of M (F n)
      by (intro outer_measure_of_mono ‹A n ⊆ F n›)
  qed

  have outer_measure_of M (⋃n. A n) ≤ outer_measure_of M (⋃n. F n)
    using ‹⋀n. A n ⊆ F n› by (intro outer_measure_of_mono) auto
  also have ... = (SUP n. emeasure M (F n))
    using F by (simp add: SUP_emeasure_incseq subset_eq)
  finally show outer_measure_of M (⋃n. A n) ≤ (SUP n. outer_measure_of M
(A n))
    by (simp add: eq F)
qed (auto intro: SUP_least outer_measure_of_mono)

definition measurable_envelope :: 'a measure ⇒ 'a set ⇒ 'a set ⇒ bool
  where measurable_envelope M A E ⟷
   (A ⊆ E ∧ E ∈ sets M ∧ (∀F∈sets M. emeasure M (F ∩ E) = outer_measure_of
M (F ∩ A)))

lemma measurable_envelopeD:
  assumes measurable_envelope M A E
  shows A ⊆ E
    and E ∈ sets M
    and ⋀F. F ∈ sets M ⟹ emeasure M (F ∩ E) = outer_measure_of M (F ∩
A)
    and A ⊆ space M
 using assms sets.sets_into_space[of E] by (auto simp: measurable_envelope_def)

lemma measurable_envelopeD1:
  assumes E: measurable_envelope M A E and F: F ∈ sets M F ⊆ E − A
  shows emeasure M F = 0
proof −
  have emeasure M F = emeasure M (F ∩ E)
    using F by (intro arg_cong2[where f=emeasure]) auto
  also have ... = outer_measure_of M (F ∩ A)
```

using *measurable_envelopeD*[*OF E*] ‹*F* ∈ *sets M*› **by** (*auto simp*: *measurable_envelope_def*)
also have ... = *outer_measure_of M* {}
using ‹*F* ⊆ *E* − *A*› **by** (*intro arg_cong2*[**where** *f*=*outer_measure_of*]) *auto*
finally show *emeasure M F* = *0*
by *simp*
qed

lemma *measurable_envelope_eq1*:
assumes *A* ⊆ *E E* ∈ *sets M*
shows *measurable_envelope M A E* ⟷ (∀ *F*∈*sets M*. *F* ⊆ *E* − *A* ⟶ *emeasure M F* = *0*)
proof *safe*
assume ∗: ∀ *F*∈*sets M*. *F* ⊆ *E* − *A* ⟶ *emeasure M F* = *0*
show *measurable_envelope M A E*
unfolding *measurable_envelope_def*
proof (*rule ccontr*, *auto simp add*: ‹*E* ∈ *sets M*› ‹*A* ⊆ *E*›)
fix *F* **assume** *F* ∈ *sets M emeasure M* (*F* ∩ *E*) ≠ *outer_measure_of M* (*F* ∩ *A*)
then have *outer_measure_of M* (*F* ∩ *A*) < *emeasure M* (*F* ∩ *E*)
using *outer_measure_of_mono*[*of F* ∩ *A F* ∩ *E M*] ‹*A* ⊆ *E*› ‹*E* ∈ *sets M*› **by** (*auto simp*: *less_le*)
then obtain *G* **where** *G*: *G* ∈ *sets M F* ∩ *A* ⊆ *G* **and** *less*: *emeasure M G* < *emeasure M* (*E* ∩ *F*)
unfolding *outer_measure_of_def INF_less_iff* **by** (*auto simp*: *ac_simps*)
have *le*: *emeasure M* (*G* ∩ *E* ∩ *F*) ≤ *emeasure M G*
using ‹*E* ∈ *sets M*› ‹*G* ∈ *sets M*› ‹*F* ∈ *sets M*› **by** (*auto intro*!: *emeasure_mono*)

from *G* **have** *E* ∩ *F* − *G* ∈ *sets M E* ∩ *F* − *G* ⊆ *E* − *A*
using ‹*F* ∈ *sets M*› ‹*E* ∈ *sets M*› **by** *auto*
with ∗ **have** *0* = *emeasure M* (*E* ∩ *F* − *G*)
by *auto*
also have *E* ∩ *F* − *G* = *E* ∩ *F* − (*G* ∩ *E* ∩ *F*)
by *auto*
also have *emeasure M* (*E* ∩ *F* − (*G* ∩ *E* ∩ *F*)) = *emeasure M* (*E* ∩ *F*) − *emeasure M* (*G* ∩ *E* ∩ *F*)
using ‹*E* ∈ *sets M*› ‹*F* ∈ *sets M*› *le less G* **by** (*intro emeasure_Diff*) (*auto simp*: *top_unique*)
also have ... > *0*
using *le less* **by** (*intro diff_gr0_ennreal*) *auto*
finally show *False* **by** *auto*
qed
qed (*rule measurable_envelopeD1*)

lemma *measurable_envelopeD2*:
assumes *E*: *measurable_envelope M A E* **shows** *emeasure M E* = *outer_measure_of M A*
proof −

from ‹*measurable_envelope M A E*› **have** *emeasure M* (*E* ∩ *E*) = *outer_measure_of M* (*E* ∩ *A*)
by (*auto simp*: *measurable_envelope_def*)
with *measurable_envelopeD*[*OF E*] **show** *emeasure M E* = *outer_measure_of M A*
by (*auto simp*: *Int_absorb1*)
qed

lemma *measurable_envelope_eq2*:
assumes *A* ⊆ *E E* ∈ *sets M emeasure M E* < ∞
shows *measurable_envelope M A E* ⟷ (*emeasure M E* = *outer_measure_of M A*)
proof *safe*
assume ∗: *emeasure M E* = *outer_measure_of M A*
show *measurable_envelope M A E*
unfolding *measurable_envelope_eq1*[*OF* ‹*A* ⊆ *E*› ‹*E* ∈ *sets M*›]
proof (*intro conjI ballI impI assms*)
fix *F* **assume** *F*: *F* ∈ *sets M F* ⊆ *E* − *A*
with ‹*E* ∈ *sets M*› **have** *le*: *emeasure M F* ≤ *emeasure M E*
by (*intro emeasure_mono*) *auto*
from *F* ‹*A* ⊆ *E*› **have** *outer_measure_of M A* ≤ *outer_measure_of M* (*E* − *F*)
by (*intro outer_measure_of_mono*) *auto*
then have *emeasure M E* − *0* ≤ *emeasure M* (*E* − *F*)
using ∗ ‹*E* ∈ *sets M*› ‹*F* ∈ *sets M*› **by** *simp*
also have ... = *emeasure M E* − *emeasure M F*
using ‹*E* ∈ *sets M*› ‹*emeasure M E* < ∞› *F le* **by** (*intro emeasure_Diff*) (*auto simp*: *top_unique*)
finally show *emeasure M F* = *0*
using *ennreal_mono_minus_cancel*[*of emeasure M E 0 emeasure M F*] *le assms* **by** *auto*
qed
qed (*auto intro*: *measurable_envelopeD2*)

lemma *measurable_envelopeI_countable*:
fixes *A* :: *nat* ⇒ *'a set*
assumes *E*: ⋀*n*. *measurable_envelope M* (*A n*) (*E n*)
shows *measurable_envelope M* (⋃*n*. *A n*) (⋃*n*. *E n*)
proof (*subst measurable_envelope_eq1*)
show (⋃*n*. *A n*) ⊆ (⋃*n*. *E n*) (⋃*n*. *E n*) ∈ *sets M*
using *measurable_envelopeD*(*1*,*2*)[*OF E*] **by** *auto*
show ∀ *F*∈*sets M*. *F* ⊆ (⋃*n*. *E n*) − (⋃*n*. *A n*) ⟶ *emeasure M F* = *0*
proof *safe*
fix *F* **assume** *F*: *F* ∈ *sets M F* ⊆ (⋃*n*. *E n*) − (⋃*n*. *A n*)
then have *F* ∩ *E n* ∈ *sets M F* ∩ *E n* ⊆ *E n* − *A n F* ⊆ (⋃*n*. *E n*) **for** *n*
using *measurable_envelopeD*(*1*,*2*)[*OF E*] **by** *auto*
then have *emeasure M* (⋃*n*. *F* ∩ *E n*) = *0*
by (*intro emeasure_UN_eq_0 measurable_envelopeD1*[*OF E*]) *auto*
then show *emeasure M F* = *0*

```
    using ‹F ⊆ (⋃n. E n)› by (auto simp: Int_absorb2)
  qed
qed

lemma measurable_envelopeI_countable_cover:
  fixes A and C :: nat ⇒ 'a set
  assumes C: A ⊆ (⋃n. C n) ⋀n. C n ∈ sets M ⋀n. emeasure M (C n) < ∞
  shows ∃E⊆(⋃n. C n). measurable_envelope M A E
proof −
  have A ∩ C n ⊆ space M for n
    using ‹C n ∈ sets M› by (auto dest: sets.sets_into_space)
  then have ∀n. ∃E∈sets M. A ∩ C n ⊆ E ∧ outer_measure_of M (A ∩ C n)
= emeasure M E
    using outer_measure_of_attain[of A ∩ C n M for n] by auto
  then obtain E
    where E: ⋀n. E n ∈ sets M ⋀n. A ∩ C n ⊆ E n
      and eq: ⋀n. outer_measure_of M (A ∩ C n) = emeasure M (E n)
    by metis

  have outer_measure_of M (A ∩ C n) ≤ outer_measure_of M (E n ∩ C n) for
n
    using E by (intro outer_measure_of_mono) auto
  moreover have outer_measure_of M (E n ∩ C n) ≤ outer_measure_of M (E
n) for n
    by (intro outer_measure_of_mono) auto
  ultimately have eq: outer_measure_of M (A ∩ C n) = emeasure M (E n ∩ C
n) for n
    using E C by (intro antisym) (auto simp: eq)

  { fix n
    have outer_measure_of M (A ∩ C n) ≤ outer_measure_of M (C n)
      by (intro outer_measure_of_mono) simp
    also have ... < ∞
      using assms by auto
    finally have emeasure M (E n ∩ C n) < ∞
      using eq by simp }
  then have measurable_envelope M (⋃n. A ∩ C n) (⋃n. E n ∩ C n)
   using E C by (intro measurable_envelopeI_countable measurable_envelope_eq2[THEN
iffD2]) (auto simp: eq)
  with ‹A ⊆ (⋃n. C n)› show ?thesis
    by (intro exI[of _ (⋃n. E n ∩ C n)]) (auto simp add: Int_absorb2)
qed

lemma (in complete_measure) complete_sets_sandwich:
  assumes [measurable]: A ∈ sets M C ∈ sets M and subset: A ⊆ B B ⊆ C
    and measure: emeasure M A = emeasure M C emeasure M A < ∞
  shows B ∈ sets M
proof −
  have B − A ∈ sets M
```

proof (*rule complete*)
show $B - A \subseteq C - A$
using *subset* **by** *auto*
show $C - A \in$ *null_sets M*
using *measure subset* **by**(*simp add*: *emeasure_Diff null_setsI*)
qed
then have $A \cup (B - A) \in$ *sets M*
by *measurable*
also have $A \cup (B - A) = B$
using ‹$A \subseteq B$› **by** *auto*
finally show *?thesis* **.**
qed

lemma (**in** *complete_measure*) *complete_sets_sandwich_fmeasurable*:
assumes [*measurable*]: $A \in$ *fmeasurable M* $C \in$ *fmeasurable M* **and** *subset*: $A \subseteq B$ $B \subseteq C$
and *measure*: *measure M A* = *measure M C*
shows $B \in$ *fmeasurable M*
proof (*rule fmeasurableI2*)
show $B \subseteq C$ $C \in$ *fmeasurable M* **by** *fact+*
show $B \in$ *sets M*
proof (*rule complete_sets_sandwich*)
show $A \in$ *sets M* $C \in$ *sets M* $A \subseteq B$ $B \subseteq C$
using *assms* **by** *auto*
show *emeasure M A* $< \infty$
using ‹$A \in$ *fmeasurable M*› **by** (*auto simp*: *fmeasurable_def*)
show *emeasure M A* = *emeasure M C*
using *assms* **by** (*simp add*: *emeasure_eq_measure2*)
qed
qed

lemma *AE_completion_iff*: (*AE x in completion M. P x*) $\longleftrightarrow$ (*AE x in M. P x*)
proof
assume *AE x in completion M. P x*
then obtain *N* **where** $N \in$ *null_sets* (*completion M*) **and** *P*: $\{x \in$*space M*$. \neg P\ x\} \subseteq N$
unfolding *eventually_ae_filter* **by** *auto*
then obtain N' **where** N': $N' \in$ *null_sets M* **and** $N \subseteq N'$
unfolding *null_sets_completion_iff2* **by** *auto*
from *P* ‹$N \subseteq N'$› **have** $\{x \in$*space M*$. \neg P\ x\} \subseteq N'$
by *auto*
with N' **show** *AE x in M. P x*
unfolding *eventually_ae_filter* **by** *auto*
qed (*rule AE_completion*)

lemma *null_part_null_sets*: $S \in$ *completion M* $\Longrightarrow$ *null_part M S* $\in$ *null_sets* (*completion M*)
by (*auto dest*!: *null_part intro*: *null_sets_completionI null_sets_completion_subset*)

lemma *AE_notin_null_part*: $S \in$ *completion* $M \Longrightarrow$ (*AE x in M*. $x \notin$ *null_part M S*)
by (*auto dest*!: *null_part_null_sets AE_not_in simp*: *AE_completion_iff*)

lemma *completion_upper*:
assumes *A*: $A \in$ *sets* (*completion M*)
shows $\exists A' \in$*sets M*. $A \subseteq A' \wedge$ *emeasure* (*completion M*) $A =$ *emeasure M* A'
proof −
from *AE_notin_null_part*[*OF A*] **obtain** *N* **where** *N*: $N \in$ *null_sets M null_part M A* $\subseteq N$
unfolding *eventually_ae_filter* **using** *null_part_null_sets*[*OF A*, *THEN null_setsD2*, *THEN sets.sets_into_space*] **by** *auto*
show *?thesis*
proof (*intro bexI conjI*)
show $A \subseteq$ *main_part M A* $\cup$ *N*
using ‹*null_part M A* $\subseteq N$› **by** (*subst main_part_null_part_Un*[*symmetric*, *OF A*]) *auto*
show *emeasure* (*completion M*) $A =$ *emeasure M* (*main_part M A* $\cup$ *N*)
using *A* ‹$N \in$ *null_sets M*› **by** (*simp add*: *emeasure_Un_null_set*)
qed (*use A N* **in** *auto*)
qed

lemma *AE_in_main_part*:
assumes *A*: $A \in$ *completion M* **shows** *AE x in M*. $x \in$ *main_part M A* $\longleftrightarrow x \in A$
using *AE_notin_null_part*[*OF A*]
by (*subst* (*2*) *main_part_null_part_Un*[*symmetric*, *OF A*]) *auto*

lemma *completion_density_eq*:
assumes *ae*: *AE x in M*. $0 <$ *f x* **and** [*measurable*]: $f \in M \to_M$ *borel*
shows *completion* (*density M f*) = *density* (*completion M*) *f*
proof (*intro measure_eqI*)
have $N' \in$ *sets M* $\wedge$ (*AE x*$\in N'$ *in M*. *f x* = *0*) $\longleftrightarrow N' \in$ *null_sets M* **for** N'
proof *safe*
assume N': $N' \in$ *sets M* **and** *ae_N'*: *AE x*$\in N'$ *in M*. *f x* = *0*
from *ae_N' ae* **have** *AE x in M*. $x \notin N'$
by *eventually_elim auto*
then show $N' \in$ *null_sets M*
using N' **by** (*simp add*: *AE_iff_null_sets*)
next
assume N': $N' \in$ *null_sets M* **then show** $N' \in$ *sets M AE x*$\in N'$ *in M*. *f x* = *0*
using *ae AE_not_in*[*OF N'*] **by** (*auto simp*: *less_le*)
qed
then show *sets_eq*: *sets* (*completion* (*density M f*)) = *sets* (*density* (*completion M*) *f*)
by (*simp add*: *sets_completion null_sets_density_iff*)

fix *A* **assume** *A*: ‹*A* $\in$ *completion* (*density M f*)›

moreover
have $A \in$ *completion* M
using A **unfolding** *sets_eq* **by** *simp*
moreover
have *main_part* (*density* $M\ f$) $A \in M$
using A *main_part_sets*[*of* A *density* $M\ f$] **unfolding** *sets_density sets_eq* **by** *simp*
moreover have *AE* x *in* M. $x \in$ *main_part* (*density* $M\ f$) $A \longleftrightarrow x \in A$
using *AE_in_main_part*[*OF* ‹$A \in$ *completion* (*density* $M\ f$)›] *ae* **by** (*auto simp add*: *AE_density*)
ultimately show *emeasure* (*completion* (*density* $M\ f$)) $A =$ *emeasure* (*density* (*completion* M) f) A
by (*auto simp add*: *emeasure_density measurable_completion nn_integral_completion intro*!: *nn_integral_cong_AE*)
qed

lemma *null_sets_subset*: $B \in$ *null_sets* $M \Longrightarrow A \in$ *sets* $M \Longrightarrow A \subseteq B \Longrightarrow A \in$ *null_sets* M
using *emeasure_mono*[*of* A B M] **by** (*simp add*: *null_sets_def*)

lemma (**in** *complete_measure*) *complete2*: $A \subseteq B \Longrightarrow B \in$ *null_sets* $M \Longrightarrow A \in$ *null_sets* M
using *complete*[*of* A B] *null_sets_subset*[*of* B M A] **by** *simp*

lemma (**in** *complete_measure*) *AE_iff_null_sets*: (*AE* x *in* M. $P\ x$) $\longleftrightarrow$ {x∈*space* M. ¬ $P\ x$} ∈ *null_sets* M
unfolding *eventually_ae_filter* **by** (*auto intro*: *complete2*)

lemma (**in** *complete_measure*) *null_sets_iff_AE*: $A \in$ *null_sets* $M \longleftrightarrow$ ((*AE* x *in* M. $x \notin A$) $\wedge$ $A \subseteq$ *space* M)
unfolding *AE_iff_null_sets* **by** (*auto cong*: *rev_conj_cong dest*: *sets.sets_into_space simp*: *subset_eq*)

lemma (**in** *complete_measure*) *in_sets_AE*:
assumes *ae*: *AE* x *in* M. $x \in A \longleftrightarrow x \in B$ **and** A: $A \in$ *sets* M **and** B: $B \subseteq$ *space* M
shows $B \in$ *sets* M
proof −
have (*AE* x *in* M. $x \notin B - A \wedge x \notin A - B$)
using *ae* **by** *eventually_elim auto*
then have $B - A \in$ *null_sets* M $A - B \in$ *null_sets* M
using A B **unfolding** *null_sets_iff_AE* **by** (*auto dest*: *sets.sets_into_space*)
then have $A \cup (B - A) - (A - B) \in$ *sets* M
using A **by** *blast*
also have $A \cup (B - A) - (A - B) = B$
by *auto*
finally show $B \in$ *sets* M **.**
qed

lemma (**in** *complete_measure*) *vimage_null_part_null_sets*:
assumes *f*: $f \in M \to_M N$ **and** *eq*: *null_sets N* $\subseteq$ *null_sets* (*distr M N f*)
and *A*: $A \in$ *completion N*
shows $f -` \mathit{null\_part}\ N\ A \cap \mathit{space}\ M \in \mathit{null\_sets}\ M$
proof −
obtain N' **where** $N' \in \mathit{null\_sets}\ N\ \mathit{null\_part}\ N\ A \subseteq N'$
using *null_part*[*OF A*] **by** *auto*
then have N': $N' \in \mathit{null\_sets}$ (*distr M N f*)
using *eq* **by** *auto*
show *?thesis*
proof (*rule complete2*)
show $f -` \mathit{null\_part}\ N\ A \cap \mathit{space}\ M \subseteq f -` N' \cap \mathit{space}\ M$
using ‹*null_part N A* $\subseteq N'$› **by** *auto*
show $f -` N' \cap \mathit{space}\ M \in \mathit{null\_sets}\ M$
using *f N'* **by** (*auto simp*: *null_sets_def emeasure_distr*)
qed
qed

lemma (**in** *complete_measure*) *vimage_null_part_sets*:
$f \in M \to_M N \Longrightarrow \mathit{null\_sets}\ N \subseteq \mathit{null\_sets}$ (*distr M N f*) $\Longrightarrow A \in \mathit{completion}\ N \Longrightarrow$
$f -` \mathit{null\_part}\ N\ A \cap \mathit{space}\ M \in \mathit{sets}\ M$
using *vimage_null_part_null_sets*[*of f N A*] **by** *auto*

lemma (**in** *complete_measure*) *measurable_completion2*:
assumes *f*: $f \in M \to_M N$ **and** *null_sets*: *null_sets N* $\subseteq$ *null_sets* (*distr M N f*)
shows $f \in M \to_M \mathit{completion}\ N$
proof (*rule measurableI*)
show $x \in \mathit{space}\ M \Longrightarrow f\ x \in \mathit{space}$ (*completion N*) **for** *x*
using *f*[*THEN measurable_space*] **by** *auto*
fix *A* **assume** *A*: $A \in \mathit{sets}$ (*completion N*)
have $f -` A \cap \mathit{space}\ M = (f -` \mathit{main\_part}\ N\ A \cap \mathit{space}\ M) \cup (f -` \mathit{null\_part}\ N\ A \cap \mathit{space}\ M)$
using *main_part_null_part_Un*[*OF A*] **by** *auto*
then show $f -` A \cap \mathit{space}\ M \in \mathit{sets}\ M$
using *f A null_sets* **by** (*auto intro*: *vimage_null_part_sets measurable_sets*)
qed

lemma (**in** *complete_measure*) *completion_distr_eq*:
assumes *X*: $X \in M \to_M N$ **and** *null_sets*: *null_sets* (*distr M N X*) = *null_sets N*
shows *completion* (*distr M N X*) = *distr M* (*completion N*) *X*
proof (*rule measure_eqI*)
show *eq*: *sets* (*completion* (*distr M N X*)) = *sets* (*distr M* (*completion N*) *X*)
by (*simp add*: *sets_completion null_sets*)

fix *A* **assume** *A*: $A \in$ *completion* (*distr M N X*)
moreover have A': $A \in \mathit{completion}\ N$

using *A* **by** (*simp add*: *eq*)
moreover have *main_part* (*distr M N X*) *A* ∈ *sets N*
using *main_part_sets*[*OF A*] **by** *simp*
moreover have *X* −‘ *A* ∩ *space M* = (*X* −‘ *main_part* (*distr M N X*) *A* ∩ *space M*) ∪ (*X* −‘ *null_part* (*distr M N X*) *A* ∩ *space M*)
using *main_part_null_part_Un*[*OF A*] **by** *auto*
moreover have *X* −‘ *null_part* (*distr M N X*) *A* ∩ *space M* ∈ *null_sets M*
using *X A* **by** (*intro vimage_null_part_null_sets*) (*auto cong*: *distr_cong*)
ultimately show *emeasure* (*completion* (*distr M N X*)) *A* = *emeasure* (*distr M* (*completion N*) *X*) *A*
using *X* **by** (*auto simp*: *emeasure_distr measurable_completion null_sets measurable_completion2*
intro!: *emeasure_Un_null_set*[*symmetric*])
qed

lemma *distr_completion*: *X* ∈ *M* →$_M$ *N* ⟹ *distr* (*completion M*) *N X* = *distr M N X*
by (*intro measure_eqI*) (*auto simp*: *emeasure_distr measurable_completion*)

proposition (**in** *complete_measure*) *fmeasurable_inner_outer*:
S ∈ *fmeasurable M* ⟷
(∀ *e*>*0*. ∃ *T*∈*fmeasurable M*. ∃ *U*∈*fmeasurable M*. *T* ⊆ *S* ∧ *S* ⊆ *U* ∧ |*measure M T* − *measure M U*| < *e*)
(**is** _ ⟷ *?approx*)
proof *safe*
let *?μ* = *measure M* **let** *?D* = λ*T U* . |*?μ T* − *?μ U*|
assume *?approx*
have ∃ *A*. ∀ *n*. (*fst* (*A n*) ∈ *fmeasurable M* ∧ *snd* (*A n*) ∈ *fmeasurable M* ∧ *fst* (*A n*) ⊆ *S* ∧ *S* ⊆ *snd* (*A n*) ∧
?D (*fst* (*A n*)) (*snd* (*A n*)) < *1*/*Suc n*) ∧ (*fst* (*A n*) ⊆ *fst* (*A* (*Suc n*)) ∧ *snd* (*A* (*Suc n*)) ⊆ *snd* (*A n*))
(**is** ∃ *A*. ∀ *n*. *?P n* (*A n*) ∧ *?Q* (*A n*) (*A* (*Suc n*)))
proof (*intro dependent_nat_choice*)
show ∃ *A*. *?P 0 A*
using ‹*?approx*›[*THEN spec*, *of 1*] **by** *auto*
next
fix *A n* **assume** *?P n A*
moreover
from ‹*?approx*›[*THEN spec*, *of 1*/*Suc* (*Suc n*)]
obtain *T U* **where** ∗: *T* ∈ *fmeasurable M U* ∈ *fmeasurable M T* ⊆ *S S* ⊆ *U* *?D T U* < *1* / *Suc* (*Suc n*)
by *auto*
ultimately have *?μ T* ≤ *?μ* (*T* ∪ *fst A*) *?μ* (*U* ∩ *snd A*) ≤ *?μ U*
?μ T ≤ *?μ U* *?μ* (*T* ∪ *fst A*) ≤ *?μ* (*U* ∩ *snd A*)
by (*auto intro*!: *measure_mono_fmeasurable*)
then have *?D* (*T* ∪ *fst A*) (*U* ∩ *snd A*) ≤ *?D T U*
by *auto*
also have *?D T U* < *1*/*Suc* (*Suc n*) **by** *fact*
finally show ∃ *B*. *?P* (*Suc n*) *B* ∧ *?Q A B*

```
    using ‹?P n A› *
    by (intro exI[of _ (T ∪ fst A, U ∩ snd A)] conjI) auto
  qed
  then obtain A
    where lm: ⋀n. fst (A n) ∈ fmeasurable M ⋀n. snd (A n) ∈ fmeasurable M
      and set_bound: ⋀n. fst (A n) ⊆ S ⋀n. S ⊆ snd (A n)
      and mono: ⋀n. fst (A n) ⊆ fst (A (Suc n)) ⋀n. snd (A (Suc n)) ⊆ snd (A
n)
      and bound: ⋀n. ?D (fst (A n)) (snd (A n)) < 1/Suc n
    by metis

  have INT_sA: (⋂n. snd (A n)) ∈ fmeasurable M
    using lm by (intro fmeasurable_INT[of _ 0]) auto
  have UN_fA: (⋃n. fst (A n)) ∈ fmeasurable M
    using lm order_trans[OF set_bound(1) set_bound(2)[of 0]] by (intro fmea-
surable_UN[of _ _ snd (A 0)]) auto

  have (λn. ?μ (fst (A n)) − ?μ (snd (A n))) ⟶ 0
    using bound
    by (subst tendsto_rabs_zero_iff[symmetric])
     (intro tendsto_sandwich[OF _ _ tendsto_const LIMSEQ_inverse_real_of_nat];
        auto intro!: always_eventually less_imp_le simp: divide_inverse)
  moreover
  have (λn. ?μ (fst (A n)) − ?μ (snd (A n))) ⟶ ?μ (⋃n. fst (A n)) − ?μ
(⋂n. snd (A n))
  proof (intro tendsto_diff Lim_measure_incseq Lim_measure_decseq)
    show range (λi. fst (A i)) ⊆ sets M range (λi. snd (A i)) ⊆ sets M
      incseq (λi. fst (A i)) decseq (λn. snd (A n))
      using mono lm by (auto simp: incseq_Suc_iff decseq_Suc_iff intro!: mea-
sure_mono_fmeasurable)
    show emeasure M (⋃x. fst (A x)) ≠ ∞ emeasure M (snd (A n)) ≠ ∞ for n
      using lm(2)[of n] UN_fA by (auto simp: fmeasurable_def)
  qed
  ultimately have eq: 0 = ?μ (⋃n. fst (A n)) − ?μ (⋂n. snd (A n))
    by (rule LIMSEQ_unique)

  show S ∈ fmeasurable M
    using UN_fA INT_sA
  proof (rule complete_sets_sandwich_fmeasurable)
    show (⋃n. fst (A n)) ⊆ S S ⊆ (⋂n. snd (A n))
      using set_bound by auto
    show ?μ (⋃n. fst (A n)) = ?μ (⋂n. snd (A n))
      using eq by auto
  qed
qed (auto intro!: bexI[of _ S])

lemma (in complete_measure) fmeasurable_measure_inner_outer:
  (S ∈ fmeasurable M ∧ m = measure M S) ⟷
    (∀ e>0. ∃ T∈fmeasurable M. T ⊆ S ∧ m − e < measure M T) ∧
```

$(\forall\, e{>}0.\ \exists\, U{\in}\mathit{fmeasurable}\ M.\ S \subseteq U \wedge \mathit{measure}\ M\ U < m + e)$
(**is** *?lhs* = *?rhs*)
proof
assume *RHS*: *?rhs*
then have *T*: $\bigwedge e.\ 0 < e \longrightarrow (\exists\, T{\in}\mathit{fmeasurable}\ M.\ T \subseteq S \wedge m - e < \mathit{measure}\ M\ T)$
and *U*: $\bigwedge e.\ 0 < e \longrightarrow (\exists\, U{\in}\mathit{fmeasurable}\ M.\ S \subseteq U \wedge \mathit{measure}\ M\ U < m + e)$
by *auto*
have $S \in \mathit{fmeasurable}\ M$
proof (*subst fmeasurable_inner_outer*, *safe*)
fix *e*::*real* **assume** $0 < e$
with *RHS* **obtain** *T U* **where** *T*: $T \in \mathit{fmeasurable}\ M\ T \subseteq S\ m - e/2 < \mathit{measure}\ M\ T$
and *U*: $U \in \mathit{fmeasurable}\ M\ S \subseteq U\ \mathit{measure}\ M\ U < m + e/2$
by (*meson half_gt_zero*)+
moreover have $\mathit{measure}\ M\ U - \mathit{measure}\ M\ T < (m + e/2) - (m - e/2)$
by (*intro diff_strict_mono*) *fact*+
moreover have $\mathit{measure}\ M\ T \leq \mathit{measure}\ M\ U$
using *T U* **by** (*intro measure_mono_fmeasurable*) *auto*
ultimately show $\exists\, T{\in}\mathit{fmeasurable}\ M.\ \exists\, U{\in}\mathit{fmeasurable}\ M.\ T \subseteq S \wedge S \subseteq U \wedge |\mathit{measure}\ M\ T - \mathit{measure}\ M\ U| < e$
apply (*rule_tac bexI*[*OF* _ ‹$T \in \mathit{fmeasurable}\ M$›])
apply (*rule_tac bexI*[*OF* _ ‹$U \in \mathit{fmeasurable}\ M$›])
by *auto*
qed
moreover have $m = \mathit{measure}\ M\ S$
using ‹$S \in \mathit{fmeasurable}\ M$› *U*[*of measure M S − m*] *T*[*of m − measure M S*]
by (*cases m measure M S rule*: *linorder_cases*)
(*auto simp*: *not_le*[*symmetric*] *measure_mono_fmeasurable*)
ultimately show *?lhs*
by *simp*
qed (*auto intro*!: *bexI*[*of* _ *S*])

lemma (**in** *complete_measure*) *null_sets_outer*:
$S \in \mathit{null\_sets}\ M \longleftrightarrow (\forall\, e{>}0.\ \exists\, T{\in}\mathit{fmeasurable}\ M.\ S \subseteq T \wedge \mathit{measure}\ M\ T < e)$
proof −
have $S \in \mathit{null\_sets}\ M \longleftrightarrow (S \in \mathit{fmeasurable}\ M \wedge 0 = \mathit{measure}\ M\ S)$
by (*auto simp*: *null_sets_def emeasure_eq_measure2 intro*: *fmeasurableI*) (*simp add*: *measure_def*)
also have ... = $(\forall\, e{>}0.\ \exists\, T{\in}\mathit{fmeasurable}\ M.\ S \subseteq T \wedge \mathit{measure}\ M\ T < e)$
unfolding *fmeasurable_measure_inner_outer* **by** *auto*
finally show *?thesis* **.**
qed

lemma (**in** *complete_measure*) *fmeasurable_measure_inner_outer_le*:
$(S \in \mathit{fmeasurable}\ M \wedge m = \mathit{measure}\ M\ S) \longleftrightarrow$
$(\forall\, e{>}0.\ \exists\, T{\in}\mathit{fmeasurable}\ M.\ T \subseteq S \wedge m - e \leq \mathit{measure}\ M\ T) \wedge$
$(\forall\, e{>}0.\ \exists\, U{\in}\mathit{fmeasurable}\ M.\ S \subseteq U \wedge \mathit{measure}\ M\ U \leq m + e)$ (**is** *?T1*)

```
  and null_sets_outer_le:
    S ∈ null_sets M ⟷ (∀ e>0. ∃ T∈fmeasurable M. S ⊆ T ∧ measure M T ≤
e) (is ?T2)
proof −
  have e > 0 ∧ m − e/2 ≤ t ⟹ m − e < t
       e > 0 ∧ t ≤ m + e/2 ⟹ t < m + e
       e > 0 ⟷ e/2 > 0
       for e t
    by auto
  then show ?T1 ?T2
    unfolding fmeasurable_measure_inner_outer null_sets_outer
    by (meson dense le_less_trans less_imp_le)+
qed

lemma (in cld_measure) notin_sets_outer_measure_of_cover:
  assumes E: E ⊆ space M E ∉ sets M
  shows ∃ B∈sets M. 0 < emeasure M B ∧ emeasure M B < ∞ ∧
    outer_measure_of M (B ∩ E) = emeasure M B ∧ outer_measure_of M (B −
E) = emeasure M B
proof −
  from locally_determined[OF ‹E ⊆ space M›] ‹E ∉ sets M›
  obtain F
    where [measurable]: F ∈ sets M and emeasure M F < ∞ E ∩ F ∉ sets M
    by blast
  then obtain H H′
    where H: measurable_envelope M (F ∩ E) H and H′: measurable_envelope
M (F − E) H′
    using measurable_envelopeI_countable_cover[of F ∩ E λ_. F M]
      measurable_envelopeI_countable_cover[of F − E λ_. F M]
    by auto
 note measurable_envelopeD(2)[OF H′, measurable] measurable_envelopeD(2)[OF
H, measurable]

  from measurable_envelopeD(1)[OF H′] measurable_envelopeD(1)[OF H]
  have subset: F − H′ ⊆ F ∩ E F ∩ E ⊆ F ∩ H
    by auto
  moreover define G where G = (F ∩ H) − (F − H′)
  ultimately have G: G = F ∩ H ∩ H′
    by auto
  have emeasure M (F ∩ H) ≠ 0
  proof
    assume emeasure M (F ∩ H) = 0
    then have F ∩ H ∈ null_sets M
      by auto
    with ‹E ∩ F ∉ sets M› show False
      using complete[OF ‹F ∩ E ⊆ F ∩ H›] by (auto simp: Int_commute)
  qed
  moreover
  have emeasure M (F − H′) ≠ emeasure M (F ∩ H)
```

proof
assume *emeasure M* $(F - H') =$ *emeasure M* $(F \cap H)$
with ‹$E \cap F \notin$ *sets M*› *emeasure_mono*[*of* $F \cap H$ *F M*] ‹*emeasure M F* $< \infty$›
have $F \cap E \in$ *sets M*
by (*intro complete_sets_sandwich*[*OF _ _ subset*]) *auto*
with ‹$E \cap F \notin$ *sets M*› **show** *False*
by (*simp add*: *Int_commute*)
qed
moreover have *emeasure M* $(F - H') \leq$ *emeasure M* $(F \cap H)$
using *subset* **by** (*intro emeasure_mono*) *auto*
ultimately have *emeasure M G* $\neq 0$
unfolding *G_def* **using** *subset*
by (*subst emeasure_Diff*) (*auto simp*: *top_unique diff_eq_0_iff_ennreal*)
show *?thesis*
proof (*intro bexI conjI*)
have *emeasure M G* $\leq$ *emeasure M F*
unfolding *G* **by** (*auto intro*!: *emeasure_mono*)
with ‹*emeasure M F* $< \infty$› **show** $0 <$ *emeasure M G emeasure M G* $< \infty$
using ‹*emeasure M G* $\neq 0$› **by** (*auto simp*: *zero_less_iff_neq_zero*)
show [*measurable*]: *G* $\in$ *sets M*
unfolding *G* **by** *auto*

have *emeasure M G* $=$ *outer_measure_of M* $(F \cap H' \cap (F \cap E))$
using *measurable_envelopeD*(*3*)[*OF H*, *of* $F \cap H'$] **unfolding** *G* **by** (*simp add*: *ac_simps*)
also have $\ldots \leq$ *outer_measure_of M* $(G \cap E)$
using *measurable_envelopeD*(*1*)[*OF H*] **by** (*intro outer_measure_of_mono*) (*auto simp*: *G*)
finally show *outer_measure_of M* $(G \cap E) =$ *emeasure M G*
using *outer_measure_of_mono*[*of* $G \cap E$ *G M*] **by** *auto*

have *emeasure M G* $=$ *outer_measure_of M* $(F \cap H \cap (F - E))$
using *measurable_envelopeD*(*3*)[*OF H'*, *of* $F \cap H$] **unfolding** *G* **by** (*simp add*: *ac_simps*)
also have $\ldots \leq$ *outer_measure_of M* $(G - E)$
using *measurable_envelopeD*(*1*)[*OF H'*] **by** (*intro outer_measure_of_mono*) (*auto simp*: *G*)
finally show *outer_measure_of M* $(G - E) =$ *emeasure M G*
using *outer_measure_of_mono*[*of* $G - E$ *G M*] **by** *auto*
qed
qed

The following theorem is a specialization of D.H. Fremlin, Measure Theory vol 4I (413G). We only show one direction and do not use a inner regular family *K*.

lemma (**in** *cld_measure*) *borel_measurable_cld*:
fixes *f* :: *'a* $\Rightarrow$ *real*
assumes $\bigwedge A\ a\ b.\ A \in$ *sets M* $\Longrightarrow 0 <$ *emeasure M A* $\Longrightarrow$ *emeasure M A* $< \infty$ $\Longrightarrow a < b \Longrightarrow$

```
    min (outer_measure_of M {x∈A. f x ≤ a}) (outer_measure_of M {x∈A. b
≤ f x}) < emeasure M A
  shows f ∈ M →M borel
proof (rule ccontr)
  let ?E = λa. {x∈space M. f x ≤ a} and ?F = λa. {x∈space M. a ≤ f x}

  assume f ∉ M →M borel
  then obtain a where ?E a ∉ sets M
    unfolding borel_measurable_iff_le by blast
  from notin_sets_outer_measure_of_cover[OF _ this]
  obtain K
    where K: K ∈ sets M 0 < emeasure M K emeasure M K < ∞
      and eq1: outer_measure_of M (K ∩ ?E a) = emeasure M K
      and eq2: outer_measure_of M (K − ?E a) = emeasure M K
    by auto
  then have me_K: measurable_envelope M (K ∩ ?E a) K
    by (subst measurable_envelope_eq2) auto

  define b where b n = a + inverse (real (Suc n)) for n
  have (SUP n. outer_measure_of M (K ∩ ?F (b n))) = outer_measure_of M
(⋃ n. K ∩ ?F (b n))
  proof (intro SUP_outer_measure_of_incseq)
    have x ≤ y ⟹ b y ≤ b x for x y
      by (auto simp: b_def field_simps)
    then show incseq (λn. K ∩ {x ∈ space M. b n ≤ f x})
      by (auto simp: incseq_def intro: order_trans)
  qed auto
  also have (⋃ n. K ∩ ?F (b n)) = K − ?E a
  proof −
    have b ⟶ a
      unfolding b_def by (rule LIMSEQ_inverse_real_of_nat_add)
    then have ∀ n. ¬ b n ≤ f x ⟹ f x ≤ a for x
      by (rule LIMSEQ_le_const) (auto intro: less_imp_le simp: not_le)
    moreover have ¬ b n ≤ a for n
      by (auto simp: b_def)
    ultimately show ?thesis
      using ‹K ∈ sets M›[THEN sets.sets_into_space] by (auto simp: subset_eq
intro: order_trans)
  qed
  finally have 0 < (SUP n. outer_measure_of M (K ∩ ?F (b n)))
    using K by (simp add: eq2)
  then obtain n where pos_b: 0 < outer_measure_of M (K ∩ ?F (b n)) and a
< b n
    unfolding less_SUP_iff by (auto simp: b_def)
  from measurable_envelopeI_countable_cover[of K ∩ ?F (b n) λ_. K M] K
  obtain K′ where K′ ⊆ K and me_K′: measurable_envelope M (K ∩ ?F (b
n)) K′
    by auto
  then have K′_le_K: emeasure M K′ ≤ emeasure M K
```

```
    by (intro emeasure_mono K)
  have K' ∈ sets M
    using me_K' by (rule measurable_envelopeD)

  have min (outer_measure_of M {x∈K'. f x ≤ a}) (outer_measure_of M {x∈K'.
b n ≤ f x}) < emeasure M K'
  proof (rule assms)
    show 0 < emeasure M K' emeasure M K' < ∞
      using measurable_envelopeD2[OF me_K'] pos_b K K'_le_K by auto
  qed fact+
  also have {x∈K'. f x ≤ a} = K' ∩ (K ∩ ?E a)
    using ‹K' ∈ sets M›[THEN sets.sets_into_space] ‹K' ⊆ K› by auto
  also have {x∈K'. b n ≤ f x} = K' ∩ (K ∩ ?F (b n))
    using ‹K' ∈ sets M›[THEN sets.sets_into_space] ‹K' ⊆ K› by auto
  finally have min (emeasure M K) (emeasure M K') < emeasure M K'
    unfolding
      measurable_envelopeD(3)[OF me_K ‹K' ∈ sets M›, symmetric]
      measurable_envelopeD(3)[OF me_K' ‹K' ∈ sets M›, symmetric]
    using ‹K' ⊆ K› by (simp add: Int_absorb1 Int_absorb2)
  with K'_le_K show False
    by (auto simp: min_def split: if_split_asm)
qed

end
```

6.11 Regularity of Measures

```
theory Regularity
imports Measure_Space Borel_Space
begin

theorem
  fixes M::'a::{second_countable_topology, complete_space} measure
  assumes sb: sets M = sets borel
  assumes emeasure M (space M) ≠ ∞
  assumes B ∈ sets borel
  shows inner_regular: emeasure M B =
    (SUP K ∈ {K. K ⊆ B ∧ compact K}. emeasure M K) (is ?inner B)
  and outer_regular: emeasure M B =
    (INF U ∈ {U. B ⊆ U ∧ open U}. emeasure M U) (is ?outer B)
proof −
  have Us: UNIV = space M by (metis assms(1) sets_eq_imp_space_eq space_borel)
  hence sU: space M = UNIV by simp
  interpret finite_measure M by rule fact
  have approx_inner: ⋀A. A ∈ sets M ⟹
    (⋀e. e > 0 ⟹ ∃K. K ⊆ A ∧ compact K ∧ emeasure M A ≤ emeasure M K
+ ennreal e) ⟹ ?inner A
    by (rule ennreal_approx_SUP)
    (force intro!: emeasure_mono simp: compact_imp_closed emeasure_eq_measure)+
```

```
  have approx_outer: ⋀A. A ∈ sets M ⟹
    (⋀e. e > 0 ⟹ ∃B. A ⊆ B ∧ open B ∧ emeasure M B ≤ emeasure M A +
ennreal e) ⟹ ?outer A
    by (rule ennreal_approx_INF)
       (force intro!: emeasure_mono simp: emeasure_eq_measure sb)+
  from countable_dense_setE obtain X :: 'a set
    where X: countable X ⋀Y :: 'a set. open Y ⟹ Y ≠ {} ⟹ ∃d∈X. d ∈ Y
    by auto
  {
    fix r::real assume r > 0 hence ⋀y. open (ball y r) ⋀y. ball y r ≠ {} by auto
    with X(2)[OF this]
    have x: space M = (⋃x∈X. cball x r)
      by (auto simp add: sU) (metis dist_commute order_less_imp_le)
    let ?U = ⋃k. (⋃n∈{0..k}. cball (from_nat_into X n) r)
    have (λk. emeasure M (⋃n∈{0..k}. cball (from_nat_into X n) r)) ⟶ M
?U
      by (rule Lim_emeasure_incseq) (auto intro!: borel_closed bexI simp: inc-
seq_def Us sb)
    also have ?U = space M
    proof safe
      fix x from X(2)[OF open_ball[of x r]] ‹r > 0› obtain d where d: d∈X d
∈ ball x r by auto
      show x ∈ ?U
        using X(1) d
        by simp (auto intro!: exI [where x = to_nat_on X d] simp: dist_commute
Bex_def)
    qed (simp add: sU)
    finally have (λk. M (⋃n∈{0..k}. cball (from_nat_into X n) r)) ⟶ M
(space M) .
  } note M_space = this
  {
    fix e ::real and n :: nat assume e > 0 n > 0
    hence 1/n > 0 e * 2 powr − n > 0 by (auto)
    from M_space[OF ‹1/n>0›]
    have (λk. measure M (⋃i∈{0..k}. cball (from_nat_into X i) (1/real n)))
⟶ measure M (space M)
      unfolding emeasure_eq_measure by (auto)
    from metric_LIMSEQ_D[OF this ‹0 < e * 2 powr −n›]
    obtain k where dist (measure M (⋃i∈{0..k}. cball (from_nat_into X i)
(1/real n))) (measure M (space M)) <
      e * 2 powr −n
      by auto
    hence measure M (⋃i∈{0..k}. cball (from_nat_into X i) (1/real n)) ≥
      measure M (space M) − e * 2 powr −real n
      by (auto simp: dist_real_def)
    hence ∃k. measure M (⋃i∈{0..k}. cball (from_nat_into X i) (1/real n)) ≥
      measure M (space M) − e * 2 powr − real n ..
  } note k=this
  hence ∀e∈{0<..}. ∀(n::nat)∈{0<..}. ∃k.
```

measure M (⋃ *i*∈{*0..k*}. *cball* (*from_nat_into X i*) (*1/real n*)) ≥ *measure M* (*space M*) − *e* ∗ *2 powr* − *real n*
by *blast*
then obtain *k* **where** *k*: ∀ *e*∈{*0*<..}. ∀ *n*∈{*0*<..}. *measure M* (*space M*) − *e* ∗ *2 powr* − *real* (*n*::*nat*)
≤ *measure M* (⋃ *i*∈{*0..k e n*}. *cball* (*from_nat_into X i*) (*1 / n*))
by *metis*
hence *k*: ⋀*e n*. *e* > *0* ⟹ *n* > *0* ⟹ *measure M* (*space M*) − *e* ∗ *2 powr* − *n*
≤ *measure M* (⋃ *i*∈{*0..k e n*}. *cball* (*from_nat_into X i*) (*1 / n*))
unfolding *Ball_def* **by** *blast*
have *approx_space*:
∃ *K* ∈ {*K*. *K* ⊆ *space M* ∧ *compact K*}. *emeasure M* (*space M*) ≤ *emeasure M K* + *ennreal e*
(**is** *?thesis e*) **if** *0* < *e* **for** *e* :: *real*
proof −
define *B* **where** [*abs_def*]:
B n = (⋃ *i*∈{*0..k e* (*Suc n*)}. *cball* (*from_nat_into X i*) (*1 / Suc n*)) **for** *n*
have ⋀*n*. *closed* (*B n*) **by** (*auto simp*: *B_def*)
hence [*simp*]: ⋀*n*. *B n* ∈ *sets M* **by** (*simp add*: *sb*)
from *k*[*OF* ‹*e* > *0*› *zero_less_Suc*]
have ⋀*n*. *measure M* (*space M*) − *measure M* (*B n*) ≤ *e* ∗ *2 powr* − *real* (*Suc n*)
by (*simp add*: *algebra_simps B_def finite_measure_compl*)
hence *B_compl_le*: ⋀*n*::*nat*. *measure M* (*space M* − *B n*) ≤ *e* ∗ *2 powr* − *real* (*Suc n*)
by (*simp add*: *finite_measure_compl*)
define *K* **where** *K* = (⋂ *n*. *B n*)
from ‹*closed* (*B* _)› **have** *closed K* **by** (*auto simp*: *K_def*)
hence [*simp*]: *K* ∈ *sets M* **by** (*simp add*: *sb*)
have *measure M* (*space M*) − *measure M K* = *measure M* (*space M* − *K*)
by (*simp add*: *finite_measure_compl*)
also have ... = *emeasure M* (⋃ *n*. *space M* − *B n*) **by** (*auto simp*: *K_def emeasure_eq_measure*)
also have ... ≤ (∑ *n*. *emeasure M* (*space M* − *B n*))
by (*rule emeasure_subadditive_countably*) (*auto simp*: *summable_def*)
also have ... ≤ (∑ *n*. *ennreal* (*e*∗*2 powr* − *real* (*Suc n*)))
using *B_compl_le* **by** (*intro suminf_le*) (*simp_all add*: *emeasure_eq_measure ennreal_leI*)
also have ... ≤ (∑ *n*. *ennreal* (*e* ∗ (*1 / 2*) ^ *Suc n*))
by (*simp add*: *powr_minus powr_realpow field_simps del*: *of_nat_Suc*)
also have ... = *ennreal e* ∗ (∑ *n*. *ennreal* ((*1 / 2*) ^ *Suc n*))
unfolding *ennreal_power*[*symmetric*]
using ‹*0* < *e*›
by (*simp add*: *ac_simps ennreal_mult' divide_ennreal*[*symmetric*] *divide_ennreal_def*
ennreal_power[*symmetric*])
also have ... = *e*
by (*subst suminf_ennreal_eq*[*OF zero_le_power power_half_series*]) *auto*
finally have *measure M* (*space M*) ≤ *measure M K* + *e*
using ‹*0* < *e*› **by** *simp*

```
    hence emeasure M (space M) ≤ emeasure M K + e
      using ‹0 < e› by (simp add: emeasure_eq_measure flip: ennreal_plus)
    moreover have compact K
      unfolding compact_eq_totally_bounded
    proof safe
      show complete K using ‹closed K› by (simp add: complete_eq_closed)
      fix e'::real assume 0 < e'
      then obtain n where n: 1 / real (Suc n) < e' by (rule nat_approx_posE)
      let ?k = from_nat_into X ` {0..k e (Suc n)}
      have finite ?k by simp
      moreover have K ⊆ (⋃x∈?k. ball x e') unfolding K_def B_def using n
by force
      ultimately show ∃k. finite k ∧ K ⊆ (⋃x∈k. ball x e') by blast
    qed
    ultimately
    show ?thesis by (auto simp: sU)
  qed
  { fix A::'a set assume closed A hence A ∈ sets borel by (simp add: com-
pact_imp_closed)
    hence [simp]: A ∈ sets M by (simp add: sb)
    have ?inner A
    proof (rule approx_inner)
      fix e::real assume e > 0
      from approx_space[OF this] obtain K where
        K: K ⊆ space M compact K emeasure M (space M) ≤ emeasure M K + e
        by (auto simp: emeasure_eq_measure)
      hence [simp]: K ∈ sets M by (simp add: sb compact_imp_closed)
      have measure M A − measure M (A ∩ K) = measure M (A − A ∩ K)
        by (subst finite_measure_Diff) auto
      also have A − A ∩ K = A ∪ K − K by auto
      also have measure M ... = measure M (A ∪ K) − measure M K
        by (subst finite_measure_Diff) auto
      also have ... ≤ measure M (space M) − measure M K
        by (simp add: emeasure_eq_measure sU sb finite_measure_mono)
      also have ... ≤ e
        using K ‹0 < e› by (simp add: emeasure_eq_measure flip: ennreal_plus)
      finally have emeasure M A ≤ emeasure M (A ∩ K) + ennreal e
          using ‹0<e› by (simp add: emeasure_eq_measure algebra_simps flip:
ennreal_plus)
       moreover have A ∩ K ⊆ A compact (A ∩ K) using ‹closed A› ‹compact
K› by auto
      ultimately show ∃K ⊆ A. compact K ∧ emeasure M A ≤ emeasure M K +
ennreal e
        by blast
    qed simp
    have ?outer A
    proof cases
      assume A ≠ {}
      let ?G = λd. {x. infdist x A < d}
```

```
    {
      fix d
      have ?G d = (λx. infdist x A) -` {..<d} by auto
      also have open ...
          by (intro continuous_open_vimage) (auto intro!: continuous_infdist
continuous_ident)
      finally have open (?G d) .
    } note open_G = this
    from in_closed_iff_infdist_zero[OF ‹closed A› ‹A ≠ {}›]
    have A = {x. infdist x A = 0} by auto
    also have ... = (⋂i. ?G (1/real (Suc i)))
    proof (auto simp del: of_nat_Suc, rule ccontr)
      fix x
      assume infdist x A ≠ 0
      then have pos: infdist x A > 0 using infdist_nonneg[of x A] by simp
        then obtain n where n: 1 / real (Suc n) < infdist x A by (rule
nat_approx_posE)
      assume ∀i. infdist x A < 1 / real (Suc i)
      then have infdist x A < 1 / real (Suc n) by auto
      with n show False by simp
    qed
    also have M ... = (INF n. emeasure M (?G (1 / real (Suc n))))
    proof (rule INF_emeasure_decseq[symmetric], safe)
      fix i::nat
      from open_G[of 1 / real (Suc i)]
      show ?G (1 / real (Suc i)) ∈ sets M by (simp add: sb borel_open)
    next
      show decseq (λi. {x. infdist x A < 1 / real (Suc i)})
        by (auto intro: less_trans intro!: divide_strict_left_mono
          simp: decseq_def le_eq_less_or_eq)
    qed simp
    finally
    have emeasure M A = (INF n. emeasure M {x. infdist x A < 1 / real (Suc
n)}) .
    moreover
    have ... ≥ (INF U∈{U. A ⊆ U ∧ open U}. emeasure M U)
    proof (intro INF_mono)
      fix m
      have ?G (1 / real (Suc m)) ∈ {U. A ⊆ U ∧ open U} using open_G by
auto
      moreover have M (?G (1 / real (Suc m))) ≤ M (?G (1 / real (Suc m)))
by simp
      ultimately show ∃U∈{U. A ⊆ U ∧ open U}.
        emeasure M U ≤ emeasure M {x. infdist x A < 1 / real (Suc m)}
        by blast
    qed
    moreover
    have emeasure M A ≤ (INF U∈{U. A ⊆ U ∧ open U}. emeasure M U)
      by (rule INF_greatest) (auto intro!: emeasure_mono simp: sb)
```

```
    ultimately show ?thesis by simp
  qed (auto intro!: INF_eqI)
  note ‹?inner A› ‹?outer A› }
 note closed_in_D = this
 from ‹B ∈ sets borel›
 have Int_stable (Collect closed) Collect closed ⊆ Pow UNIV B ∈ sigma_sets
UNIV (Collect closed)
  by (auto simp: Int_stable_def borel_eq_closed)
 then show ?inner B ?outer B
 proof (induct B rule: sigma_sets_induct_disjoint)
  case empty
  { case 1 show ?case by (intro SUP_eqI[symmetric]) auto }
  { case 2 show ?case by (intro INF_eqI[symmetric]) (auto elim!: meta_allE[of
_ {}]) }
 next
  case (basic B)
  { case 1 from basic closed_in_D show ?case by auto }
  { case 2 from basic closed_in_D show ?case by auto }
 next
  case (compl B)
  note inner = compl(2) and outer = compl(3)
  from compl have [simp]: B ∈ sets M by (auto simp: sb borel_eq_closed)
  case 2
   have M (space M − B) = M (space M) − emeasure M B by (auto simp:
emeasure_compl)
  also have ... = (INF K∈{K. K ⊆ B ∧ compact K}. M (space M) − M K)
   by (subst ennreal_SUP_const_minus) (auto simp: less_top[symmetric] inner)
  also have ... = (INF U∈{U. U ⊆ B ∧ compact U}. M (space M − U))
   by (auto simp add: emeasure_compl sb compact_imp_closed)
  also have ... ≥ (INF U∈{U. U ⊆ B ∧ closed U}. M (space M − U))
   by (rule INF_superset_mono) (auto simp add: compact_imp_closed)
  also have (INF U∈{U. U ⊆ B ∧ closed U}. M (space M − U)) =
     (INF U∈{U. space M − B ⊆ U ∧ open U}. emeasure M U)
   apply (rule arg_cong [of _ _ Inf])
   using sU
   apply (auto simp add: image_iff)
   apply (rule exI [of _ UNIV − y for y])
   apply safe
    apply (auto simp add: double_diff)
   done
  finally have
    (INF U∈{U. space M − B ⊆ U ∧ open U}. emeasure M U) ≤ emeasure M
(space M − B) .
  moreover have
    (INF U∈{U. space M − B ⊆ U ∧ open U}. emeasure M U) ≥ emeasure M
(space M − B)
    by (auto simp: sb sU intro!: INF_greatest emeasure_mono)
 ultimately show ?case by (auto intro!: antisym simp: sets_eq_imp_space_eq[OF
sb])
```

```
    case 1
     have M (space M − B) = M (space M) − emeasure M B by (auto simp:
emeasure_compl)
    also have ... = (SUP U∈ {U. B ⊆ U ∧ open U}. M (space M) −  M U)
      unfolding outer by (subst ennreal_INF_const_minus) auto
    also have ... = (SUP U∈{U. B ⊆ U ∧ open U}. M (space M − U))
      by (auto simp add: emeasure_compl sb compact_imp_closed)
     also have ... = (SUP K∈{K. K ⊆ space M − B ∧ closed K}. emeasure M
K)
       unfolding SUP_image [of _ λu. space M − u _, symmetric, unfolded
comp_def]
      apply (rule arg_cong [of _ _ Sup])
      using sU apply (auto intro!: imageI)
      done
    also have ... = (SUP K∈{K. K ⊆ space M − B ∧ compact K}. emeasure M
K)
    proof (safe intro!: antisym SUP_least)
      fix K assume closed K K ⊆ space M − B
      from closed_in_D[OF ‹closed K›]
       have K_inner: emeasure M K = (SUP K∈{Ka. Ka ⊆ K ∧ compact Ka}.
emeasure M K) by simp
       show emeasure M K ≤ (SUP K∈{K. K ⊆ space M − B ∧ compact K}.
emeasure M K)
        unfolding K_inner using ‹K ⊆ space M − B›
        by (auto intro!: SUP_upper SUP_least)
    qed (fastforce intro!: SUP_least SUP_upper simp: compact_imp_closed)
    finally show ?case by (auto intro!: antisym simp: sets_eq_imp_space_eq[OF
sb])
  next
    case (union D)
    then have range D ⊆ sets M by (auto simp: sb borel_eq_closed)
     with union have M[symmetric]: (∑ i. M (D i)) = M (⋃ i. D i) by (intro
suminf_emeasure)
    also have (λn. ∑ i<n. M (D i)) ⟶ (∑ i. M (D i))
      by (intro summable_LIMSEQ) auto
   finally have measure_LIMSEQ: (λn. ∑ i<n. measure M (D i)) ⟶ measure
M (⋃ i. D i)
      by (simp add: emeasure_eq_measure sum_nonneg)
    have (⋃ i. D i) ∈ sets M using ‹range D ⊆ sets M› by auto

    case 1
    show ?case
    proof (rule approx_inner)
      fix e::real assume e > 0
      with measure_LIMSEQ
       have ∃ no. ∀ n≥no. |(∑ i<n. measure M (D i)) −measure M (⋃ x. D x)| <
e/2
      by (auto simp: lim_sequentially dist_real_def simp del: less_divide_eq_numeral1)
```

hence $\exists n0.\ |(\sum i{<}n0.\ measure\ M\ (D\ i)) - measure\ M\ (\bigcup x.\ D\ x)| < e/2$ **by** *auto*
then obtain *n0* **where** *n0*: $|(\sum i{<}n0.\ measure\ M\ (D\ i)) - measure\ M\ (\bigcup i.\ D\ i)| < e/2$
unfolding *choice_iff* **by** *blast*
have $ennreal\ (\sum i{<}n0.\ measure\ M\ (D\ i)) = (\sum i{<}n0.\ M\ (D\ i))$
by (*auto simp add*: *emeasure_eq_measure*)
also have $\ldots \leq (\sum i.\ M\ (D\ i))$ **by** (*rule sum_le_suminf*) *auto*
also have $\ldots = M\ (\bigcup i.\ D\ i)$ **by** (*simp add*: *M*)
also have $\ldots = measure\ M\ (\bigcup i.\ D\ i)$ **by** (*simp add*: *emeasure_eq_measure*)
finally have *n0*: $measure\ M\ (\bigcup i.\ D\ i) - (\sum i{<}n0.\ measure\ M\ (D\ i)) < e/2$
using *n0* **by** (*auto simp*: *sum_nonneg*)
have $\forall i.\ \exists K.\ K \subseteq D\ i \wedge compact\ K \wedge emeasure\ M\ (D\ i) \leq emeasure\ M\ K + e/(2{*}Suc\ n0)$
proof
fix *i*
from ‹$0 < e$› **have** $0 < e/(2{*}Suc\ n0)$ **by** *simp*
have $emeasure\ M\ (D\ i) = (SUP\ K{\in}\{K.\ K \subseteq (D\ i) \wedge compact\ K\}.\ emeasure\ M\ K)$
using *union* **by** *blast*
from $SUP\_approx\_ennreal[OF\ ‹0 < e/(2{*}Suc\ n0)›\ \_\ this]$
show $\exists K.\ K \subseteq D\ i \wedge compact\ K \wedge emeasure\ M\ (D\ i) \leq emeasure\ M\ K + e/(2{*}Suc\ n0)$
by (*auto simp*: *emeasure_eq_measure intro*: *less_imp_le compact_empty*)
qed
then obtain *K* **where** *K*: $\bigwedge i.\ K\ i \subseteq D\ i\ \bigwedge i.\ compact\ (K\ i)$
$\bigwedge i.\ emeasure\ M\ (D\ i) \leq emeasure\ M\ (K\ i) + e/(2{*}Suc\ n0)$
unfolding *choice_iff* **by** *blast*
let $?K = \bigcup i{\in}\{..{<}n0\}.\ K\ i$
have *disjoint_family_on K* $\{..{<}n0\}$ **using** *K* ‹*disjoint_family D*›
unfolding *disjoint_family_on_def* **by** *blast*
hence *mK*: $measure\ M\ ?K = (\sum i{<}n0.\ measure\ M\ (K\ i))$ **using** *K*
by (*intro finite_measure_finite_Union*) (*auto simp*: *sb compact_imp_closed*)
have $measure\ M\ (\bigcup i.\ D\ i) < (\sum i{<}n0.\ measure\ M\ (D\ i)) + e/2$ **using** *n0* **by** *simp*
also have $(\sum i{<}n0.\ measure\ M\ (D\ i)) \leq (\sum i{<}n0.\ measure\ M\ (K\ i) + e/(2{*}Suc\ n0))$
using *K* ‹$0 < e$›
by (*auto intro*: *sum_mono simp*: *emeasure_eq_measure simp flip*: *ennreal_plus*)
also have $\ldots = (\sum i{<}n0.\ measure\ M\ (K\ i)) + (\sum i{<}n0.\ e/(2{*}Suc\ n0))$
by (*simp add*: *sum.distrib*)
also have $\ldots \leq (\sum i{<}n0.\ measure\ M\ (K\ i)) + e\ /\ 2$ **using** ‹$0 < e$›
by (*auto simp*: *field_simps intro*!: *mult_left_mono*)
finally
have $measure\ M\ (\bigcup i.\ D\ i) < (\sum i{<}n0.\ measure\ M\ (K\ i)) + e\ /\ 2 + e\ /\ 2$
by *auto*
hence $M\ (\bigcup i.\ D\ i) < M\ ?K + e$
using ‹$0{<}e$› **by** (*auto simp*: *mK emeasure_eq_measure sum_nonneg en-*

nreal_less_iff simp flip: *ennreal_plus*)
moreover
have *?K* ⊆ (⋃ *i*. *D i*) **using** *K* **by** *auto*
moreover
have *compact ?K* **using** *K* **by** *auto*
ultimately
have *?K*⊆(⋃ *i*. *D i*) ∧ *compact ?K* ∧ *emeasure M* (⋃ *i*. *D i*) ≤ *emeasure M ?K* + *ennreal e* **by** *simp*
thus ∃ *K*⊆⋃ *i*. *D i*. *compact K* ∧ *emeasure M* (⋃ *i*. *D i*) ≤ *emeasure M K* + *ennreal e* **..**
qed *fact*
case *2*
show *?case*
proof (*rule approx_outer*[*OF* ‹(⋃ *i*. *D i*) ∈ *sets M*›])
fix *e*::*real* **assume** *e* > *0*
have ∀ *i*::*nat*. ∃ *U*. *D i* ⊆ *U* ∧ *open U* ∧ *e*/(*2 powr Suc i*) > *emeasure M U* − *emeasure M* (*D i*)
proof
fix *i*::*nat*
from ‹*0* < *e*› **have** *0* < *e*/(*2 powr Suc i*) **by** *simp*
have *emeasure M* (*D i*) = (*INF U*∈{*U*. (*D i*) ⊆ *U* ∧ *open U*}. *emeasure M U*)
using *union* **by** *blast*
from *INF_approx_ennreal*[*OF* ‹*0* < *e*/(*2 powr Suc i*)› *this*]
show ∃ *U*. *D i* ⊆ *U* ∧ *open U* ∧ *e*/(*2 powr Suc i*) > *emeasure M U* − *emeasure M* (*D i*)
using ‹*0*<*e*›
by (*auto simp*: *emeasure_eq_measure sum_nonneg ennreal_less_iff ennreal_minus*
finite_measure_mono sb
simp flip: *ennreal_plus*)
qed
then obtain *U* **where** *U*: ⋀ *i*. *D i* ⊆ *U i* ⋀ *i*. *open* (*U i*)
⋀ *i*. *e*/(*2 powr Suc i*) > *emeasure M* (*U i*) − *emeasure M* (*D i*)
unfolding *choice_iff* **by** *blast*
let *?U* = ⋃ *i*. *U i*
have *ennreal* (*measure M ?U* − *measure M* (⋃ *i*. *D i*)) = *M ?U* − *M* (⋃ *i*. *D i*)
using *U*(*1*,*2*)
by (*subst ennreal_minus*[*symmetric*])
(*auto intro*!: *finite_measure_mono simp*: *sb emeasure_eq_measure*)
also have ... = *M* (*?U* − (⋃ *i*. *D i*)) **using** *U* ‹(⋃ *i*. *D i*) ∈ *sets M*›
by (*subst emeasure_Diff*) (*auto simp*: *sb*)
also have ... ≤ *M* (⋃ *i*. *U i* − *D i*) **using** *U* ‹*range D* ⊆ *sets M*›
by (*intro emeasure_mono*) (*auto simp*: *sb intro*!: *sets.countable_nat_UN sets.Diff*)
also have ... ≤ (∑ *i*. *M* (*U i* − *D i*)) **using** *U* ‹*range D* ⊆ *sets M*›
by (*intro emeasure_subadditive_countably*) (*auto intro*!: *sets.Diff simp*: *sb*)
also have ... ≤ (∑ *i*. *ennreal e*/(*2 powr Suc i*)) **using** *U* ‹*range D* ⊆ *sets*

```
M›
      using ‹0<e›
      by (intro suminf_le, subst emeasure_Diff)
         (auto simp: emeasure_Diff emeasure_eq_measure sb ennreal_minus
                     finite_measure_mono divide_ennreal ennreal_less_iff
               intro: less_imp_le)
    also have ... ≤ (∑ n. ennreal (e * (1 / 2) ^ Suc n))
      using ‹0<e›
       by (simp add: powr_minus powr_realpow field_simps divide_ennreal del:
of_nat_Suc)
    also have ... = ennreal e * (∑ n. ennreal ((1 / 2) ^ Suc n))
      unfolding ennreal_power[symmetric]
      using ‹0 < e›
         by (simp add: ac_simps ennreal_mult' divide_ennreal[symmetric] di-
vide_ennreal_def
                  ennreal_power[symmetric])
    also have ... = ennreal e
      by (subst suminf_ennreal_eq[OF zero_le_power power_half_series]) auto
    finally have emeasure M ?U ≤ emeasure M (⋃ i. D i) + ennreal e
      using ‹0<e› by (simp add: emeasure_eq_measure flip: ennreal_plus)
    moreover
    have (⋃ i. D i) ⊆ ?U using U by auto
    moreover
    have open ?U using U by auto
    ultimately
    have (⋃ i. D i) ⊆ ?U ∧ open ?U ∧ emeasure M ?U ≤ emeasure M (⋃ i. D
i) + ennreal e by simp
    thus ∃ B. (⋃ i. D i) ⊆ B ∧ open B ∧ emeasure M B ≤ emeasure M (⋃ i. D
i) + ennreal e ..
  qed
 qed
qed

end
```

6.12 Lebesgue Measure

```
theory Lebesgue_Measure
imports
  Finite_Product_Measure
  Caratheodory
  Complete_Measure
  Summation_Tests
  Regularity
begin

lemma measure_eqI_lessThan:
  fixes M N :: real measure
  assumes sets: sets M = sets borel sets N = sets borel
```

assumes *fin*: $\bigwedge x.$ *emeasure* M $\{x <..\} < \infty$
assumes $\bigwedge x.$ *emeasure* M $\{x <..\}$ = *emeasure* N $\{x <..\}$
shows $M = N$
proof (*rule measure_eqI_generator_eq_countable*)
let *?LT* = λa::*real*. $\{a <..\}$ **let** *?E* = *range ?LT*
show *Int_stable ?E*
by (*auto simp*: *Int_stable_def lessThan_Int_lessThan*)

show *?E* $\subseteq$ *Pow UNIV sets M* = *sigma_sets UNIV ?E sets N* = *sigma_sets UNIV ?E*
unfolding *sets borel_Ioi* **by** *auto*

show *?LT*‘*Rats* $\subseteq$ *?E* $(\bigcup i \in$ *Rats*. *?LT i*$)$ = *UNIV* $\bigwedge a.$ $a \in$ *?LT*‘*Rats* $\Longrightarrow$ *emeasure M a* $\neq \infty$
using *fin* **by** (*auto intro*: *Rats_no_bot_less simp*: *less_top*)
qed (*auto intro*: *assms countable_rat*)

6.12.1 Measures defined by monotonous functions

Every right-continuous and nondecreasing function gives rise to a measure on the reals:

definition *interval_measure* :: (*real* $\Rightarrow$ *real*) $\Rightarrow$ *real measure* **where**
interval_measure F =
extend_measure UNIV $\{(a, b).\ a \leq b\}$ $(\lambda(a, b).\ \{a<..b\})$ $(\lambda(a, b).$ *ennreal* $(F\ b - F\ a))$

lemma *emeasure_interval_measure_Ioc*:
assumes $a \leq b$
assumes *mono_F*: $\bigwedge x\ y.\ x \leq y \Longrightarrow F\ x \leq F\ y$
assumes *right_cont_F* : $\bigwedge a.$ *continuous* (*at_right a*) *F*
shows *emeasure* (*interval_measure F*) $\{a<..b\} = F\ b - F\ a$
proof (*rule extend_measure_caratheodory_pair*[*OF interval_measure_def* ‹$a \leq b$›])
show *semiring_of_sets UNIV* $\{\{a<..b\}\ |a\ b ::$ *real*. $a \leq b\}$
proof (*unfold_locales*, *safe*)
fix $a\ b\ c\ d ::$ *real* **assume** $*$: $a \leq b$ $c \leq d$
then show $\exists C \subseteq \{\{a<..b\}\ |a\ b.\ a \leq b\}.$ *finite* $C \wedge$ *disjoint* $C \wedge \{a<..b\} - \{c<..d\} = \bigcup C$
proof *cases*
let *?C* = $\{\{a<..b\}\}$
assume $b < c \vee d \leq a \vee d \leq c$
with $*$ **have** *?C* $\subseteq \{\{a<..b\}\ |a\ b.\ a \leq b\} \wedge$ *finite ?C* $\wedge$ *disjoint ?C* $\wedge \{a<..b\} - \{c<..d\} = \bigcup$ *?C*
by (*auto simp add*: *disjoint_def*)
thus *?thesis* **..**
next
let *?C* = $\{\{a<..c\}, \{d<..b\}\}$
assume $\neg\ (b < c \vee d \leq a \vee d \leq c)$
with $*$ **have** *?C* $\subseteq \{\{a<..b\}\ |a\ b.\ a \leq b\} \wedge$ *finite ?C* $\wedge$ *disjoint ?C* $\wedge \{a<..b\}$

```
− {c<..d} = ⋃ ?C
      by (auto simp add: disjoint_def Ioc_inj) (metis linear)+
    thus ?thesis ..
  qed
 qed (auto simp: Ioc_inj, metis linear)
next
 fix l r :: nat ⇒ real and a b :: real
 assume l_r[simp]: ⋀n. l n ≤ r n and a ≤ b and disj: disjoint_family (λn. {l
n<..r n})
 assume lr_eq_ab: (⋃ i. {l i<..r i}) = {a<..b}

 have [intro, simp]: ⋀a b. a ≤ b ⟹ F a ≤ F b
  by (auto intro!: l_r mono_F)

 { fix S :: nat set assume finite S
  moreover note ‹a ≤ b›
  moreover have ⋀i. i ∈ S ⟹ {l i <.. r i} ⊆ {a <.. b}
   unfolding lr_eq_ab[symmetric] by auto
  ultimately have (∑ i∈S. F (r i) − F (l i)) ≤ F b − F a
  proof (induction S arbitrary: a rule: finite_psubset_induct)
   case (psubset S)
   show ?case
   proof cases
    assume ∃ i∈S. l i < r i
    with ‹finite S› have Min (l ‘ {i∈S. l i < r i}) ∈ l ‘ {i∈S. l i < r i}
     by (intro Min_in) auto
    then obtain m where m: m ∈ S l m < r m l m = Min (l ‘ {i∈S. l i < r
i})
     by fastforce

    have (∑ i∈S. F (r i) − F (l i)) = (F (r m) − F (l m)) + (∑ i∈S − {m}.
F (r i) − F (l i))
     using m psubset by (intro sum.remove) auto
    also have (∑ i∈S − {m}. F (r i) − F (l i)) ≤ F b − F (r m)
    proof (intro psubset.IH)
     show S − {m} ⊂ S
      using ‹m∈S› by auto
     show r m ≤ b
      using psubset.prems(2)[OF ‹m∈S›] ‹l m < r m› by auto
    next
     fix i assume i ∈ S − {m}
     then have i: i ∈ S i ≠ m by auto
     { assume i': l i < r i l i < r m
      with ‹finite S› i m have l m ≤ l i
       by auto
      with i' have {l i <.. r i} ∩ {l m <.. r m} ≠ {}
       by auto
      then have False
       using disjoint_family_onD[OF disj, of i m] i by auto }
```

```
      then have l i ≠ r i ⟹ r m ≤ l i
        unfolding not_less[symmetric] using l_r[of i] by auto
      then show {l i <.. r i} ⊆ {r m <.. b}
        using psubset.prems(2)[OF ‹i∈S›] by auto
    qed
    also have F (r m) − F (l m) ≤ F (r m) − F a
      using psubset.prems(2)[OF ‹m ∈ S›] ‹l m < r m›
      by (auto simp add: Ioc_subset_iff intro!: mono_F)
    finally show ?case
      by (auto intro: add_mono)
  qed (auto simp add: ‹a ≤ b› less_le)
 qed }
note claim1 = this


{ fix S u v and l r :: nat ⇒ real
 assume finite S ⋀i. i∈S ⟹ l i < r i {u..v} ⊆ (⋃i∈S. {l i<..< r i})
 then have F v − F u ≤ (∑i∈S. F (r i) − F (l i))
 proof (induction arbitrary: v u rule: finite_psubset_induct)
   case (psubset S)
   show ?case
   proof cases
     assume S = {} then show ?case
       using psubset by (simp add: mono_F)
   next
     assume S ≠ {}
     then obtain j where j ∈ S
       by auto

     let ?R = r j < u ∨ l j > v ∨ (∃ i∈S−{j}. l i ≤ l j ∧ r j ≤ r i)
     show ?case
     proof cases
       assume ?R
       with ‹j ∈ S› psubset.prems have {u..v} ⊆ (⋃i∈S−{j}. {l i<..< r i})
         apply (simp add: subset_eq Ball_def Bex_def)
       by (metis order_le_less_trans order_less_le_trans order_less_not_sym)
       with ‹j ∈ S› have F v − F u ≤ (∑i∈S − {j}. F (r i) − F (l i))
         by (intro psubset) auto
       also have ... ≤ (∑i∈S. F (r i) − F (l i))
         using psubset.prems
         by (intro sum_mono2 psubset) (auto intro: less_imp_le)
       finally show ?thesis .
     next
       assume ¬ ?R
       then have j: u ≤ r j l j ≤ v ⋀i. i ∈ S − {j} ⟹ r i < r j ∨ l i > l j
         by (auto simp: not_less)
       let ?S1 = {i ∈ S. l i < l j}
       let ?S2 = {i ∈ S. r i > r j}
```

```
        have ∗: ?S1 ∩ ?S2 = {}
          using j by (fastforce simp add: disjoint_iff)
        have (∑ i∈S. F (r i) − F (l i)) ≥ (∑ i∈?S1 ∪ ?S2 ∪ {j}. F (r i) − F
(l i))
          using ‹j ∈ S› ‹finite S› psubset.prems j
          by (intro sum_mono2) (auto intro: less_imp_le)
        also have (∑ i∈?S1 ∪ ?S2 ∪ {j}. F (r i) − F (l i)) =
          (∑ i∈?S1. F (r i) − F (l i)) + (∑ i∈?S2 . F (r i) − F (l i)) + (F (r
j) − F (l j))
       using psubset(1) by (simp add: ∗ sum.union_disjoint disjoint_iff_not_equal)
        also (xtrans) have (∑ i∈?S1. F (r i) − F (l i)) ≥ F (l j) − F u
          using ‹j ∈ S› ‹finite S› psubset.prems j
          apply (intro psubset.IH psubset)
          apply (auto simp: subset_eq Ball_def)
          apply (metis less_le_trans not_le)
          done
        also (xtrans) have (∑ i∈?S2. F (r i) − F (l i)) ≥ F v − F (r j)
          using ‹j ∈ S› ‹finite S› psubset.prems j
          apply (intro psubset.IH psubset)
          apply (auto simp: subset_eq Ball_def)
          apply (metis le_less_trans not_le)
          done
        finally (xtrans) show ?case
          by (auto simp: add_mono)
      qed
    qed
  qed }
 note claim2 = this


 have ennreal (F b − F a) ≤ (∑ i. ennreal (F (r i) − F (l i)))
 proof (rule ennreal_le_epsilon)
  fix epsilon :: real assume egt0: epsilon > 0
  have ∀ i. ∃ d>0. F (r i + d) < F (r i) + epsilon / 2^(i+2)
  proof
   fix i
   note right_cont_F [of r i]
   then have ∃ d>0. F (r i + d) − F (r i) < epsilon / 2 ^(i + 2)
    by (simp add: continuous_at_right_real_increasing egt0)
   thus ∃ d>0. F (r i + d) < F (r i) + epsilon / 2^(i+2)
    by force
  qed
  then obtain delta where
     deltai_gt0: ⋀i. delta i > 0 and
     deltai_prop: ⋀i. F (r i + delta i) < F (r i) + epsilon / 2^(i+2)
   by metis
  have ∃ a' > a. F a' − F a < epsilon / 2
   using right_cont_F [of a]
  by (metis continuous_at_right_real_increasing egt0 half_gt_zero less_add_same_cancel1
```

mono_F)
then obtain a' **where** *a'lea* [*arith*]: $a' > a$ **and**
a_prop: $F\ a' - F\ a < epsilon\ /\ 2$
by *auto*
define S' **where** $S' = \{i.\ l\ i < r\ i\}$
obtain $S :: nat\ set$ **where** $S \subseteq S'$ **and** *finS*: *finite S*
and *Sprop*: $\{a'..b\} \subseteq (\bigcup i \in S.\ \{l\ i<..<r\ i + delta\ i\})$
proof (*rule compactE_image*)
show *compact* $\{a'..b\}$
by (*rule compact_Icc*)
show $\bigwedge i.\ i \in S' \Longrightarrow$ *open* $(\{l\ i<..<r\ i + delta\ i\})$ **by** *auto*
have $\{a'..b\} \subseteq \{a <..\ b\}$
by *auto*
also have $\{a <..\ b\} = (\bigcup i{\in}S'.\ \{l\ i<..r\ i\})$
unfolding *lr_eq_ab*[*symmetric*] **by** (*fastforce simp add*: *S'_def intro*: *less_le_trans*)
also have $\ldots \subseteq (\bigcup i \in S'.\ \{l\ i<..<r\ i + delta\ i\})$
by (*intro UN_mono*; *simp add*: *add.commute add_strict_increasing deltai_gt0 subset_iff*)
finally show $\{a'..b\} \subseteq (\bigcup i \in S'.\ \{l\ i<..<r\ i + delta\ i\})$ **.**
qed
with *S'_def* **have** *Sprop2*: $\bigwedge i.\ i \in S \Longrightarrow l\ i < r\ i$ **by** *auto*
obtain n **where** *Sbound*: $\bigwedge i.\ i \in S \Longrightarrow i \leq n$
using *Max_ge finS* **by** *blast*
have $F\ b - F\ a = (F\ b - F\ a') + (F\ a' - F\ a)$
by *auto*
also have ... $\leq (F\ b - F\ a') + epsilon\ /\ 2$
using *a_prop* **by** (*intro add_left_mono*) *simp*
also have ... $\leq (\sum i{\in}S.\ F\ (r\ i) - F\ (l\ i)) + epsilon\ /\ 2 + epsilon\ /\ 2$
proof −
have $F\ b - F\ a' \leq (\sum i{\in}S.\ F\ (r\ i + delta\ i) - F\ (l\ i))$
using *claim2 l_r Sprop* **by** (*simp add*: *deltai_gt0 finS add.commute add_strict_increasing*)
also have ... $\leq (\sum i \in S.\ F(r\ i) - F(l\ i) + epsilon\ /\ 2\hat{}(i+2))$
by (*smt* (*verit*) *sum_mono deltai_prop*)
also have ... $= (\sum i \in S.\ F(r\ i) - F(l\ i)) +$
$(epsilon\ /\ 4) * (\sum i \in S.\ (1\ /\ 2)\hat{}i)$ (**is** $\_ = ?t + \_$)
by (*subst sum.distrib*) (*simp add*: *field_simps sum_distrib_left*)
also have ... $\leq ?t + (epsilon\ /\ 4) * (\sum\ i < Suc\ n.\ (1\ /\ 2)\hat{}i)$
using *egt0 Sbound* **by** (*intro add_left_mono mult_left_mono sum_mono2*) *force*+
also have ... $\leq ?t + (epsilon\ /\ 2)$
using *egt0* **by** (*simp add*: *geometric_sum add_left_mono mult_left_mono*)
finally have $F\ b - F\ a' \leq (\sum i{\in}S.\ F\ (r\ i) - F\ (l\ i)) + epsilon\ /\ 2$
by *simp*
then show *?thesis*
by *linarith*
qed
also have ... $= (\sum i{\in}S.\ F\ (r\ i) - F\ (l\ i)) + epsilon$

by *auto*
also have ... $\leq (\sum i{\leq}n.\ F\ (r\ i) - F\ (l\ i)) + \mathit{epsilon}$
using *finS Sbound Sprop* **by** (*auto intro*!: *add_right_mono sum_mono2*)
finally have *ennreal* $(F\ b - F\ a) \leq (\sum i{\leq}n.\ \mathit{ennreal}\ (F\ (r\ i) - F\ (l\ i))) + \mathit{epsilon}$
using *egt0* **by** (*simp add*: *sum_nonneg flip*: *ennreal_plus*)
then show *ennreal* $(F\ b - F\ a) \leq (\sum i.\ \mathit{ennreal}\ (F\ (r\ i) - F\ (l\ i))) + (\mathit{epsilon} :: \mathit{real})$
by (*rule order_trans*) (*auto intro*!: *add_mono sum_le_suminf simp del*: *sum_ennreal*)
qed
moreover have $(\sum i.\ \mathit{ennreal}\ (F\ (r\ i) - F\ (l\ i))) \leq \mathit{ennreal}\ (F\ b - F\ a)$
using ‹$a \leq b$› **by** (*auto intro*!: *suminf_le_const ennreal_le_iff*[*THEN iffD2*] *claim1*)
ultimately show $(\sum n.\ \mathit{ennreal}\ (F\ (r\ n) - F\ (l\ n))) = \mathit{ennreal}\ (F\ b - F\ a)$
by (*rule antisym*[*rotated*])
qed (*auto simp*: *Ioc_inj mono_F*)

lemma *measure_interval_measure_Ioc*:
assumes $a \leq b$ **and** $\bigwedge x\ y.\ x \leq y \Longrightarrow F\ x \leq F\ y$ **and** $\bigwedge a.\ \mathit{continuous}\ (\mathit{at\_right}\ a)\ F$
shows *measure* (*interval_measure F*) $\{a <..\ b\} = F\ b - F\ a$
unfolding *measure_def*
by (*simp add*: *assms emeasure_interval_measure_Ioc*)

lemma *emeasure_interval_measure_Ioc_eq*:
$(\bigwedge x\ y.\ x \leq y \Longrightarrow F\ x \leq F\ y) \Longrightarrow (\bigwedge a.\ \mathit{continuous}\ (\mathit{at\_right}\ a)\ F) \Longrightarrow$
emeasure (*interval_measure F*) $\{a <..\ b\} = (\mathit{if}\ a \leq b\ \mathit{then}\ F\ b - F\ a\ \mathit{else}\ 0)$
using *emeasure_interval_measure_Ioc*[*of a b F*] **by** *auto*

lemma *sets_interval_measure* [*simp*, *measurable_cong*]:
sets (*interval_measure F*) = *sets borel*
apply (*simp add*: *sets_extend_measure interval_measure_def borel_sigma_sets_Ioc image_def split*: *prod.split*)
by (*metis greaterThanAtMost_empty nle_le*)

lemma *space_interval_measure* [*simp*]: *space* (*interval_measure F*) = *UNIV*
by (*simp add*: *interval_measure_def space_extend_measure*)

lemma *emeasure_interval_measure_Icc*:
assumes $a \leq b$
assumes *mono_F*: $\bigwedge x\ y.\ x \leq y \Longrightarrow F\ x \leq F\ y$
assumes *cont_F* : *continuous_on UNIV F*
shows *emeasure* (*interval_measure F*) $\{a\ ..\ b\} = F\ b - F\ a$
proof (*rule tendsto_unique*)
{ fix *a b* :: *real* **assume** $a \leq b$ **then have** *emeasure* (*interval_measure F*) $\{a <..\ b\} = F\ b - F\ a$
using *cont_F*
by (*subst emeasure_interval_measure_Ioc*)

```
      (auto intro: mono_F continuous_within_subset simp: continuous_on_eq_continuous_within)
}
  note ∗ = this

  let ?F = interval_measure F
  show ((λa. F b − F a) ⟶ emeasure ?F {a..b}) (at_left a)
  proof (rule tendsto_at_left_sequentially)
    show a − 1 < a by simp
    fix X assume X: ⋀n. X n < a incseq X X ⟶ a
    then have emeasure (interval_measure F) {X n<..b} ≠ ∞ for n
      by (smt (verit) ∗ ‹a ≤ b› ennreal_neq_top infinity_ennreal_def)
    with X have (λn. emeasure ?F {X n<..b}) ⟶ emeasure ?F (⋂n. {X n
<..b})
      by (intro Lim_emeasure_decseq; force simp: decseq_def incseq_def emea-
sure_interval_measure_Ioc ∗)
    also have (⋂n. {X n <..b}) = {a..b}
      apply auto
      apply (rule LIMSEQ_le_const2[OF ‹X ⟶ a›])
      using less_eq_real_def apply presburger
      using X(1) order_less_le_trans by blast
    also have (λn. emeasure ?F {X n<..b}) = (λn. F b − F (X n))
      using ‹⋀n. X n < a› ‹a ≤ b› by (subst ∗) (auto intro: less_imp_le
less_le_trans)
    finally show (λn. F b − F (X n)) ⟶ emeasure ?F {a..b} .
  qed
  show ((λa. ennreal (F b − F a)) ⟶ F b − F a) (at_left a)
    by (rule continuous_on_tendsto_compose[where g=λx. x and s=UNIV])
      (auto simp: continuous_on_ennreal continuous_on_diff cont_F)
qed (rule trivial_limit_at_left_real)

lemma sigma_finite_interval_measure:
  assumes mono_F: ⋀x y. x ≤ y ⟹ F x ≤ F y
  assumes right_cont_F : ⋀a. continuous (at_right a) F
  shows sigma_finite_measure (interval_measure F)
  apply unfold_locales
  apply (intro exI[of _ (λ(a, b). {a <.. b}) ‘ (ℚ × ℚ)])
 apply (auto intro!: Rats_no_top_le Rats_no_bot_less countable_rat simp: emea-
sure_interval_measure_Ioc_eq[OF assms])
  done
```

6.12.2 Lebesgue-Borel measure

```
definition lborel :: ('a :: euclidean_space) measure where
  lborel = distr (Π_M b∈Basis. interval_measure (λx. x)) borel (λf. ∑b∈Basis. f
b ∗_R b)

abbreviation lebesgue :: 'a::euclidean_space measure
  where lebesgue ≡ completion lborel
```

abbreviation *lebesgue_on* :: *'a set* ⇒ *'a*::*euclidean_space measure*
where *lebesgue_on* Ω ≡ *restrict_space* (*completion lborel*) Ω

lemma *lebesgue_on_mono*:
assumes *major*: *AE x in lebesgue_on S*. *P x* **and** *minor*: ⋀*x*.⟦*P x*; *x* ∈ *S*⟧ ⟹ *Q x*
shows *AE x in lebesgue_on S*. *Q x*
proof −
have *AE a in lebesgue_on S*. *P a* ⟶ *Q a*
using *minor space_restrict_space* **by** *fastforce*
then show *?thesis*
using *major* **by** *auto*
qed

lemma *integral_eq_zero_null_sets*:
assumes *S* ∈ *null_sets lebesgue*
shows *integral*^{L} (*lebesgue_on S*) *f* = *0*
proof (*rule integral_eq_zero_AE*)
show *AE x in lebesgue_on S*. *f x* = *0*
by (*metis* (*no_types*, *lifting*) *assms AE_not_in lebesgue_on_mono null_setsD2 null_sets_restrict_space order_refl*)
qed

lemma
shows *sets_lborel*[*simp*, *measurable_cong*]: *sets lborel* = *sets borel*
and *space_lborel*[*simp*]: *space lborel* = *space borel*
and *measurable_lborel1*[*simp*]: *measurable M lborel* = *measurable M borel*
and *measurable_lborel2*[*simp*]: *measurable lborel M* = *measurable borel M*
by (*simp_all add*: *lborel_def*)

lemma *space_lebesgue_on* [*simp*]: *space* (*lebesgue_on S*) = *S*
by (*simp add*: *space_restrict_space*)

lemma *sets_lebesgue_on_refl* [*iff*]: *S* ∈ *sets* (*lebesgue_on S*)
by (*metis inf_top.right_neutral sets.top space_borel space_completion space_lborel space_restrict_space*)

lemma *Compl_in_sets_lebesgue*: −*A* ∈ *sets lebesgue* ⟷ *A* ∈ *sets lebesgue*
by (*metis Compl_eq_Diff_UNIV double_compl space_borel space_completion space_lborel Sigma_Algebra.sets.compl_sets*)

lemma *measurable_lebesgue_cong*:
assumes ⋀*x*. *x* ∈ *S* ⟹ *f x* = *g x*
shows *f* ∈ *measurable* (*lebesgue_on S*) *M* ⟷ *g* ∈ *measurable* (*lebesgue_on S*) *M*
by (*metis* (*mono_tags*, *lifting*) *IntD1 assms measurable_cong_simp space_restrict_space*)

lemma *lebesgue_on_UNIV_eq*: *lebesgue_on UNIV* = *lebesgue*
by (*simp add*: *emeasure_restrict_space measure_eqI*)

lemma *integral_restrict_Int*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes $S \in sets\ lebesgue\ T \in sets\ lebesgue$
shows $integral^L\ (lebesgue\_on\ T)\ (\lambda x.\ if\ x \in S\ then\ f\ x\ else\ 0) = integral^L\ (lebesgue\_on\ (S \cap T))\ f$
proof −
have $(\lambda x.\ indicat\_real\ T\ x *_R (if\ x \in S\ then\ f\ x\ else\ 0)) = (\lambda x.\ indicat\_real\ (S \cap T)\ x *_R f\ x)$
by (*force simp*: *indicator_def*)
then show *?thesis*
by (*simp add*: *assms sets.Int Bochner_Integration.integral_restrict_space*)
qed

lemma *integral_restrict*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes $S \subseteq T\ S \in sets\ lebesgue\ T \in sets\ lebesgue$
shows $integral^L\ (lebesgue\_on\ T)\ (\lambda x.\ if\ x \in S\ then\ f\ x\ else\ 0) = integral^L\ (lebesgue\_on\ S)\ f$
using *integral_restrict_Int* [*of S T f*] *assms*
by (*simp add*: *Int_absorb2*)

lemma *integral_restrict_UNIV*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes $S \in sets\ lebesgue$
shows $integral^L\ lebesgue\ (\lambda x.\ if\ x \in S\ then\ f\ x\ else\ 0) = integral^L\ (lebesgue\_on\ S)\ f$
using *integral_restrict_Int* [*of S UNIV f*] *assms*
by (*simp add*: *lebesgue_on_UNIV_eq*)

lemma *integrable_lebesgue_on_empty* [*iff*]:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::\{second\_countable\_topology,banach\}$
shows $integrable\ (lebesgue\_on\ \{\})\ f$
by (*simp add*: *integrable_restrict_space*)

lemma *integral_lebesgue_on_empty* [*simp*]:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::\{second\_countable\_topology,banach\}$
shows $integral^L\ (lebesgue\_on\ \{\})\ f = 0$
by (*simp add*: *Bochner_Integration.integral_empty*)
lemma *has_bochner_integral_restrict_space*:
fixes $f :: 'a \Rightarrow 'b::\{banach, second\_countable\_topology\}$
assumes Ω: $\Omega \cap space\ M \in sets\ M$
shows $has\_bochner\_integral\ (restrict\_space\ M\ \Omega)\ f\ i$
$\longleftrightarrow has\_bochner\_integral\ M\ (\lambda x.\ indicator\ \Omega\ x *_R f\ x)\ i$
by (*simp add*: *integrable_restrict_space* [*OF assms*] *integral_restrict_space* [*OF assms*] *has_bochner_integral_iff*)

lemma *integrable_restrict_UNIV*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::\{banach, second\_countable\_topology\}$

assumes S: $S \in$ *sets lebesgue*
shows *integrable lebesgue* (λx. *if* $x \in S$ *then* $f\ x$ *else* 0) $\longleftrightarrow$ *integrable* (*lebesgue_on* S) f
using *has_bochner_integral_restrict_space* [*of* S *lebesgue* f] *assms*
by (*simp add*: *integrable.simps indicator_scaleR_eq_if*)

lemma *integral_mono_lebesgue_on_AE*:
fixes f::_ $\Rightarrow$ *real*
assumes f: *integrable* (*lebesgue_on* T) f
and *gf*: *AE* x *in* (*lebesgue_on* S). $g\ x \leq f\ x$
and *f0*: *AE* x *in* (*lebesgue_on* T). $0 \leq f\ x$
and $S \subseteq T$ **and** S: $S \in$ *sets lebesgue* **and** T: $T \in$ *sets lebesgue*
shows ($\int x.\ g\ x\ \partial$(*lebesgue_on* S)) $\leq$ ($\int x.\ f\ x\ \partial$(*lebesgue_on* T))
proof −
have ($\int x.\ g\ x\ \partial$(*lebesgue_on* S)) = ($\int x.$ (*if* $x \in S$ *then* $g\ x$ *else* 0) ∂*lebesgue*)
by (*simp add*: *Lebesgue_Measure.integral_restrict_UNIV* S)
also have ... $\leq$ ($\int x.$ (*if* $x \in T$ *then* $f\ x$ *else* 0) ∂*lebesgue*)
proof (*rule Bochner_Integration.integral_mono_AE'*)
show *integrable lebesgue* (λx. *if* $x \in T$ *then* $f\ x$ *else* 0)
by (*simp add*: *integrable_restrict_UNIV* T f)
show *AE* x *in lebesgue*. (*if* $x \in S$ *then* $g\ x$ *else* 0) $\leq$ (*if* $x \in T$ *then* $f\ x$ *else* 0)
using *assms* **by** (*auto simp*: *AE_restrict_space_iff*)
show *AE* x *in lebesgue*. $0 \leq$ (*if* $x \in T$ *then* $f\ x$ *else* 0)
using *f0* **by** (*simp add*: *AE_restrict_space_iff* T)
qed
also have ... = ($\int x.\ f\ x\ \partial$(*lebesgue_on* T))
using *Lebesgue_Measure.integral_restrict_UNIV* T **by** *blast*
finally show *?thesis* **.**
qed

6.12.3 Borel measurability

lemma *borel_measurable_if_I*:
fixes f :: 'a::*euclidean_space* $\Rightarrow$ 'b::*euclidean_space*
assumes f: $f \in$ *borel_measurable* (*lebesgue_on* S) **and** S: $S \in$ *sets lebesgue*
shows (λx. *if* $x \in S$ *then* $f\ x$ *else* 0) $\in$ *borel_measurable lebesgue*
proof −
have *eq*: $\{x.\ x \notin S\} \cup \{x.\ f\ x \in Y\} = \{x.\ x \notin S\} \cup \{x.\ f\ x \in Y\} \cap S$ **for** Y
by *blast*
show *?thesis*
using f S
apply (*simp add*: *vimage_def in_borel_measurable_borel Ball_def*)
apply (*elim all_forward imp_forward asm_rl*)
apply (*simp only*: *Collect_conj_eq Collect_disj_eq imp_conv_disj eq*)
apply (*auto simp*: *Compl_eq* [*symmetric*] *Compl_in_sets_lebesgue sets_restrict_space_iff*)
done
qed

lemma *borel_measurable_if_D*:

fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes $(\lambda x.\ if\ x \in S\ then\ f\ x\ else\ 0) \in borel\_measurable\ lebesgue$
shows $f \in borel\_measurable\ (lebesgue\_on\ S)$
using *assms* **by** (*smt* (*verit*) *measurable_lebesgue_cong measurable_restrict_space1*)

lemma *borel_measurable_if*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes $S \in sets\ lebesgue$
shows $(\lambda x.\ if\ x \in S\ then\ f\ x\ else\ 0) \in borel\_measurable\ lebesgue \longleftrightarrow f \in borel\_measurable\ (lebesgue\_on\ S)$
using *assms borel_measurable_if_D borel_measurable_if_I* **by** *blast*

lemma *borel_measurable_if_lebesgue_on*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes $S \in sets\ lebesgue\ T \in sets\ lebesgue\ S \subseteq T$
shows $(\lambda x.\ if\ x \in S\ then\ f\ x\ else\ 0) \in borel\_measurable\ (lebesgue\_on\ T) \longleftrightarrow f \in borel\_measurable\ (lebesgue\_on\ S)$
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs* **then show** *?rhs*
using *measurable_restrict_mono* [*OF* _ ‹$S \subseteq T$›]
by (*subst measurable_lebesgue_cong* [**where** $g = (\lambda x.\ if\ x \in S\ then\ f\ x\ else\ 0)$]) *auto*
next
assume *?rhs* **then show** *?lhs*
by (*simp add*: ‹$S \in sets\ lebesgue$› *borel_measurable_if_I measurable_restrict_space1*)
qed

lemma *borel_measurable_vimage_halfspace_component_lt*:
$f \in borel\_measurable\ (lebesgue\_on\ S) \longleftrightarrow (\forall a\ i.\ i \in Basis \longrightarrow \{x \in S.\ f\ x \cdot i < a\} \in sets\ (lebesgue\_on\ S))$
by (*force simp add*: *space_restrict_space trans* [*OF borel_measurable_iff_halfspace_less*])

lemma *borel_measurable_vimage_halfspace_component_ge*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
shows $f \in borel\_measurable\ (lebesgue\_on\ S) \longleftrightarrow (\forall a\ i.\ i \in Basis \longrightarrow \{x \in S.\ f\ x \cdot i \geq a\} \in sets\ (lebesgue\_on\ S))$
by (*force simp add*: *space_restrict_space trans* [*OF borel_measurable_iff_halfspace_ge*])

lemma *borel_measurable_vimage_halfspace_component_gt*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
shows $f \in borel\_measurable\ (lebesgue\_on\ S) \longleftrightarrow (\forall a\ i.\ i \in Basis \longrightarrow \{x \in S.\ f\ x \cdot i > a\} \in sets\ (lebesgue\_on\ S))$
by (*force simp add*: *space_restrict_space trans* [*OF borel_measurable_iff_halfspace_greater*])

lemma *borel_measurable_vimage_halfspace_component_le*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
shows $f \in borel\_measurable\ (lebesgue\_on\ S) \longleftrightarrow (\forall a\ i.\ i \in Basis \longrightarrow \{x \in S.\ f\ x \cdot i \leq a\} \in sets\ (lebesgue\_on\ S))$

by (*force simp add*: *space_restrict_space trans* [*OF borel_measurable_iff_halfspace_le*])

lemma
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
shows *borel_measurable_vimage_open_interval*:
$f \in borel\_measurable\ (lebesgue\_on\ S) \longleftrightarrow$
$(\forall\, a\ b.\ \{x \in S.\ f\ x \in box\ a\ b\} \in sets\ (lebesgue\_on\ S))$ (**is** *?thesis1*)
and *borel_measurable_vimage_open*:
$f \in borel\_measurable\ (lebesgue\_on\ S) \longleftrightarrow$
$(\forall\, T.\ open\ T \longrightarrow \{x \in S.\ f\ x \in T\} \in sets\ (lebesgue\_on\ S))$ (**is** *?thesis2*)
proof −
have $\{x \in S.\ f\ x \in box\ a\ b\} \in sets\ (lebesgue\_on\ S)$ **if** $f \in borel\_measurable\ (lebesgue\_on\ S)$ **for** $a\ b$
proof −
have $S = S \cap space\ lebesgue$
by *simp*
then have $S \cap (f -` box\ a\ b) \in sets\ (lebesgue\_on\ S)$
by (*metis* (*no_types*) *box_borel in_borel_measurable_borel inf_sup_aci*(*1*) *space_restrict_space that*)
then show *?thesis*
by (*simp add*: *Collect_conj_eq vimage_def*)
qed
moreover
have $\{x \in S.\ f\ x \in T\} \in sets\ (lebesgue\_on\ S)$
if T: $\bigwedge a\ b.\ \{x \in S.\ f\ x \in box\ a\ b\} \in sets\ (lebesgue\_on\ S)$ *open* T **for** T
proof −
obtain $\mathcal{D}$ **where** *countable* $\mathcal{D}$ **and** $\mathcal{D}$: $\bigwedge X.\ X \in \mathcal{D} \Longrightarrow \exists\, a\ b.\ X = box\ a\ b$ $\bigcup \mathcal{D} = T$
using *open_countable_Union_open_box that* ‹*open T*› **by** *metis*
then have *eq*: $\{x \in S.\ f\ x \in T\} = (\bigcup U \in \mathcal{D}.\ \{x \in S.\ f\ x \in U\})$
by *blast*
have $\{x \in S.\ f\ x \in U\} \in sets\ (lebesgue\_on\ S)$ **if** $U \in \mathcal{D}$ **for** U
using *that* T $\mathcal{D}$ **by** *blast*
then show *?thesis*
by (*auto simp*: *eq intro*: *Sigma_Algebra.sets.countable_UN'* [*OF* ‹*countable* $\mathcal{D}$›])
qed
moreover
have *eq*: $\{x \in S.\ f\ x \cdot i < a\} = \{x \in S.\ f\ x \in \{y.\ y \cdot i < a\}\}$ **for** $i\ a$
by *auto*
have $f \in borel\_measurable\ (lebesgue\_on\ S)$
if $\bigwedge T.\ open\ T \Longrightarrow \{x \in S.\ f\ x \in T\} \in sets\ (lebesgue\_on\ S)$
by (*metis* (*no_types*) *eq borel_measurable_vimage_halfspace_component_lt open_halfspace_component_lt that*)
ultimately show *?thesis1 ?thesis2*
by *blast*+
qed

lemma *borel_measurable_vimage_closed*:

fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
shows $f \in borel\_measurable\ (lebesgue\_on\ S) \longleftrightarrow$
$(\forall T.\ closed\ T \longrightarrow \{x \in S.\ f\ x \in T\} \in sets\ (lebesgue\_on\ S))$
proof −
have *eq*: $\{x \in S.\ f\ x \in T\} = S - (S \cap f\ -`\ (-\ T))$ **for** T
by *auto*
show *?thesis*
unfolding *borel_measurable_vimage_open eq*
by (*metis Diff_Diff_Int closed_Compl diff_eq open_Compl sets.Diff sets_lebesgue_on_refl vimage_Compl*)
qed

lemma *borel_measurable_vimage_closed_interval*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
shows $f \in borel\_measurable\ (lebesgue\_on\ S) \longleftrightarrow$
$(\forall a\ b.\ \{x \in S.\ f\ x \in cbox\ a\ b\} \in sets\ (lebesgue\_on\ S))$
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs* **then show** *?rhs*
using *borel_measurable_vimage_closed* **by** *blast*
next
assume *RHS*: *?rhs*
have $\{x \in S.\ f\ x \in T\} \in sets\ (lebesgue\_on\ S)$ **if** *open T* **for** T
proof −
obtain $\mathcal{D}$ **where** *countable* $\mathcal{D}$ **and** $\mathcal{D}$: $\mathcal{D} \subseteq Pow\ T\ \bigwedge X.\ X \in \mathcal{D} \Longrightarrow \exists a\ b.\ X = cbox\ a\ b\ \bigcup \mathcal{D} = T$
using *open_countable_Union_open_cbox that* ‹*open T*› **by** *metis*
then have *eq*: $\{x \in S.\ f\ x \in T\} = (\bigcup U \in \mathcal{D}.\ \{x \in S.\ f\ x \in U\})$
by *blast*
have $\{x \in S.\ f\ x \in U\} \in sets\ (lebesgue\_on\ S)$ **if** $U \in \mathcal{D}$ **for** U
using *that* $\mathcal{D}$ **by** (*metis RHS*)
then show *?thesis*
by (*auto simp*: *eq intro*: *Sigma_Algebra.sets.countable_UN'* [*OF* ‹*countable* $\mathcal{D}$›])
qed
then show *?lhs*
by (*simp add*: *borel_measurable_vimage_open*)
qed

lemma *borel_measurable_vimage_borel*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
shows $f \in borel\_measurable\ (lebesgue\_on\ S) \longleftrightarrow$
$(\forall T.\ T \in sets\ borel \longrightarrow \{x \in S.\ f\ x \in T\} \in sets\ (lebesgue\_on\ S))$
(**is** *?lhs* = *?rhs*)
proof
assume *f*: *?lhs*
then show *?rhs*
using *measurable_sets* [*OF f*]
by (*simp add*: *Collect_conj_eq inf_sup_aci*(1) *space_restrict_space vim-*

age_def)
qed (*simp add*: *borel_measurable_vimage_open_interval*)

lemma *lebesgue_measurable_vimage_borel*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes $f \in borel\_measurable\ lebesgue$ $T \in sets\ borel$
shows $\{x.\ f\ x \in T\} \in sets\ lebesgue$
using *assms borel_measurable_vimage_borel* [*of f UNIV*] **by** *auto*

lemma *borel_measurable_lebesgue_preimage_borel*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
shows $f \in borel\_measurable\ lebesgue \longleftrightarrow$
$(\forall T.\ T \in sets\ borel \longrightarrow \{x.\ f\ x \in T\} \in sets\ lebesgue)$
by (*smt* (*verit*, *best*) *Collect_cong UNIV_I borel_measurable_vimage_borel lebesgue_on_UNIV_eq*)

6.12.4 Measurability of continuous functions

lemma *continuous_imp_measurable_on_sets_lebesgue*:
assumes *f*: *continuous_on S f* **and** *S*: $S \in sets\ lebesgue$
shows $f \in borel\_measurable\ (lebesgue\_on\ S)$
by (*metis borel_measurable_continuous_on_restrict borel_measurable_subalgebra f*
lebesgue_on_UNIV_eq mono_restrict_space sets_completionI_sets sets_lborel space_borel
space_lebesgue_on space_restrict_space subsetI)

lemma *id_borel_measurable_lebesgue* [*iff*]: $id \in borel\_measurable\ lebesgue$
by (*simp add*: *measurable_completion*)

lemma *id_borel_measurable_lebesgue_on* [*iff*]: $id \in borel\_measurable\ (lebesgue\_on\ S)$
by (*simp add*: *measurable_completion measurable_restrict_space1*)

context
begin

interpretation *sigma_finite_measure interval_measure* $(\lambda x.\ x)$
by (*rule sigma_finite_interval_measure*) *auto*
interpretation *finite_product_sigma_finite* $\lambda\_.$ *interval_measure* $(\lambda x.\ x)$ *Basis*
proof qed *simp*

lemma *lborel_eq_real*: $lborel = interval\_measure\ (\lambda x.\ x)$
unfolding *lborel_def Basis_real_def*
using *distr_id*[*of interval_measure* $(\lambda x.\ x)$]
by (*subst distr_component*[*symmetric*])
(*simp_all add*: *distr_distr comp_def del*: *distr_id cong*: *distr_cong*)

lemma *lborel_eq*: $lborel = distr\ (\Pi_M\ b{\in}Basis.\ lborel)\ borel\ (\lambda f.\ \sum b{\in}Basis.\ f\ b *_R\ b)$

by (*subst lborel_def*) (*simp add*: *lborel_eq_real*)

lemma *nn_integral_lborel_prod*:
assumes [*measurable*]: $\bigwedge b.\ b \in Basis \Longrightarrow f\ b \in borel\_measurable\ borel$
assumes *nn*[*simp*]: $\bigwedge b\ x.\ b \in Basis \Longrightarrow 0 \le f\ b\ x$
shows $(\int^+ x.\ (\prod b{\in}Basis.\ f\ b\ (x \cdot b))\ \partial lborel) = (\prod b{\in}Basis.\ (\int^+ x.\ f\ b\ x\ \partial lborel))$
by (*simp add*: *lborel_def nn_integral_distr product_nn_integral_prod product_nn_integral_singleton*)

lemma *emeasure_lborel_Icc*[*simp*]:
fixes $l\ u :: real$
assumes [*simp*]: $l \le u$
shows $emeasure\ lborel\ \{l\ ..\ u\} = u - l$
by (*simp add*: *emeasure_interval_measure_Icc lborel_eq_real*)

lemma *emeasure_lborel_Icc_eq*: $emeasure\ lborel\ \{l\ ..\ u\} = ennreal\ (if\ l \le u\ then\ u - l\ else\ 0)$
by *simp*

lemma *emeasure_lborel_cbox*[*simp*]:
assumes [*simp*]: $\bigwedge b.\ b \in Basis \Longrightarrow l \cdot b \le u \cdot b$
shows $emeasure\ lborel\ (cbox\ l\ u) = (\prod b{\in}Basis.\ (u - l) \cdot b)$
proof −
have $(\lambda x.\ \prod b{\in}Basis.\ indicator\ \{l{\cdot}b\ ..\ u{\cdot}b\}\ (x \cdot b) :: ennreal) = indicator\ (cbox\ l\ u)$
by (*auto simp*: *fun_eq_iff cbox_def split*: *split_indicator*)
then have $emeasure\ lborel\ (cbox\ l\ u) = (\int^+ x.\ (\prod b{\in}Basis.\ indicator\ \{l{\cdot}b\ ..\ u{\cdot}b\}\ (x \cdot b))\ \partial lborel)$
by *simp*
also have $\ldots = (\prod b{\in}Basis.\ (u - l) \cdot b)$
by (*subst nn_integral_lborel_prod*) (*simp_all add*: *prod_ennreal inner_diff_left*)
finally show *?thesis* **.**
qed

lemma *AE_lborel_singleton*: $AE\ x\ in\ lborel::'a::euclidean\_space\ measure.\ x \neq c$
using *SOME_Basis AE_discrete_difference* [*of* {*c*} *lborel*] *emeasure_lborel_cbox* [*of c c*]
by (*auto simp add*: *power_0_left*)

lemma *emeasure_lborel_Ioo*[*simp*]:
assumes [*simp*]: $l \le u$
shows $emeasure\ lborel\ \{l\ <..<\ u\} = ennreal\ (u - l)$
proof −
have $emeasure\ lborel\ \{l\ <..<\ u\} = emeasure\ lborel\ \{l\ ..\ u\}$
using *AE_lborel_singleton*[*of u*] *AE_lborel_singleton*[*of l*] **by** (*intro emeasure_eq_AE*) *auto*
then show *?thesis*
by *simp*
qed

lemma *emeasure_lborel_Ioc*[*simp*]:
assumes [*simp*]: $l \leq u$
shows *emeasure lborel* $\{l <.. u\} = ennreal\ (u - l)$
by (*simp add*: *emeasure_interval_measure_Ioc lborel_eq_real*)

lemma *emeasure_lborel_Ico*[*simp*]:
assumes [*simp*]: $l \leq u$
shows *emeasure lborel* $\{l ..< u\} = ennreal\ (u - l)$
proof −
have *emeasure lborel* $\{l ..< u\} =$ *emeasure lborel* $\{l .. u\}$
using *AE_lborel_singleton*[*of u*] *AE_lborel_singleton*[*of l*] **by** (*intro emeasure_eq_AE*) *auto*
then show *?thesis*
by *simp*
qed

lemma *emeasure_lborel_box*[*simp*]:
assumes [*simp*]: $\bigwedge b.\ b \in Basis \Longrightarrow l \cdot b \leq u \cdot b$
shows *emeasure lborel* $(box\ l\ u) = (\prod b{\in}Basis.\ (u - l) \cdot b)$
proof −
have $(\lambda x.\ \prod b{\in}Basis.\ indicator\ \{l{\cdot}b <..< u{\cdot}b\}\ (x \cdot b) :: ennreal) = indicator\ (box\ l\ u)$
by (*auto simp*: *fun_eq_iff box_def split*: *split_indicator*)
then have *emeasure lborel* $(box\ l\ u) = (\int^+ x.\ (\prod b{\in}Basis.\ indicator\ \{l{\cdot}b <..< u{\cdot}b\}\ (x \cdot b))\ \partial lborel)$
by *simp*
also have $\ldots = (\prod b{\in}Basis.\ (u - l) \cdot b)$
by (*subst nn_integral_lborel_prod*) (*simp_all add*: *prod_ennreal inner_diff_left*)
finally show *?thesis* **.**
qed

lemma *emeasure_lborel_cbox_eq*:
emeasure lborel $(cbox\ l\ u) = (if\ \forall\ b{\in}Basis.\ l \cdot b \leq u \cdot b\ then\ \prod b{\in}Basis.\ (u - l) \cdot b\ else\ 0)$
using *box_eq_empty*(*2*)[*THEN iffD2*, *of u l*] **by** (*auto simp*: *not_le*)

lemma *emeasure_lborel_box_eq*:
emeasure lborel $(box\ l\ u) = (if\ \forall\ b{\in}Basis.\ l \cdot b \leq u \cdot b\ then\ \prod b{\in}Basis.\ (u - l) \cdot b\ else\ 0)$
using *box_eq_empty*(*1*)[*THEN iffD2*, *of u l*] **by** (*auto simp*: *not_le dest*!: *less_imp_le*) *force*

lemma *emeasure_lborel_singleton*[*simp*]: *emeasure lborel* $\{x\} = 0$
using *emeasure_lborel_cbox*[*of x x*] *nonempty_Basis*
by (*auto simp del*: *emeasure_lborel_cbox nonempty_Basis*)

lemma *emeasure_lborel_cbox_finite*: *emeasure lborel* $(cbox\ a\ b) < \infty$
by (*auto simp*: *emeasure_lborel_cbox_eq*)

lemma *emeasure_lborel_box_finite*: *emeasure lborel* (*box a b*) $< \infty$
by (*auto simp*: *emeasure_lborel_box_eq*)

lemma *emeasure_lborel_ball_finite*: *emeasure lborel* (*ball c r*) $< \infty$
by (*metis bounded_ball bounded_subset_cbox_symmetric cbox_borel emeasure_lborel_cbox_finite emeasure_mono order_le_less_trans sets_lborel*)

lemma *emeasure_lborel_cball_finite*: *emeasure lborel* (*cball c r*) $< \infty$
by (*metis bounded_cball bounded_subset_cbox_symmetric cbox_borel emeasure_lborel_cbox_finite emeasure_mono order_le_less_trans sets_lborel*)

lemma *fmeasurable_cbox* [*iff*]: *cbox a b* $\in$ *fmeasurable lborel*
and *fmeasurable_box* [*iff*]: *box a b* $\in$ *fmeasurable lborel*
by (*auto simp*: *fmeasurable_def emeasure_lborel_box_eq emeasure_lborel_cbox_eq*)

lemma
fixes *l u* :: *real*
assumes [*simp*]: $l \le u$
shows *measure_lborel_Icc*[*simp*]: *measure lborel* $\{l .. u\} = u - l$
and *measure_lborel_Ico*[*simp*]: *measure lborel* $\{l ..< u\} = u - l$
and *measure_lborel_Ioc*[*simp*]: *measure lborel* $\{l <.. u\} = u - l$
and *measure_lborel_Ioo*[*simp*]: *measure lborel* $\{l <..< u\} = u - l$
by (*simp_all add*: *measure_def*)

lemma
assumes [*simp*]: $\bigwedge b.\ b \in Basis \Longrightarrow l \cdot b \le u \cdot b$
shows *measure_lborel_box*[*simp*]: *measure lborel* (*box l u*) = $(\prod b{\in}Basis.\ (u - l) \cdot b)$
and *measure_lborel_cbox*[*simp*]: *measure lborel* (*cbox l u*) = $(\prod b{\in}Basis.\ (u - l) \cdot b)$
by (*simp_all add*: *measure_def inner_diff_left prod_nonneg*)

lemma *measure_lborel_cbox_eq*:
measure lborel (*cbox l u*) = (*if* $\forall\, b{\in}Basis.\ l \cdot b \le u \cdot b$ *then* $\prod b{\in}Basis.\ (u - l) \cdot b$ *else* 0)
using *box_eq_empty*(2)[*THEN iffD2*, *of u l*] **by** (*auto simp*: *not_le*)

lemma *measure_lborel_box_eq*:
measure lborel (*box l u*) = (*if* $\forall\, b{\in}Basis.\ l \cdot b \le u \cdot b$ *then* $\prod b{\in}Basis.\ (u - l) \cdot b$ *else* 0)
using *box_eq_empty*(1)[*THEN iffD2*, *of u l*] **by** (*auto simp*: *not_le dest*!: *less_imp_le*)
force

lemma *measure_lborel_singleton*[*simp*]: *measure lborel* $\{x\} = 0$
by (*simp add*: *measure_def*)

lemma *sigma_finite_lborel*: *sigma_finite_measure lborel*

proof
show $\exists A::'a\ set\ set.\ countable\ A \wedge A \subseteq sets\ lborel \wedge \bigcup A = space\ lborel \wedge (\forall a{\in}A.\ emeasure\ lborel\ a \neq \infty)$
by (*intro exI*[*of* _ *range* ($\lambda n::nat.\ box\ (-\ real\ n *_R One)\ (real\ n *_R One)$)])
(*auto simp*: *emeasure_lborel_cbox_eq UN_box_eq_UNIV*)
qed

end

lemma *emeasure_lborel_UNIV* [*simp*]: *emeasure lborel* (*UNIV*::'*a*::*euclidean_space set*) = ∞
proof −
{ fix *n*::*nat*
let *?Ba* = *Basis* :: '*a set*
have *real n* ≤ (*2*::*real*) ^ *card ?Ba* ∗ *real n*
by (*simp add*: *mult_le_cancel_right1*)
also
have ... ≤ (*2*::*real*) ^ *card ?Ba* ∗ *real* (*Suc n*) ^ *card ?Ba*
apply (*rule mult_left_mono*)
apply (*metis DIM_positive One_nat_def less_eq_Suc_le less_imp_le of_nat_le_iff of_nat_power self_le_power zero_less_Suc*)
apply (*simp*)
done
finally have *real n* ≤ (*2*::*real*) ^ *card ?Ba* ∗ *real* (*Suc n*) ^ *card ?Ba* .
} note [*intro*!] = *this*
show *?thesis*
unfolding *UN_box_eq_UNIV*[*symmetric*]
apply (*subst SUP_emeasure_incseq*[*symmetric*])
apply (*auto simp*: *incseq_def subset_box inner_add_left*
simp del: *Sup_eq_top_iff SUP_eq_top_iff*
intro!: *ennreal_SUP_eq_top*)
done
qed

lemma *emeasure_lborel_countable*:
fixes *A* :: '*a*::*euclidean_space set*
assumes *countable A*
shows *emeasure lborel A* = *0*
proof −
have *A* ⊆ ($\bigcup i.$ {*from_nat_into A i*}) **using** *from_nat_into_surj assms* **by** *force*
then have *emeasure lborel A* ≤ *emeasure lborel* ($\bigcup i.$ {*from_nat_into A i*})
by (*intro emeasure_mono*) *auto*
also have *emeasure lborel* ($\bigcup i.$ {*from_nat_into A i*}) = *0*
by (*rule emeasure_UN_eq_0*) *auto*
finally show *?thesis*
by *simp*
qed

lemma *countable_imp_null_set_lborel*: *countable A* $\Longrightarrow$ *A* $\in$ *null_sets lborel*
by (*simp add*: *null_sets_def emeasure_lborel_countable sets.countable*)

lemma *finite_imp_null_set_lborel*: *finite A* $\Longrightarrow$ *A* $\in$ *null_sets lborel*
by (*intro countable_imp_null_set_lborel countable_finite*)

lemma *insert_null_sets_iff* [*simp*]: *insert a N* $\in$ *null_sets lebesgue* $\longleftrightarrow$ *N* $\in$ *null_sets lebesgue*
by (*meson completion.complete2 finite.simps finite_imp_null_set_lborel null_sets.insert_in_sets*

null_sets_completionI subset_insertI)

lemma *insert_null_sets_lebesgue_on_iff* [*simp*]:
assumes *a* $\in$ *S S* $\in$ *sets lebesgue*
shows *insert a N* $\in$ *null_sets* (*lebesgue_on S*) $\longleftrightarrow$ *N* $\in$ *null_sets* (*lebesgue_on S*)
by (*simp add*: *assms null_sets_restrict_space*)

lemma *lborel_neq_count_space*[*simp*]:
fixes *A* :: ('*a*::*ordered_euclidean_space*) *set*
shows *lborel* $\neq$ *count_space A*
by (*metis finite.simps finite_imp_null_set_lborel insert_not_empty null_sets_count_space singleton_iff*)

lemma *mem_closed_if_AE_lebesgue_open*:
assumes *open S closed C*
assumes *AE x* $\in$ *S in lebesgue. x* $\in$ *C*
assumes *x* $\in$ *S*
shows *x* $\in$ *C*
proof (*rule ccontr*)
assume *xC*: *x* $\notin$ *C*
with *openE*[*of S* $-$ *C*] *assms*
obtain *e* **where** *e*: *0* $<$ *e ball x e* $\subseteq$ *S* $-$ *C*
by *blast*
then obtain *a b* **where** *box*: *x* $\in$ *box a b box a b* $\subseteq$ *S* $-$ *C*
by (*metis rational_boxes order_trans*)
then have *0* $<$ *emeasure lebesgue* (*box a b*)
by (*auto simp*: *emeasure_lborel_box_eq mem_box algebra_simps intro*!: *prod_pos*)
also have ... $\leq$ *emeasure lebesgue* (*S* $-$ *C*)
using *assms box*
by (*auto intro*!: *emeasure_mono*)
also have ... $=$ *0*
using *assms*
by (*auto simp*: *eventually_ae_filter completion.complete2 set_diff_eq null_setsD1*)
finally show *False* **by** *simp*
qed

lemma *mem_closed_if_AE_lebesgue*: *closed C* $\Longrightarrow$ (*AE x in lebesgue. x* $\in$ *C*) $\Longrightarrow$ *x* $\in$ *C*

using *mem_closed_if_AE_lebesgue_open*[*OF open_UNIV*] **by** *simp*

6.12.5 Affine transformation on the Lebesgue-Borel

lemma *lborel_eqI*:
fixes *M* :: *'a*::*euclidean_space measure*
assumes *emeasure_eq*: $\bigwedge l\ u.\ (\bigwedge b.\ b \in Basis \Longrightarrow l \cdot b \leq u \cdot b) \Longrightarrow emeasure\ M\ (box\ l\ u) = (\prod b{\in}Basis.\ (u - l) \cdot b)$
assumes *sets_eq*: *sets M* = *sets borel*
shows *lborel* = *M*
proof (*rule measure_eqI_generator_eq*)
let *?E* = *range* ($\lambda(a,\ b).\ box\ a\ b$::*'a set*)
show *Int_stable ?E*
by (*auto simp*: *Int_stable_def box_Int_box*)

show *?E* $\subseteq$ *Pow UNIV sets lborel* = *sigma_sets UNIV ?E sets M* = *sigma_sets UNIV ?E*
by (*simp_all add*: *borel_eq_box sets_eq*)

let *?A* = λn::*nat. box* (− (*real n* $*_R$ *One*)) (*real n* $*_R$ *One*) :: *'a set*
show *range ?A* $\subseteq$ *?E* ($\bigcup i.$ *?A i*) = *UNIV*
unfolding *UN_box_eq_UNIV* **by** *auto*
show *emeasure lborel* (*?A i*) $\neq \infty$ **for** *i* **by** *auto*
show *emeasure lborel X* = *emeasure M X* **if** *X* $\in$ *?E* **for** *X*
using *that box_eq_empty*(*1*) **by** (*fastforce simp*: *emeasure_eq emeasure_lborel_box_eq*)
qed

lemma *lborel_affine_euclidean*:
fixes *c* :: *'a*::*euclidean_space* $\Rightarrow$ *real* **and** *t*
defines *T x* $\equiv$ *t* + ($\sum j{\in}Basis.\ (c\ j * (x \cdot j))\ *_R\ j$)
assumes *c*: $\bigwedge j.\ j \in Basis \Longrightarrow c\ j \neq 0$
shows *lborel* = *density* (*distr lborel borel T*) (λ_. ($\prod j{\in}Basis.\ |c\ j|$)) (**is** _ = *?D*)
proof (*rule lborel_eqI*)
let *?B* = *Basis* :: *'a set*
fix *l u* **assume** *le*: $\bigwedge b.\ b \in ?B \Longrightarrow l \cdot b \leq u \cdot b$
have [*measurable*]: *T* $\in$ *borel* $\rightarrow_M$ *borel*
by (*simp add*: *T_def*[*abs_def*])
have *eq*: *T* −‘ *box l u* = *box*
($\sum j{\in}Basis.\ (((if\ 0 < c\ j\ then\ l - t\ else\ u - t) \cdot j)\ /\ c\ j)\ *_R\ j$)
($\sum j{\in}Basis.\ (((if\ 0 < c\ j\ then\ u - t\ else\ l - t) \cdot j)\ /\ c\ j)\ *_R\ j$)
using *c* **by** (*auto simp*: *box_def T_def field_simps inner_simps divide_less_eq*)
with *le c* **show** *emeasure ?D* (*box l u*) = ($\prod b{\in}?B.\ (u - l) \cdot b$)
by (*auto simp*: *emeasure_density emeasure_distr nn_integral_multc emeasure_lborel_box_eq inner_simps*
field_split_simps ennreal_mult'[*symmetric*] *prod_nonneg prod.distrib*[*symmetric*]
intro!: *prod.cong*)
qed *simp*

lemma *lborel_affine*:
fixes $t :: 'a::euclidean\_space$
shows $c \neq 0 \Longrightarrow lborel = density\ (distr\ lborel\ borel\ (\lambda x.\ t + c *_R x))\ (\lambda\_.\ |c| \hat{}\ DIM('a))$
using *lborel_affine_euclidean*[**where** $c=\lambda\_::'a.\ c$ **and** $t=t$]
unfolding *scaleR_scaleR*[*symmetric*] *scaleR_sum_right*[*symmetric*] *euclidean_representation prod_constant* **by** *simp*

lemma *lborel_real_affine*:
$c \neq 0 \Longrightarrow lborel = density\ (distr\ lborel\ borel\ (\lambda x.\ t + c * x))\ (\lambda\_.\ ennreal\ (abs\ c))$
using *lborel_affine*[*of c t*] **by** *simp*

lemma *AE_borel_affine*:
fixes $P :: real \Rightarrow bool$
shows $c \neq 0 \Longrightarrow Measurable.pred\ borel\ P \Longrightarrow AE\ x\ in\ lborel.\ P\ x \Longrightarrow AE\ x\ in\ lborel.\ P\ (t + c * x)$
by (*subst lborel_real_affine*[**where** $t=-\ t\ /\ c$ **and** $c=1\ /\ c$])
(*simp_all add*: *AE_density AE_distr_iff field_simps*)

lemma *nn_integral_real_affine*:
fixes $c :: real$ **assumes** [*measurable*]: $f \in borel\_measurable\ borel$ **and** c: $c \neq 0$
shows $(\int^+ x.\ f\ x\ \partial lborel) = |c| * (\int^+ x.\ f\ (t + c * x)\ \partial lborel)$
by (*subst lborel_real_affine*[*OF c, of t*])
(*simp add*: *nn_integral_density nn_integral_distr nn_integral_cmult*)

lemma *lborel_integrable_real_affine*:
fixes $f :: real \Rightarrow 'a :: \{banach, second\_countable\_topology\}$
assumes f: *integrable lborel f*
shows $c \neq 0 \Longrightarrow integrable\ lborel\ (\lambda x.\ f\ (t + c * x))$
using *f f*[*THEN borel_measurable_integrable*] **unfolding** *integrable_iff_bounded*
by (*subst* (*asm*) *nn_integral_real_affine*[**where** $c=c$ **and** $t=t$]) (*auto simp*: *ennreal_mult_less_top*)

lemma *lborel_integrable_real_affine_iff*:
fixes $f :: real \Rightarrow 'a :: \{banach, second\_countable\_topology\}$
shows $c \neq 0 \Longrightarrow integrable\ lborel\ (\lambda x.\ f\ (t + c * x)) \longleftrightarrow integrable\ lborel\ f$
using
lborel_integrable_real_affine[*of f c t*]
lborel_integrable_real_affine[*of* $\lambda x.\ f\ (t + c * x)\ 1/c\ -t/c$]
by (*auto simp add*: *field_simps*)

lemma *lborel_integral_real_affine*:
fixes $f :: real \Rightarrow 'a :: \{banach, second\_countable\_topology\}$ **and** $c :: real$
assumes c: $c \neq 0$ **shows** $(\int x.\ f\ x\ \partial\ lborel) = |c| *_R (\int x.\ f\ (t + c * x)\ \partial lborel)$
proof *cases*
assume *f*[*measurable*]: *integrable lborel f* **then show** *?thesis*
using *c f f*[*THEN borel_measurable_integrable*] *f*[*THEN lborel_integrable_real_affine, of c t*]

by (*subst lborel_real_affine*[*OF c, of t*])
(*simp add*: *integral_density integral_distr*)
next
assume ¬ *integrable lborel f* **with** *c* **show** *?thesis*
by (*simp add*: *lborel_integrable_real_affine_iff not_integrable_integral_eq*)
qed

lemma
fixes *c* :: *'a*::*euclidean_space* ⇒ *real* **and** *t*
assumes *c*: ⋀*j*. *j* ∈ *Basis* ⟹ *c j* ≠ *0*
defines *T* == (λ*x*. *t* + (∑ *j*∈*Basis*. (*c j* ∗ (*x* ∙ *j*)) ∗$_R$ *j*))
shows *lebesgue_affine_euclidean*: *lebesgue* = *density* (*distr lebesgue lebesgue T*) (λ_. (∏ *j*∈*Basis*. |*c j*|)) (**is** _ = *?D*)
and *lebesgue_affine_measurable*: *T* ∈ *lebesgue* →$_M$ *lebesgue*
proof −
have *T_borel*[*measurable*]: *T* ∈ *borel* →$_M$ *borel*
by (*auto simp*: *T_def*[*abs_def*])
{ fix *A* :: *'a set* **assume** *A*: *A* ∈ *sets borel*
then have *emeasure lborel A* = *0* ⟷ *emeasure* (*density* (*distr lborel borel T*) (λ_. (∏ *j*∈*Basis*. |*c j*|))) *A* = *0*
unfolding *T_def* **using** *c* **by** (*subst lborel_affine_euclidean*[*symmetric*]) *auto*
also have ... ⟷ *emeasure* (*distr lebesgue lborel T*) *A* = *0*
using *A c* **by** (*simp add*: *distr_completion emeasure_density nn_integral_cmult prod_nonneg cong*: *distr_cong*)
finally have *emeasure lborel A* = *0* ⟷ *emeasure* (*distr lebesgue lborel T*) *A* = *0* **. }**
then have *eq*: *null_sets lborel* = *null_sets* (*distr lebesgue lborel T*)
by (*auto simp*: *null_sets_def*)

show *T* ∈ *lebesgue* →$_M$ *lebesgue*
by (*simp add*: *completion.measurable_completion2 eq measurable_completion*)

have *lebesgue* = *completion* (*density* (*distr lborel borel T*) (λ_. (∏ *j*∈*Basis*. |*c j*|)))
using *c* **by** (*subst lborel_affine_euclidean*[*of c t*]) (*simp_all add*: *T_def*[*abs_def*])
also have ... = *density* (*completion* (*distr lebesgue lborel T*)) (λ_. (∏ *j*∈*Basis*. |*c j*|))
using *c* **by** (*auto intro*!: *always_eventually prod_pos completion_density_eq simp*: *distr_completion cong*: *distr_cong*)
also have ... = *density* (*distr lebesgue lebesgue T*) (λ_. (∏ *j*∈*Basis*. |*c j*|))
by (*subst completion.completion_distr_eq*) (*auto simp*: *eq measurable_completion*)
finally show *lebesgue* = *density* (*distr lebesgue lebesgue T*) (λ_. (∏ *j*∈*Basis*. |*c j*|)) **.**
qed

corollary *lebesgue_real_affine*:
c ≠ *0* ⟹ *lebesgue* = *density* (*distr lebesgue lebesgue* (λ*x*. *t* + *c* ∗ *x*)) (λ_. *ennreal* (*abs c*))
using *lebesgue_affine_euclidean* [**where** *c*= λ*x*::*real*. *c*] **by** *simp*

lemma *nn_integral_real_affine_lebesgue*:
fixes c :: *real* **assumes** f[*measurable*]: $f \in$ *borel_measurable lebesgue* **and** c: $c \neq 0$
shows $(\int^+ x.\ f\ x\ \partial lebesgue) =$ *ennreal*$|c| * (\int^+ x.\ f(t + c * x)\ \partial lebesgue)$
proof −
have $(\int^+ x.\ f\ x\ \partial lebesgue) = (\int^+ x.\ f\ x\ \partial density\ (distr\ lebesgue\ lebesgue\ (\lambda x.\ t + c * x))\ (\lambda x.\ ennreal\ |c|))$
using *lebesgue_real_affine* c **by** *auto*
also have $\ldots = \int^+ x.\ ennreal\ |c| * f\ x\ \partial distr\ lebesgue\ lebesgue\ (\lambda x.\ t + c * x)$
by (*subst nn_integral_density*) *auto*
also have $\ldots =$ *ennreal* $|c| *$ *integral*^{N} $(distr\ lebesgue\ lebesgue\ (\lambda x.\ t + c * x))$ f
using f *measurable_distr_eq1 nn_integral_cmult* **by** *blast*
also have $\ldots = |c| * (\int^+ x.\ f(t + c * x)\ \partial lebesgue)$
using *lebesgue_affine_measurable*[**where** $c = \lambda x{::}real.\ c$]
by (*subst nn_integral_distr*) (*force*+)
finally show *?thesis* **.**
qed

lemma *lebesgue_measurable_scaling*[*measurable*]: $(*_R)\ x \in lebesgue \rightarrow_M lebesgue$
proof *cases*
assume $x = 0$
then have $(*_R)\ x = (\lambda x.\ 0{::}'a)$
by (*auto simp*: *fun_eq_iff*)
then show *?thesis* **by** *auto*
next
assume $x \neq 0$ **then show** *?thesis*
using *lebesgue_affine_measurable*[*of* $\lambda\_.\ x\ 0$]
unfolding *scaleR_scaleR*[*symmetric*] *scaleR_sum_right*[*symmetric*] *euclidean_representation*
by (*auto simp add*: *ac_simps*)
qed

lemma
fixes m :: *real* **and** δ :: $'a{::}euclidean\_space$
defines $T\ r\ d\ x \equiv r *_R x + d$
shows *emeasure_lebesgue_affine*: *emeasure lebesgue* $(T\ m\ \delta\ `\ S) = |m|\ \hat{}\ DIM('a) *$ *emeasure lebesgue* S (**is** *?e*)
and *measure_lebesgue_affine*: *measure lebesgue* $(T\ m\ \delta\ `\ S) = |m|\ \hat{}\ DIM('a) *$ *measure lebesgue* S (**is** *?m*)
proof −
show *?e*
proof *cases*
assume $m = 0$ **then show** *?thesis*
by (*simp add*: *image_constant_conv T_def*[*abs_def*])
next
let *?T* $= T\ m\ \delta$ **and** *?T'* $= T\ (1\ /\ m)\ (-\ ((1/m) *_R \delta))$
assume $m \neq 0$
then have *s_comp_s*: *?T'* $\circ$ *?T* $=$ *id ?T* $\circ$ *?T'* $=$ *id*

```
    by (auto simp: T_def[abs_def] fun_eq_iff scaleR_add_right scaleR_diff_right)
   then have inv ?T' = ?T bij ?T'
    by (auto intro: inv_unique_comp o_bij)
   then have eq: T m δ ‘ S = T (1 / m) ((−1/m) *R δ) −‘ S ∩ space lebesgue
    using bij_vimage_eq_inv_image[OF ‹bij ?T'›, of S] by auto

  have trans_eq_T: (λx. δ + (∑j∈Basis. (m * (x • j)) *R j)) = T m δ for m δ
   unfolding T_def[abs_def] scaleR_scaleR[symmetric] scaleR_sum_right[symmetric]
    by (auto simp add: euclidean_representation ac_simps)

   have T[measurable]: T r d ∈ lebesgue →M lebesgue for r d
    using lebesgue_affine_measurable[of λ_. r d]
    by (cases r = 0) (auto simp: trans_eq_T T_def[abs_def])

   show ?thesis
   proof cases
    assume S ∈ sets lebesgue with ‹m ≠ 0› show ?thesis
     unfolding eq
     apply (subst lebesgue_affine_euclidean[of λ_. m δ])
      apply (simp_all add: emeasure_density trans_eq_T nn_integral_cmult
emeasure_distr
                    del: space_completion emeasure_completion)
     apply (simp add: vimage_comp s_comp_s)
     done
   next
    assume S ∉ sets lebesgue
    moreover have ?T ‘ S ∉ sets lebesgue
    proof
     assume ?T ‘ S ∈ sets lebesgue
     then have ?T −‘ (?T ‘ S) ∩ space lebesgue ∈ sets lebesgue
      by (rule measurable_sets[OF T])
     also have ?T −‘ (?T ‘ S) ∩ space lebesgue = S
      by (simp add: vimage_comp s_comp_s eq)
     finally show False using ‹S ∉ sets lebesgue› by auto
    qed
    ultimately show ?thesis
     by (simp add: emeasure_notin_sets)
   qed
  qed
  show ?m
   unfolding measure_def ‹?e› by (simp add: enn2real_mult prod_nonneg)
qed

lemma lebesgue_real_scale:
  assumes c ≠ 0
  shows   lebesgue = density (distr lebesgue lebesgue (λx. c * x)) (λx. ennreal |c|)
  using assms by (subst lebesgue_affine_euclidean[of λ_. c 0]) simp_all

lemma divideR_right:
```

fixes $x\ y :: 'a{::}real\_normed\_vector$
shows $r \neq 0 \Longrightarrow y = x\ /_R\ r \longleftrightarrow r *_R y = x$
using *scaleR_cancel_left*[*of r y x* $/_R$ *r*] **by** *simp*

lemma *lborel_has_bochner_integral_real_affine_iff*:
fixes $x :: 'a :: \{banach, second\_countable\_topology\}$
shows $c \neq 0 \Longrightarrow$
has_bochner_integral lborel f x $\longleftrightarrow$
has_bochner_integral lborel $(\lambda x.\ f\ (t + c * x))\ (x\ /_R\ |c|)$
unfolding *has_bochner_integral_iff lborel_integrable_real_affine_iff*
by (*simp_all add*: *lborel_integral_real_affine*[*symmetric*] *divideR_right cong*: *conj_cong*)

lemma *lborel_distr_uminus*: *distr lborel borel uminus* = (*lborel* :: *real measure*)
by (*subst lborel_real_affine*[*of* -1 *0*])
(*auto simp*: *density_1 one_ennreal_def*[*symmetric*])

lemma *lborel_distr_mult*:
assumes (*c*::*real*) $\neq$ *0*
shows *distr lborel borel* $((*)\ c)$ = *density lborel* $(\lambda\_.\ inverse\ |c|)$
proof−
have *distr lborel borel* $((*)\ c)$ = *distr lborel lborel* $((*)\ c)$ **by** (*simp cong*: *distr_cong*)
also from *assms* **have** ... = *density lborel* $(\lambda\_.\ inverse\ |c|)$
by (*subst lborel_real_affine*[*of inverse c 0*]) (*auto simp*: *o_def distr_density_distr*)
finally show *?thesis* **.**
qed

lemma *lborel_distr_mult'*:
assumes (*c*::*real*) $\neq$ *0*
shows *lborel* = *density* (*distr lborel borel* $((*)\ c)$) $(\lambda\_.\ |c|)$
proof−
have *lborel* = *density lborel* $(\lambda\_.\ 1)$ **by** (*rule density_1*[*symmetric*])
also from *assms* **have** $(\lambda\_.\ 1 :: ennreal) = (\lambda\_.\ inverse\ |c| * |c|)$ **by** (*intro ext*) *simp*
also have *density lborel* ... = *density* (*density lborel* $(\lambda\_.\ inverse\ |c|)$) $(\lambda\_.\ |c|)$
by (*subst density_density_eq*) (*auto simp*: *ennreal_mult*)
also from *assms* **have** *density lborel* $(\lambda\_.\ inverse\ |c|)$ = *distr lborel borel* $((*)\ c)$
by (*rule lborel_distr_mult*[*symmetric*])
finally show *?thesis* **.**
qed

lemma *lborel_distr_plus*:
fixes $c :: 'a{::}euclidean\_space$
shows *distr lborel borel* $((+)\ c)$ = *lborel*
by (*subst lborel_affine*[*of 1 c*], *auto simp*: *density_1*)

interpretation *lborel*: *sigma_finite_measure lborel*
by (*rule sigma_finite_lborel*)

interpretation *lborel_pair*: *pair_sigma_finite lborel lborel* **..**

lemma *lborel_prod*:
lborel $\bigotimes_M$ *lborel* = (*lborel* :: (*'a*::*euclidean_space* × *'b*::*euclidean_space*) *measure*)
proof (*rule lborel_eqI*[*symmetric*], *clarify*)
fix *la ua* :: *'a* **and** *lb ub* :: *'b*
assume *lu*: ⋀*a b*. (*a*, *b*) ∈ *Basis* ⟹ (*la*, *lb*) · (*a*, *b*) ≤ (*ua*, *ub*) · (*a*, *b*)
have [*simp*]:
⋀*b*. *b* ∈ *Basis* ⟹ *la* · *b* ≤ *ua* · *b*
⋀*b*. *b* ∈ *Basis* ⟹ *lb* · *b* ≤ *ub* · *b*
inj_on (λ*u*. (*u*, *0*)) *Basis inj_on* (λ*u*. (*0*, *u*)) *Basis*
(λ*u*. (*u*, *0*)) ' *Basis* ∩ (λ*u*. (*0*, *u*)) ' *Basis* = {}
box (*la*, *lb*) (*ua*, *ub*) = *box la ua* × *box lb ub*
using *lu*[*of* _ *0*] *lu*[*of 0*] **by** (*auto intro*!: *inj_onI simp add*: *Basis_prod_def ball_Un box_def*)
show *emeasure* (*lborel* $\bigotimes_M$ *lborel*) (*box* (*la*, *lb*) (*ua*, *ub*)) =
ennreal (*prod* ((·) ((*ua*, *ub*) − (*la*, *lb*))) *Basis*)
by (*simp add*: *lborel.emeasure_pair_measure_Times Basis_prod_def prod.union_disjoint prod.reindex ennreal_mult inner_diff_left prod_nonneg*)
qed (*simp add*: *borel_prod*[*symmetric*])

lemma *lborelD_Collect*[*measurable* (*raw*)]: {*x*∈*space borel*. *P x*} ∈ *sets borel* ⟹ {*x*∈*space lborel*. *P x*} ∈ *sets lborel*
by *simp*

lemma *lborelD*[*measurable* (*raw*)]: *A* ∈ *sets borel* ⟹ *A* ∈ *sets lborel*
by *simp*

lemma *emeasure_bounded_finite*:
assumes *bounded A* **shows** *emeasure lborel A* < ∞
proof −
obtain *a b* **where** *A* ⊆ *cbox a b*
by (*meson bounded_subset_cbox_symmetric* ‹*bounded A*›)
then have *emeasure lborel A* ≤ *emeasure lborel* (*cbox a b*)
by (*intro emeasure_mono*) *auto*
then show *?thesis*
by (*auto simp*: *emeasure_lborel_cbox_eq prod_nonneg less_top*[*symmetric*] *top_unique split*: *if_split_asm*)
qed

lemma *emeasure_compact_finite*: *compact A* ⟹ *emeasure lborel A* < ∞
using *emeasure_bounded_finite*[*of A*] **by** (*auto intro*: *compact_imp_bounded*)

lemma *borel_integrable_compact*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::{*banach*, *second_countable_topology*}
assumes *compact S continuous_on S f*
shows *integrable lborel* (λ*x*. *indicator S x* ∗$_R$ *f x*)

proof *cases*
assume $S \neq \{\}$
have *continuous_on S* ($\lambda x.$ *norm* (*f x*))
using *assms* **by** (*intro continuous_intros*)
from *continuous_attains_sup*[*OF* ‹*compact S*› ‹$S \neq \{\}$› *this*]
obtain *M* **where** *M*: $\bigwedge x.\ x \in S \Longrightarrow$ *norm* (*f x*) $\leq$ *M*
by *auto*
show *?thesis*
proof (*rule integrable_bound*)
show *integrable lborel* ($\lambda x.$ *indicator S x* $*$ *M*)
using *assms* **by** (*auto intro*!: *emeasure_compact_finite borel_compact integrable_mult_left*)
show ($\lambda x.$ *indicator S x* $*_R$ *f x*) $\in$ *borel_measurable lborel*
using *assms* **by** (*auto intro*!: *borel_measurable_continuous_on_indicator borel_compact*)
show *AE x in lborel.* *norm* (*indicator S x* $*_R$ *f x*) $\leq$ *norm* (*indicator S x* $*$ *M*)
by (*auto split*: *split_indicator simp*: *abs_real_def dest*!: *M*)
qed
qed *simp*

lemma *borel_integrable_atLeastAtMost*:
fixes *f* :: *real* $\Rightarrow$ *real*
assumes *f*: $\bigwedge x.\ a \leq x \Longrightarrow x \leq b \Longrightarrow$ *isCont f x*
shows *integrable lborel* ($\lambda x.$ *f x* $*$ *indicator* {*a* .. *b*} *x*) (**is** *integrable* _ *?f*)
proof −
have *integrable lborel* ($\lambda x.$ *indicator* {*a* .. *b*} *x* $*_R$ *f x*)
proof (*rule borel_integrable_compact*)
from *f* **show** *continuous_on* {*a..b*} *f*
by (*auto intro*: *continuous_at_imp_continuous_on*)
qed *simp*
then show *?thesis*
by (*auto simp*: *mult.commute*)
qed

6.12.6 Lebesgue measurable sets

abbreviation *lmeasurable* :: *'a*::*euclidean_space set set*
where
lmeasurable ≡ *fmeasurable lebesgue*

lemma *not_measurable_UNIV* [*simp*]: *UNIV* $\notin$ *lmeasurable*
by (*simp add*: *fmeasurable_def*)

lemma *lmeasurable_iff_integrable*:
S $\in$ *lmeasurable* $\longleftrightarrow$ *integrable lebesgue* (*indicator S* :: *'a*::*euclidean_space* $\Rightarrow$ *real*)
by (*auto simp*: *fmeasurable_def integrable_iff_bounded borel_measurable_indicator_iff ennreal_indicator*)

lemma *lmeasurable_cbox* [*iff*]: *cbox a b* ∈ *lmeasurable*
and *lmeasurable_box* [*iff*]: *box a b* ∈ *lmeasurable*
by (*auto simp*: *fmeasurable_def emeasure_lborel_box_eq emeasure_lborel_cbox_eq*)

lemma
fixes *a*::*real*
shows *lmeasurable_interval* [*iff*]: {*a*..*b*} ∈ *lmeasurable* {*a*<..<*b*} ∈ *lmeasurable*
by (*metis box_real lmeasurable_box lmeasurable_cbox*)+

lemma *fmeasurable_compact*: *compact S* ⟹ *S* ∈ *fmeasurable lborel*
using *emeasure_compact_finite*[*of S*] **by** (*intro fmeasurableI*) (*auto simp*: *borel_compact*)

lemma *lmeasurable_compact*: *compact S* ⟹ *S* ∈ *lmeasurable*
using *fmeasurable_compact* **by** (*force simp*: *fmeasurable_def*)

lemma *measure_frontier*:
bounded S ⟹ *measure lebesgue* (*frontier S*) = *measure lebesgue* (*closure S*) − *measure lebesgue* (*interior S*)
using *closure_subset interior_subset*
by (*auto simp*: *frontier_def fmeasurable_compact intro*!: *measurable_measure_Diff*)

lemma *lmeasurable_closure*:
bounded S ⟹ *closure S* ∈ *lmeasurable*
by (*simp add*: *lmeasurable_compact*)

lemma *lmeasurable_frontier*:
bounded S ⟹ *frontier S* ∈ *lmeasurable*
by (*simp add*: *compact_frontier_bounded lmeasurable_compact*)

lemma *lmeasurable_open*: *bounded S* ⟹ *open S* ⟹ *S* ∈ *lmeasurable*
using *emeasure_bounded_finite*[*of S*] **by** (*intro fmeasurableI*) (*auto simp*: *borel_open*)

lemma *lmeasurable_ball* [*simp*]: *ball a r* ∈ *lmeasurable*
by (*simp add*: *lmeasurable_open*)

lemma *lmeasurable_cball* [*simp*]: *cball a r* ∈ *lmeasurable*
by (*simp add*: *lmeasurable_compact*)

lemma *lmeasurable_interior*: *bounded S* ⟹ *interior S* ∈ *lmeasurable*
by (*simp add*: *bounded_interior lmeasurable_open*)

lemma *null_sets_cbox_Diff_box*: *cbox a b* − *box a b* ∈ *null_sets lborel*
by (*simp add*: *emeasure_Diff emeasure_lborel_box_eq emeasure_lborel_cbox_eq null_setsI subset_box*)

lemma *bounded_set_imp_lmeasurable*:
assumes *bounded S S* ∈ *sets lebesgue* **shows** *S* ∈ *lmeasurable*
by (*metis assms bounded_Un emeasure_bounded_finite emeasure_completion fmeasurableI main_part_null_part_Un*)

lemma *finite_measure_lebesgue_on*: $S \in$ *lmeasurable* $\Longrightarrow$ *finite_measure* (*lebesgue_on* S)
by (*auto simp*: *finite_measureI fmeasurable_def emeasure_restrict_space*)

lemma *integrable_const_ivl* [*iff*]:
fixes *a*::'*a*::*ordered_euclidean_space*
shows *integrable* (*lebesgue_on* {*a*..*b*}) ($\lambda x.$ *c*)
by (*metis cbox_interval finite_measure.integrable_const finite_measure_lebesgue_on lmeasurable_cbox*)

6.12.7 Translation preserves Lebesgue measure

lemma *sigma_sets_image*:
assumes *S*: $S \in$ *sigma_sets* Ω *M* **and** $M \subseteq$ *Pow* Ω f ' $\Omega = \Omega$ *inj_on f* Ω
and *M*: $\bigwedge y.\ y \in M \Longrightarrow f$ ' $y \in M$
shows (f ' *S*) $\in$ *sigma_sets* Ω *M*
using *S*
proof (*induct S rule*: *sigma_sets.induct*)
case (*Basic a*) **then show** *?case*
by (*simp add*: *M*)
next
case *Empty* **then show** *?case*
by (*simp add*: *sigma_sets.Empty*)
next
case (*Compl a*)
with *assms* **show** *?case*
by (*metis inj_on_image_set_diff sigma_sets.Compl sigma_sets_into_sp*)
next
case (*Union a*) **then show** *?case*
by (*metis image_UN sigma_sets.simps*)
qed

lemma *null_sets_translation*:
assumes $N \in$ *null_sets lborel* **shows** $\{x.\ x - a \in N\} \in$ *null_sets lborel*
proof −
have [*simp*]: ($\lambda x.\ x + a$) ' $N = \{x.\ x - a \in N\}$
by *force*
show *?thesis*
using *assms emeasure_lebesgue_affine* [*of 1 a N*] **by** (*auto simp*: *null_sets_def*)
qed

lemma *lebesgue_sets_translation*:
fixes $f :: 'a \Rightarrow 'a$::*euclidean_space*
assumes *S*: $S \in$ *sets lebesgue*
shows (($\lambda x.\ a + x$) ' *S*) $\in$ *sets lebesgue*
proof −
have *im_eq*: (+) *a* ' $A = \{x.\ x - a \in A\}$ **for** *A*
by *force*

have $((\lambda x.\ a + x)\ `\ S) = ((\lambda x.\ -a + x)\ -`\ S) \cap (space\ lebesgue)$
using *image_iff* **by** *fastforce*
also have $\ldots \in sets\ lebesgue$
proof (*rule measurable_sets* [*OF measurableI assms*])
fix $A :: 'b\ set$
assume *A*: $A \in sets\ lebesgue$
have *vim_eq*: $(\lambda x.\ x - a)\ -`\ A = (+)\ a\ `\ A$ **for** *A*
by *force*
have $\exists s\ n\ N'.\ (+)\ a\ `\ (S \cup N) = s \cup n \wedge s \in sets\ borel \wedge N' \in null\_sets\ lborel \wedge n \subseteq N'$
if $S \in sets\ borel$ **and** $N' \in null\_sets\ lborel$ **and** $N \subseteq N'$ **for** *S N N'*
proof (*intro exI conjI*)
show $(+)\ a\ `\ (S \cup N) = (\lambda x.\ a + x)\ `\ S \cup (\lambda x.\ a + x)\ `\ N$
by *auto*
show $(\lambda x.\ a + x)\ `\ N' \in null\_sets\ lborel$
using *that* **by** (*auto simp*: *null_sets_translation im_eq*)
qed (*use that im_eq* **in** *auto*)
with *A* **have** $(\lambda x.\ x - a)\ -`\ A \in sets\ lebesgue$
by (*force simp*: *vim_eq completion_def intro*!: *sigma_sets_image*)
then show $(+)\ (-\ a)\ -`\ A \cap space\ lebesgue \in sets\ lebesgue$
by (*auto simp*: *vimage_def im_eq*)
qed *auto*
finally show *?thesis* **.**
qed

lemma *measurable_translation*:
$S \in lmeasurable \Longrightarrow ((+)\ a\ `\ S) \in lmeasurable$
using *emeasure_lebesgue_affine* [*of 1 a S*]
by (*smt* (*verit*, *best*) *add.commute ennreal_1 fmeasurable_def image_cong lambda_one*

lebesgue_sets_translation mem_Collect_eq power_one scaleR_one)

lemma *measurable_translation_subtract*:
$S \in lmeasurable \Longrightarrow ((\lambda x.\ x - a)\ `\ S) \in lmeasurable$
using *measurable_translation* [*of S − a*] **by** (*simp cong*: *image_cong_simp*)

lemma *measure_translation*:
measure lebesgue $((+)\ a\ `\ S) =$ *measure lebesgue S*
using *measure_lebesgue_affine* [*of 1 a S*] **by** (*simp add*: *ac_simps cong*: *image_cong_simp*)

lemma *measure_translation_subtract*:
measure lebesgue $((\lambda x.\ x - a)\ `\ S) =$ *measure lebesgue S*
using *measure_translation* [*of − a*] **by** (*simp cong*: *image_cong_simp*)

6.12.8 A nice lemma for negligibility proofs

lemma *summable_iff_suminf_neq_top*: $(\bigwedge n.\ f\ n \geq 0) \Longrightarrow \neg\ summable\ f \Longrightarrow (\sum i.\ ennreal\ (f\ i)) = top$

```
  by (metis summable_suminf_not_top)

proposition starlike_negligible_bounded_gmeasurable:
  fixes S :: 'a :: euclidean_space set
  assumes S: S ∈ sets lebesgue and bounded S
      and eq1: ⋀c x. ⟦(c *R x) ∈ S; 0 ≤ c; x ∈ S⟧ ⟹ c = 1
    shows S ∈ null_sets lebesgue
proof −
  obtain M where 0 < M S ⊆ ball 0 M
    using ‹bounded S› by (auto dest: bounded_subset_ballD)

  let ?f = λn. root DIM('a) (Suc n)

  have ?f n *R x ∈ S ⟹ x ∈ (*R) (1 / ?f n) ‘ S for x n
    by (rule image_eqI[of _ _ ?f n *R x]) auto
  then have vimage_eq_image: (*R) (?f n) −‘ S = (*R) (1 / ?f n) ‘ S for n
    by auto

  have eq: (1 / ?f n) ^ DIM('a) = 1 / Suc n for n
    by (simp add: field_simps)

  { fix n x assume x: root DIM('a) (1 + real n) *R x ∈ S
    have 1 * norm x ≤ root DIM('a) (1 + real n) * norm x
      by (rule mult_mono) auto
    also have ... < M
      using x ‹S ⊆ ball 0 M› by auto
    finally have norm x < M by simp }
  note less_M = this

  have (∑ n. ennreal (1 / Suc n)) = top
      using not_summable_harmonic[where 'a=real] summable_Suc_iff[where
f=λn. 1 / (real n)]
    by (intro summable_iff_suminf_neq_top) (auto simp add: inverse_eq_divide)
  then have top * emeasure lebesgue S = (∑ n. (1 / ?f n)^DIM('a) * emeasure
lebesgue S)
    unfolding ennreal_suminf_multc eq by simp
  also have ... = (∑ n. emeasure lebesgue ((*R) (?f n) −‘ S))
    unfolding vimage_eq_image using emeasure_lebesgue_affine[of 1 / ?f n 0 S
for n] by simp
  also have ... = emeasure lebesgue (⋃ n. (*R) (?f n) −‘ S)
  proof (intro suminf_emeasure)
    show disjoint_family (λn. (*R) (?f n) −‘ S)
      unfolding disjoint_family_on_def
    proof safe
      fix m n :: nat and x assume m ≠ n ?f m *R x ∈ S ?f n *R x ∈ S
      with eq1[of ?f m / ?f n ?f n *R x] show x ∈ {}
        by auto
    qed
    have (*R) (?f i) −‘ S ∈ sets lebesgue for i
```

```
    using measurable_sets[OF lebesgue_measurable_scaling[of ?f i] S] by auto
  then show range (λi. (*R) (?f i) -` S) ⊆ sets lebesgue
    by auto
 qed
 also have ... ≤ emeasure lebesgue (ball 0 M :: 'a set)
  using less_M by (intro emeasure_mono) auto
 also have ... < top
  using lmeasurable_ball by (auto simp: fmeasurable_def)
 finally have emeasure lebesgue S = 0
  by (simp add: ennreal_top_mult split: if_split_asm)
 then show S ∈ null_sets lebesgue
  unfolding null_sets_def using ‹S ∈ sets lebesgue› by auto
qed

corollary starlike_negligible_compact:
 compact S ⟹ (⋀c x. ⟦(c *R x) ∈ S; 0 ≤ c; x ∈ S⟧ ⟹ c = 1) ⟹ S ∈ null_sets
lebesgue
 using starlike_negligible_bounded_gmeasurable[of S] by (auto simp: compact_eq_bounded_closed)

proposition outer_regular_lborel_le:
 assumes B[measurable]: B ∈ sets borel and 0 < (e::real)
 obtains U where open U B ⊆ U and emeasure lborel (U - B) ≤ e
proof -
 let ?μ = emeasure lborel
 let ?B = λn::nat. ball 0 n :: 'a set
 let ?e = λn. e*((1/2)^Suc n)
 have ∀ n. ∃ U. open U ∧ ?B n ∩ B ⊆ U ∧ ?μ (U - B) < ?e n
 proof
  fix n :: nat
  let ?A = density lborel (indicator (?B n))
  have emeasure_A: X ∈ sets borel ⟹ emeasure ?A X = ?μ (?B n ∩ X) for X
   by (auto simp: emeasure_density borel_measurable_indicator indicator_inter_arith[symmetric])

  have finite_A: emeasure ?A (space ?A) ≠ ∞
   using emeasure_bounded_finite[of ?B n] by (auto simp: emeasure_A)
  interpret A: finite_measure ?A
   by rule fact
  have emeasure ?A B + ?e n > (INF U∈{U. B ⊆ U ∧ open U}. emeasure ?A
U)
   using ‹0<e› by (auto simp: outer_regular[OF _ finite_A B, symmetric])
  then obtain U where U: B ⊆ U open U and muU: ?μ (?B n ∩ B) + ?e n
> ?μ (?B n ∩ U)
   unfolding INF_less_iff by (auto simp: emeasure_A)
  moreover
  { have ?μ ((?B n ∩ U) - B) = ?μ ((?B n ∩ U) - (?B n ∩ B))
     using U by (intro arg_cong[where f=?μ]) auto
    also have ... = ?μ (?B n ∩ U) - ?μ (?B n ∩ B)
     using U A.emeasure_finite[of B]
      by (intro emeasure_Diff) (auto simp del: A.emeasure_finite simp: emea-
```

```
sure_A)
      also have ... < ?e n
        using U muU A.emeasure_finite[of B]
        by (subst minus_less_iff_ennreal)
           (auto simp del: A.emeasure_finite simp: emeasure_A less_top ac_simps
intro!: emeasure_mono)
      finally have ?μ ((?B n ∩ U) − B) < ?e n . }
    ultimately show ∃ U. open U ∧ ?B n ∩ B ⊆ U ∧ ?μ (U − B) < ?e n
      by (intro exI[of _ ?B n ∩ U]) auto
  qed
  then obtain U
    where U: ⋀n. open (U n) ⋀n. ?B n ∩ B ⊆ U n ⋀n. ?μ (U n − B) < ?e n
    by metis
  show ?thesis
  proof
    { fix x assume x ∈ B
      moreover
      obtain n where norm x < real n
        using reals_Archimedean2 by blast
      ultimately have x ∈ (⋃ n. U n)
        using U(2)[of n] by auto }
    note ∗ = this
    then show open (⋃ n. U n) B ⊆ (⋃ n. U n)
      using U by auto
    have ?μ (⋃ n. U n − B) ≤ (∑ n. ?μ (U n − B))
      using U(1) by (intro emeasure_subadditive_countably) auto
    also have ... ≤ (∑ n. ennreal (?e n))
      using U(3) by (intro suminf_le) (auto intro: less_imp_le)
    also have ... = ennreal (e ∗ 1)
      using ‹0<e› by (intro suminf_ennreal_eq sums_mult power_half_series)
auto
    finally show emeasure lborel ((⋃ n. U n) − B) ≤ ennreal e
      by simp
  qed
qed

lemma outer_regular_lborel:
  assumes B: B ∈ sets borel and 0 < (e::real)
  obtains U where open U B ⊆ U emeasure lborel (U − B) < e
proof −
  obtain U where U: open U B ⊆ U and emeasure lborel (U−B) ≤ e/2
    using outer_regular_lborel_le [OF B, of e/2] ‹e > 0›
    by force
  moreover have ennreal (e/2) < ennreal e
    using ‹e > 0› by (simp add: ennreal_lessI)
  ultimately have emeasure lborel (U−B) < e
    by auto
  with U show ?thesis
    using that by auto
```

qed

lemma *completion_upper*:
assumes *A*: $A \in sets\ (completion\ M)$
obtains A' **where** $A \subseteq A'\ A' \in sets\ M\ A' - A \in null\_sets\ (completion\ M)$
$emeasure\ (completion\ M)\ A = emeasure\ M\ A'$
proof −
from *AE_notin_null_part*[*OF A*] **obtain** *N* **where** *N*: $N \in null\_sets\ M\ null\_part\ M\ A \subseteq N$
by (*meson assms null_part*)
let $?A' = main\_part\ M\ A \cup N$
show *?thesis*
proof
show $A \subseteq ?A'$
using ‹*null_part M A* ⊆ *N*› *assms main_part_null_part_Un* **by** *blast*
have *main_part M A* ⊆ *A*
using *assms main_part_null_part_Un* **by** *auto*
then have $?A' - A \subseteq N$
by *blast*
with *N* **show** $?A' - A \in null\_sets\ (completion\ M)$
by (*blast intro*: *null_sets_completionI completion.complete_measure_axioms complete_measure.complete2*)
show *emeasure* (*completion M*) *A* = *emeasure M* (*main_part M A* ∪ *N*)
using *A* ‹*N* ∈ *null_sets M*› **by** (*simp add*: *emeasure_Un_null_set*)
qed (*use A N* **in** *auto*)
qed

lemma *sets_lebesgue_outer_open*:
fixes *e*::*real*
assumes *S*: $S \in sets\ lebesgue$ **and** $e > 0$
obtains *T* **where** *open T S* ⊆ *T* (*T* − *S*) ∈ *lmeasurable emeasure lebesgue* (*T* − *S*) < *ennreal e*
proof −
obtain S' **where** S': $S \subseteq S'\ S' \in sets\ borel$
and *null*: $S' - S \in null\_sets\ lebesgue$
and *em*: *emeasure lebesgue S* = *emeasure lborel S*′
using *completion_upper*[*of S lborel*] *S* **by** *auto*
then have *f_S*′: $S' \in sets\ borel$
using *S* **by** (*auto simp*: *fmeasurable_def*)
with *outer_regular_lborel*[*OF _* ‹*0*<*e*›]
obtain *U* **where** *U*: *open U S*′ ⊆ *U emeasure lborel* (*U* − *S*′) < *e*
by *blast*
show *thesis*
proof
show *open U S* ⊆ *U*
using *f_S*′ *U S*′ **by** *auto*
have (*U* − *S*) = (*U* − *S*′) ∪ (*S*′ − *S*)
using *S*′ *U* **by** *auto*
then have *eq*: *emeasure lebesgue* (*U* − *S*) = *emeasure lborel* (*U* − *S*′)

```
    using null by (simp add: U(1) emeasure_Un_null_set f_S' sets.Diff)
  have (U − S) ∈ sets lebesgue
    by (simp add: S U(1) sets.Diff)
  then show (U − S) ∈ lmeasurable
    unfolding fmeasurable_def using U(3) eq less_le_trans by fastforce
  with eq U show emeasure lebesgue (U − S) < e
    by (simp add: eq)
  qed
qed

lemma sets_lebesgue_inner_closed:
  fixes e::real
  assumes S ∈ sets lebesgue e > 0
  obtains T where closed T T ⊆ S S−T ∈ lmeasurable emeasure lebesgue (S −
T) < ennreal e
proof −
  have −S ∈ sets lebesgue
    using assms by (simp add: Compl_in_sets_lebesgue)
  then obtain T where open T −S ⊆ T
        and T: (T − −S) ∈ lmeasurable emeasure lebesgue (T − −S) < e
    using sets_lebesgue_outer_open assms by blast
  show thesis
  proof
    show closed (−T)
      using ‹open T› by blast
    show −T ⊆ S
      using ‹− S ⊆ T› by auto
    show S − ( −T) ∈ lmeasurable emeasure lebesgue (S − (− T)) < e
      using T by (auto simp: Int_commute)
  qed
qed

lemma lebesgue_openin:
   ⟦openin (top_of_set S) T; S ∈ sets lebesgue⟧ ⟹ T ∈ sets lebesgue
  by (metis borel_open openin_open sets.Int sets_completionI_sets sets_lborel)

lemma lebesgue_closedin:
   ⟦closedin (top_of_set S) T; S ∈ sets lebesgue⟧ ⟹ T ∈ sets lebesgue
 by (metis borel_closed closedin_closed sets.Int sets_completionI_sets sets_lborel)
```

6.12.9 *F_sigma* and *G_delta* sets.

```
inductive fsigma :: 'a::topological_space set ⇒ bool where
  (⋀n::nat. closed (F n)) ⟹ fsigma (⋃(F ‘ UNIV))

inductive gdelta :: 'a::topological_space set ⇒ bool where
  (⋀n::nat. open (F n)) ⟹ gdelta (⋂(F ‘ UNIV))

lemma fsigma_UNIV [iff]: fsigma (UNIV :: 'a::real_inner set)
```

```
proof −
  have (UNIV ::'a set) = (⋃ i. cball 0 (of_nat i))
    by (auto simp: real_arch_simple)
  then show ?thesis
    by (metis closed_cball fsigma.intros)
qed

lemma fsigma_Union_compact:
  fixes S :: 'a::{real_normed_vector,heine_borel} set
  shows fsigma S ⟷ (∃ F::nat ⇒ 'a set. range F ⊆ Collect compact ∧ S = ⋃(F
‘ UNIV))
proof safe
  assume fsigma S
  then obtain F :: nat ⇒ 'a set where F: range F ⊆ Collect closed S = ⋃(F ‘
UNIV)
    by (meson fsigma.cases image_subsetI mem_Collect_eq)
  then have ∃ D::nat ⇒ 'a set. range D ⊆ Collect compact ∧ ⋃(D ‘ UNIV) = F
i for i
    using closed_Union_compact_subsets [of F i]
    by (metis image_subsetI mem_Collect_eq range_subsetD)
  then obtain D :: nat ⇒ nat ⇒ 'a set
    where D: ⋀i. range (D i) ⊆ Collect compact ∧ ⋃((D i) ‘ UNIV) = F i
    by metis
  let ?DD = λn. (λ(i,j). D i j) (prod_decode n)
  show ∃ F::nat ⇒ 'a set. range F ⊆ Collect compact ∧ S = ⋃(F ‘ UNIV)
  proof (intro exI conjI)
    show range ?DD ⊆ Collect compact
      using D by clarsimp (metis mem_Collect_eq rangeI split_conv subsetCE
surj_pair)
    show S = ⋃ (range ?DD)
    proof
      show S ⊆ ⋃ (range ?DD)
        using D F
      by clarsimp (metis UN_iff old.prod.case prod_decode_inverse prod_encode_eq)
      show ⋃ (range ?DD) ⊆ S
        using D F  by fastforce
    qed
  qed
next
  fix F :: nat ⇒ 'a set
  assume range F ⊆ Collect compact and S = ⋃(F ‘ UNIV)
  then show fsigma (⋃(F ‘ UNIV))
    by (simp add: compact_imp_closed fsigma.intros image_subset_iff)
qed

lemma gdelta_imp_fsigma: gdelta S ⟹ fsigma (− S)
proof (induction rule: gdelta.induct)
  case (1 F)
  have − ⋂(F ‘ UNIV) = (⋃ i. −(F i))
```

```
    by auto
  then show ?case
    by (simp add: fsigma.intros closed_Compl 1)
qed

lemma fsigma_imp_gdelta: fsigma S ⟹ gdelta (− S)
proof (induction rule: fsigma.induct)
  case (1 F)
  have − ⋃(F ‘ UNIV) = (⋂i. −(F i))
    by auto
  then show ?case
    by (simp add: 1 gdelta.intros open_closed)
qed

lemma gdelta_complement: gdelta(− S) ⟷ fsigma S
  using fsigma_imp_gdelta gdelta_imp_fsigma by force

lemma lebesgue_set_almost_fsigma:
  assumes S ∈ sets lebesgue
  obtains C T where fsigma C T ∈ null_sets lebesgue C ∪ T = S disjnt C T
proof −
  { fix n::nat
    obtain T where closed T T ⊆ S S−T ∈ lmeasurable emeasure lebesgue (S −
T) < ennreal (1 / Suc n)
      using sets_lebesgue_inner_closed [OF assms]
      by (metis of_nat_0_less_iff zero_less_Suc zero_less_divide_1_iff)
    then have ∃ T. closed T ∧ T ⊆ S ∧ S − T ∈ lmeasurable ∧ measure lebesgue
(S−T) < 1 / Suc n
      by (metis emeasure_eq_measure2 ennreal_leI not_le)
  }
  then obtain F where F: ⋀n::nat. closed (F n) ∧ F n ⊆ S ∧ S − F n ∈
lmeasurable ∧ measure lebesgue (S − F n) < 1 / Suc n
    by metis
  let ?C = ⋃(F ‘ UNIV)
  show thesis
  proof
    show fsigma ?C
      using F by (simp add: fsigma.intros)
    show (S − ?C) ∈ null_sets lebesgue
    proof (clarsimp simp add: completion.null_sets_outer_le)
      fix e :: real
      assume 0 < e
      then obtain n where n: 1 / Suc n < e
        using nat_approx_posE by metis
      show ∃ T ∈ lmeasurable. S − (⋃x. F x) ⊆ T ∧ measure lebesgue T ≤ e
      proof (intro bexI conjI)
        show measure lebesgue (S − F n) ≤ e
          by (meson F n less_trans not_le order.asym)
      qed (use F in auto)
```

```
    qed
    show ?C ∪ (S − ?C) = S
      using F by blast
    show disjnt ?C (S − ?C)
      by (auto simp: disjnt_def)
  qed
qed

lemma lebesgue_set_almost_gdelta:
  assumes S ∈ sets lebesgue
  obtains C T where gdelta C T ∈ null_sets lebesgue S ∪ T = C disjnt S T
proof −
  have −S ∈ sets lebesgue
    using assms Compl_in_sets_lebesgue by blast
  then obtain C T where C: fsigma C T ∈ null_sets lebesgue C ∪ T = −S
disjnt C T
    using lebesgue_set_almost_fsigma by metis
  show thesis
  proof
    show gdelta (−C)
      by (simp add: ‹fsigma C› fsigma_imp_gdelta)
    show S ∪ T = −C disjnt S T
      using C by (auto simp: disjnt_def)
  qed (use C in auto)
qed

end
```

6.13 Tagged Divisions for Henstock-Kurzweil Integration

```
theory Tagged_Division
  imports Topology_Euclidean_Space
begin

lemma sum_Sigma_product:
  assumes finite S
    and ⋀i. i ∈ S ⟹ finite (T i)
  shows (∑ i∈S. sum (x i) (T i)) = (∑ (i, j)∈Sigma S T. x i j)
  using assms
  by induction (auto simp: sum.Sigma)

lemmas scaleR_simps = scaleR_zero_left scaleR_minus_left scaleR_left_diff_distrib
  scaleR_zero_right scaleR_minus_right scaleR_right_diff_distrib scaleR_eq_0_iff
  scaleR_cancel_left scaleR_cancel_right scaleR_add_right scaleR_add_left real_vector_class.scaleR_
```

6.13.1 Some useful lemmas about intervals

lemma *interior_subset_union_intervals*:
fixes *a b c d*
defines $i \equiv cbox\ a\ b$
defines $j \equiv cbox\ c\ d$
assumes $interior\ j \neq \{\}$
and $i \subseteq j \cup S$
and $interior\ i \cap interior\ j = \{\}$
shows $interior\ i \subseteq interior\ S$
by (*smt* (*verit*, *del_insts*) *IntI Int_interval_mixed_eq_empty UnE assms empty_iff interior_cbox interior_maximal interior_subset open_interior subset_eq*)

lemma *interior_Union_subset_cbox*:
assumes *finite* $\mathcal{F}$
assumes $\mathcal{F}$: $\bigwedge S.\ S \in \mathcal{F} \Longrightarrow \exists a\ b.\ S = cbox\ a\ b\ \bigwedge S.\ S \in \mathcal{F} \Longrightarrow interior\ S \subseteq T$
and T: *closed* T
shows $interior\ (\bigcup \mathcal{F}) \subseteq T$
proof −
have *clo*[*simp*]: $S \in \mathcal{F} \Longrightarrow closed\ S$ **for** S
using $\mathcal{F}$ **by** *auto*
define E **where** $E = \{s{\in}\mathcal{F}.\ interior\ s = \{\}\}$
then have *finite* E $E \subseteq \{s{\in}\mathcal{F}.\ interior\ s = \{\}\}$
using ‹*finite* $\mathcal{F}$› **by** *auto*
then have $interior\ (\bigcup \mathcal{F}) = interior\ (\bigcup (\mathcal{F} - E))$
proof (*induction E rule*: *finite_subset_induct'*)
case (*insert s f'*)
have $interior\ (\bigcup (\mathcal{F} - insert\ s\ f') \cup s) = interior\ (\bigcup (\mathcal{F} - insert\ s\ f'))$
using *insert.hyps* ‹*finite* $\mathcal{F}$› **by** (*intro interior_closed_Un_empty_interior*) *auto*
also have $\bigcup (\mathcal{F} - insert\ s\ f') \cup s = \bigcup (\mathcal{F} - f')$
using *insert.hyps* **by** *auto*
finally show *?case*
by (*simp add*: *insert.IH*)
qed *simp*
also have $\ldots \subseteq \bigcup (\mathcal{F} - E)$
by (*rule interior_subset*)
also have $\ldots \subseteq T$
proof (*rule Union_least*)
fix s **assume** $s \in \mathcal{F} - E$
with $\mathcal{F}$[*of s*] **obtain** $a\ b$ **where** s: $s \in \mathcal{F}\ s = cbox\ a\ b\ box\ a\ b \neq \{\}$
by (*fastforce simp*: *E_def*)
have $closure\ (interior\ s) \subseteq closure\ T$
by (*intro closure_mono* $\mathcal{F}$ ‹$s \in \mathcal{F}$›)
with s ‹*closed* T› **show** $s \subseteq T$
by *simp*
qed
finally show *?thesis* **.**
qed

lemma *Int_interior_Union_intervals*:
  ⟦*finite* $\mathcal{F}$; *open* S; $\bigwedge T.\ T{\in}\mathcal{F} \Longrightarrow \exists\, a\ b.\ T = cbox\ a\ b$; $\bigwedge T.\ T{\in}\mathcal{F} \Longrightarrow S \cap (interior\ T) = \{\}$⟧
  $\Longrightarrow S \cap interior\ (\bigcup \mathcal{F}) = \{\}$
 using *interior_Union_subset_cbox*[*of* $\mathcal{F}$ *UNIV* − S] **by** *auto*

lemma *interval_split*:
 fixes $a :: 'a{::}euclidean\_space$
 assumes $k \in Basis$
 shows
  $cbox\ a\ b \cap \{x.\ x{\cdot}k \le c\} = cbox\ a\ (\sum i{\in}Basis.\ (if\ i = k\ then\ min\ (b{\cdot}k)\ c\ else\ b{\cdot}i) *_R i)$
  $cbox\ a\ b \cap \{x.\ x{\cdot}k \ge c\} = cbox\ (\sum i{\in}Basis.\ (if\ i = k\ then\ max\ (a{\cdot}k)\ c\ else\ a{\cdot}i) *_R i)\ b$
 using *assms* **by** (*rule_tac set_eqI*; *auto simp*: *mem_box*)+

lemma *interval_not_empty*: $\forall\, i{\in}Basis.\ a{\cdot}i \le b{\cdot}i \Longrightarrow cbox\ a\ b \neq \{\}$
 by (*simp add*: *box_ne_empty*)

6.13.2 Bounds on intervals where they exist

definition *interval_upperbound* :: $('a{::}euclidean\_space)\ set \Rightarrow 'a$
 where $interval\_upperbound\ s = (\sum i{\in}Basis.\ (SUP\ x{\in}s.\ x{\cdot}i) *_R i)$

definition *interval_lowerbound* :: $('a{::}euclidean\_space)\ set \Rightarrow 'a$
 where $interval\_lowerbound\ s = (\sum i{\in}Basis.\ (INF\ x{\in}s.\ x{\cdot}i) *_R i)$

lemma *interval_upperbound*[*simp*]:
 $\forall\, i{\in}Basis.\ a{\cdot}i \le b{\cdot}i \Longrightarrow$
  $interval\_upperbound\ (cbox\ a\ b) = (b{::}'a{::}euclidean\_space)$
 unfolding *interval_upperbound_def euclidean_representation_sum cbox_def*
 by (*safe intro*!: *cSup_eq*) *auto*

lemma *interval_lowerbound*[*simp*]:
 $\forall\, i{\in}Basis.\ a{\cdot}i \le b{\cdot}i \Longrightarrow$
  $interval\_lowerbound\ (cbox\ a\ b) = (a{::}'a{::}euclidean\_space)$
 unfolding *interval_lowerbound_def euclidean_representation_sum cbox_def*
 by (*safe intro*!: *cInf_eq*) *auto*

lemmas *interval_bounds* = *interval_upperbound interval_lowerbound*

lemma
 fixes $X{::}real\ set$
 shows *interval_upperbound_real*[*simp*]: $interval\_upperbound\ X = Sup\ X$
  and *interval_lowerbound_real*[*simp*]: $interval\_lowerbound\ X = Inf\ X$
 by (*auto simp*: *interval_upperbound_def interval_lowerbound_def*)

lemma *interval_bounds'*[*simp*]:
 assumes $cbox\ a\ b \neq \{\}$

shows *interval_upperbound* (*cbox a b*) = *b*
and *interval_lowerbound* (*cbox a b*) = *a*
using *assms* **unfolding** *box_ne_empty* **by** *auto*

lemma *interval_upperbound_Times*:
assumes $A \neq \{\}$ **and** $B \neq \{\}$
shows *interval_upperbound* $(A \times B)$ = (*interval_upperbound A*, *interval_upperbound B*)
proof−
from *assms* **have** *fst_image_times'*: $A = fst\ `(A \times B)$ **by** *simp*
have $(\sum i{\in}Basis.\ (SUP\ x{\in}A \times B.\ x \cdot (i,\ 0)) *_R i) = (\sum i{\in}Basis.\ (SUP\ x{\in}A.\ x \cdot i) *_R i)$
by (*subst* (2) *fst_image_times'*) (*simp del*: *fst_image_times add*: *image_comp inner_Pair_0*)
moreover from *assms* **have** *snd_image_times'*: $B = snd\ `(A \times B)$ **by** *simp*
have $(\sum i{\in}Basis.\ (SUP\ x{\in}A \times B.\ x \cdot (0,\ i)) *_R i) = (\sum i{\in}Basis.\ (SUP\ x{\in}B.\ x \cdot i) *_R i)$
by (*subst* (2) *snd_image_times'*) (*simp del*: *snd_image_times add*: *image_comp inner_Pair_0*)
ultimately show *?thesis* **unfolding** *interval_upperbound_def*
by (*subst sum_Basis_prod_eq*) (*auto simp add*: *sum_prod*)
qed

lemma *interval_lowerbound_Times*:
assumes $A \neq \{\}$ **and** $B \neq \{\}$
shows *interval_lowerbound* $(A \times B)$ = (*interval_lowerbound A*, *interval_lowerbound B*)
proof−
from *assms* **have** *fst_image_times'*: $A = fst\ `(A \times B)$ **by** *simp*
have $(\sum i{\in}Basis.\ (INF\ x{\in}A \times B.\ x \cdot (i,\ 0)) *_R i) = (\sum i{\in}Basis.\ (INF\ x{\in}A.\ x \cdot i) *_R i)$
by (*subst* (2) *fst_image_times'*) (*simp del*: *fst_image_times add*: *image_comp inner_Pair_0*)
moreover from *assms* **have** *snd_image_times'*: $B = snd\ `(A \times B)$ **by** *simp*
have $(\sum i{\in}Basis.\ (INF\ x{\in}A \times B.\ x \cdot (0,\ i)) *_R i) = (\sum i{\in}Basis.\ (INF\ x{\in}B.\ x \cdot i) *_R i)$
by (*subst* (2) *snd_image_times'*) (*simp del*: *snd_image_times add*: *image_comp inner_Pair_0*)
ultimately show *?thesis* **unfolding** *interval_lowerbound_def*
by (*subst sum_Basis_prod_eq*) (*auto simp add*: *sum_prod*)
qed

6.13.3 The notion of a gauge — simply an open set containing the point

definition *gauge* $\gamma \longleftrightarrow (\forall x.\ x \in \gamma\ x \wedge open\ (\gamma\ x))$

lemma *gaugeI*:
assumes $\bigwedge x.\ x \in \gamma\ x$ **and** $\bigwedge x.\ open\ (\gamma\ x)$

shows *gauge* γ
using *assms* **unfolding** *gauge_def* **by** *auto*

lemma *gaugeD*[*dest*]:
assumes *gauge* γ **shows** $x \in \gamma\ x$
and *open* $(\gamma\ x)$
using *assms* **unfolding** *gauge_def* **by** *auto*

lemma *gauge_ball_dependent*: $\forall x.\ 0 < e\ x \Longrightarrow$ *gauge* $(\lambda x.\ ball\ x\ (e\ x))$
unfolding *gauge_def* **by** *auto*

lemma *gauge_ball*[*intro*]: $0 < e \Longrightarrow$ *gauge* $(\lambda x.\ ball\ x\ e)$
unfolding *gauge_def* **by** *auto*

lemma *gauge_trivial*[*intro*!]: *gauge* $(\lambda x.\ ball\ x\ 1)$
by *auto*

lemma *gauge_Int*[*intro*]: *gauge* $\gamma 1 \Longrightarrow$ *gauge* $\gamma 2 \Longrightarrow$ *gauge* $(\lambda x.\ \gamma 1\ x \cap \gamma 2\ x)$
unfolding *gauge_def* **by** *auto*

lemma *gauge_reflect*:
fixes $\gamma :: 'a::euclidean\_space \Rightarrow 'a\ set$
shows *gauge* $\gamma \Longrightarrow$ *gauge* $(\lambda x.\ uminus\ `\ \gamma\ (-\ x))$
by (*metis* (*mono_tags*, *lifting*) *gauge_def imageI open_negations minus_minus*)

lemma *gauge_Inter*:
assumes *finite* S
and $\bigwedge u.\ u \in S \Longrightarrow$ *gauge* $(f\ u)$
shows *gauge* $(\lambda x.\ \bigcap \{f\ u\ x \mid u.\ u \in S\})$
proof −
have $*$: $\bigwedge x.\ \{f\ u\ x \mid u.\ u \in S\} = (\lambda u.\ f\ u\ x)\ `\ S$
by *auto*
show *?thesis*
unfolding *gauge_def* **unfolding** $*$
by (*simp add*: *assms gaugeD open_INT*)
qed

lemma *gauge_existence_lemma*:
$(\forall x.\ \exists d :: real.\ p\ x \longrightarrow 0 < d \wedge q\ d\ x) \longleftrightarrow (\forall x.\ \exists d{>}0.\ p\ x \longrightarrow q\ d\ x)$
by (*metis zero_less_one*)

6.13.4 Attempt a systematic general set of "offset" results for components

lemma *gauge_modify*:
assumes $(\forall S.\ open\ S \longrightarrow open\ \{x.\ f(x) \in S\})$ *gauge* d
shows *gauge* $(\lambda x.\ \{y.\ f\ y \in d\ (f\ x)\})$
using *assms* **unfolding** *gauge_def* **by** *force*

6.13.5 Divisions

definition *division_of* (**infixl** *division'_of 40*)
where
s division_of i ⟷
finite s ∧
(∀ *K*∈*s*. *K* ⊆ *i* ∧ *K* ≠ {} ∧ (∃ *a b*. *K* = *cbox a b*)) ∧
(∀ *K1*∈*s*. ∀ *K2*∈*s*. *K1* ≠ *K2* ⟶ *interior*(*K1*) ∩ *interior*(*K2*) = {}) ∧
(⋃ *s* = *i*)

lemma *division_ofD*[*dest*]:
assumes *s division_of i*
shows *finite s*
and ⋀*K*. *K* ∈ *s* ⟹ *K* ⊆ *i*
and ⋀*K*. *K* ∈ *s* ⟹ *K* ≠ {}
and ⋀*K*. *K* ∈ *s* ⟹ ∃ *a b*. *K* = *cbox a b*
and ⋀*K1 K2*. *K1* ∈ *s* ⟹ *K2* ∈ *s* ⟹ *K1* ≠ *K2* ⟹ *interior*(*K1*) ∩ *interior*(*K2*) = {}
and ⋃ *s* = *i*
using *assms* **unfolding** *division_of_def* **by** *auto*

lemma *division_ofI*:
assumes *finite s*
and ⋀*K*. *K* ∈ *s* ⟹ *K* ⊆ *i*
and ⋀*K*. *K* ∈ *s* ⟹ *K* ≠ {}
and ⋀*K*. *K* ∈ *s* ⟹ ∃ *a b*. *K* = *cbox a b*
and ⋀*K1 K2*. *K1* ∈ *s* ⟹ *K2* ∈ *s* ⟹ *K1* ≠ *K2* ⟹ *interior K1* ∩ *interior K2* = {}
and ⋃ *s* = *i*
shows *s division_of i*
using *assms* **unfolding** *division_of_def* **by** *auto*

lemma *division_of_finite*: *s division_of i* ⟹ *finite s*
by *auto*

lemma *division_of_self*[*intro*]: *cbox a b* ≠ {} ⟹ {*cbox a b*} *division_of* (*cbox a b*)
unfolding *division_of_def* **by** *auto*

lemma *division_of_trivial*[*simp*]: *s division_of* {} ⟷ *s* = {}
unfolding *division_of_def* **by** *auto*

lemma *division_of_sing*[*simp*]:
s division_of cbox a (*a*::'*a*::*euclidean_space*) ⟷ *s* = {*cbox a a*}
(**is** *?l* = *?r*)
proof
assume *?l*
have *x* = {*a*} **if** *x* ∈ *s* **for** *x*
by (*metis* ‹*s division_of cbox a a*› *cbox_idem division_ofD*(2) *division_ofD*(3) *subset_singletonD that*)

moreover have $s \neq \{\}$
using ‹s *division_of cbox a a*› **by** *auto*
ultimately show *?r*
unfolding *cbox_idem* **by** *auto*
qed (*use cbox_idem* **in** *blast*)

lemma *elementary_empty*: **obtains** p **where** p *division_of* $\{\}$
by *simp*

lemma *elementary_interval*: **obtains** p **where** p *division_of* (*cbox a b*)
by (*metis division_of_trivial division_of_self*)

lemma *division_contains*: s *division_of* $i \Longrightarrow \forall x{\in}i.\ \exists k{\in}s.\ x \in k$
unfolding *division_of_def* **by** *auto*

lemma *forall_in_division*:
d *division_of* $i \Longrightarrow (\forall x{\in}d.\ P\ x) \longleftrightarrow (\forall a\ b.\ \mathit{cbox}\ a\ b \in d \longrightarrow P\ (\mathit{cbox}\ a\ b))$
unfolding *division_of_def* **by** *fastforce*

lemma *cbox_division_memE*:
assumes $\mathcal{D}$: $K \in \mathcal{D}$ $\mathcal{D}$ *division_of* S
obtains $a\ b$ **where** $K = \mathit{cbox}\ a\ b\ K \neq \{\}\ K \subseteq S$
using *assms* **unfolding** *division_of_def* **by** *metis*

lemma *division_of_subset*:
assumes p *division_of* $(\bigcup p)$
and $q \subseteq p$
shows q *division_of* $(\bigcup q)$
proof (*rule division_ofI*)
show *finite q*
using *assms finite_subset* **by** *blast*
next
fix k
assume $k \in q$
show $k \subseteq \bigcup q$
using ‹$k \in q$› **by** *auto*
show $\exists a\ b.\ k = \mathit{cbox}\ a\ b\ k \neq \{\}$
using *assms* ‹$k \in q$› **by** *blast+*
next
fix *k1 k2*
assume *k1* $\in$ *q k2* $\in$ *q k1* $\neq$ *k2*
then show *interior k1* $\cap$ *interior k2* $= \{\}$
using *assms* **by** *blast*
qed *auto*

lemma *division_of_union_self*[*intro*]: p *division_of* $s \Longrightarrow p$ *division_of* $(\bigcup p)$
by *blast*

lemma *division_inter*:

```
  fixes s1 s2 :: 'a::euclidean_space set
  assumes p1 division_of s1
    and p2 division_of s2
  shows {k1 ∩ k2 | k1 k2. k1 ∈ p1 ∧ k2 ∈ p2 ∧ k1 ∩ k2 ≠ {}} division_of (s1
∩ s2)
  (is ?A' division_of _)
proof −
  let ?A = {s. s ∈ (λ(k1,k2). k1 ∩ k2) ‘ (p1 × p2) ∧ s ≠ {}}
  have ∗: ?A' = ?A by auto
  show ?thesis
    unfolding ∗
  proof (rule division_ofI)
    have ?A ⊆ (λ(x, y). x ∩ y) ‘ (p1 × p2)
      by auto
    moreover have finite (p1 × p2)
      using assms unfolding division_of_def by auto
    ultimately show finite ?A by auto
    have ∗: ⋀s. ⋃{x∈s. x ≠ {}} = ⋃s
      by auto
    show UA: ⋃?A = s1 ∩ s2
      unfolding ∗
      using division_ofD(6)[OF assms(1)] and division_ofD(6)[OF assms(2)] by
auto
    {
      fix k
      assume kA: k ∈ ?A
      then obtain k1 k2 where k: k = k1 ∩ k2 k1 ∈ p1 k2 ∈ p2 k ≠ {}
        by auto
      then show k ≠ {}
        by auto
      show k ⊆ s1 ∩ s2
        using UA kA by blast
      show ∃a b. k = cbox a b
        using k by (metis (no_types, lifting) Int_interval assms division_ofD(4))
    }
    fix k1 k2
    assume k1 ∈ ?A
    then obtain x1 y1 where k1: k1 = x1 ∩ y1 x1 ∈ p1 y1 ∈ p2 k1 ≠ {}
      by auto
    assume k2 ∈ ?A
    then obtain x2 y2 where k2: k2 = x2 ∩ y2 x2 ∈ p1 y2 ∈ p2 k2 ≠ {}
      by auto
    assume k1 ≠ k2
    then show interior k1 ∩ interior k2 = {}
      unfolding k1 k2
      using assms division_ofD(5) k1 k2 by auto
  qed
qed
```

lemma *division_inter_1*:
assumes *d division_of i*
and *cbox a* (*b*::*'a*::*euclidean_space*) ⊆ *i*
shows {*cbox a b* ∩ *k* | *k*. *k* ∈ *d* ∧ *cbox a b* ∩ *k* ≠ {}} *division_of* (*cbox a b*)
proof (*cases cbox a b* = {})
case *True*
show *?thesis*
unfolding *True* **and** *division_of_trivial* **by** *auto*
next
case *False*
have ∗: *cbox a b* ∩ *i* = *cbox a b* **using** *assms*(*2*) **by** *auto*
show *?thesis*
using *division_inter*[*OF division_of_self*[*OF False*] *assms*(*1*)]
unfolding ∗ **by** *auto*
qed

lemma *elementary_Int*:
fixes *s t* :: *'a*::*euclidean_space set*
assumes *p1 division_of s* **and** *p2 division_of t*
shows ∃ *p*. *p division_of* (*s* ∩ *t*)
using *assms division_inter* **by** *blast*

lemma *elementary_Inter*:
assumes *finite* $\mathcal{F}$
and $\mathcal{F} \neq \{\}$
and ∀ *s*∈$\mathcal{F}$. ∃ *p*. *p division_of* (*s*::(*'a*::*euclidean_space*) *set*)
shows ∃ *p*. *p division_of* (⋂ $\mathcal{F}$)
using *assms*
proof (*induct* $\mathcal{F}$ *rule*: *finite_induct*)
case (*insert x* $\mathcal{F}$)
then show *?case*
by (*metis cInf_singleton complete_lattice_class.Inf_insert elementary_Int insert_iff*)
qed *auto*

lemma *division_disjoint_union*:
assumes *p1 division_of s1*
and *p2 division_of s2*
and *interior s1* ∩ *interior s2* = {}
shows (*p1* ∪ *p2*) *division_of* (*s1* ∪ *s2*)
proof (*rule division_ofI*)
note *d1* = *division_ofD*[*OF assms*(*1*)]
note *d2* = *division_ofD*[*OF assms*(*2*)]
fix *k1 k2*
assume *k1* ∈ *p1* ∪ *p2 k2* ∈ *p1* ∪ *p2 k1* ≠ *k2*
with *assms* **show** *interior k1* ∩ *interior k2* = {}
by (*smt* (*verit*, *best*) *IntE Un_iff disjoint_iff_not_equal division_ofD*(*2*) *division_ofD*(*5*) *inf*.*orderE interior_Int*)
qed (*use division_ofD assms* **in** *auto*)

```
lemma partial_division_extend_1:
  fixes a b c d :: 'a::euclidean_space
  assumes incl: cbox c d ⊆ cbox a b
    and nonempty: cbox c d ≠ {}
  obtains p where p division_of (cbox a b) cbox c d ∈ p
proof
  let ?B = λf::'a⇒'a × 'a.
    cbox (∑ i∈Basis. (fst (f i) · i) *R i) (∑ i∈Basis. (snd (f i) · i) *R i)
  define p where p = ?B ` (Basis →E {(a, c), (c, d), (d, b)})

  show cbox c d ∈ p
    unfolding p_def
   by (auto simp add: box_eq_empty cbox_def intro!: image_eqI[where x=λ(i::'a)∈Basis.
(c, d)])
  have ord: a · i ≤ c · i c · i ≤ d · i d · i ≤ b · i if i ∈ Basis for i
    using incl nonempty that
    unfolding box_eq_empty subset_box by (auto simp: not_le)

  show p division_of (cbox a b)
  proof (rule division_ofI)
    show finite p
      unfolding p_def by (auto intro!: finite_PiE)
    {
      fix K
      assume K ∈ p
      then obtain f where f: f ∈ Basis →E {(a, c), (c, d), (d, b)} and k: K =
?B f
        by (auto simp: p_def)
      then show ∃ a b. K = cbox a b
        by auto
      {
        fix l
        assume l ∈ p
        then obtain g where g: g ∈ Basis →E {(a, c), (c, d), (d, b)} and l: l =
?B g
          by (auto simp: p_def)
        assume l ≠ K
        have ∃ i∈Basis. f i ≠ g i
          using ‹l ≠ K› l k f g by auto
        then obtain i where *: i ∈ Basis f i ≠ g i ..
        then have f i = (a, c) ∨ f i = (c, d) ∨ f i = (d, b)
          g i = (a, c) ∨ g i = (c, d) ∨ g i = (d, b)
          using f g by (auto simp: PiE_iff)
        with * ord[of i] show interior l ∩ interior K = {}
          by (auto simp add: l k disjoint_interval intro!: bexI[of _ i])
      }
      have a · i ≤ fst (f i) · i snd (f i) · i ≤ b · i fst (f i) · i ≤ snd (f i) · i
        if i ∈ Basis for i
```

```
      proof −
        have f i = (a, c) ∨ f i = (c, d) ∨ f i = (d, b)
          using that f by (auto simp: PiE_iff)
        with that ord[of i]
        show a · i ≤ fst (f i) · i snd (f i) · i ≤ b · i fst (f i) · i ≤ snd (f i) · i
          by auto
      qed
      then show K ≠ {} K ⊆ cbox a b
        by (auto simp add: k box_eq_empty subset_box not_less)
    }
    moreover
    have ∃k∈p. x ∈ k if x: x ∈ cbox a b for x
    proof −
      have ∃l. x · i ∈ {fst l · i .. snd l · i} ∧ l ∈ {(a, c), (c, d), (d, b)} if i ∈
Basis for i
      proof −
        have (a · i ≤ x · i ∧ x · i ≤ c · i) ∨ (c · i ≤ x · i ∧ x · i ≤ d · i) ∨
            (d · i ≤ x · i ∧ x · i ≤ b · i)
          using that x ord[of i]
          by (auto simp: cbox_def)
        then show ∃l. x · i ∈ {fst l · i .. snd l · i} ∧ l ∈ {(a, c), (c, d), (d, b)}
          by auto
      qed
      then obtain f where
       f: ∀i∈Basis. x · i ∈ {fst (f i) · i..snd (f i) · i} ∧ f i ∈ {(a, c), (c, d), (d, b)}
        by metis
      moreover from f have x ∈ ?B (restrict f Basis) restrict f Basis ∈ Basis →E
{(a,c), (c,d), (d,b)}
        by (auto simp: mem_box)
      ultimately show ?thesis
        unfolding p_def by blast
    qed
    ultimately show ⋃p = cbox a b
      by auto
  qed
qed

proposition partial_division_extend_interval:
  assumes p division_of (⋃p) (⋃p) ⊆ cbox a b
  obtains q where p ⊆ q q division_of cbox a (b::'a::euclidean_space)
proof (cases p = {})
  case True
  then show ?thesis
    using elementary_interval that by auto
next
  case False
  note p = division_ofD[OF assms(1)]
  have ∀k∈p. ∃q. q division_of cbox a b ∧ k ∈ q
  proof
```

fix k
assume *kp*: $k \in p$
obtain c d **where** k: $k = cbox\ c\ d$
using $p(4)[OF\ kp]$ **by** *blast*
have $*$: $cbox\ c\ d \subseteq cbox\ a\ b\ cbox\ c\ d \neq \{\}$
using $p(2,3)[OF\ kp,\ unfolded\ k]$ **using** *assms*(2)
by (*blast intro*: *order.trans*)+
obtain q **where** $q\ division\_of\ cbox\ a\ b\ cbox\ c\ d \in q$
by (*rule partial_division_extend_1*[*OF* $*$])
then show $\exists q.\ q\ division\_of\ cbox\ a\ b \wedge k \in q$
unfolding k **by** *auto*
qed
then obtain q **where** q: $\bigwedge x.\ x \in p \Longrightarrow q\ x\ division\_of\ cbox\ a\ b\ \bigwedge x.\ x \in p \Longrightarrow x \in q\ x$
by *metis*
have $q\ x\ division\_of \bigcup (q\ x)$ **if** x: $x \in p$ **for** x
using $q(1)$ x **by** *blast*
then have *di*: $\bigwedge x.\ x \in p \Longrightarrow \exists d.\ d\ division\_of \bigcup (q\ x - \{x\})$
by (*meson Diff_subset division_of_subset*)
have $\forall s \in (\lambda i.\ \bigcup\ (q\ i - \{i\}))\ `\ p.\ \exists d.\ d\ division\_of\ s$
using *di* **by** *blast*
then obtain d **where** d: $d\ division\_of \bigcap ((\lambda i.\ \bigcup (q\ i - \{i\}))\ `\ p)$
by (*meson False elementary_Inter finite_imageI image_is_empty* $p(1)$)
have $d \cup p\ division\_of\ cbox\ a\ b$
proof −
have *te*: $\bigwedge S\ f\ T.\ S \neq \{\} \Longrightarrow \forall i \in S.\ f\ i \cup i = T \Longrightarrow T = \bigcap (f\ `\ S) \cup \bigcup S$ **by** *auto*
have *cbox_eq*: $cbox\ a\ b = \bigcap ((\lambda i.\ \bigcup (q\ i - \{i\}))\ `\ p) \cup \bigcup p$
proof (*rule te*[*OF False*], *clarify*)
fix i
assume $i \in p$
then show $\bigcup (q\ i - \{i\}) \cup i = cbox\ a\ b$
by (*metis Un_commute complete_lattice_class.Sup_insert division_ofD*(6) *insert_Diff q*)
qed
have [*simp*]: $interior\ (\bigcap i \in p.\ \bigcup (q\ i - \{i\})) \cap interior\ K = \{\}$ **if** K: $K \in p$ **for** K
proof −
note $qk = division\_ofD[OF\ q(1)[OF\ K]]$
have $*$: $\bigwedge U\ T\ S.\ T \cap S = \{\} \Longrightarrow U \subseteq S \Longrightarrow U \cap T = \{\}$
by *auto*
show *?thesis*
proof (*rule* $*$[*OF Int_interior_Union_intervals*])
show $\bigwedge T.\ T \in q\ K - \{K\} \Longrightarrow interior\ K \cap interior\ T = \{\}$
using K $q(2)$ $qk(5)$ **by** *auto*
show $interior\ (\bigcap i \in p.\ \bigcup (q\ i - \{i\})) \subseteq interior\ (\bigcup (q\ K - \{K\}))$
by (*meson INF_lower K interior_mono*)
qed (*use qk* **in** *auto*)
qed

```
    show d ∪ p division_of (cbox a b)
    by (simp add: Int_interior_Union_intervals assms(1) cbox_eq d division_disjoint_union
p(1) p(4))
  qed
  then show ?thesis
    by (meson Un_upper2 that)
qed

lemma elementary_bounded[dest]:
  shows p division_of S ⟹ bounded S
  unfolding division_of_def by (metis bounded_Union bounded_cbox)

lemma elementary_subset_cbox:
  p division_of S ⟹ ∃ a b. S ⊆ cbox a b
  by (meson bounded_subset_cbox_symmetric elementary_bounded)

proposition division_union_intervals_exists:
  assumes cbox a b ≠ {}
  obtains p where (insert (cbox a b) p) division_of (cbox a b ∪ cbox c d)
proof (cases cbox c d = {})
  case True
  with assms that show ?thesis by force
next
  case False
  show ?thesis
  proof (cases cbox a b ∩ cbox c d = {})
    case True
    then show ?thesis
      by (metis that False assms division_disjoint_union division_of_self in-
sert_is_Un interior_Int interior_empty)
  next
    case False
    obtain u v where uv: cbox a b ∩ cbox c d = cbox u v
      unfolding Int_interval by auto
    have uv_sub: cbox u v ⊆ cbox c d using uv by auto
    obtain p where pd: p division_of cbox c d and cbox u v ∈ p
      by (rule partial_division_extend_1[OF uv_sub False[unfolded uv]])
    note p = this division_ofD[OF pd]
    have interior (cbox a b ∩ ⋃(p − {cbox u v})) = interior(cbox u v) ∩ interior
(⋃(p − {cbox u v}))
      by (metis Diff_subset Int_absorb1 Int_assoc Sup_subset_mono interior_Int
p(8) uv)
    also have ... = {}
      using p(6) p(7)[OF p(2)] ‹finite p›
      by (intro Int_interior_Union_intervals) auto
    finally have disj: interior (cbox a b) ∩ interior (⋃(p − {cbox u v})) = {}
      by simp
    have cbe: cbox a b ∪ cbox c d = cbox a b ∪ ⋃(p − {cbox u v})
      using p(8) unfolding uv[symmetric] by auto
```

have *insert* (*cbox a b*) (*p* − {*cbox u v*}) *division_of cbox a b* ∪ ⋃(*p* − {*cbox u v*})
by (*metis Diff_subset assms disj division_disjoint_union division_of_self division_of_subset insert_is_Un p*(*8*) *pd*)
with *that*[*of p* − {*cbox u v*}] **show** *?thesis*
by (*simp add*: *cbe*)
qed
qed

lemma *division_of_Union*:
assumes *finite f*
and ⋀*p*. *p* ∈ *f* ⟹ *p division_of* (⋃*p*)
and ⋀*k1 k2*. *k1* ∈ ⋃*f* ⟹ *k2* ∈ ⋃*f* ⟹ *k1* ≠ *k2* ⟹ *interior k1* ∩ *interior k2* = {}
shows ⋃*f division_of* ⋃(⋃*f*)
using *assms* **by** (*auto intro*!: *division_ofI*)

lemma *elementary_union_interval*:
fixes *a b* :: *'a*::*euclidean_space*
assumes *p division_of* ⋃*p*
obtains *q* **where** *q division_of* (*cbox a b* ∪ ⋃*p*)
proof (*cases p* = {} ∨ *cbox a b* = {})
case *True*
obtain *p* **where** *p division_of* (*cbox a b*)
by (*rule elementary_interval*)
then show *thesis*
using *True assms that* **by** *auto*
next
case *False*
then have *p* ≠ {} *cbox a b* ≠ {}
by *auto*
note *pdiv* = *division_ofD*[*OF assms*]
show *?thesis*
proof (*cases interior* (*cbox a b*) = {})
case *True*
show *?thesis*
by (*metis True assms division_disjoint_union elementary_interval inf_bot_left that*)
next
case *nonempty*: *False*
have ∀ *K*∈*p*. ∃ *q*. (*insert* (*cbox a b*) *q*) *division_of* (*cbox a b* ∪ *K*)
by (*metis* ‹*cbox a b* ≠ {}› *division_union_intervals_exists pdiv*(*4*))
then obtain *q* **where** ∀ *x*∈*p*. *insert* (*cbox a b*) (*q x*) *division_of* (*cbox a b*) ∪ *x*
by *metis*
note *q* = *division_ofD*[*OF this*[*rule_format*]]
let *?D* = ⋃{*insert* (*cbox a b*) (*q K*) | *K*. *K* ∈ *p*}
show *thesis*
proof (*rule that*[*OF division_ofI*])

```
    have ∗: {insert (cbox a b) (q K) |K. K ∈ p} = (λK. insert (cbox a b) (q K))
‘ p
      by auto
    show finite ?D
      using ∗ pdiv(1) q(1) by auto
    have ⋃ ?D = (⋃x∈p. ⋃(insert (cbox a b) (q x)))
      by auto
    also have ... = ⋃{cbox a b ∪ t |t. t ∈ p}
      using q(6) by auto
    also have ... = cbox a b ∪ ⋃p
      using ‹p ≠ {}› by auto
    finally show ⋃ ?D = cbox a b ∪ ⋃p .
    show K ⊆ cbox a b ∪ ⋃p K ≠ {} if K ∈ ?D for K
      using q that by blast+
    show ∃ a b. K = cbox a b if K ∈ ?D for K
      using q(4) that by auto
  next
    fix K′ K
    assume K: K ∈ ?D and K′: K′ ∈ ?D K ≠ K′
    obtain x where x: K ∈ insert (cbox a b) (q x) x∈p
      using K by auto
    obtain x′ where x′: K′∈insert (cbox a b) (q x′) x′∈p
      using K′ by auto
    show interior K ∩ interior K′ = {}
    proof (cases x = x′ ∨ K = cbox a b ∨ K′ = cbox a b)
      case True
      then show ?thesis
        using True K′ q(5) x′ x by auto
    next
      case False
      then have as′: K ≠ cbox a b K′ ≠ cbox a b
        by auto
      obtain c d where K: K = cbox c d
        using q(4) x by blast
      have interior K ∩ interior (cbox a b) = {}
        using as′ K′(2) q(5) x by blast
      then have interior K ⊆ interior x
     by (metis ‹interior (cbox a b) ≠ {}› K q(2) x interior_subset_union_intervals)
      moreover
      obtain c d where c_d: K′ = cbox c d
        using q(4) x′(1) x′(2) by presburger
      have interior K′ ∩ interior (cbox a b) = {}
        using as′(2) q(5) x′ by blast
      then have interior K′ ⊆ interior x′
        by (metis nonempty c_d interior_subset_union_intervals q(2) x′)
      moreover have interior x ∩ interior x′ = {}
        by (meson False assms division_ofD(5) x′(2) x(2))
      ultimately show ?thesis
        using ‹interior K ⊆ interior x› ‹interior K′ ⊆ interior x′› by auto
```

```
    qed
   qed
  qed
qed



lemma elementary_unions_intervals:
  assumes fin: finite ℱ
    and ⋀s. s ∈ ℱ ⟹ ∃a b. s = cbox a (b::'a::euclidean_space)
  obtains p where p division_of (⋃ℱ)
proof −
  have ∃p. p division_of (⋃ℱ)
  proof (induct_tac ℱ rule:finite_subset_induct)
    show ∃p. p division_of ⋃{} using elementary_empty by auto
  next
    fix x F
    assume as: finite F x ∉ F ∃p. p division_of ⋃F x∈ℱ
    then obtain a b where x: x = cbox a b
      by (meson assms(2))
    then show ∃p. p division_of ⋃(insert x F)
    using elementary_union_interval by (metis Union_insert as(3) division_ofD(6))
  qed (use assms in auto)
  then show ?thesis
    using that by auto
qed

lemma elementary_union:
  assumes ps division_of S pt division_of T
  obtains p where p division_of (S ∪ T)
proof −
  have S ∪ T = ⋃ps ∪ ⋃pt
    using assms unfolding division_of_def by auto
  with elementary_unions_intervals[of ps ∪ pt] assms
  show ?thesis
    by (metis Un_iff Union_Un_distrib division_of_def finite_Un that)
qed

lemma partial_division_extend:
  fixes T :: 'a::euclidean_space set
  assumes p division_of S
    and q division_of T
    and S ⊆ T
  obtains r where p ⊆ r and r division_of T
proof −
  note divp = division_ofD[OF assms(1)] and divq = division_ofD[OF assms(2)]
  obtain a b where ab: T ⊆ cbox a b
    using elementary_subset_cbox[OF assms(2)] by auto
  obtain r1 where p ⊆ r1 r1 division_of (cbox a b)
```

```
    using assms
    by (metis ab dual_order.trans partial_division_extend_interval divp(6))
  note r1 = this division_ofD[OF this(2)]
  obtain p' where p' division_of ⋃(r1 - p)
    by (metis Diff_subset division_of_subset r1(2) r1(8))
  then obtain r2 where r2: r2 division_of (⋃(r1 - p)) ∩ (⋃ q)
    by (metis assms(2) divq(6) elementary_Int)
  {
    fix x
    assume x: x ∈ T x ∉ S
    then obtain R where r: R ∈ r1 x ∈ R
      unfolding r1 using ab
      by (meson division_contains r1(2) subsetCE)
    moreover have R ∉ p
      using divp(6) r(2) x(2) by blast
    ultimately have x∈⋃(r1 - p) by auto
  }
  then have Teq: T = ⋃ p ∪ (⋃(r1 - p) ∩ ⋃ q)
    unfolding divp divq using assms(3) by auto
  have interior S ∩ interior (⋃(r1-p)) = {}
  proof (rule Int_interior_Union_intervals)
    have *: ⋀S. (⋀x. x ∈ S ⟹ False) ⟹ S = {}
      by auto
    show interior S ∩ interior m = {} if m ∈ r1 - p for m
    proof -
      have interior m ∩ interior (⋃ p) = {}
      proof (rule Int_interior_Union_intervals)
        show ⋀T. T∈p ⟹ interior m ∩ interior T = {}
          by (metis DiffD1 DiffD2 that r1(1) r1(7) rev_subsetD)
      qed (use divp in auto)
      then show interior S ∩ interior m = {}
        unfolding divp by auto
    qed
  qed (use r1 in auto)
  then have interior S ∩ interior (⋃(r1-p) ∩ (⋃ q)) = {}
    using interior_subset by auto
  then have div: p ∪ r2 division_of ⋃ p ∪ ⋃(r1 - p) ∩ ⋃ q
    by (simp add: assms(1) division_disjoint_union divp(6) r2)
  show ?thesis
    by (metis Teq div sup_ge1 that)
qed


lemma division_split:
  assumes p division_of (cbox a b)
    and k: k∈Basis
  shows {l ∩ {x. x•k ≤ c} | l. l ∈ p ∧ l ∩ {x. x•k ≤ c} ≠ {}} division_of(cbox a
b ∩ {x. x•k ≤ c})
      (is ?p1 division_of ?I1)
```

and $\{l \cap \{x.\ x{\cdot}k \geq c\} \mid l.\ l \in p \wedge l \cap \{x.\ x{\cdot}k \geq c\} \neq \{\}\}$ *division_of* $(cbox\ a\ b \cap \{x.\ x{\cdot}k \geq c\})$
(**is** *?p2 division_of ?I2*)
proof (*rule_tac*[!] *division_ofI*)
note p = *division_ofD*[*OF assms*(*1*)]
show *finite ?p1 finite ?p2*
using $p(1)$ **by** *auto*
show $\bigcup$ *?p1* = *?I1* $\bigcup$ *?p2* = *?I2*
unfolding $p(6)$[*symmetric*] **by** *auto*
{
fix K
assume $K \in$ *?p1*
then obtain l **where** l: $K = l \cap \{x.\ x \cdot k \leq c\}$ $l \in p$ $l \cap \{x.\ x \cdot k \leq c\} \neq \{\}$
by *blast*
obtain $u\ v$ **where** uv: $l = cbox\ u\ v$
using *assms*(*1*) $l(2)$ **by** *blast*
show $K \subseteq$ *?I1*
using l $p(2)$ uv **by** *force*
show $K \neq \{\}$
by (*simp add*: l)
have $\exists a\ b.\ cbox\ u\ v \cap \{x.\ x \cdot k \leq c\} = cbox\ a\ b$
unfolding *interval_split*[*OF k*] **by** (*auto intro*: *order.trans*)
then show $\exists a\ b.\ K = cbox\ a\ b$
by (*simp add*: $l(1)$ uv)
fix K'
assume $K' \in$ *?p1*
then obtain l' **where** l': $K' = l' \cap \{x.\ x \cdot k \leq c\}$ $l' \in p$ $l' \cap \{x.\ x \cdot k \leq c\} \neq \{\}$
by *blast*
assume $K \neq K'$
then show *interior* $K \cap$ *interior* $K' = \{\}$
unfolding $l\ l'$ **using** $p(5)$[*OF* $l(2)$ $l'(2)$] **by** *auto*
}
{
fix K
assume $K \in$ *?p2*
then obtain l **where** l: $K = l \cap \{x.\ c \leq x \cdot k\}$ $l \in p$ $l \cap \{x.\ c \leq x \cdot k\} \neq \{\}$
by *blast*
obtain $u\ v$ **where** uv: $l = cbox\ u\ v$
using $l(2)$ $p(4)$ **by** *blast*
show $K \subseteq$ *?I2*
using l $p(2)$ uv **by** *force*
show $K \neq \{\}$
by (*simp add*: l)
have $\exists a\ b.\ cbox\ u\ v \cap \{x.\ c \leq x \cdot k\} = cbox\ a\ b$
unfolding *interval_split*[*OF k*] **by** (*auto intro*: *order.trans*)
then show $\exists a\ b.\ K = cbox\ a\ b$
by (*simp add*: $l(1)$ uv)
fix K'

assume $K' \in ?p2$
then obtain l' **where** l': $K' = l' \cap \{x.\ c \leq x \cdot k\}$ $l' \in p$ $l' \cap \{x.\ c \leq x \cdot k\} \neq \{\}$
by *blast*
assume $K \neq K'$
then show *interior* $K \cap$ *interior* $K' = \{\}$
unfolding l l' **using** $p(5)[OF\ l(2)\ l'(2)]$ **by** *auto*
}
qed

6.13.6 Tagged (partial) divisions

definition *tagged_partial_division_of* (**infixr** *tagged'_partial'_division'_of 40*)
where *s tagged_partial_division_of i* $\longleftrightarrow$
finite s $\wedge$
$(\forall x\ K.\ (x,\ K) \in s \longrightarrow x \in K \wedge K \subseteq i \wedge (\exists a\ b.\ K = cbox\ a\ b)) \wedge$
$(\forall x1\ K1\ x2\ K2.\ (x1,\ K1) \in s \wedge (x2,\ K2) \in s \wedge (x1,\ K1) \neq (x2,\ K2) \longrightarrow$
interior K1 $\cap$ *interior K2* $= \{\}$)

lemma *tagged_partial_division_ofD*:
assumes *s tagged_partial_division_of i*
shows *finite s*
and $\bigwedge x\ K.\ (x,K) \in s \Longrightarrow x \in K$
and $\bigwedge x\ K.\ (x,K) \in s \Longrightarrow K \subseteq i$
and $\bigwedge x\ K.\ (x,K) \in s \Longrightarrow \exists a\ b.\ K = cbox\ a\ b$
and $\bigwedge x1\ K1\ x2\ K2.\ (x1,K1) \in s \Longrightarrow$
$(x2,\ K2) \in s \Longrightarrow (x1,\ K1) \neq (x2,\ K2) \Longrightarrow$ *interior K1* $\cap$ *interior K2* $= \{\}$
using *assms* **unfolding** *tagged_partial_division_of_def* **by** *blast+*

definition *tagged_division_of* (**infixr** *tagged'_division'_of 40*)
where *s tagged_division_of i* $\longleftrightarrow$ *s tagged_partial_division_of i* $\wedge$ $(\bigcup \{K.\ \exists x.\ (x,K) \in s\} = i)$

lemma *tagged_division_of_finite*: *s tagged_division_of i* $\Longrightarrow$ *finite s*
unfolding *tagged_division_of_def tagged_partial_division_of_def* **by** *auto*

lemma *tagged_division_of*:
s tagged_division_of i $\longleftrightarrow$
finite s $\wedge$
$(\forall x\ K.\ (x,\ K) \in s \longrightarrow x \in K \wedge K \subseteq i \wedge (\exists a\ b.\ K = cbox\ a\ b)) \wedge$
$(\forall x1\ K1\ x2\ K2.\ (x1,\ K1) \in s \wedge (x2,\ K2) \in s \wedge (x1,\ K1) \neq (x2,\ K2) \longrightarrow$
interior K1 $\cap$ *interior K2* $= \{\}) \wedge$
$(\bigcup \{K.\ \exists x.\ (x,K) \in s\} = i)$
unfolding *tagged_division_of_def tagged_partial_division_of_def* **by** *auto*

lemma *tagged_division_ofI*:
assumes *finite s*
and $\bigwedge x\ K.\ (x,K) \in s \Longrightarrow x \in K$
and $\bigwedge x\ K.\ (x,K) \in s \Longrightarrow K \subseteq i$

```
    and ⋀x K. (x,K) ∈ s ⟹ ∃ a b. K = cbox a b
    and ⋀x1 K1 x2 K2. (x1,K1) ∈ s ⟹ (x2, K2) ∈ s ⟹ (x1, K1) ≠ (x2, K2)
⟹
      interior K1 ∩ interior K2 = {}
    and (⋃{K. ∃ x. (x,K) ∈ s} = i)
  shows s tagged_division_of i
  unfolding tagged_division_of
  using assms by fastforce

lemma tagged_division_ofD[dest]:
  assumes s tagged_division_of i
  shows finite s
    and ⋀x K. (x,K) ∈ s ⟹ x ∈ K
    and ⋀x K. (x,K) ∈ s ⟹ K ⊆ i
    and ⋀x K. (x,K) ∈ s ⟹ ∃ a b. K = cbox a b
    and ⋀x1 K1 x2 K2. (x1, K1) ∈ s ⟹ (x2, K2) ∈ s ⟹ (x1, K1) ≠ (x2, K2)
⟹
      interior K1 ∩ interior K2 = {}
    and (⋃{K. ∃ x. (x,K) ∈ s} = i)
  using assms unfolding tagged_division_of by blast+

lemma division_of_tagged_division:
  assumes s tagged_division_of i
  shows (snd ` s) division_of i
proof (rule division_ofI)
  note assm = tagged_division_ofD[OF assms]
  show ⋃(snd ` s) = i finite (snd ` s)
    using assm by auto
  fix K
  assume k: K ∈ snd ` s
  then show K ⊆ i K ≠ {} ∃ a b. K = cbox a b
    using assm by fastforce+
  fix K'
  assume k': K' ∈ snd ` s K ≠ K'
  then show interior K ∩ interior K' = {}
   by (metis (no_types, lifting) assms imageE k prod.collapse tagged_division_ofD(5))
qed

lemma partial_division_of_tagged_division:
  assumes s tagged_partial_division_of i
  shows (snd ` s) division_of ⋃(snd ` s)
proof (rule division_ofI)
  note assm = tagged_partial_division_ofD[OF assms]
  show finite (snd ` s) ⋃(snd ` s) = ⋃(snd ` s)
    using assm by auto
  fix K
  assume K: K ∈ snd ` s
  then show K ≠ {} ∃ a b. K = cbox a b K ⊆ ⋃(snd ` s)
    using assm by fastforce+
```

fix K'
assume $K' \in snd\ `\ s\ K \neq K'$
then show *interior* $K \cap$ *interior* $K' = \{\}$
using *assm*(5) K **by** *force*
qed

lemma *tagged_partial_division_subset*:
assumes s *tagged_partial_division_of* i **and** $t \subseteq s$
shows t *tagged_partial_division_of* i
using *assms finite_subset*[*OF assms*(2)]
unfolding *tagged_partial_division_of_def*
by *blast*

lemma *tag_in_interval*: p *tagged_division_of* $i \Longrightarrow (x,\ k) \in p \Longrightarrow x \in i$
by *auto*

lemma *tagged_division_of_empty*: $\{\}$ *tagged_division_of* $\{\}$
unfolding *tagged_division_of* **by** *auto*

lemma *tagged_partial_division_of_trivial*[*simp*]: p *tagged_partial_division_of* $\{\}$
$\longleftrightarrow p = \{\}$
unfolding *tagged_partial_division_of_def* **by** *auto*

lemma *tagged_division_of_trivial*[*simp*]: p *tagged_division_of* $\{\} \longleftrightarrow p = \{\}$
unfolding *tagged_division_of* **by** *auto*

lemma *tagged_division_of_self*: $x \in cbox\ a\ b \Longrightarrow \{(x,cbox\ a\ b)\}$ *tagged_division_of*
$(cbox\ a\ b)$
by (*rule tagged_division_ofI*) *auto*

lemma *tagged_division_of_self_real*: $x \in \{a\ ..\ b::real\} \Longrightarrow \{(x,\{a\ ..\ b\})\}$ *tagged_division_of*
$\{a\ ..\ b\}$
by (*metis box_real*(2) *tagged_division_of_self*)

lemma *tagged_division_Un*:
assumes *p1 tagged_division_of s1*
and *p2 tagged_division_of s2*
and *interior s1* $\cap$ *interior s2* $= \{\}$
shows (*p1* $\cup$ *p2*) *tagged_division_of* (*s1* $\cup$ *s2*)
proof (*rule tagged_division_ofI*)
note *p1* = *tagged_division_ofD*[*OF assms*(1)]
note *p2* = *tagged_division_ofD*[*OF assms*(2)]
show *finite* (*p1* $\cup$ *p2*)
using *p1*(1) *p2*(1) **by** *auto*
show $\bigcup\{k.\ \exists x.\ (x,\ k) \in p1 \cup p2\} = s1 \cup s2$
using *p1*(6) *p2*(6) **by** *blast*
fix $x\ K$
assume xK: $(x,\ K) \in p1 \cup p2$
show $x \in K\ \exists a\ b.\ K = cbox\ a\ b\ K \subseteq s1 \cup s2$

```
    using xK p1 p2 by auto
  fix x' K'
  assume xk': (x', K') ∈ p1 ∪ p2 (x, K) ≠ (x', K')
  have *: ⋀a b. a ⊆ s1 ⟹ b ⊆ s2 ⟹ interior a ∩ interior b = {}
    using assms(3) interior_mono by blast
  with assms show interior K ∩ interior K' = {}
    by (metis Un_iff inf_commute p1(3) p2(3) tagged_division_ofD(5) xK xk')
qed

lemma tagged_division_Union:
  assumes finite I
    and tag: ⋀i. i∈I ⟹ pfn i tagged_division_of i
    and disj: ⋀i1 i2. ⟦i1 ∈ I; i2 ∈ I; i1 ≠ i2⟧ ⟹ interior(i1) ∩ interior(i2) =
{}
  shows ⋃(pfn ‘ I) tagged_division_of (⋃I)
proof (rule tagged_division_ofI)
  note assm = tagged_division_ofD[OF tag]
  show finite (⋃(pfn ‘ I))
    using assms by auto
  have ⋃{k. ∃x. (x, k) ∈ ⋃(pfn ‘ I)} = ⋃((λi. ⋃{k. ∃x. (x, k) ∈ pfn i}) ‘ I)
    by blast
  also have ... = ⋃I
    using assm(6) by auto
  finally show ⋃{k. ∃x. (x, k) ∈ ⋃(pfn ‘ I)} = ⋃I .
  fix x k
  assume xk: (x, k) ∈ ⋃(pfn ‘ I)
  then obtain i where i: i ∈ I (x, k) ∈ pfn i
    by auto
  show x ∈ k ∃a b. k = cbox a b k ⊆ ⋃I
    using assm(2−4)[OF i] using i(1) by auto
  fix x' k'
  assume xk': (x', k') ∈ ⋃(pfn ‘ I) (x, k) ≠ (x', k')
  then obtain i' where i': i' ∈ I (x', k') ∈ pfn i'
    by auto
  have *: ⋀a b. i ≠ i' ⟹ a ⊆ i ⟹ b ⊆ i' ⟹ interior a ∩ interior b = {}
    using i(1) i'(1) disj interior_mono by blast
  show interior k ∩ interior k' = {}
  proof (cases i = i')
    case True then show ?thesis
      using assm(5) i' i xk'(2) by blast
  next
    case False then show ?thesis
    using * assm(3) i' i by auto
  qed
qed

lemma tagged_partial_division_of_Union_self:
  assumes p tagged_partial_division_of s
  shows p tagged_division_of (⋃(snd ‘ p))
```

using *tagged_partial_division_ofD*[*OF assms*]
by (*intro tagged_division_ofI*) *auto*

lemma *tagged_division_of_union_self*:
assumes *p tagged_division_of s*
shows *p tagged_division_of* (⋃(*snd* ‘ *p*))
using *tagged_division_ofD*[*OF assms*]
by (*intro tagged_division_ofI*) *auto*

lemma *tagged_division_Un_interval*:
assumes *p1 tagged_division_of* (*cbox a b* ∩ {*x*. *x*•*k* ≤ (*c*::*real*)})
and *p2 tagged_division_of* (*cbox a b* ∩ {*x*. *x*•*k* ≥ *c*})
and *k*: *k* ∈ *Basis*
shows (*p1* ∪ *p2*) *tagged_division_of* (*cbox a b*)
proof −
have ∗: *cbox a b* = (*cbox a b* ∩ {*x*. *x*•*k* ≤ *c*}) ∪ (*cbox a b* ∩ {*x*. *x*•*k* ≥ *c*})
by *auto*
have *interior* (*cbox a b* ∩ {*x*. *x* • *k* ≤ *c*}) ∩ *interior* (*cbox a b* ∩ {*x*. *c* ≤ *x* • *k*})
= {}
using *k* **by** (*force simp*: *interval_split*[*OF k*] *box_def*)
with *assms* **show** *?thesis*
by (*metis* ∗ *tagged_division_Un*)
qed

lemma *tagged_division_Un_interval_real*:
fixes *a* :: *real*
assumes *p1 tagged_division_of* ({*a* .. *b*} ∩ {*x*. *x*•*k* ≤ (*c*::*real*)})
and *p2 tagged_division_of* ({*a* .. *b*} ∩ {*x*. *x*•*k* ≥ *c*})
and *k*: *k* ∈ *Basis*
shows (*p1* ∪ *p2*) *tagged_division_of* {*a* .. *b*}
using *assms* **by** (*metis box_real*(*2*) *tagged_division_Un_interval*)

lemma *tagged_division_split_left_inj*:
assumes *d*: *d tagged_division_of i*
and *tags*: (*x1*, *K1*) ∈ *d* (*x2*, *K2*) ∈ *d*
and *K1* ≠ *K2*
and *eq*: *K1* ∩ {*x*. *x* • *k* ≤ *c*} = *K2* ∩ {*x*. *x* • *k* ≤ *c*}
shows *interior* (*K1* ∩ {*x*. *x*•*k* ≤ *c*}) = {}
by (*smt* (*verit*) *Int_Un_eq*(*1*) *Un_Int_distrib interior_Int prod.inject sup_bot.right_neutral tagged_division_ofD*(*5*) *assms*)

lemma *tagged_division_split_right_inj*:
assumes *d*: *d tagged_division_of i*
and *tags*: (*x1*, *K1*) ∈ *d* (*x2*, *K2*) ∈ *d*
and *K1* ≠ *K2*
and *eq*: *K1* ∩ {*x*. *x*•*k* ≥ *c*} = *K2* ∩ {*x*. *x*•*k* ≥ *c*}
shows *interior* (*K1* ∩ {*x*. *x*•*k* ≥ *c*}) = {}
by (*smt* (*verit*) *Int_Un_eq*(*1*) *Un_Int_distrib interior_Int prod.inject sup_bot.right_neutral tagged_division_ofD*(*5*) *assms*)

lemma (**in** *comm_monoid_set*) *over_tagged_division_lemma*:
assumes *p tagged_division_of i*
and $\bigwedge u\ v.\ box\ u\ v = \{\} \Longrightarrow d\ (cbox\ u\ v) = \mathbf{1}$
shows *F* $(\lambda(\_,\ k).\ d\ k)\ p = F\ d\ (snd\ `\ p)$
proof −
have ∗: $(\lambda(\_\ ,k).\ d\ k) = d \circ snd$
by (*simp add*: *fun_eq_iff*)
note *assm* = *tagged_division_ofD*[*OF assms*(*1*)]
show *?thesis*
unfolding ∗
proof (*rule reindex_nontrivial*[*symmetric*])
show *finite p*
using *assm* **by** *auto*
fix *x y*
assume *x*∈*p y*∈*p x*≠*y snd x* = *snd y*
obtain *a b* **where** *ab*: *snd x* = *cbox a b*
using *assm*(*4*)[*of fst x snd x*] ‹*x*∈*p*› **by** *auto*
have (*fst x*, *snd y*) ∈ *p* (*fst x*, *snd y*) ≠ *y*
using ‹*x* ∈ *p*› ‹*x* ≠ *y*› ‹*snd x* = *snd y*› [*symmetric*] **by** *auto*
with ‹*x*∈*p*› ‹*y*∈*p*› **have** *box a b* = {}
by (*metis* ‹*snd x* = *snd y*› *ab assm*(*5*) *inf.idem interior_cbox prod.collapse*)
then show *d* (*snd x*) = **1**
by (*simp add*: *ab assms*(*2*))
qed
qed

6.13.7 Functions closed on boxes: morphisms from boxes to monoids

This auxiliary structure is used to sum up over the elements of a division. Main theorem is *operative_division*. Instances for the monoid are *'a option*, *real*, and *bool*.

Using additivity of lifted function to encode definedness. **definition** *lift_option* :: ('*a* ⇒ '*b* ⇒ '*c*) ⇒ '*a option* ⇒ '*b option* ⇒ '*c option*
where
lift_option f a' b' = *Option.bind a'* (λ*a*. *Option.bind b'* (λ*b*. *Some* (*f a b*)))

lemma *lift_option_simps*[*simp*]:
lift_option f (*Some a*) (*Some b*) = *Some* (*f a b*)
lift_option f None b' = *None*
lift_option f a' None = *None*
by (*auto simp*: *lift_option_def*)

lemma *comm_monoid_lift_option*:
assumes *comm_monoid f z*
shows *comm_monoid* (*lift_option f*) (*Some z*)

proof −
from *assms* **interpret** *comm_monoid f z* **.**
show *?thesis*
by *standard* (*auto simp*: *lift_option_def ac_simps split*: *bind_split*)
qed

lemma *comm_monoid_set_and*: *comm_monoid_set HOL.conj True*
by (*simp add*: *comm_monoid_set.intro conj.comm_monoid_axioms*)

Misc **lemma** *interval_real_split*:
$\{a$ .. b::*real*$\} \cap \{x.\ x \le c\} = \{a$ .. *min* $b\ c\}$
$\{a$ .. $b\} \cap \{x.\ c \le x\} = \{$*max* $a\ c$ .. $b\}$
by *auto*

lemma *bgauge_existence_lemma*: $(\forall x{\in}s.\ \exists d{::}$*real*$.\ 0 < d \wedge q\ d\ x) \longleftrightarrow (\forall x.\ \exists d{>}0.\ x{\in}s \longrightarrow q\ d\ x)$
by (*meson zero_less_one*)

Division points **definition** *division_points* (k::($'a$::*euclidean_space*) *set*) $d =$
$\{(j,x).\ j \in$ *Basis* $\wedge$ (*interval_lowerbound* k)$\cdot j < x \wedge x <$ (*interval_upperbound* k)$\cdot j \wedge$
$(\exists i{\in}d.$ (*interval_lowerbound* i)$\cdot j = x \vee$ (*interval_upperbound* i)$\cdot j = x)\}$

lemma *division_points_finite*:
fixes i :: $'a$::*euclidean_space set*
assumes d *division_of* i
shows *finite* (*division_points* $i\ d$)
proof −
note *assm* = *division_ofD*[*OF assms*]
let *?M* = $\lambda j.\ \{(j,x)|x.$ (*interval_lowerbound* i)$\cdot j < x \wedge x <$ (*interval_upperbound* i)$\cdot j \wedge$
$(\exists i{\in}d.$ (*interval_lowerbound* i)$\cdot j = x \vee$ (*interval_upperbound* i)$\cdot j = x)\}$
have $*$: *division_points* $i\ d = \bigcup$(*?M* ‘ *Basis*)
unfolding *division_points_def* **by** *auto*
show *?thesis*
unfolding $*$ **using** *assm* **by** *auto*
qed

lemma *division_points_subset*:
fixes a :: $'a$::*euclidean_space*
assumes d *division_of* (*cbox* $a\ b$)
and $\forall i{\in}$*Basis*. $a{\cdot}i < b{\cdot}i\ \ a{\cdot}k < c\ c < b{\cdot}k$
and k: $k \in$ *Basis*
shows *division_points* (*cbox* $a\ b \cap \{x.\ x{\cdot}k \le c\}$) $\{l \cap \{x.\ x{\cdot}k \le c\} \mid l\ .\ l \in d \wedge l \cap \{x.\ x{\cdot}k \le c\} \neq \{\}\} \subseteq$
division_points (*cbox* $a\ b$) d (**is** *?t1*)
and *division_points* (*cbox* $a\ b \cap \{x.\ x{\cdot}k \ge c\}$) $\{l \cap \{x.\ x{\cdot}k \ge c\} \mid l\ .\ l \in d \wedge \neg(l \cap \{x.\ x{\cdot}k \ge c\} = \{\})\} \subseteq$
division_points (*cbox* $a\ b$) d (**is** *?t2*)

proof −
note *assm* = *division_ofD*[*OF assms*(*1*)]
have ∗: ∀ *i*∈*Basis*. *a*•*i* ≤ *b*•*i*
∀ *i*∈*Basis*. *a*•*i* ≤ (∑ *i*∈*Basis*. (*if i* = *k then min* (*b* · *k*) *c else* *b* · *i*) ∗$_R$ *i*) · *i*
∀ *i*∈*Basis*. (∑ *i*∈*Basis*. (*if i* = *k then max* (*a* · *k*) *c else a* · *i*) ∗$_R$ *i*) · *i* ≤ *b*•*i*
min (*b* · *k*) *c* = *c max* (*a* · *k*) *c* = *c*
using *assms* **using** *less_imp_le* **by** *auto*
have ∃ *i*∈*d*. *interval_lowerbound i* · *x* = *y* ∨ *interval_upperbound i* · *x* = *y*
if *a* · *x* < *y y* < (*if x* = *k then c else b* · *x*)
interval_lowerbound i · *x* = *y* ∨ *interval_upperbound i* · *x* = *y*
i = *l* ∩ {*x*. *x* · *k* ≤ *c*} *l* ∈ *d l* ∩ {*x*. *x* · *k* ≤ *c*} ≠ {}
x ∈ *Basis* **for** *i l x y*
proof −
obtain *u v* **where** *l*: *l* = *cbox u v*
using ‹*l* ∈ *d*› *assms*(*1*) **by** *blast*
have ∀ *i*∈*Basis*. *u*•*i* ≤ *v*•*i*
using *l* **using** *that*(*6*) **unfolding** *box_ne_empty*[*symmetric*] **by** *auto*
then show *?thesis*
using *that* ‹*x* ∈ *Basis*› **unfolding** *l interval_split*[*OF k*]
by (*force split*: *if_split_asm*)
qed
moreover have ⋀*x y*. ⟦*y* < (*if x* = *k then c else b* · *x*)⟧ ⟹ *y* < *b* · *x*
using ‹*c* < *b*•*k*› **by** (*auto split*: *if_split_asm*)
ultimately show *?t1*
unfolding *division_points_def interval_split*[*OF k*, *of a b*]
unfolding *interval_bounds*[*OF* ∗(*1*)] *interval_bounds*[*OF* ∗(*2*)] *interval_bounds*[*OF* ∗(*3*)]
unfolding ∗ **by** *force*
have ⋀*x y i l*. (*if x* = *k then c else a* · *x*) < *y* ⟹ *a* · *x* < *y*
using ‹*a*•*k* < *c*› **by** (*auto split*: *if_split_asm*)
moreover have ∃ *i*∈*d*. *interval_lowerbound i* · *x* = *y* ∨
interval_upperbound i · *x* = *y*
if (*if x* = *k then c else a* · *x*) < *y y* < *b* · *x*
interval_lowerbound i · *x* = *y* ∨ *interval_upperbound i* · *x* = *y*
i = *l* ∩ {*x*. *c* ≤ *x* · *k*} *l* ∈ *d l* ∩ {*x*. *c* ≤ *x* · *k*} ≠ {}
x ∈ *Basis* **for** *x y i l*
proof −
obtain *u v* **where** *l*: *l* = *cbox u v*
using ‹*l* ∈ *d*› *assm*(*4*) **by** *blast*
have ∀ *i*∈*Basis*. *u*•*i* ≤ *v*•*i*
using *l* **using** *that*(*6*) **unfolding** *box_ne_empty*[*symmetric*] **by** *auto*
then show ∃ *i*∈*d*. *interval_lowerbound i* · *x* = *y* ∨ *interval_upperbound i* · *x* = *y*
using *that* ‹*x* ∈ *Basis*› **unfolding** *l interval_split*[*OF k*]
by (*force split*: *if_split_asm*)
qed
ultimately show *?t2*
unfolding *division_points_def interval_split*[*OF k*, *of a b*]
unfolding *interval_bounds*[*OF* ∗(*1*)] *interval_bounds*[*OF* ∗(*2*)] *interval_bounds*[*OF*

```
∗(3)]
    unfolding ∗
    by force
qed

lemma division_points_psubset:
  fixes a :: 'a::euclidean_space
  assumes d: d division_of (cbox a b)
      and altb: ∀ i∈Basis. a•i < b•i  a•k < c c < b•k
      and l ∈ d
      and disj: interval_lowerbound l•k = c ∨ interval_upperbound l•k = c
      and k: k ∈ Basis
  shows division_points (cbox a b ∩ {x. x•k ≤ c}) {l ∩ {x. x•k ≤ c} | l. l∈d ∧ l
∩ {x. x•k ≤ c} ≠ {}} ⊂
        division_points (cbox a b) d (is ?D1 ⊂ ?D)
    and division_points (cbox a b ∩ {x. x•k ≥ c}) {l ∩ {x. x•k ≥ c} | l. l∈d ∧ l ∩
{x. x•k ≥ c} ≠ {}} ⊂
        division_points (cbox a b) d (is ?D2 ⊂ ?D)
proof −
  have ab: ∀ i∈Basis. a•i ≤ b•i
    using altb by (auto intro!:less_imp_le)
  obtain u v where l: l = cbox u v
    using d ‹l ∈ d› by blast
  have uv: ∀ i∈Basis. u•i ≤ v•i ∀ i∈Basis. a•i ≤ u•i ∧ v•i ≤ b•i
    apply (metis assms(5) box_ne_empty(1) cbox_division_memE d l)
   by (metis assms(5) box_ne_empty(1) cbox_division_memE d l subset_box(1))
  have ∗: interval_upperbound (cbox a b ∩ {x. x • k ≤ interval_upperbound l • k})
• k = interval_upperbound l • k
        interval_upperbound (cbox a b ∩ {x. x • k ≤ interval_lowerbound l • k}) •
k = interval_lowerbound l • k
    unfolding l interval_split[OF k] interval_bounds[OF uv(1)]
    using uv[rule_format, of k] ab k
    by auto
  have ∃ x. x ∈ ?D − ?D1
    using assms(3−)
    unfolding division_points_def interval_bounds[OF ab]
    by (force simp add: ∗)
  moreover have ?D1 ⊆ ?D
    by (auto simp add: assms division_points_subset)
  ultimately show ?D1 ⊂ ?D
    by blast
  have ∗: interval_lowerbound (cbox a b ∩ {x. x • k ≥ interval_lowerbound l • k})
• k = interval_lowerbound l • k
    interval_lowerbound (cbox a b ∩ {x. x • k ≥ interval_upperbound l • k}) • k =
interval_upperbound l • k
    unfolding l interval_split[OF k] interval_bounds[OF uv(1)]
    using uv[rule_format, of k] ab k
    by auto
  have ∃ x. x ∈ ?D − ?D2
```

using *assms*(3−)
unfolding *division_points_def interval_bounds*[*OF ab*]
by (*force simp add*: ∗)
moreover have *?D2* ⊆ *?D*
by (*auto simp add*: *assms division_points_subset*)
ultimately show *?D2* ⊂ *?D*
by *blast*
qed

lemma *division_split_left_inj*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *div*: $\mathcal{D}$ *division_of S*
and *eq*: *K1* ∩ {*x*::*'a*. *x*•*k* ≤ *c*} = *K2* ∩ {*x*. *x*•*k* ≤ *c*}
and *K1* ∈ $\mathcal{D}$ *K2* ∈ $\mathcal{D}$ *K1* ≠ *K2*
shows *interior* (*K1* ∩ {*x*. *x*•*k* ≤ *c*}) = {}
proof −
have *interior K2* ∩ *interior* {*a*. *a* • *k* ≤ *c*} = *interior K1* ∩ *interior* {*a*. *a* • *k* ≤ *c*}
by (*metis* (*no_types*) *eq interior_Int*)
moreover have ⋀*A*. *interior A* ∩ *interior K2* = {} ∨ *A* = *K2* ∨ *A* ∉ $\mathcal{D}$
by (*meson div* ‹*K2* ∈ $\mathcal{D}$› *division_of_def*)
ultimately show *?thesis*
using ‹*K1* ∈ $\mathcal{D}$› ‹*K1* ≠ *K2*› **by** *auto*
qed

lemma *division_split_right_inj*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *div*: $\mathcal{D}$ *division_of S*
and *eq*: *K1* ∩ {*x*::*'a*. *x*•*k* ≥ *c*} = *K2* ∩ {*x*. *x*•*k* ≥ *c*}
and *K1* ∈ $\mathcal{D}$ *K2* ∈ $\mathcal{D}$ *K1* ≠ *K2*
shows *interior* (*K1* ∩ {*x*. *x*•*k* ≥ *c*}) = {}
proof −
have *interior K2* ∩ *interior* {*a*. *a* • *k* ≥ *c*} = *interior K1* ∩ *interior* {*a*. *a* • *k* ≥ *c*}
by (*metis* (*no_types*) *eq interior_Int*)
moreover have ⋀*A*. *interior A* ∩ *interior K2* = {} ∨ *A* = *K2* ∨ *A* ∉ $\mathcal{D}$
by (*meson div* ‹*K2* ∈ $\mathcal{D}$› *division_of_def*)
ultimately show *?thesis*
using ‹*K1* ∈ $\mathcal{D}$› ‹*K1* ≠ *K2*› **by** *auto*
qed

lemma *interval_doublesplit*:
fixes *a* :: *'a*::*euclidean_space*
assumes *k* ∈ *Basis*
shows *cbox a b* ∩ {*x* . |*x*•*k* − *c*| ≤ (*e*::*real*)} =
cbox (∑ *i*∈*Basis*. (*if i* = *k then max* (*a*•*k*) (*c* − *e*) *else a*•*i*) ∗$_R$ *i*)
(∑ *i*∈*Basis*. (*if i* = *k then min* (*b*•*k*) (*c* + *e*) *else b*•*i*) ∗$_R$ *i*)
proof −
have ∗: ⋀*x c e*::*real*. |*x* − *c*| ≤ *e* ⟷ *x* ≥ *c* − *e* ∧ *x* ≤ *c* + *e*

```
    by auto
  have **: ⋀s P Q. s ∩ {x. P x ∧ Q x} = (s ∩ {x. Q x}) ∩ {x. P x}
    by blast
  show ?thesis
    unfolding * ** interval_split[OF assms] by (rule refl)
qed

lemma division_doublesplit:
  fixes a :: 'a::euclidean_space
  assumes p division_of (cbox a b)
    and k: k ∈ Basis
  shows (λl. l ∩ {x. |x•k − c| ≤ e}) ` {l∈p. l ∩ {x. |x•k − c| ≤ e} ≠ {}}
        division_of (cbox a b ∩ {x. |x•k − c| ≤ e})
proof −
  have **: ⋀p q p' q'. p division_of q ⟹ p = p' ⟹ q = q' ⟹ p' division_of q'
    by auto
  note division_split(1)[OF assms, where c=c+e,unfolded interval_split[OF k]]
  note division_split(2)[OF this, where c=c−e and k=k,OF k]
  then show ?thesis
    apply (rule **)
    subgoal
    apply (simp add: abs_diff_le_iff field_simps Collect_conj_eq setcompr_eq_image
[symmetric] cong: image_cong_simp)
      apply (rule equalityI)
      apply blast
      apply clarsimp
      apply (rename_tac S)
      apply (rule_tac x=S ∩ {x. c + e ≥ x • k} in exI)
      apply auto
      done
    by (simp add: interval_split k interval_doublesplit)
qed

Operative  locale operative = comm_monoid_set +
  fixes g :: 'b::euclidean_space set ⇒ 'a
  assumes box_empty_imp: ⋀a b. box a b = {} ⟹ g (cbox a b) = 1
    and Basis_imp: ⋀a b c k. k ∈ Basis ⟹ g (cbox a b) = g (cbox a b ∩ {x. x•k
≤ c}) * g (cbox a b ∩ {x. x•k ≥ c})
begin

lemma empty [simp]: g {} = 1
  by (metis box_empty_imp box_subset_cbox empty_as_interval subset_empty)

lemma division:
  F g d = g (cbox a b) if d division_of (cbox a b)
proof −
  define C where [abs_def]: C = card (division_points (cbox a b) d)
  then show ?thesis
  using that proof (induction C arbitrary: a b d rule: less_induct)
```

case (*less a b d*)
show *?case*
proof (*cases box a b = {}*)
case *True*
{ fix *k* **assume** *k∈d*
then obtain *a' b'* **where** *k*: *k = cbox a' b'*
using *division_ofD(4)[OF less.prems]* **by** *blast*
then have *cbox a' b' ⊆ cbox a b*
using ‹*k ∈ d*› *less.prems* **by** *blast*
then have *box a' b' ⊆ box a b*
unfolding *subset_box* **by** *auto*
then have *g k = 1*
using *box_empty_imp* [*of a' b'*] *k* **by** (*simp add*: *True*)
}
with *True* **show** *F g d = g* (*cbox a b*)
by (*auto intro*!: *neutral simp*: *box_empty_imp*)
next
case *False*
then have *ab*: ∀ *i∈Basis. a•i < b•i* **and** *ab'*: ∀ *i∈Basis. a•i ≤ b•i*
by (*auto simp*: *box_ne_empty*)
show *F g d = g* (*cbox a b*)
proof (*cases division_points* (*cbox a b*) *d = {}*)
case *True*
{ fix *u v* **and** *j* :: *'b*
assume *j*: *j ∈ Basis* **and** *as*: *cbox u v ∈ d*
then have *cbox u v ≠ {}*
using *less.prems* **by** *blast*
then have *uv*: ∀ *i∈Basis. u•i ≤ v•i u•j ≤ v•j*
using *j* **unfolding** *box_ne_empty* **by** *auto*
have ∗: ⋀*p r Q*. ¬ *j∈Basis* ∨ *p* ∨ *r* ∨ (∀ *x∈d. Q x*) ⟹ *p* ∨ *r* ∨ *Q* (*cbox u v*)
using *as j* **by** *auto*
have (*j*, *u•j*) ∉ *division_points* (*cbox a b*) *d*
(*j*, *v•j*) ∉ *division_points* (*cbox a b*) *d* **using** *True* **by** *auto*
note *this*[*unfolded de_Morgan_conj division_points_def mem_Collect_eq split_conv interval_bounds*[*OF ab'*] *bex_simps*]
note ∗[*OF this*(*1*)] ∗[*OF this*(*2*)] **note** *this*[*unfolded interval_bounds*[*OF uv*(*1*)]]
moreover
have *a•j ≤ u•j v•j ≤ b•j*
by (*meson as division_ofD*(*2*) *j less.prems subset_box*(*1*) *uv*(*1*))+
ultimately have *u•j = a•j ∧ v•j = a•j ∨ u•j = b•j ∧ v•j = b•j ∨ u•j = a•j ∧ v•j = b•j*
using *uv*(*2*) **by** *force*
}
then have *d'*: ∀ *i∈d*. ∃ *u v*. *i = cbox u v ∧*
(∀ *j∈Basis. u•j = a•j ∧ v•j = a•j ∨ u•j = b•j ∧ v•j = b•j ∨ u•j = a•j ∧ v•j = b•j*)
unfolding *forall_in_division*[*OF less.prems*] **by** *blast*

```
      have (1/2) *R (a+b) ∈ cbox a b
        unfolding mem_box using ab by (auto simp: inner_simps)
      note this[unfolded division_ofD(6)[OF ‹d division_of cbox a b›,symmetric]
Union_iff]
      then obtain i where i: i ∈ d (1 / 2) *R (a + b) ∈ i ..
      obtain u v where uv: i = cbox u v
                  ∀j∈Basis. u · j = a · j ∧ v · j = a · j ∨
                            u · j = b · j ∧ v · j = b · j ∨
                            u · j = a · j ∧ v · j = b · j
        using d' i(1) by auto
      have cbox a b ∈ d
      proof −
        have u = a v = b
          unfolding euclidean_eq_iff[where 'a='b]
        proof safe
          fix j :: 'b
          assume j: j ∈ Basis
          note i(2)[unfolded uv mem_box]
          then show u · j = a · j and v · j = b · j
            by (smt (verit) False box_midpoint j mem_box(1) uv(2))+
        qed
        then have i = cbox a b using uv by auto
        then show ?thesis using i by auto
      qed
      then have deq: d = insert (cbox a b) (d − {cbox a b})
        by auto
      have F g (d − {cbox a b}) = 1
      proof (intro neutral ballI)
        fix x
        assume x: x ∈ d − {cbox a b}
        then have x∈d
          by auto note d'[rule_format,OF this]
        then obtain u v where uv: x = cbox u v
                    ∀j∈Basis. u · j = a · j ∧ v · j = a · j ∨
                              u · j = b · j ∧ v · j = b · j ∨
                              u · j = a · j ∧ v · j = b · j
          by blast
        have u ≠ a ∨ v ≠ b
          using x[unfolded uv] by auto
        then obtain j where u·j ≠ a·j ∨ v·j ≠ b·j and j: j ∈ Basis
          unfolding euclidean_eq_iff[where 'a='b] by auto
        then have u·j = v·j
          using uv(2)[rule_format,OF j] by auto
        then show g x = 1
          by (smt (verit) box_empty_imp box_eq_empty(1) j uv(1))
      qed
      then show F g d = g (cbox a b)
    by (metis deq division_of_finite insert_Diff_single insert_remove less.prems
right_neutral)
```

```
    next
      case False
      then have ∃x. x ∈ division_points (cbox a b) d
        by auto
      then obtain k c
        where k ∈ Basis interval_lowerbound (cbox a b) • k < c
              c < interval_upperbound (cbox a b) • k
              ∃i∈d. interval_lowerbound i • k = c ∨ interval_upperbound i • k = c
        unfolding division_points_def by auto
      then obtain j where a • k < c c < b • k
          and j ∈ d and j: interval_lowerbound j • k = c ∨ interval_upperbound
j • k = c
        by (metis division_of_trivial empty_iff interval_bounds' less.prems)
      let ?lec = {x. x•k ≤ c} let ?gec = {x. x•k ≥ c}
      define d1 where d1 = {l ∩ ?lec | l. l ∈ d ∧ l ∩ ?lec ≠ {}}
      define d2 where d2 = {l ∩ ?gec | l. l ∈ d ∧ l ∩ ?gec ≠ {}}
      define cb where cb = (∑i∈Basis. (if i = k then c else b•i) *R i)
      define ca where ca = (∑i∈Basis. (if i = k then c else a•i) *R i)
      have division_points (cbox a b ∩ ?lec) {l ∩ ?lec |l. l ∈ d ∧ l ∩ ?lec ≠ {}}
            ⊂ division_points (cbox a b) d
        by (rule division_points_psubset[OF ‹d division_of cbox a b› ab ‹a • k <
c› ‹c < b • k› ‹j ∈ d› j ‹k ∈ Basis›])
      with division_points_finite[OF ‹d division_of cbox a b›]
      have card
       (division_points (cbox a b ∩ ?lec) {l ∩ ?lec |l. l ∈ d ∧ l ∩ ?lec ≠ {}})
         < card (division_points (cbox a b) d)
        by (rule psubset_card_mono)
      moreover have division_points (cbox a b ∩ {x. c ≤ x • k}) {l ∩ {x. c ≤ x
• k} |l. l ∈ d ∧ l ∩ {x. c ≤ x • k} ≠ {}}
            ⊂ division_points (cbox a b) d
        by (rule division_points_psubset[OF ‹d division_of cbox a b› ab ‹a • k <
c› ‹c < b • k› ‹j ∈ d› j ‹k ∈ Basis›])
      with division_points_finite[OF ‹d division_of cbox a b›]
      have card (division_points (cbox a b ∩ ?gec) {l ∩ ?gec |l. l ∈ d ∧ l ∩ ?gec
≠ {}})
            < card (division_points (cbox a b) d)
        by (rule psubset_card_mono)
      ultimately have *: F g d1 = g (cbox a b ∩ ?lec) F g d2 = g (cbox a b ∩
?gec)
        unfolding interval_split[OF ‹k ∈ Basis›]
        apply (rule_tac[!] less.hyps)
        using division_split[OF ‹d division_of cbox a b›, where k=k and c=c]
‹k ∈ Basis›
        by (simp_all add: interval_split d1_def d2_def division_points_finite[OF
‹d division_of cbox a b›])
      have fxk_le: g (l ∩ ?lec) = 1
        if l ∈ d y ∈ d l ∩ ?lec = y ∩ ?lec l ≠ y for l y
      proof −
        obtain u v where leq: l = cbox u v
```

```
        using ‹l ∈ d› less.prems by auto
      have interior (cbox u v ∩ ?lec) = {}
        using that division_split_left_inj leq less.prems by blast
      then show ?thesis
        unfolding leq interval_split [OF ‹k ∈ Basis›]
        by (auto intro: box_empty_imp)
    qed
    have fxk_ge: g (l ∩ {x. x · k ≥ c}) = 1
      if l ∈ d y ∈ d l ∩ ?gec = y ∩ ?gec l ≠ y for l y
    proof −
      obtain u v where leq: l = cbox u v
        using ‹l ∈ d› less.prems by auto
      have interior (cbox u v ∩ ?gec) = {}
        using that division_split_right_inj leq less.prems by blast
      then show ?thesis
        unfolding leq interval_split[OF ‹k ∈ Basis›]
        by (auto intro: box_empty_imp)
    qed
    have d1_alt: d1 = (λl. l ∩ ?lec) ‘ {l ∈ d. l ∩ ?lec ≠ {}}
      using d1_def by auto
    have d2_alt: d2 = (λl. l ∩ ?gec) ‘ {l ∈ d. l ∩ ?gec ≠ {}}
      using d2_def by auto
    have g (cbox a b) = F g d1 ∗ F g d2 (is _ = ?prev)
      unfolding ∗ using ‹k ∈ Basis›
      by (auto dest: Basis_imp)
    also have F g d1 = F (λl. g (l ∩ ?lec)) d
      unfolding d1_alt using division_of_finite[OF less.prems] fxk_le
      by (subst reindex_nontrivial) (auto intro!: mono_neutral_cong_left)
    also have F g d2 = F (λl. g (l ∩ ?gec)) d
      unfolding d2_alt using division_of_finite[OF less.prems] fxk_ge
      by (subst reindex_nontrivial) (auto intro!: mono_neutral_cong_left)
    also have ∗: ∀ x∈d. g x = g (x ∩ ?lec) ∗ g (x ∩ ?gec)
      unfolding forall_in_division[OF ‹d division_of cbox a b›]
      using ‹k ∈ Basis›
      by (auto dest: Basis_imp)
    have F (λl. g (l ∩ ?lec)) d ∗ F (λl. g (l ∩ ?gec)) d = F g d
      using ∗ by (simp add: distrib)
    finally show ?thesis by auto
  qed
 qed
qed

proposition tagged_division:
 assumes d tagged_division_of (cbox a b)
 shows F (λ(_, l). g l) d = g (cbox a b)
 by (metis assms box_empty_imp division division_of_tagged_division over_tagged_division_lemma)

end
```

```
locale operative_real = comm_monoid_set +
  fixes g :: real set ⇒ 'a
  assumes neutral: b ≤ a ⟹ g {a..b} = 1
  assumes coalesce_less: a < c ⟹ c < b ⟹ g {a..c} * g {c..b} = g {a..b}
begin

sublocale operative where g = g
  rewrites box = (greaterThanLessThan :: real ⇒ _)
    and cbox = (atLeastAtMost :: real ⇒ _)
    and ⋀x::real. x ∈ Basis ⟷ x = 1
proof −
  show operative f z g
  proof
    show g (cbox a b) = 1 if box a b = {} for a b
      using that by (simp add: neutral)
    show g (cbox a b) = g (cbox a b ∩ {x. x • k ≤ c}) * g (cbox a b ∩ {x. c ≤ x •
k})
      if k ∈ Basis for a b c k
    proof −
      from that have [simp]: k = 1
        by simp
      from neutral [of 0 1] neutral [of a a for a] coalesce_less
      have [simp]: g {} = 1 ⋀a. g {a} = 1
        ⋀a b c. a < c ⟹ c < b ⟹ g {a..c} * g {c..b} = g {a..b}
        by auto
      have g {a..b} = g {a..min b c} * g {max a c..b}
        by (auto simp: min_def max_def le_less)
      then show g (cbox a b) = g (cbox a b ∩ {x. x • k ≤ c}) * g (cbox a b ∩ {x.
c ≤ x • k})
        by (simp add: atMost_def [symmetric] atLeast_def [symmetric])
    qed
  qed
  show box = (greaterThanLessThan :: real ⇒ _)
    and cbox = (atLeastAtMost :: real ⇒ _)
    and ⋀x::real. x ∈ Basis ⟷ x = 1
    by (simp_all add: fun_eq_iff)
qed

lemma coalesce_less_eq:
  g {a..c} * g {c..b} = g {a..b} if a ≤ c c ≤ b
  by (metis coalesce_less commute left_neutral less_eq_real_def neutral that)

end

lemma operative_realI:
  operative_real f z g if operative f z g
proof −
  interpret operative f z g
```

```
    using that .
  show ?thesis
  proof
    show g {a..b} = z if b ≤ a for a b
      using that box_empty_imp by simp
    show f (g {a..c}) (g {c..b}) = g {a..b} if a < c c < b for a b c
      using that Basis_imp [of 1 a b c]
     by (simp_all add: atMost_def [symmetric] atLeast_def [symmetric] max_def
min_def)
  qed
qed
```

6.13.8 Special case of additivity we need for the FTC

```
lemma additive_tagged_division_1:
  fixes f :: real ⇒ 'a::real_normed_vector
  assumes a ≤ b
    and p tagged_division_of {a..b}
  shows sum (λ(x,K). f(Sup K) − f(Inf K)) p = f b − f a
proof −
   let ?f = (λK::real set. if K = {} then 0 else f(interval_upperbound K) −
f(interval_lowerbound K))
  interpret operative_real plus 0 ?f
    rewrites comm_monoid_set.F (+) 0 = sum
    by standard[1] (auto simp add: sum_def)
  have p_td: p tagged_division_of cbox a b
    using assms(2) box_real(2) by presburger
  have **: cbox a b ≠ {}
    using assms(1) by auto
  then have f b − f a = (∑(x, l)∈p. if l = {} then 0 else f (interval_upperbound
l) − f (interval_lowerbound l))
    using assms(2) tagged_division by force
  then show ?thesis
    using assms by (auto intro!: sum.cong)
qed
```

6.13.9 Fine-ness of a partition w.r.t. a gauge

```
definition fine  (infixr fine 46)
  where d fine s ⟷ (∀(x,k) ∈ s. k ⊆ d x)

lemma fineI:
  assumes ⋀x k. (x, k) ∈ s ⟹ k ⊆ d x
  shows d fine s
  using assms unfolding fine_def by auto

lemma fineD[dest]:
  assumes d fine s
  shows ⋀x k. (x,k) ∈ s ⟹ k ⊆ d x
  using assms unfolding fine_def by auto
```

lemma *fine_Int*: $(\lambda x.\ d1\ x \cap d2\ x)$ *fine* $p \longleftrightarrow d1$ *fine* $p \wedge d2$ *fine* p
unfolding *fine_def* **by** *auto*

lemma *fine_Inter*:
$(\lambda x. \bigcap \{f\ d\ x \mid d.\ \ d \in s\})$ *fine* $p \longleftrightarrow (\forall d{\in}s.\ (f\ d)$ *fine* $p)$
unfolding *fine_def* **by** *blast*

lemma *fine_Un*: d *fine* $p1 \Longrightarrow d$ *fine* $p2 \Longrightarrow d$ *fine* $(p1 \cup p2)$
unfolding *fine_def* **by** *blast*

lemma *fine_Union*: $(\bigwedge p.\ p \in ps \Longrightarrow d$ *fine* $p) \Longrightarrow d$ *fine* $(\bigcup ps)$
unfolding *fine_def* **by** *auto*

lemma *fine_subset*: $p \subseteq q \Longrightarrow d$ *fine* $q \Longrightarrow d$ *fine* p
unfolding *fine_def* **by** *blast*

6.13.10 Some basic combining lemmas

lemma *tagged_division_Union_exists*:
assumes *finite* I
and $\forall i{\in}I.\ \exists p.\ p$ *tagged_division_of* $i \wedge d$ *fine* p
and $\forall i1{\in}I.\ \forall i2{\in}I.\ i1 \neq i2 \longrightarrow$ *interior* $i1 \cap$ *interior* $i2 = \{\}$
and $\bigcup I = i$
obtains p **where** p *tagged_division_of* i **and** d *fine* p
proof −
obtain *pfn* **where** *pfn*:
$\bigwedge x.\ x \in I \Longrightarrow$ *pfn* x *tagged_division_of* x
$\bigwedge x.\ x \in I \Longrightarrow d$ *fine* *pfn* x
using *assms* **by** *metis*
show *thesis*
proof
show $\bigcup$ (*pfn* ‘ I) *tagged_division_of* i
using *assms pfn*(1) *tagged_division_Union* **by** *force*
show d *fine* $\bigcup$ (*pfn* ‘ I)
by (*metis* (*mono_tags*, *lifting*) *fine_Union imageE pfn*(2))
qed
qed

6.13.11 The set we’re concerned with must be closed

lemma *division_of_closed*:
fixes $i :: {'n}{::}$*euclidean_space set*
shows s *division_of* $i \Longrightarrow$ *closed* i
by *blast*

6.13.12 General bisection principle for intervals; might be useful elsewhere

lemma *interval_bisection_step*:

fixes *type* :: *'a*::*euclidean_space*
assumes *emp*: *P* {}
and *Un*: ⋀*S T*. ⟦*P S*; *P T*; *interior*(*S*) ∩ *interior*(*T*) = {}⟧ ⟹ *P* (*S* ∪ *T*)
and *non*: ¬ *P* (*cbox a* (*b*::*'a*))
obtains *c d* **where** ¬ *P* (*cbox c d*)
and ⋀*i*. *i* ∈ *Basis* ⟹ *a*•*i* ≤ *c*•*i* ∧ *c*•*i* ≤ *d*•*i* ∧ *d*•*i* ≤ *b*•*i* ∧ 2 ∗ (*d*•*i* − *c*•*i*) ≤ *b*•*i* − *a*•*i*
proof −
have *cbox a b* ≠ {}
using *emp non* **by** *metis*
then have *ab*: ⋀*i*. *i*∈*Basis* ⟹ *a* • *i* ≤ *b* • *i*
by (*force simp*: *mem_box*)
have *UN_cases*: ⟦*finite* ℱ;
⋀*S*. *S*∈ℱ ⟹ *P S*;
⋀*S*. *S*∈ℱ ⟹ ∃ *a b*. *S* = *cbox a b*;
⋀*S T*. *S*∈ℱ ⟹ *T*∈ℱ ⟹ *S* ≠ *T* ⟹ *interior S* ∩ *interior T* = {}⟧ ⟹ *P* (⋃ℱ) **for** ℱ
proof (*induct* ℱ *rule*: *finite_induct*)
case *empty* **show** *?case*
using *emp* **by** *auto*
next
case (*insert x f*)
then show *?case*
unfolding *Union_insert* **by** (*metis Int_interior_Union_intervals Un insert_iff open_interior*)
qed
let *?ab* = λ*i*. (*a*•*i* + *b*•*i*) / 2
let *?A* = {*cbox c d* | *c d*::*'a*. ∀ *i*∈*Basis*. (*c*•*i* = *a*•*i*) ∧ (*d*•*i* = *?ab i*) ∨
(*c*•*i* = *?ab i*) ∧ (*d*•*i* = *b*•*i*)}
have *P* (⋃ *?A*)
if ⋀*c d*. ∀ *i*∈*Basis*. *a*•*i* ≤ *c*•*i* ∧ *c*•*i* ≤ *d*•*i* ∧ *d*•*i* ≤ *b*•*i* ∧ 2 ∗ (*d*•*i* − *c*•*i*) ≤ *b*•*i* − *a*•*i* ⟹ *P* (*cbox c d*)
proof (*rule UN_cases*)
let *?B* = (λ*S*. *cbox* (∑ *i*∈*Basis*. (*if i* ∈ *S then a*•*i else ?ab i*) ∗$_R$ *i*::*'a*)
(∑ *i*∈*Basis*. (*if i* ∈ *S then ?ab i else b*•*i*) ∗$_R$ *i*)) ‘ {*s*. *s* ⊆ *Basis*}
have *?A* ⊆ *?B*
proof
fix *x*
assume *x* ∈ *?A*
then obtain *c d*
where *x*: *x* = *cbox c d*
⋀*i*. *i* ∈ *Basis* ⟹ *c* • *i* = *a* • *i* ∧ *d* • *i* = *?ab i* ∨ *c* • *i* = *?ab i* ∧ *d* • *i* = *b* • *i*
by *blast*
have *c* = (∑ *i*∈*Basis*. (*if c* • *i* = *a* • *i then a* • *i else ?ab i*) ∗$_R$ *i*)
d = (∑ *i*∈*Basis*. (*if c* • *i* = *a* • *i then ?ab i else b* • *i*) ∗$_R$ *i*)
using *x*(2) *ab* **by** (*fastforce simp add*: *euclidean_eq_iff*[**where** *'a*=*'a*])+
then show *x* ∈ *?B*
unfolding *x* **by** (*rule_tac x*={*i*. *i*∈*Basis* ∧ *c*•*i* = *a*•*i*} **in** *image_eqI*) *auto*

```
  qed
  then show finite ?A
    by (rule finite_subset) auto
next
  fix S
  assume S ∈ ?A
  then obtain c d
    where s: S = cbox c d
     ⋀i. i ∈ Basis ⟹ c · i = a · i ∧ d · i = ?ab i ∨ c · i = ?ab i ∧ d · i = b · i
    by blast
  show P S
    unfolding s using ab s(2) by (fastforce intro!: that)
  show ∃ a b. S = cbox a b
    unfolding s by auto
  fix T
  assume T ∈ ?A
  then obtain e f where t:
    T = cbox e f
    ⋀i. i ∈ Basis ⟹ e · i = a · i ∧ f · i = ?ab i ∨ e · i = ?ab i ∧ f · i = b · i
    by blast
  assume S ≠ T
  then have ¬ (c = e ∧ d = f)
    unfolding s t by auto
  then obtain i where c•i ≠ e•i ∨ d•i ≠ f•i and i': i ∈ Basis
    unfolding euclidean_eq_iff[where 'a='a] by auto
  then have i: c•i ≠ e•i d•i ≠ f•i
    using s(2) t(2) by fastforce+
  have *: ⋀s t. (⋀a. a ∈ s ⟹ a ∈ t ⟹ False) ⟹ s ∩ t = {}
    by auto
  show interior S ∩ interior T = {}
    unfolding s t interior_cbox
  proof (rule *)
    fix x
    assume x ∈ box c d x ∈ box e f
    then have c•i < f•i e•i < d•i
      unfolding mem_box using i' by force+
    with i i' show False
      using s(2) t(2) by fastforce
  qed
qed
also have ⋃ ?A = cbox a b
proof (rule set_eqI,rule)
  fix x
  assume x ∈ ⋃ ?A
  then obtain c d where x:
    x ∈ cbox c d
    ⋀i. i ∈ Basis ⟹ c · i = a · i ∧ d · i = ?ab i ∨ c · i = ?ab i ∧ d · i = b · i
    by blast
  then show x∈cbox a b
```

```
    unfolding mem_box by force
  next
    fix x
    assume x: x ∈ cbox a b
    then have ∀i∈Basis. ∃c d. (c = a•i ∧ d = ?ab i ∨ c = ?ab i ∧ d = b•i) ∧
c≤x•i ∧ x•i ≤ d
      (is ∀i∈Basis. ∃c d. ?P i c d)
      unfolding mem_box by (metis linear)
    then obtain α β where ∀i∈Basis. (α · i = a · i ∧ β · i = ?ab i ∨
        α · i = ?ab i ∧ β · i = b · i) ∧ α · i ≤ x · i ∧ x · i ≤ β · i
      by (auto simp: choice_Basis_iff)
    then show x∈⋃ ?A
      by (force simp add: mem_box)
  qed
  finally show thesis
    by (metis (no_types, lifting) assms(3) that)
qed

lemma interval_bisection:
  fixes type :: 'a::euclidean_space
  assumes P {}
    and Un: ⋀S T. ⟦P S; P T; interior(S) ∩ interior(T) = {}⟧ ⟹ P (S ∪ T)
    and ¬ P (cbox a (b::'a))
  obtains x where x ∈ cbox a b
    and ∀e>0. ∃c d. x ∈ cbox c d ∧ cbox c d ⊆ ball x e ∧ cbox c d ⊆ cbox a b ∧
¬ P (cbox c d)
proof −
  have ∀x. ∃y. ¬ P (cbox (fst x) (snd x)) ⟶ (¬ P (cbox (fst y) (snd y)) ∧
    (∀i∈Basis. fst x•i ≤ fst y•i ∧ fst y•i ≤ snd y•i ∧ snd y•i ≤ snd x•i ∧
      2 ∗ (snd y•i − fst y•i) ≤ snd x•i − fst x•i)) (is ∀x. ?P x)
  proof
    show ?P x for x
    proof (cases P (cbox (fst x) (snd x)))
      case True
      then show ?thesis by auto
    next
      case False
      obtain c d where ¬ P (cbox c d)
        ⋀i. i ∈ Basis ⟹
          fst x · i ≤ c · i ∧
          c · i ≤ d · i ∧
          d · i ≤ snd x · i ∧
          2 ∗ (d · i − c · i) ≤ snd x · i − fst x · i
        by (blast intro: interval_bisection_step[of P, OF assms(1−2) False])
      then show ?thesis
        by (rule_tac x=(c,d) in exI) auto
    qed
  qed
  then obtain f where f:
```

```
    ∀ x.
      ¬ P (cbox (fst x) (snd x)) ⟶
      ¬ P (cbox (fst (f x)) (snd (f x))) ∧
        (∀ i∈Basis.
            fst x ∙ i ≤ fst (f x) ∙ i ∧
            fst (f x) ∙ i ≤ snd (f x) ∙ i ∧
            snd (f x) ∙ i ≤ snd x ∙ i ∧
            2 ∗ (snd (f x) ∙ i − fst (f x) ∙ i) ≤ snd x ∙ i − fst x ∙ i) by metis
  define AB A B where ab_def: AB n = (f ^^ n) (a,b) A n = fst(AB n) B n =
snd(AB n) for n
  have [simp]: A 0 = a B 0 = b
    and ABRAW: ⋀n. ¬ P (cbox (A(Suc n)) (B(Suc n))) ∧
              (∀ i∈Basis. A(n)∙i ≤ A(Suc n)∙i ∧ A(Suc n)∙i ≤ B(Suc n)∙i ∧ B(Suc
n)∙i ≤ B(n)∙i ∧
                  2 ∗ (B(Suc n)∙i − A(Suc n)∙i) ≤ B(n)∙i − A(n)∙i) (is ⋀n. ?P n)
  proof −
    show A 0 = a B 0 = b
      unfolding ab_def by auto
    note S = ab_def funpow.simps o_def id_apply
    show ?P n for n
    proof (induct n)
      case 0
      then show ?case
        unfolding S using ‹¬ P (cbox a b)› f by auto
    next
      case (Suc n)
      then show ?case
        unfolding S
        by (force intro!: f[rule_format])
    qed
  qed
  then have AB: A(n)∙i ≤ A(Suc n)∙i A(Suc n)∙i ≤ B(Suc n)∙i
                B(Suc n)∙i ≤ B(n)∙i 2 ∗ (B(Suc n)∙i − A(Suc n)∙i) ≤ B(n)∙i −
A(n)∙i
              if i∈Basis for i n
    using that by blast+
  have notPAB: ¬ P (cbox (A(Suc n)) (B(Suc n))) for n
    using ABRAW by blast
  have interv: ∃ n. ∀ x∈cbox (A n) (B n). ∀ y∈cbox (A n) (B n). dist x y < e
    if e: 0 < e for e
  proof −
    obtain n where n: (∑ i∈Basis. b ∙ i − a ∙ i) / e < 2 ^ n
      using real_arch_pow[of 2 (sum (λi. b∙i − a∙i) Basis) / e] by auto
    show ?thesis
    proof (rule exI [where x=n], clarify)
      fix x y
      assume xy: x∈cbox (A n) (B n) y∈cbox (A n) (B n)
      have dist x y ≤ sum (λi. |(x − y)∙i|) Basis
        unfolding dist_norm by(rule norm_le_l1)
```

```
    also have ... ≤ sum (λi. B n•i − A n•i) Basis
    proof (rule sum_mono)
      fix i :: 'a
      assume i ∈ Basis
      with xy show |(x − y) · i| ≤ B n · i − A n · i
        by (smt (verit, best) inner_diff_left mem_box(2))
    qed
    also have ... ≤ sum (λi. b•i − a•i) Basis / 2^n
      unfolding sum_divide_distrib
    proof (rule sum_mono)
      show B n · i − A n · i ≤ (b · i − a · i) / 2 ^ n if i: i ∈ Basis for i
      proof (induct n)
        case 0
        then show ?case
          unfolding AB by auto
      next
        case (Suc n)
        have B (Suc n) · i − A (Suc n) · i ≤ (B n · i − A n · i) / 2
          using AB(3) that AB(4)[of i n] using i by auto
        also have ... ≤ (b · i − a · i) / 2 ^ Suc n
          using Suc by (auto simp add: field_simps)
        finally show ?case .
      qed
    qed
    also have ... < e
      using n using e by (auto simp add: field_simps)
    finally show dist x y < e .
  qed
qed
have ABsubset: cbox (A n) (B n) ⊆ cbox (A m) (B m) if m ≤ n for m n
  using that
proof (induction rule: inc_induct)
  case (step i) show ?case
    by (smt (verit, ccfv_SIG) ABRAW in_mono mem_box(2) step.IH subsetI)
qed simp
have ⋀n. cbox (A n) (B n) ≠ {}
  by (meson AB dual_order.trans interval_not_empty)
then obtain x0 where x0: ⋀n. x0 ∈ cbox (A n) (B n)
  using decreasing_closed_nest [OF closed_cbox] ABsubset interv by blast
show thesis
proof (intro that strip)
  show x0 ∈ cbox a b
    using ‹A 0 = a› ‹B 0 = b› x0 by blast
next
  fix e :: real
  assume e > 0
  from interv[OF this] obtain n
    where n: ∀x∈cbox (A n) (B n). ∀y∈cbox (A n) (B n). dist x y < e ..
  have ¬ P (cbox (A n) (B n))
```

```
    proof (cases 0 < n)
      case True then show ?thesis
        by (metis Suc_pred' notPAB)
    next
      case False then show ?thesis
        using ‹A 0 = a› ‹B 0 = b› ‹¬ P (cbox a b)› by blast
    qed
    moreover have cbox (A n) (B n) ⊆ ball x0 e
      using n using x0[of n] by auto
    moreover have cbox (A n) (B n) ⊆ cbox a b
      using ABsubset ‹A 0 = a› ‹B 0 = b› by blast
    ultimately show ∃c d. x0 ∈ cbox c d ∧ cbox c d ⊆ ball x0 e ∧ cbox c d ⊆
cbox a b ∧ ¬ P (cbox c d)
      by (meson x0)
  qed
qed
```

6.13.13 Cousin's lemma

```
lemma fine_division_exists:
  fixes a b :: 'a::euclidean_space
  assumes gauge g
  obtains p where p tagged_division_of (cbox a b) g fine p
proof (cases ∃p. p tagged_division_of (cbox a b) ∧ g fine p)
  case True
  then show ?thesis
    using that by auto
next
  case False
  assume ¬ (∃p. p tagged_division_of (cbox a b) ∧ g fine p)
  obtain x where x:
      x ∈ (cbox a b)
      ⋀e. 0 < e ⟹
        ∃c d.
          x ∈ cbox c d ∧
          cbox c d ⊆ ball x e ∧
          cbox c d ⊆ (cbox a b) ∧
          ¬ (∃p. p tagged_division_of cbox c d ∧ g fine p)
    proof (rule interval_bisection[of λs. ∃p. p tagged_division_of s ∧ _ p, OF _
_ False])
      show ∃p. p tagged_division_of {} ∧ g fine p
        by (simp add: fine_def)
    qed (meson tagged_division_Un fine_Un)+
  obtain e where e: e > 0 ball x e ⊆ g x
    by (meson assms gauge_def openE)
  then obtain c d where c_d: x ∈ cbox c d
                           cbox c d ⊆ ball x e
                           cbox c d ⊆ cbox a b
                           ¬ (∃p. p tagged_division_of cbox c d ∧ g fine p)
```

by (*meson x(2)*)
have *g fine* {(*x*, *cbox c d*)}
unfolding *fine_def* **using** *e* **using** *c_d(2)* **by** *auto*
then show *?thesis*
using *tagged_division_of_self*[*OF c_d(1)*] **using** *c_d* **by** *simp*
qed

lemma *fine_division_exists_real*:
fixes *a b* :: *real*
assumes *gauge g*
obtains *p* **where** *p tagged_division_of* {*a* .. *b*} *g fine p*
by (*metis assms box_real(2) fine_division_exists*)

6.13.14 A technical lemma about "refinement" of division

lemma *tagged_division_finer*:
fixes *p* :: ('*a*::*euclidean_space* × ('*a*::*euclidean_space set*)) *set*
assumes *ptag*: *p tagged_division_of* (*cbox a b*)
and *gauge d*
obtains *q* **where** *q tagged_division_of* (*cbox a b*)
and *d fine q*
and ∀(*x*,*K*) ∈ *p*. *K* ⊆ *d*(*x*) ⟶ (*x*,*K*) ∈ *q*
proof −
have *p*: *finite p p tagged_partial_division_of* (*cbox a b*)
using *ptag tagged_division_of_def* **by** *blast*+
have (∃ *q*. *q tagged_division_of* (⋃{*k*. ∃ *x*. (*x*,*k*) ∈ *p*}) ∧ *d fine q* ∧ (∀(*x*,*k*) ∈ *p*. *k* ⊆ *d*(*x*) ⟶ (*x*,*k*) ∈ *q*))
if *finite p p tagged_partial_division_of* (*cbox a b*) *gauge d* **for** *p*
using *that*
proof (*induct p*)
case *empty*
show *?case*
by (*force simp add*: *fine_def*)
next
case (*insert xk p*)
obtain *x k* **where** *xk*: *xk* = (*x*, *k*)
using *surj_pair*[*of xk*] **by** *metis*
obtain *q1* **where** *q1*: *q1 tagged_division_of* ⋃{*k*. ∃ *x*. (*x*, *k*) ∈ *p*}
and *d fine q1*
and *q1I*: ⋀*x k*. ⟦(*x*, *k*)∈*p*; *k* ⊆ *d x*⟧ ⟹ (*x*, *k*) ∈ *q1*
using *insert*
by (*metis* (*mono_tags*, *lifting*) *case_prodD subset_insertI tagged_partial_division_subset*)
have ∗: ⋃{*l*. ∃ *y*. (*y*,*l*) ∈ *insert xk p*} = *k* ∪ ⋃{*l*. ∃ *y*. (*y*,*l*) ∈ *p*}
unfolding *xk* **by** *auto*
note *p* = *tagged_partial_division_ofD*[*OF insert(4)*]
obtain *u v* **where** *uv*: *k* = *cbox u v*
using *p*(*4*) *xk* **by** *blast*
have {*K*. ∃ *x*. (*x*, *K*) ∈ *p*} ⊆ *snd* ‘ *p*
by *force*

```
    then have finite {K. ∃x. (x, K) ∈ p}
      using finite_surj insert.hyps(1) by blast
    then have int: interior (cbox u v) ∩ interior (⋃{k. ∃x. (x, k) ∈ p}) = {}
    proof (rule Int_interior_Union_intervals)
      show open (interior (cbox u v))
        by auto
      show ⋀T. T ∈ {K. ∃x. (x, K) ∈ p} ⟹ ∃a b. T = cbox a b
        using p(4) by auto
      show ⋀T. T ∈ {K. ∃x. (x, K) ∈ p} ⟹ interior (cbox u v) ∩ interior T =
{}
        using insert.hyps(2) p(5) uv xk by blast
    qed
    show ?case
    proof (cases cbox u v ⊆ d x)
      case True
      have {(x, cbox u v)} tagged_division_of cbox u v
        by (simp add: p(2) uv xk tagged_division_of_self)
       then have {(x, cbox u v)} ∪ q1 tagged_division_of ⋃{K. ∃x. (x, K) ∈
insert xk p}
        by (smt (verit, best) ∗ int q1 tagged_division_Un uv)
      moreover have d fine ({(x,cbox u v)} ∪ q1)
        using True ‹d fine q1› fine_def by fastforce
      ultimately show ?thesis
        by (metis (no_types, lifting) case_prodI2 insert_iff insert_is_Un q1I uv
xk)
    next
      case False
      obtain q2 where q2: q2 tagged_division_of cbox u v d fine q2
        using fine_division_exists[OF assms(2)] by blast
      show ?thesis
      proof (intro exI conjI)
        show q2 ∪ q1 tagged_division_of ⋃ {k. ∃x. (x, k) ∈ insert xk p}
          by (smt (verit, best) ∗ int q1 q2(1) tagged_division_Un uv)
        show d fine q2 ∪ q1
          by (simp add: ‹d fine q1› fine_Un q2(2))
      qed (use False uv xk q1I in auto)
    qed
  qed
  with p obtain q where q: q tagged_division_of ⋃{k. ∃x. (x, k) ∈ p} d fine q
∀(x, k)∈p. k ⊆ d x ⟶ (x, k) ∈ q
    by (meson ‹gauge d›)
  with ptag that show ?thesis by auto
qed
```

Covering lemma

Some technical lemmas used in the approximation results that follow. Proof of the covering lemma is an obvious multidimensional generalization of Lemma 3, p65 of Swartz's "Introduction to Gauge Integrals".

proposition *covering_lemma*:
assumes $S \subseteq$ *cbox a b box a b* $\neq \{\}$ *gauge g*
obtains $\mathcal{D}$ **where**
$\bigcup \mathcal{D} \subseteq$ *cbox a b*
$\bigwedge K.\ K \in \mathcal{D} \Longrightarrow$ *interior K* $\neq \{\} \wedge (\exists\, c\ d.\ K =$ *cbox c d*$)$
pairwise $(\lambda A\ B.$ *interior A* $\cap$ *interior B* $= \{\})\ \mathcal{D}$
$\bigwedge K.\ K \in \mathcal{D} \Longrightarrow \exists\, x \in S \cap K.\ K \subseteq g\ x$
$\bigwedge u\ v.$ *cbox u v* $\in \mathcal{D} \Longrightarrow \exists\, n.\ \forall\, i \in$ *Basis*. $v \cdot i - u \cdot i = (b \cdot i - a \cdot i)$ / *2^n*
$S \subseteq \bigcup \mathcal{D}$
proof −
have *aibi*: $\bigwedge i.\ i \in$ *Basis* $\Longrightarrow a \cdot i \leq b \cdot i$ **and** *normab*: $0 <$ *norm*$(b - a)$
using ‹*box a b* $\neq \{\}$› *box_eq_empty box_idem* **by** *fastforce+*
let *?K0* $= \lambda(n, f{::}'a{\Rightarrow}nat).$
cbox $(\sum i \in$ *Basis*. $(a \cdot i + (f\, i$ / *2^n*$) * (b \cdot i - a \cdot i)) *_R\ i)$
$(\sum i \in$ *Basis*. $(a \cdot i + ((f\, i + 1)$ / *2^n*$) * (b \cdot i - a \cdot i)) *_R\ i)$
let *?D0* $=$ *?K0* ‘ $(SIGMA\ n{:}UNIV.\ Pi_E\ Basis\ (\lambda i{::}'a.\ lessThan\ (2\hat{}n)))$
obtain $\mathcal{D}0$ **where** *count*: *countable* $\mathcal{D}0$
and *sub*: $\bigcup \mathcal{D}0 \subseteq$ *cbox a b*
and *int*: $\bigwedge K.\ K \in \mathcal{D}0 \Longrightarrow$ *(interior K* $\neq \{\}) \wedge (\exists\, c\ d.\ K =$ *cbox c d*$)$
and *intdj*: $\bigwedge A\ B.\ [\![A \in \mathcal{D}0;\ B \in \mathcal{D}0]\!] \Longrightarrow A \subseteq B \vee B \subseteq A \vee$ *interior* $A \cap$ *interior* $B = \{\}$
and *SK*: $\bigwedge x.\ x \in S \Longrightarrow \exists\, K \in \mathcal{D}0.\ x \in K \wedge K \subseteq g\ x$
and *cbox*: $\bigwedge u\ v.$ *cbox u v* $\in \mathcal{D}0 \Longrightarrow \exists\, n.\ \forall\, i \in$ *Basis*. $v \cdot i - u \cdot i = (b \cdot i - a \cdot i)$ / *2^n*
and *fin*: $\bigwedge K.\ K \in \mathcal{D}0 \Longrightarrow$ *finite* $\{L \in \mathcal{D}0.\ K \subseteq L\}$
proof
show *countable ?D0*
by (*simp add*: *countable_PiE*)
next
show $\bigcup$ *?D0* $\subseteq$ *cbox a b*
apply (*simp add*: *UN_subset_iff*)
apply (*intro conjI allI ballI subset_box_imp*)
apply (*simp add*: *field_simps aibi mult_right_mono*)
apply (*force simp*: *aibi scaling_mono nat_less_real_le dest*: *PiE_mem intro*: *mult_right_mono*)
done
next
show $\bigwedge K.\ K \in$ *?D0* $\Longrightarrow$ *interior K* $\neq \{\} \wedge (\exists\, c\ d.\ K =$ *cbox c d*$)$
using ‹*box a b* $\neq \{\}$›
by (*clarsimp simp*: *box_eq_empty*) (*fastforce simp add*: *field_split_simps dest*: *PiE_mem*)
next
have *realff*: *(real w)* $*$ *2^m* $<$ *(real v)* $*$ *2^n* $\longleftrightarrow$ *w* $*$ *2^m* $<$ *v* $*$ *2^n* **for** *m n v w*
using *of_nat_less_iff less_imp_of_nat_less* **by** *fastforce*
have $*$: $\forall\, v\ w.$ *?K0(m,v)* $\subseteq$ *?K0(n,w)* $\vee$ *?K0(n,w)* $\subseteq$ *?K0(m,v)* $\vee$ *interior(?K0(m,v))* $\cap$ *interior(?K0(n,w))* $= \{\}$
for *m n* — The symmetry argument requires a single HOL formula
proof (*rule linorder_wlog* [**where** *a*=*m* **and** *b*=*n*], *intro allI impI*)

fix *v w m* **and** *n*::*nat*
assume $m \leq n$ — WLOG we can assume $m \leq n$, when the first disjunct becomes impossible
have *?K0(n,w)* $\subseteq$ *?K0(m,v)* $\vee$ *interior*(*?K0(m,v)*) $\cap$ *interior*(*?K0(n,w)*) = {}
apply (*rule ccontr*)
apply (*simp add*: *subset_box disjoint_interval*)
apply (*clarsimp simp add*: *aibi mult_le_cancel_right divide_le_cancel not_less not_le*)
apply (*drule_tac x=i* **in** *bspec*, *assumption*)
using ‹*m*≤*n*› *realff* [*of* _ _ *1+_*] *realff* [*of 1+_ 1+_*]
apply (*auto simp*: *divide_simps add.commute not_le nat_le_iff_add realff*)
apply (*metis* (*no_types*, *opaque_lifting*) *power_add mult_Suc mult_less_cancel2 not_less_eq mult.assoc*)+
done
then show *?K0(m,v)* $\subseteq$ *?K0(n,w)* $\vee$ *?K0(n,w)* $\subseteq$ *?K0(m,v)* $\vee$ *interior*(*?K0(m,v)*) $\cap$ *interior*(*?K0(n,w)*) = {}
by *meson*
qed *auto*
show $\bigwedge A\ B.\ [\![A \in ?D0;\ B \in ?D0]\!] \Longrightarrow A \subseteq B \vee B \subseteq A \vee \mathit{interior}\ A \cap \mathit{interior}\ B = \{\}$
using ∗ **by** *fastforce*
next
show $\exists K \in ?D0.\ x \in K \wedge K \subseteq g\ x$ **if** $x \in S$ **for** *x*
proof (*simp only*: *bex_simps split_paired_Bex_Sigma*)
show $\exists n.\ \exists f \in \mathit{Basis} \to_E \{..<2\ \hat{}\ n\}.\ x \in ?K0(n,f) \wedge ?K0(n,f) \subseteq g\ x$
proof −
obtain *e* **where** *0 < e*
and *e*: $\bigwedge y.\ (\bigwedge i.\ i \in \mathit{Basis} \Longrightarrow |x \cdot i - y \cdot i| \leq e) \Longrightarrow y \in g\ x$
proof −
have *x* ∈ *g x open* (*g x*)
using ‹*gauge g*› **by** (*auto simp*: *gauge_def*)
then obtain *ε* **where** *0 < ε* **and** *ε*: *ball x ε* ⊆ *g x*
using *openE* **by** *blast*
have *norm* (*x* − *y*) < *ε*
if ($\bigwedge i.\ i \in \mathit{Basis} \Longrightarrow |x \cdot i - y \cdot i| \leq \varepsilon\ /\ (2 * \mathit{real}\ \mathit{DIM}('a))$) **for** *y*
proof −
have *norm* (*x* − *y*) ≤ ($\sum i \in \mathit{Basis}.\ |x \cdot i - y \cdot i|$)
by (*metis* (*no_types*, *lifting*) *inner_diff_left norm_le_l1 sum.cong*)
also have ... ≤ *DIM*(′*a*) ∗ (*ε* / (2 ∗ *real DIM*(′*a*)))
by (*meson sum_bounded_above that*)
also have ... = *ε* / 2
by (*simp add*: *field_split_simps*)
finally show *?thesis*
using ‹*0 < ε*› **by** *linarith*
qed
then show *?thesis*
by (*rule_tac e* = *ε* / 2 / *DIM*(′*a*) **in** *that*) (*simp_all add*: ‹*0 < ε*› *dist_norm subsetD* [*OF ε*])

```
    qed
    have xab: x ∈ cbox a b
      using ‹x ∈ S› ‹S ⊆ cbox a b› by blast
    obtain n where n: norm (b − a) / 2^n < e
      using real_arch_pow_inv [of e / norm(b − a) 1/2] normab ‹0 < e›
      by (auto simp: field_split_simps)
    then have norm (b − a) < e * 2^n
      by (auto simp: field_split_simps)
    then have bai: b · i − a · i < e * 2 ^ n if i ∈ Basis for i
    by (smt (verit, del_insts) Basis_le_norm diff_add_cancel inner_simps(1)
that)
     have D: (a + n ≤ x ∧ x ≤ a + m) ⟹ (a + n ≤ y ∧ y ≤ a + m) ⟹
abs(x − y) ≤ m − n
          for a m n x and y::real
      by auto
    have ∀ i∈Basis. ∃ k<2 ^ n. (a · i + real k * (b · i − a · i)) / 2 ^ n ≤ x · i ∧
          x · i ≤ a · i + (real k + 1) * (b · i − a · i) / 2 ^ n)
    proof
      fix i::'a assume i ∈ Basis
      consider x · i = b · i | x · i < b · i
        using ‹i ∈ Basis› mem_box(2) xab by force
      then show ∃ k<2 ^ n. (a · i + real k * (b · i − a · i)) / 2 ^ n ≤ x · i ∧
                    x · i ≤ a · i + (real k + 1) * (b · i − a · i) / 2 ^ n)
      proof cases
        case 1 then show ?thesis
          by (rule_tac x = 2^n − 1 in exI) (auto simp: algebra_simps
field_split_simps of_nat_diff ‹i ∈ Basis› aibi)
      next
        case 2
        then have abi_less: a · i < b · i
          using ‹i ∈ Basis› xab by (auto simp: mem_box)
        let ?k = nat ⌊2 ^ n * (x · i − a · i) / (b · i − a · i)⌋
        show ?thesis
        proof (intro exI conjI)
          show ?k < 2 ^ n
            using aibi xab ‹i ∈ Basis›
              by (force simp: nat_less_iff floor_less_iff field_split_simps 2
mem_box)
        next
          have a · i + real ?k * (b · i − a · i) / 2 ^ n ≤
              a · i + (2 ^ n * (x · i − a · i) / (b · i − a · i)) * (b · i − a · i) /
2 ^ n
            using aibi [OF ‹i ∈ Basis›] xab 2
            apply (intro add_left_mono mult_right_mono divide_right_mono
of_nat_floor)
              apply (auto simp: ‹i ∈ Basis› mem_box field_split_simps)
            done
          also have ... = x · i
            using abi_less by (simp add: field_split_simps)
```

```
          finally show a · i + real ?k * (b · i − a · i) / 2 ^ n ≤ x · i .
        next
          have x · i ≤ a · i + (2 ^ n * (x · i − a · i) / (b · i − a · i)) * (b · i
− a · i) / 2 ^ n
            using abi_less by (simp add: field_split_simps)
          also have ... ≤ a · i + (real ?k + 1) * (b · i − a · i) / 2 ^ n
            using aibi [OF ‹i ∈ Basis›] xab
            apply (intro add_left_mono mult_right_mono divide_right_mono
of_nat_floor)
              apply (auto simp: ‹i ∈ Basis› mem_box divide_simps)
            done
          finally show x · i ≤ a · i + (real ?k + 1) * (b · i − a · i) / 2 ^ n .
        qed
      qed
    qed
    then have ∃f∈Basis →E {..<2 ^ n}. x ∈ ?K0(n,f)
      apply (simp add: mem_box Bex_def)
      apply (clarify dest!: bchoice)
      apply (rule_tac x=restrict f Basis in exI, simp)
      done
    moreover have ⋀f. x ∈ ?K0(n,f) ⟹ ?K0(n,f) ⊆ g x
      apply (clarsimp simp add: mem_box)
      apply (rule e)
      apply (drule bspec D, assumption)+
      apply (erule order_trans)
      apply (simp add: divide_simps)
      using bai apply (force simp add: algebra_simps)
      done
    ultimately show ?thesis by auto
  qed
 qed
next
  show ⋀u v. cbox u v ∈ ?D0 ⟹ ∃n. ∀i ∈ Basis. v · i − u · i = (b · i − a ·
i) / 2^n
    by (force simp: eq_cbox box_eq_empty field_simps dest!: aibi)
next
  obtain j::'a where j ∈ Basis
    using nonempty_Basis by blast
  have finite {L ∈ ?D0. ?K0(n,f) ⊆ L} if f ∈ Basis →E {..<2 ^ n} for n f
  proof (rule finite_subset)
    let ?B = (λ(n, f::'a⇒nat). cbox (∑i∈Basis. (a · i + (f i) / 2^n * (b · i −
a · i)) *R i)
                                    (∑i∈Basis. (a · i + ((f i) + 1) / 2^n * (b · i −
a · i)) *R i))
             ' (SIGMA m:atMost n. Pi_E Basis (λi::'a. lessThan (2^m)))
    have ?K0(m,g) ∈ ?B if g ∈ Basis →E {..<2 ^ m} ?K0(n,f) ⊆ ?K0(m,g)
for m g
    proof −
      have dd: w / m ≤ v / n ∧ (v+1) / n ≤ (w+1) / m
```

$\Longrightarrow$ *inverse n* $\leq$ *inverse m* **for** *w m v n*::*real*
by (*auto simp*: *field_split_simps*)
have *bjaj*: $b \cdot j - a \cdot j > 0$
using ‹$j \in$ *Basis*› ‹*box a b* $\neq \{\}$› *box_eq_empty*(*1*) **by** *fastforce*
have $((g\ j)\ /\ 2\ \hat{}\ m) * (b \cdot j - a \cdot j) \leq ((f\ j)\ /\ 2\ \hat{}\ n) * (b \cdot j - a \cdot j) \wedge$
$(((f\ j) + 1)\ /\ 2\ \hat{}\ n) * (b \cdot j - a \cdot j) \leq (((g\ j) + 1)\ /\ 2\ \hat{}\ m) * (b \cdot j - a \cdot j)$
using *that* ‹$j \in$ *Basis*› **by** (*simp add*: *subset_box field_split_simps aibi*)
then have $((g\ j)\ /\ 2\ \hat{}\ m) \leq ((f\ j)\ /\ 2\ \hat{}\ n) \wedge$
$((real(f\ j) + 1)\ /\ 2\ \hat{}\ n) \leq ((real(g\ j) + 1)\ /\ 2\ \hat{}\ m)$
by (*metis bjaj mult.commute of_nat_1 of_nat_add mult_le_cancel_iff2*)
then have *inverse* $(2\hat{}n) \leq$ (*inverse* $(2\hat{}m)$ :: *real*)
by (*rule dd*)
then have $m \leq n$
by *auto*
show *?thesis*
by (*rule imageI*) (*simp add*: ‹$m \leq n$› *that*)
qed
then show $\{L \in$ *?D0*. *?K0*(*n*,*f*) $\subseteq L\} \subseteq$ *?B*
by *auto*
show *finite ?B*
by (*intro finite_imageI finite_SigmaI finite_atMost finite_lessThan finite_PiE finite_Basis*)
qed
then show *finite* $\{L \in$ *?D0*. $K \subseteq L\}$ **if** $K \in$ *?D0* **for** *K*
using *that* **by** *auto*
qed
let *?D1* $= \{K \in \mathcal{D}0.\ \exists x \in S \cap K.\ K \subseteq g\ x\}$
obtain $\mathcal{D}$ **where** *count*: *countable* $\mathcal{D}$
and *sub*: $\bigcup\mathcal{D} \subseteq$ *cbox a b* $S \subseteq \bigcup\mathcal{D}$
and *int*: $\bigwedge K.\ K \in \mathcal{D} \Longrightarrow$ (*interior* $K \neq \{\}$) $\wedge$ ($\exists\, c\ d.\ K =$ *cbox c d*)
and *intdj*: $\bigwedge A\ B.\ [\![A \in \mathcal{D};\ B \in \mathcal{D}]\!] \Longrightarrow A \subseteq B \vee B \subseteq A \vee$ *interior A* $\cap$ *interior B* $= \{\}$
and *SK*: $\bigwedge K.\ K \in \mathcal{D} \Longrightarrow \exists x.\ x \in S \cap K \wedge K \subseteq g\ x$
and *cbox*: $\bigwedge u\ v.$ *cbox u v* $\in \mathcal{D} \Longrightarrow \exists n.\ \forall i \in$ *Basis*. $v \cdot i - u \cdot i = (b \cdot i - a \cdot i)\ /\ 2\hat{}n$
and *fin*: $\bigwedge K.\ K \in \mathcal{D} \Longrightarrow$ *finite* $\{L.\ L \in \mathcal{D} \wedge K \subseteq L\}$
proof
show *countable ?D1* **using** *count countable_subset*
by (*simp add*: *count countable_subset*)
show $\bigcup$ *?D1* $\subseteq$ *cbox a b*
using *sub* **by** *blast*
show $S \subseteq \bigcup$ *?D1*
using *SK* **by** (*force simp*:)
show $\bigwedge K.\ K \in$ *?D1* $\Longrightarrow$ (*interior* $K \neq \{\}$) $\wedge$ ($\exists\, c\ d.\ K =$ *cbox c d*)
using *int* **by** *blast*
show $\bigwedge A\ B.\ [\![A \in$ *?D1*; $B \in$ *?D1*$]\!] \Longrightarrow A \subseteq B \vee B \subseteq A \vee$ *interior A* $\cap$ *interior B* $= \{\}$
using *intdj* **by** *blast*

```
    show ⋀K. K ∈ ?D1 ⟹ ∃x. x ∈ S ∩ K ∧ K ⊆ g x
      by auto
    show ⋀u v. cbox u v ∈ ?D1 ⟹ ∃n. ∀i∈Basis. v ∙ i − u ∙ i = (b ∙ i − a ∙
i) / 2^n
      using cbox by blast
    show ⋀K. K ∈ ?D1 ⟹ finite {L. L ∈ ?D1 ∧ K ⊆ L}
    using fin by simp (metis (mono_tags, lifting) Collect_mono rev_finite_subset)
  qed
  let ?D = {K ∈ 𝒟. ∀K'. K' ∈ 𝒟 ∧ K ≠ K' ⟶ ¬(K ⊆ K')}
  show ?thesis
  proof
    show countable ?D
      by (blast intro: countable_subset [OF _ count])
    show ⋃?D ⊆ cbox a b
      using sub by blast
    show S ⊆ ⋃?D
    proof clarsimp
      fix x
      assume x ∈ S
      then obtain X where x ∈ X X ∈ 𝒟 using ‹S ⊆ ⋃𝒟› by blast
      let ?R = {(K,L). K ∈ 𝒟 ∧ L ∈ 𝒟 ∧ L ⊂ K}
      have irrR: irrefl ?R by (force simp: irrefl_def)
      have traR: trans ?R by (force simp: trans_def)
      have finR: ⋀x. finite {y. (y, x) ∈ ?R}
     by simp (metis (mono_tags, lifting) fin ‹X ∈ 𝒟› finite_subset mem_Collect_eq
psubset_imp_subset subsetI)
      have {X ∈ 𝒟. x ∈ X} ≠ {}
        using ‹X ∈ 𝒟› ‹x ∈ X› by blast
      then obtain Y where Y ∈ {X ∈ 𝒟. x ∈ X} ⋀Y'. (Y', Y) ∈ ?R ⟹ Y' ∉
{X ∈ 𝒟. x ∈ X}
        by (rule wfE_min' [OF wf_finite_segments [OF irrR traR finR]]) blast
      then show ∃Y. Y ∈ 𝒟 ∧ (∀K'. K' ∈ 𝒟 ∧ Y ≠ K' ⟶ ¬ Y ⊆ K') ∧ x ∈ Y
        by blast
    qed
    show ⋀K. K ∈ ?D ⟹ interior K ≠ {} ∧ (∃c d. K = cbox c d)
      by (blast intro: dest: int)
    show pairwise (λA B. interior A ∩ interior B = {}) ?D
      using intdj by (simp add: pairwise_def) metis
    show ⋀K. K ∈ ?D ⟹ ∃x ∈ S ∩ K. K ⊆ g x
      using SK by force
    show ⋀u v. cbox u v ∈ ?D ⟹ ∃n. ∀i∈Basis. v ∙ i − u ∙ i = (b ∙ i − a ∙ i) /
2^n
      using cbox by force
    qed
qed
```

6.13.15 Division filter

Divisions over all gauges towards finer divisions.

definition *division_filter* :: *'a*::*euclidean_space set* ⇒ (*'a* × *'a set*) *set filter*
where *division_filter s* = (*INF g*∈{*g*. *gauge g*}. *principal* {*p*. *p tagged_division_of s* ∧ *g fine p*})

proposition *eventually_division_filter*:
(∀$_F$ *p in division_filter s*. *P p*) ⟷
(∃ *g*. *gauge g* ∧ (∀ *p*. *p tagged_division_of s* ∧ *g fine p* ⟶ *P p*))
unfolding *division_filter_def*
proof (*subst eventually_INF_base*; *clarsimp*)
fix *g1 g2* :: *'a* ⇒ *'a set* **show** *gauge g1* ⟹ *gauge g2* ⟹ ∃ *x*. *gauge x* ∧
{*p*. *p tagged_division_of s* ∧ *x fine p*} ⊆ {*p*. *p tagged_division_of s* ∧ *g1 fine p*} ∧
{*p*. *p tagged_division_of s* ∧ *x fine p*} ⊆ {*p*. *p tagged_division_of s* ∧ *g2 fine p*}
by (*intro exI*[*of* _ λ*x*. *g1 x* ∩ *g2 x*]) (*auto simp*: *fine_Int*)
qed (*auto simp*: *eventually_principal*)

lemma *division_filter_not_empty*: *division_filter* (*cbox a b*) ≠ *bot*
unfolding *trivial_limit_def eventually_division_filter*
by (*auto elim*: *fine_division_exists*)

lemma *eventually_division_filter_tagged_division*:
eventually (λ*p*. *p tagged_division_of s*) (*division_filter s*)
using *eventually_division_filter* **by** *auto*

end

6.14 Henstock-Kurzweil Gauge Integration in Many Dimensions

theory *Henstock_Kurzweil_Integration*
imports
Lebesgue_Measure Tagged_Division
begin

lemma *norm_diff2*: ⟦*y* = *y1* + *y2*; *x* = *x1* + *x2*; *e* = *e1* + *e2*; *norm*(*y1* − *x1*) ≤ *e1*; *norm*(*y2* − *x2*) ≤ *e2*⟧
⟹ *norm*(*y*−*x*) ≤ *e*
by (*smt* (*verit*, *ccfv_SIG*) *norm_diff_triangle_ineq*)

lemma *setcomp_dot1*: {*z*. *P* (*z* ∙ (*i*,*0*))} = {(*x*,*y*). *P*(*x* ∙ *i*)}
by *auto*

lemma *setcomp_dot2*: {*z*. *P* (*z* ∙ (*0*,*i*))} = {(*x*,*y*). *P*(*y* ∙ *i*)}
by *auto*

lemma *Sigma_Int_Paircomp1*: (*Sigma A B*) ∩ {(*x*, *y*). *P x*} = *Sigma* (*A* ∩ {*x*. *P x*}) *B*

by *blast*

lemma *Sigma_Int_Paircomp2*: (*Sigma A B*) ∩ {(*x*, *y*). *P y*} = *Sigma A* (λ*z*. *B z* ∩ {*y*. *P y*})
by *blast*

6.14.1 Content (length, area, volume...) of an interval

abbreviation *content* :: *'a*::*euclidean_space set* ⇒ *real*
where *content s* ≡ *measure lborel s*

lemma *content_cbox_cases*:
content (*cbox a b*) = (*if* ∀ *i*∈*Basis*. *a*•*i* ≤ *b*•*i then prod* (λ*i*. *b*•*i* − *a*•*i*) *Basis else* 0)
by (*simp add*: *measure_lborel_cbox_eq inner_diff*)

lemma *content_cbox*: ∀ *i*∈*Basis*. *a*•*i* ≤ *b*•*i* ⟹ *content* (*cbox a b*) = (∏ *i*∈*Basis*. *b*•*i* − *a*•*i*)
unfolding *content_cbox_cases* **by** *simp*

lemma *content_cbox'*: *cbox a b* ≠ {} ⟹ *content* (*cbox a b*) = (∏ *i*∈*Basis*. *b*•*i* − *a*•*i*)
by (*simp add*: *box_ne_empty inner_diff*)

lemma *content_cbox_if*: *content* (*cbox a b*) = (*if cbox a b* = {} *then 0 else* ∏ *i*∈*Basis*. *b*•*i* − *a*•*i*)
by (*simp add*: *content_cbox'*)

lemma *content_cbox_cart*:
cbox a b ≠ {} ⟹ *content*(*cbox a b*) = *prod* (λ*i*. *b*$*i* − *a*$*i*) *UNIV*
by (*simp add*: *content_cbox_if Basis_vec_def cart_eq_inner_axis axis_eq_axis prod.UNION_disjoint*)

lemma *content_cbox_if_cart*:
content(*cbox a b*) = (*if cbox a b* = {} *then 0 else prod* (λ*i*. *b*$*i* − *a*$*i*) *UNIV*)
by (*simp add*: *content_cbox_cart*)

lemma *content_division_of*:
assumes *K* ∈ 𝒟 𝒟 *division_of S*
shows *content K* = (∏ *i* ∈ *Basis*. *interval_upperbound K* • *i* − *interval_lowerbound K* • *i*)
proof −
obtain *a b* **where** *K* = *cbox a b*
using *cbox_division_memE assms* **by** *metis*
then show *?thesis*
using *assms* **by** (*force simp*: *division_of_def content_cbox'*)
qed

lemma *content_real*: *a* ≤ *b* ⟹ *content* {*a*..*b*} = *b* − *a*

by *simp*

lemma *abs_eq_content*: $|y - x| = (\mathit{if}\ x{\le}y\ \mathit{then}\ \mathit{content}\ \{x..y\}\ \mathit{else}\ \mathit{content}\ \{y..x\})$
by (*auto simp*: *content_real*)

lemma *content_singleton*: $\mathit{content}\ \{a\} = 0$
by *simp*

lemma *content_unit*[*iff*]: $\mathit{content}\ (\mathit{cbox}\ 0\ (\mathit{One}::'a::\mathit{euclidean\_space})) = 1$
by *simp*

lemma *content_pos_le* [*iff*]: $0 \le \mathit{content}\ X$
by *simp*

corollary *content_nonneg* [*simp*]: $\neg\ \mathit{content}\ (\mathit{cbox}\ a\ b) < 0$
using *not_le* **by** *blast*

lemma *content_pos_lt*: $\forall\, i{\in}\mathit{Basis}.\ a{\cdot}i < b{\cdot}i \Longrightarrow 0 < \mathit{content}\ (\mathit{cbox}\ a\ b)$
by (*auto simp*: *less_imp_le inner_diff box_eq_empty intro*!: *prod_pos*)

lemma *content_eq_0*: $\mathit{content}\ (\mathit{cbox}\ a\ b) = 0 \longleftrightarrow (\exists\, i{\in}\mathit{Basis}.\ b{\cdot}i \le a{\cdot}i)$
by (*auto simp*: *content_cbox_cases not_le intro*: *less_imp_le antisym eq_refl*)

lemma *content_eq_0_interior*: $\mathit{content}\ (\mathit{cbox}\ a\ b) = 0 \longleftrightarrow \mathit{interior}(\mathit{cbox}\ a\ b) = \{\}$
unfolding *content_eq_0 interior_cbox box_eq_empty* **by** *auto*

lemma *content_pos_lt_eq*: $0 < \mathit{content}\ (\mathit{cbox}\ a\ (b::'a::\mathit{euclidean\_space})) \longleftrightarrow (\forall\, i{\in}\mathit{Basis}.\ a{\cdot}i < b{\cdot}i)$
by (*auto simp add*: *content_cbox_cases less_le prod_nonneg*)

lemma *content_empty* [*simp*]: $\mathit{content}\ \{\} = 0$
by *simp*

lemma *content_real_if* [*simp*]: $\mathit{content}\ \{a..b\} = (\mathit{if}\ a \le b\ \mathit{then}\ b - a\ \mathit{else}\ 0)$
by (*simp add*: *content_real*)

lemma *content_subset*: $\mathit{cbox}\ a\ b \subseteq \mathit{cbox}\ c\ d \Longrightarrow \mathit{content}\ (\mathit{cbox}\ a\ b) \le \mathit{content}\ (\mathit{cbox}\ c\ d)$
unfolding *measure_def*
by (*intro enn2real_mono emeasure_mono*) (*auto simp*: *emeasure_lborel_cbox_eq*)

lemma *content_lt_nz*: $0 < \mathit{content}\ (\mathit{cbox}\ a\ b) \longleftrightarrow \mathit{content}\ (\mathit{cbox}\ a\ b) \neq 0$
by (*fact zero_less_measure_iff*)

lemma *content_Pair*: $\mathit{content}\ (\mathit{cbox}\ (a,c)\ (b,d)) = \mathit{content}\ (\mathit{cbox}\ a\ b) * \mathit{content}\ (\mathit{cbox}\ c\ d)$
unfolding *measure_lborel_cbox_eq Basis_prod_def*
apply (*subst prod.union_disjoint*)

apply (*auto simp*: *bex_Un ball_Un*)
apply (*subst* (*1 2*) *prod.reindex_nontrivial*)
apply *auto*
done

lemma *content_cbox_pair_eq0_D*:
content (*cbox* (*a*,*c*) (*b*,*d*)) = *0* ⟹ *content* (*cbox a b*) = *0* ∨ *content* (*cbox c d*) = *0*
by (*simp add*: *content_Pair*)

lemma *content_cbox_plus*:
fixes *x* :: *'a*::*euclidean_space*
shows *content*(*cbox x* (*x* + *h* ∗$_R$ *One*)) = (*if h* ≥ *0 then h* ^ *DIM*(*'a*) *else 0*)
by (*simp add*: *algebra_simps content_cbox_if box_eq_empty*)

lemma *content_0_subset*: *content*(*cbox a b*) = *0* ⟹ *s* ⊆ *cbox a b* ⟹ *content s* = *0*
using *emeasure_mono*[*of s cbox a b lborel*]
by (*auto simp*: *measure_def enn2real_eq_0_iff emeasure_lborel_cbox_eq*)

lemma *content_ball_pos*:
assumes *r* > *0*
shows *content* (*ball c r*) > *0*
proof −
from *rational_boxes*[*OF assms*, *of c*] **obtain** *a b* **where** *ab*: *c* ∈ *box a b box a b* ⊆ *ball c r*
by *auto*
then have *0* < *content* (*box a b*)
by (*subst measure_lborel_box_eq*) (*auto intro*!: *prod_pos simp*: *algebra_simps box_def*)
have *emeasure lborel* (*box a b*) ≤ *emeasure lborel* (*ball c r*)
using *ab* **by** (*intro emeasure_mono*) *auto*
then show *?thesis*
using ‹*content* (*box a b*) > *0*›
by (*smt* (*verit*, *best*) *Sigma_Algebra.measure_def emeasure_lborel_ball_finite enn2real_mono infinity_ennreal_def*)
qed

lemma *content_cball_pos*:
assumes *r* > *0*
shows *content* (*cball c r*) > *0*
proof −
from *rational_boxes*[*OF assms*, *of c*] **obtain** *a b* **where** *ab*: *c* ∈ *box a b box a b* ⊆ *ball c r*
by *auto*
then have *0* < *content* (*box a b*)
by (*subst measure_lborel_box_eq*) (*auto intro*!: *prod_pos simp*: *algebra_simps box_def*)
have *emeasure lborel* (*box a b*) ≤ *emeasure lborel* (*cball c r*)

```
    using ab by (intro emeasure_mono) auto
  also have emeasure lborel (box a b) = ennreal (content (box a b))
   using emeasure_lborel_box_finite[of a b] by (intro emeasure_eq_ennreal_measure)
auto
  also have emeasure lborel (cball c r) = ennreal (content (cball c r))
   using emeasure_lborel_cball_finite[of c r] by (intro emeasure_eq_ennreal_measure)
auto
  finally show ?thesis
    using ‹content (box a b) > 0› by simp
qed

lemma content_split:
  fixes a :: 'a::euclidean_space
  assumes k ∈ Basis
  shows content (cbox a b) = content(cbox a b ∩ {x. x•k ≤ c}) + content(cbox a
b ∩ {x. x•k ≥ c})
  — Prove using measure theory
proof (cases ∀ i∈Basis. a • i ≤ b • i)
  case True
   have 1: ⋀X Y Z. (∏ i∈Basis. Z i (if i = k then X else Y i)) = Z k X *
(∏ i∈Basis−{k}. Z i (Y i))
    by (simp add: if_distrib prod.delta_remove assms)
  note simps = interval_split[OF assms] content_cbox_cases
  have 2: (∏ i∈Basis. b•i − a•i) = (∏ i∈Basis−{k}. b•i − a•i) * (b•k − a•k)
    by (metis (no_types, lifting) assms finite_Basis mult.commute prod.remove)
  have ⋀x. min (b • k) c = max (a • k) c ⟹
    x * (b•k − a•k) = x * (max (a • k) c − a • k) + x * (b • k − max (a • k) c)
    by  (auto simp add: field_simps)
  moreover
  have **: (∏ i∈Basis. ((∑ i∈Basis. (if i = k then min (b • k) c else b • i) *R i)
• i − a • i)) =
      (∏ i∈Basis. (if i = k then min (b • k) c else b • i) − a • i)
    (∏ i∈Basis. b • i − ((∑ i∈Basis. (if i = k then max (a • k) c else a • i) *R i)
• i)) =
      (∏ i∈Basis. b • i − (if i = k then max (a • k) c else a • i))
    by (auto intro!: prod.cong)
  have ¬ a • k ≤ c ⟹ ¬ c ≤ b • k ⟹ False
    unfolding not_le using True assms by auto
  ultimately show ?thesis
    using assms unfolding simps ** 1[of λi x. b•i − x] 1[of λi x. x − a•i] 2
    by auto
next
  case False
  then have cbox a b = {}
    unfolding box_eq_empty by (auto simp: not_le)
  then show ?thesis
    by (auto simp: not_le)
qed
```

lemma *division_of_content_0*:
assumes *content* (*cbox a b*) = *0 d division_of* (*cbox a b*) *K* ∈ *d*
shows *content K* = *0*
unfolding *forall_in_division*[*OF assms*(*2*)]
by (*meson assms content_0_subset division_of_def*)

lemma *sum_content_null*:
assumes *content* (*cbox a b*) = *0*
and *p tagged_division_of* (*cbox a b*)
shows (∑(*x*,*K*)∈*p*. *content K* ∗$_R$ *f x*) = (*0*::′*a*::*real_normed_vector*)
proof (*intro sum.neutral strip*)
fix *y*
assume *y*: *y* ∈ *p*
obtain *x K* **where** *xk*: *y* = (*x*, *K*)
using *surj_pair*[*of y*] **by** *blast*
then obtain *c d* **where** *k*: *K* = *cbox c d K* ⊆ *cbox a b*
by (*metis assms*(*2*) *tagged_division_ofD*(*3*) *tagged_division_ofD*(*4*) *y*)
have (λ(*x*′,*K*′). *content K*′ ∗$_R$ *f x*′) *y* = *content K* ∗$_R$ *f x*
unfolding *xk* **by** *auto*
also have ... = *0*
using *assms*(*1*) *content_0_subset k*(*2*) **by** *auto*
finally show (λ(*x*, *k*). *content k* ∗$_R$ *f x*) *y* = *0* **.**
qed

global_interpretation *sum_content*: *operative plus 0 content*
rewrites *comm_monoid_set.F plus 0* = *sum*
proof −
interpret *operative plus 0 content*
by *standard* (*auto simp add*: *content_split* [*symmetric*] *content_eq_0_interior*)
show *operative plus 0 content*
by *standard*
show *comm_monoid_set.F plus 0* = *sum*
by (*simp add*: *sum_def*)
qed

lemma *additive_content_division*: *d division_of* (*cbox a b*) ⟹ *sum content d* = *content* (*cbox a b*)
by (*fact sum_content.division*)

lemma *additive_content_tagged_division*:
d tagged_division_of (*cbox a b*) ⟹ *sum* (λ(*x*,*l*). *content l*) *d* = *content* (*cbox a b*)
by (*fact sum_content.tagged_division*)

lemma *subadditive_content_division*:
assumes *D division_of S S* ⊆ *cbox a b*
shows *sum content D* ≤ *content*(*cbox a b*)
proof −
have *D division_of* ⋃*D* ⋃*D* ⊆ *cbox a b*

using *assms* **by** *auto*
then obtain $\mathcal{D}'$ **where** $\mathcal{D} \subseteq \mathcal{D}'$ $\mathcal{D}'$ *division_of cbox a b*
using *partial_division_extend_interval* **by** *metis*
then have *sum content* $\mathcal{D} \leq$ *sum content* $\mathcal{D}'$
using *sum_mono2* **by** *blast*
also have ... $\leq$ *content*(*cbox a b*)
by (*simp add*: ‹$\mathcal{D}'$ *division_of cbox a b*› *additive_content_division less_eq_real_def*)
finally show *?thesis* **.**
qed

lemma *content_real_eq_0*: *content* {*a*..*b*::*real*} = *0* $\longleftrightarrow$ $a \geq b$
by *simp*

lemma *property_empty_interval*: $\forall$ *a b*. *content* (*cbox a b*) = *0* $\longrightarrow$ *P* (*cbox a b*) $\Longrightarrow$ *P* {}
using *content_empty* **unfolding** *empty_as_interval* **by** *auto*

lemma *interval_bounds_nz_content* [*simp*]:
assumes *content* (*cbox a b*) $\neq$ *0*
shows *interval_upperbound* (*cbox a b*) = *b*
and *interval_lowerbound* (*cbox a b*) = *a*
by (*metis assms content_empty interval_bounds'*)+

6.14.2 Gauge integral

Case distinction to define it first on compact intervals first, then use a limit. This is only much later unified. In Fremlin: Measure Theory, Volume 4I this is generalized using residual sets.

definition *has_integral* :: ('*n*::*euclidean_space* $\Rightarrow$ '*b*::*real_normed_vector*) $\Rightarrow$ '*b* $\Rightarrow$ '*n set* $\Rightarrow$ *bool*
(**infixr** *has'_integral 46*)
where (*f has_integral I*) *s* $\longleftrightarrow$
(*if* $\exists$ *a b*. *s* = *cbox a b*
then (λ*p*. $\sum$ (*x*,*k*)$\in$*p*. *content k* $*_R$ *f x*) $\longrightarrow$ *I*) (*division_filter s*)
else ($\forall$ *e*>*0*. $\exists$ *B*>*0*. $\forall$ *a b*. *ball 0 B* $\subseteq$ *cbox a b* $\longrightarrow$
($\exists$ *z*. (λ*p*. $\sum$ (*x*,*k*)$\in$*p*. *content k* $*_R$ (*if x* $\in$ *s then f x else 0*)) $\longrightarrow$ *z*) (*division_filter* (*cbox a b*)) $\wedge$
norm (*z* $-$ *I*) < *e*)))

lemma *has_integral_cbox*:
(*f has_integral I*) (*cbox a b*) $\longleftrightarrow$ ((λ*p*. $\sum$ (*x*,*k*)$\in$*p*. *content k* $*_R$ *f x*) $\longrightarrow$ *I*) (*division_filter* (*cbox a b*))
by (*auto simp add*: *has_integral_def*)

lemma *has_integral*:
(*f has_integral y*) (*cbox a b*) $\longleftrightarrow$
($\forall$ *e*>*0*. $\exists\gamma$. *gauge* γ $\wedge$
($\forall\mathcal{D}$. $\mathcal{D}$ *tagged_division_of* (*cbox a b*) $\wedge$ γ *fine* $\mathcal{D}$ $\longrightarrow$
norm (*sum* (λ(*x*,*k*). *content*(*k*) $*_R$ *f x*) $\mathcal{D}$ $-$ *y*) < *e*))

by (*auto simp*: *dist_norm eventually_division_filter has_integral_def tendsto_iff*)

lemma *has_integral_real*:
(*f has_integral y*) {*a..b*::*real*} ⟷
(∀ *e*>*0*. ∃ *γ*. *gauge γ* ∧
(∀ *𝒟*. *𝒟 tagged_division_of* {*a..b*} ∧ *γ fine 𝒟* ⟶
norm (*sum* (λ(*x*,*k*). *content*(*k*) *∗*$_R$ *f x*) *𝒟* − *y*) < *e*))
unfolding *box_real*[*symmetric*] **by** (*rule has_integral*)

lemma *has_integralD*[*dest*]:
assumes (*f has_integral y*) (*cbox a b*)
and *e* > *0*
obtains *γ*
where *gauge γ*
and ⋀*𝒟*. *𝒟 tagged_division_of* (*cbox a b*) ⟹ *γ fine 𝒟* ⟹
norm ((∑ (*x*,*k*)∈*𝒟*. *content k ∗*$_R$ *f x*) − *y*) < *e*
using *assms* **unfolding** *has_integral* **by** *auto*

lemma *has_integral_alt*:
(*f has_integral y*) *i* ⟷
(*if* ∃ *a b*. *i* = *cbox a b*
then (*f has_integral y*) *i*
else (∀ *e*>*0*. ∃ *B*>*0*. ∀ *a b*. *ball 0 B* ⊆ *cbox a b* ⟶
(∃ *z*. ((λ*x*. *if x* ∈ *i then f x else 0*) *has_integral z*) (*cbox a b*) ∧ *norm* (*z* − *y*)
< *e*)))
by (*subst has_integral_def*) (*auto simp add*: *has_integral_cbox*)

lemma *has_integral_altD*:
assumes (*f has_integral y*) *i*
and ¬ (∃ *a b*. *i* = *cbox a b*)
and *e*>*0*
obtains *B* **where** *B* > *0*
and ∀ *a b*. *ball 0 B* ⊆ *cbox a b* ⟶
(∃ *z*. ((λ*x*. *if x* ∈ *i then f*(*x*) *else 0*) *has_integral z*) (*cbox a b*) ∧ *norm*(*z* − *y*)
< *e*)
using *assms has_integral_alt*[*of f y i*] **by** *auto*

definition *integrable_on* (**infixr** *integrable′_on 46*)
where *f integrable_on i* ⟷ (∃ *y*. (*f has_integral y*) *i*)

definition *integral i f* = (*SOME y*. (*f has_integral y*) *i* ∨ ¬ *f integrable_on i* ∧
y=*0*)

lemma *integrable_integral*[*intro*]: *f integrable_on i* ⟹ (*f has_integral* (*integral i*
f)) *i*
unfolding *integrable_on_def integral_def* **by** (*metis* (*mono_tags*, *lifting*))

lemma *not_integrable_integral*: ¬ *f integrable_on i* ⟹ *integral i f* = *0*
unfolding *integrable_on_def integral_def* **by** *blast*

lemma *has_integral_integrable*[*dest*]: (*f has_integral i*) *s* ⟹ *f integrable_on s*
unfolding *integrable_on_def* **by** *auto*

lemma *has_integral_integral*: *f integrable_on s* ⟷ (*f has_integral* (*integral s f*)) *s*
by *auto*

6.14.3 Basic theorems about integrals

lemma *has_integral_eq_rhs*: (*f has_integral j*) *S* ⟹ *i* = *j* ⟹ (*f has_integral i*) *S*
by (*rule forw_subst*)

lemma *has_integral_unique_cbox*:
fixes *f* :: *'n*::*euclidean_space* ⇒ *'a*::*real_normed_vector*
shows (*f has_integral k1*) (*cbox a b*) ⟹ (*f has_integral k2*) (*cbox a b*) ⟹ *k1* = *k2*
by (*meson division_filter_not_empty has_integral_cbox tendsto_unique*)

lemma *has_integral_unique*:
fixes *f* :: *'n*::*euclidean_space* ⇒ *'a*::*real_normed_vector*
assumes (*f has_integral k1*) *i* (*f has_integral k2*) *i*
shows *k1* = *k2*
proof (*rule ccontr*)
let *?e* = *norm* (*k1* − *k2*)/*2*
let *?F* = (λ*x*. *if x* ∈ *i then f x else* 0)
assume *k1* ≠ *k2*
then have *e*: *?e* > 0
by *auto*
have *nonbox*: ¬ (∃ *a b*. *i* = *cbox a b*)
using ‹*k1* ≠ *k2*› *assms has_integral_unique_cbox* **by** *blast*
obtain *B1* **where** *B1*:
0 < *B1*
⋀*a b*. *ball 0 B1* ⊆ *cbox a b* ⟹
∃ *z*. (*?F has_integral z*) (*cbox a b*) ∧ *norm* (*z* − *k1*) < *norm* (*k1* − *k2*)/*2*
by (*rule has_integral_altD*[*OF assms*(*1*) *nonbox*,*OF e*]) *blast*
obtain *B2* **where** *B2*:
0 < *B2*
⋀*a b*. *ball 0 B2* ⊆ *cbox a b* ⟹
∃ *z*. (*?F has_integral z*) (*cbox a b*) ∧ *norm* (*z* − *k2*) < *norm* (*k1* − *k2*)/*2*
by (*rule has_integral_altD*[*OF assms*(*2*) *nonbox*,*OF e*]) *blast*
obtain *a b* :: *'n* **where** *ab*: *ball 0 B1* ⊆ *cbox a b ball 0 B2* ⊆ *cbox a b*
by (*metis Un_subset_iff bounded_Un bounded_ball bounded_subset_cbox_symmetric*)
obtain *w* **where** *w*: (*?F has_integral w*) (*cbox a b*) *norm* (*w* − *k1*) < *norm* (*k1* − *k2*)/*2*
using *B1*(*2*)[*OF ab*(*1*)] **by** *blast*
obtain *z* **where** *z*: (*?F has_integral z*) (*cbox a b*) *norm* (*z* − *k2*) < *norm* (*k1* − *k2*)/*2*

using *B2(2)*[*OF ab(2)*] **by** *blast*
have $z = w$
using *has_integral_unique_cbox*[*OF w(1) z(1)*] **by** *auto*
then have *norm* (*k1* − *k2*) ≤ *norm* (*z* − *k2*) + *norm* (*w* − *k1*)
using *norm_triangle_ineq4* [*of k1* − *w k2* − *z*]
by (*auto simp add*: *norm_minus_commute*)
also have ... < *norm* (*k1* − *k2*)/*2* + *norm* (*k1* − *k2*)/*2*
by (*metis add_strict_mono z(2) w(2)*)
finally show *False* **by** *auto*
qed

lemma *integral_unique* [*intro*]: (*f has_integral y*) *k* ⟹ *integral k f* = *y*
unfolding *integral_def*
by (*rule some_equality*) (*auto intro*: *has_integral_unique*)

lemma *has_integral_iff*: (*f has_integral i*) *S* ⟷ (*f integrable_on S* ∧ *integral S f* = *i*)
by *blast*

lemma *eq_integralD*: *integral k f* = *y* ⟹ (*f has_integral y*) *k* ∨ ¬ *f integrable_on k* ∧ *y*=*0*
unfolding *integral_def integrable_on_def*
by (*metis* (*mono_tags*, *lifting*))

lemma *has_integral_const* [*intro*]:
fixes *a b* :: *'a*::*euclidean_space*
shows ((λ*x*. *c*) *has_integral* (*content* (*cbox a b*) ∗$_R$ *c*)) (*cbox a b*)
using *eventually_division_filter_tagged_division*[*of cbox a b*]
additive_content_tagged_division[*of* _ *a b*]
by (*auto simp*: *has_integral_cbox split_beta' scaleR_sum_left*[*symmetric*]
elim!: *eventually_mono intro*!: *tendsto_cong*[*THEN iffD1*, *OF* _ *tendsto_const*])

lemma *has_integral_const_real* [*intro*]:
fixes *a b* :: *real*
shows ((λ*x*. *c*) *has_integral* (*content* {*a*..*b*} ∗$_R$ *c*)) {*a*..*b*}
by (*metis box_real*(*2*) *has_integral_const*)

lemma *has_integral_integrable_integral*: (*f has_integral i*) *s* ⟷ *f integrable_on s* ∧ *integral s f* = *i*
by *blast*

lemma *integral_const* [*simp*]:
fixes *a b* :: *'a*::*euclidean_space*
shows *integral* (*cbox a b*) (λ*x*. *c*) = *content* (*cbox a b*) ∗$_R$ *c*
by *blast*

lemma *integral_const_real* [*simp*]:
fixes *a b* :: *real*

```
  shows integral {a..b} (λx. c) = content {a..b} *R c
  by blast

lemma has_integral_is_0_cbox:
  fixes f :: 'n::euclidean_space ⇒ 'a::real_normed_vector
  assumes ⋀x. x ∈ cbox a b ⟹ f x = 0
  shows (f has_integral 0) (cbox a b)
    unfolding has_integral_cbox
    using eventually_division_filter_tagged_division[of cbox a b] assms
    by (subst tendsto_cong[where g=λ_. 0])
       (auto elim!: eventually_mono intro!: sum.neutral simp: tag_in_interval)

lemma has_integral_is_0:
  fixes f :: 'n::euclidean_space ⇒ 'a::real_normed_vector
  assumes ⋀x. x ∈ S ⟹ f x = 0
  shows (f has_integral 0) S
proof (cases (∃ a b. S = cbox a b))
  case True with assms has_integral_is_0_cbox show ?thesis
    by blast
next
  case False
  have *: (λx. if x ∈ S then f x else 0) = (λx. 0)
    by (auto simp add: assms)
  show ?thesis
    using has_integral_is_0_cbox False
    by (subst has_integral_alt) (force simp add: *)
qed

lemma has_integral_0[simp]: ((λx::'n::euclidean_space. 0) has_integral 0) S
  by (rule has_integral_is_0) auto

lemma has_integral_0_eq[simp]: ((λx. 0) has_integral i) S ⟷ i = 0
  using has_integral_unique[OF has_integral_0] by auto

lemma has_integral_linear_cbox:
  fixes f :: 'n::euclidean_space ⇒ 'a::real_normed_vector
  assumes f: (f has_integral y) (cbox a b)
    and h: bounded_linear h
  shows ((h ∘ f) has_integral (h y)) (cbox a b)
proof −
  interpret bounded_linear h using h .
  show ?thesis
   unfolding has_integral_cbox using tendsto [OF f [unfolded has_integral_cbox]]
    by (simp add: sum scaleR split_beta')
qed

lemma has_integral_linear:
  fixes f :: 'n::euclidean_space ⇒ 'a::real_normed_vector
  assumes f: (f has_integral y) S
```

```
    and h: bounded_linear h
  shows ((h ∘ f) has_integral (h y)) S
proof (cases (∃ a b. S = cbox a b))
  case True with f h has_integral_linear_cbox show ?thesis
    by blast
next
  case False
  interpret bounded_linear h using h .
  from pos_bounded obtain B where B: 0 < B ⋀x. norm (h x) ≤ norm x * B
    by blast
  let ?S = λf x. if x ∈ S then f x else 0
  show ?thesis
  proof (subst has_integral_alt, clarsimp simp: False)
    fix e :: real
    assume e: e > 0
    have *: 0 < e/B using e B(1) by simp
    obtain M where M:
      M > 0
      ⋀a b. ball 0 M ⊆ cbox a b ⟹
        ∃ z. (?S f has_integral z) (cbox a b) ∧ norm (z − y) < e/B
      using has_integral_altD[OF f False *] by blast
    show ∃ B>0. ∀ a b. ball 0 B ⊆ cbox a b ⟶
      (∃ z. (?S(h ∘ f) has_integral z) (cbox a b) ∧ norm (z − h y) < e)
    proof (rule exI, intro allI conjI impI)
      show M > 0 using M by metis
    next
      fix a b::'n
      assume sb: ball 0 M ⊆ cbox a b
      obtain z where z: (?S f has_integral z) (cbox a b) norm (z − y) < e/B
        using M(2)[OF sb] by blast
      have *: ?S(h ∘ f) = h ∘ ?S f
        using zero by auto
      show ∃ z. (?S(h ∘ f) has_integral z) (cbox a b) ∧ norm (z − h y) < e
      proof (intro exI conjI)
        show (?S(h ∘ f) has_integral h z) (cbox a b)
          by (simp add: * has_integral_linear_cbox[OF z(1) h])
        show norm (h z − h y) < e
          by (metis B diff le_less_trans pos_less_divide_eq z(2))
      qed
    qed
  qed
qed

lemma has_integral_scaleR_left:
  (f has_integral y) S ⟹ ((λx. f x *R c) has_integral (y *R c)) S
   using has_integral_linear[OF _ bounded_linear_scaleR_left] by (simp add:
comp_def)

lemma integrable_on_scaleR_left:
```

assumes *f integrable_on A*
shows $(\lambda x.\ f\ x *_R y)$ *integrable_on A*
using *assms has_integral_scaleR_left* **unfolding** *integrable_on_def* **by** *blast*

lemma *has_integral_mult_left*:
fixes *c* :: *_* :: *real_normed_algebra*
shows *(f has_integral y) S* $\Longrightarrow$ $((\lambda x.\ f\ x * c)$ *has_integral* $(y * c))$ *S*
using *has_integral_linear*[*OF _ bounded_linear_mult_left*] **by** (*simp add*: *comp_def*)

lemma *integrable_on_mult_left*:
fixes *c* :: *'a* :: *real_normed_algebra*
assumes *f integrable_on A*
shows $(\lambda x.\ f\ x * c)$ *integrable_on A*
using *assms has_integral_mult_left* **by** *blast*

lemma *has_integral_divide*:
fixes *c* :: *_* :: *real_normed_div_algebra*
shows *(f has_integral y) S* $\Longrightarrow$ $((\lambda x.\ f\ x\ /\ c)$ *has_integral* $(y\ /\ c))$ *S*
unfolding *divide_inverse* **by** (*simp add*: *has_integral_mult_left*)

lemma *integrable_on_divide*:
fixes *c* :: *'a* :: *real_normed_div_algebra*
assumes *f integrable_on A*
shows $(\lambda x.\ f\ x\ /\ c)$ *integrable_on A*
using *assms has_integral_divide* **by** *blast*

The case analysis eliminates the condition *f integrable_on S* at the cost of the type class constraint *division_ring*

corollary *integral_mult_left* [*simp*]:
fixes *c*:: *'a*::{*real_normed_algebra*,*division_ring*}
shows *integral S* $(\lambda x.\ f\ x * c)$ = *integral S f* $*$ *c*
proof (*cases f integrable_on S* $\vee$ *c* = *0*)
case *True* **then show** *?thesis*
by (*force intro*: *has_integral_mult_left*)
next
case *False* **then have** $\neg$ $(\lambda x.\ f\ x * c)$ *integrable_on S*
using *has_integral_mult_left* [*of* $(\lambda x.\ f\ x * c)$ *_ S inverse c*]
by (*auto simp add*: *mult.assoc*)
with *False* **show** *?thesis* **by** (*simp add*: *not_integrable_integral*)
qed

corollary *integral_mult_right* [*simp*]:
fixes *c*:: *'a*::{*real_normed_field*}
shows *integral S* $(\lambda x.\ c * f\ x)$ = *c* $*$ *integral S f*
by (*simp add*: *mult.commute* [*of c*])

corollary *integral_divide* [*simp*]:
fixes *z* :: *'a*::*real_normed_field*
shows *integral S* $(\lambda x.\ f\ x\ /\ z)$ = *integral S* $(\lambda x.\ f\ x)$ / *z*

using *integral_mult_left* [*of S f inverse z*]
by (*simp add*: *divide_inverse_commute*)

lemma *has_integral_mult_right*:
fixes *c* :: *'a* :: *real_normed_algebra*
shows (*f has_integral y*) *A* ⟹ ((λ*x*. *c* ∗ *f x*) *has_integral* (*c* ∗ *y*)) *A*
using *has_integral_linear*[*OF _ bounded_linear_mult_right*] **by** (*simp add*: *comp_def*)

lemma *integrable_on_mult_right*:
fixes *c* :: *'a* :: *real_normed_algebra*
assumes *f integrable_on A*
shows (λ*x*. *c* ∗ *f x*) *integrable_on A*
using *assms has_integral_mult_right* **by** *blast*

lemma *integrable_on_mult_right_iff* [*simp*]:
fixes *c* :: *'a* :: *real_normed_field*
assumes *c* ≠ *0*
shows (λ*x*. *c* ∗ *f x*) *integrable_on A* ⟷ *f integrable_on A*
using *integrable_on_mult_right*[*of f A c*]
integrable_on_mult_right[*of* λ*x*. *c* ∗ *f x A inverse c*] *assms*
by (*auto simp*: *field_simps*)

lemma *integrable_on_mult_left_iff* [*simp*]:
fixes *c* :: *'a* :: *real_normed_field*
assumes *c* ≠ *0*
shows (λ*x*. *f x* ∗ *c*) *integrable_on A* ⟷ *f integrable_on A*
using *integrable_on_mult_right_iff*[*OF assms*, *of f A*] **by** (*simp add*: *mult.commute*)

lemma *integrable_on_div_iff* [*simp*]:
fixes *c* :: *'a* :: *real_normed_field*
assumes *c* ≠ *0*
shows (λ*x*. *f x* / *c*) *integrable_on A* ⟷ *f integrable_on A*
using *integrable_on_mult_right_iff*[*of inverse c f A*] *assms* **by** (*simp add*: *field_simps*)

lemma *has_integral_cmul*: (*f has_integral k*) *S* ⟹ ((λ*x*. *c* ∗*R* *f x*) *has_integral* (*c* ∗*R* *k*)) *S*
unfolding *o_def*[*symmetric*]
by (*metis has_integral_linear bounded_linear_scaleR_right*)

lemma *has_integral_cmult_real*:
fixes *c* :: *real*
assumes *c* ≠ *0* ⟹ (*f has_integral x*) *A*
shows ((λ*x*. *c* ∗ *f x*) *has_integral c* ∗ *x*) *A*
by (*metis assms has_integral_is_0 has_integral_mult_right lambda_zero*)

lemma *has_integral_neg*: (*f has_integral k*) *S* ⟹ ((λ*x*. −(*f x*)) *has_integral* −*k*) *S*

```
  by (drule_tac c=-1 in has_integral_cmul) auto

lemma has_integral_neg_iff: ((λx. - f x) has_integral k) S ⟷ (f has_integral
- k) S
  using has_integral_neg[of f - k] has_integral_neg[of λx. - f x k] by auto

lemma has_integral_add_cbox:
  fixes f :: 'n::euclidean_space ⇒ 'a::real_normed_vector
  assumes (f has_integral k) (cbox a b) (g has_integral l) (cbox a b)
  shows ((λx. f x + g x) has_integral (k + l)) (cbox a b)
  using assms
    unfolding has_integral_cbox
   by (simp add: split_beta' scaleR_add_right sum.distrib[abs_def] tendsto_add)

lemma has_integral_add:
  fixes f :: 'n::euclidean_space ⇒ 'a::real_normed_vector
  assumes f: (f has_integral k) S and g: (g has_integral l) S
  shows ((λx. f x + g x) has_integral (k + l)) S
proof (cases ∃ a b. S = cbox a b)
  case True with has_integral_add_cbox assms show ?thesis
    by blast
next
  let ?S = λf x. if x ∈ S then f x else 0
  case False
  then show ?thesis
  proof (subst has_integral_alt, clarsimp, goal_cases)
    case (1 e)
    then have e2: e/2 > 0
      by auto
    obtain Bf where 0 < Bf
      and Bf: ⋀a b. ball 0 Bf ⊆ cbox a b ⟹
                    ∃ z. (?S f has_integral z) (cbox a b) ∧ norm (z - k) < e/2
      using has_integral_altD[OF f False e2] by blast
    obtain Bg where 0 < Bg
      and Bg: ⋀a b. ball 0 Bg ⊆ (cbox a b) ⟹
                    ∃ z. (?S g has_integral z) (cbox a b) ∧ norm (z - l) < e/2
      using has_integral_altD[OF g False e2] by blast
    show ?case
   proof (rule_tac x=max Bf Bg in exI, clarsimp simp add: max.strict_coboundedI1
‹0 < Bf›)
      fix a b
      assume ball 0 (max Bf Bg) ⊆ cbox a (b::'n)
      then have fs: ball 0 Bf ⊆ cbox a (b::'n) and gs: ball 0 Bg ⊆ cbox a (b::'n)
        by auto
      obtain w where w: (?S f has_integral w) (cbox a b) norm (w - k) < e/2
        using Bf[OF fs] by blast
      obtain z where z: (?S g has_integral z) (cbox a b) norm (z - l) < e/2
        using Bg[OF gs] by blast
      have ∗: ⋀x. (if x ∈ S then f x + g x else 0) = (?S f x) + (?S g x)
```

```
      by auto
     show ∃z. (?S(λx. f x + g x) has_integral z) (cbox a b) ∧ norm (z − (k +
l)) < e
     proof (intro exI conjI)
      show (?S(λx. f x + g x) has_integral (w + z)) (cbox a b)
      by (simp add: has_integral_add_cbox[OF w(1) z(1), unfolded ∗[symmetric]])
      show norm (w + z − (k + l)) < e
       by (metis dist_norm dist_triangle_add_half w(2) z(2))
     qed
    qed
  qed
qed

lemma has_integral_diff:
  (f has_integral k) S ⟹ (g has_integral l) S ⟹
    ((λx. f x − g x) has_integral (k − l)) S
  using has_integral_add[OF _ has_integral_neg, of f k S g l]
  by (auto simp: algebra_simps)

lemma integral_0 [simp]:
  integral S (λx::'n::euclidean_space. 0::'m::real_normed_vector) = 0
  by auto

lemma integral_add: f integrable_on S ⟹ g integrable_on S ⟹
    integral S (λx. f x + g x) = integral S f + integral S g
  by (rule integral_unique) (metis integrable_integral has_integral_add)

lemma integral_cmul [simp]: integral S (λx. c ∗R f x) = c ∗R integral S f
proof (cases f integrable_on S ∨ c = 0)
  case True with has_integral_cmul integrable_integral show ?thesis
    by fastforce
next
  case False then have ¬ (λx. c ∗R f x) integrable_on S
    using has_integral_cmul [of (λx. c ∗R f x) _ S inverse c] by auto
  with False show ?thesis by (simp add: not_integrable_integral)
qed

lemma integral_mult:
  fixes K::real
  shows f integrable_on X ⟹ K ∗ integral X f = integral X (λx. K ∗ f x)
  by simp

lemma integral_neg [simp]: integral S (λx. − f x) = − integral S f
 by (metis eq_integralD equation_minus_iff has_integral_iff has_integral_neg_iff
neg_equal_0_iff_equal)

lemma integral_diff: f integrable_on S ⟹ g integrable_on S ⟹
    integral S (λx. f x − g x) = integral S f − integral S g
  by (rule integral_unique) (metis integrable_integral has_integral_diff)
```

lemma *integrable_0*: $(\lambda x.\ 0)$ *integrable_on S*
unfolding *integrable_on_def* **using** *has_integral_0* **by** *auto*

lemma *integrable_add*: *f integrable_on S* $\Longrightarrow$ *g integrable_on S* $\Longrightarrow$ $(\lambda x.\ f\ x + g\ x)$ *integrable_on S*
unfolding *integrable_on_def* **by**(*auto intro*: *has_integral_add*)

lemma *integrable_cmul*: *f integrable_on S* $\Longrightarrow$ $(\lambda x.\ c *_R f(x))$ *integrable_on S*
unfolding *integrable_on_def* **by**(*auto intro*: *has_integral_cmul*)

lemma *integrable_on_scaleR_iff* [*simp*]:
fixes *c* :: *real*
assumes $c \neq 0$
shows $(\lambda x.\ c *_R f\ x)$ *integrable_on S* $\longleftrightarrow$ *f integrable_on S*
using *integrable_cmul*[*of* $\lambda x.\ c *_R f\ x$ *S 1 / c*] *integrable_cmul*[*of f S c*] ‹$c \neq 0$›
by *auto*

lemma *integrable_on_cmult_iff* [*simp*]:
fixes *c* :: *real*
assumes $c \neq 0$
shows $(\lambda x.\ c * f\ x)$ *integrable_on S* $\longleftrightarrow$ *f integrable_on S*
using *integrable_on_scaleR_iff* [*of c f*] *assms* **by** *simp*

lemma *integrable_on_cmult_left*:
assumes *f integrable_on S*
shows $(\lambda x.\ of\_real\ c * f\ x)$ *integrable_on S*
using *integrable_cmul*[*of f S of_real c*] *assms*
by (*simp add*: *scaleR_conv_of_real*)

lemma *integrable_neg*: *f integrable_on S* $\Longrightarrow$ $(\lambda x.\ -f(x))$ *integrable_on S*
unfolding *integrable_on_def* **by**(*auto intro*: *has_integral_neg*)

lemma *integrable_neg_iff*: $(\lambda x.\ -f(x))$ *integrable_on S* $\longleftrightarrow$ *f integrable_on S*
using *integrable_neg* **by** *fastforce*

lemma *integrable_diff*:
f integrable_on S $\Longrightarrow$ *g integrable_on S* $\Longrightarrow$ $(\lambda x.\ f\ x - g\ x)$ *integrable_on S*
unfolding *integrable_on_def* **by**(*auto intro*: *has_integral_diff*)

lemma *integrable_linear*:
f integrable_on S $\Longrightarrow$ *bounded_linear h* $\Longrightarrow$ $(h \circ f)$ *integrable_on S*
unfolding *integrable_on_def* **by**(*auto intro*: *has_integral_linear*)

lemma *integral_linear*:
f integrable_on S $\Longrightarrow$ *bounded_linear h* $\Longrightarrow$ *integral S* $(h \circ f) = h$ (*integral S f*)
by (*meson has_integral_iff has_integral_linear*)

lemma *integrable_on_cnj_iff*:

(λx. cnj (f x)) integrable_on A ⟷ f integrable_on A
using integrable_linear[OF _ bounded_linear_cnj, of f A]
integrable_linear[OF _ bounded_linear_cnj, of cnj ∘ f A]
by (auto simp: o_def)

lemma integral_cnj: cnj (integral A f) = integral A (λx. cnj (f x))
by (cases f integrable_on A)
(simp_all add: integral_linear[OF _ bounded_linear_cnj, symmetric]
o_def integrable_on_cnj_iff not_integrable_integral)

lemma integral_component_eq[simp]:
fixes f :: 'n::euclidean_space ⇒ 'm::euclidean_space
assumes f integrable_on S
shows integral S (λx. f x • k) = integral S f • k
unfolding integral_linear[OF assms(1) bounded_linear_inner_left,unfolded o_def]
..

lemma has_integral_sum:
assumes finite T
and ⋀a. a ∈ T ⟹ ((f a) has_integral (i a)) S
shows ((λx. sum (λa. f a x) T) has_integral (sum i T)) S
using ‹finite T› subset_refl[of T]
by (induct rule: finite_subset_induct) (use assms **in** ‹auto simp: has_integral_add›)

lemma integral_sum:
⟦finite I; ⋀a. a ∈ I ⟹ f a integrable_on S⟧ ⟹
integral S (λx. ∑ a∈I. f a x) = (∑ a∈I. integral S (f a))
by (simp add: has_integral_sum integrable_integral integral_unique)

lemma integrable_sum:
⟦finite I; ⋀a. a ∈ I ⟹ f a integrable_on S⟧ ⟹ (λx. ∑ a∈I. f a x) integrable_on S
unfolding integrable_on_def **using** has_integral_sum[of I] **by** metis

lemma has_integral_eq:
assumes ⋀x. x ∈ s ⟹ f x = g x
and f: (f has_integral k) s
shows (g has_integral k) s
using has_integral_diff[OF f, of λx. f x − g x 0]
using has_integral_is_0[of s λx. f x − g x]
using assms
by auto

lemma integrable_eq: ⟦f integrable_on s; ⋀x. x ∈ s ⟹ f x = g x⟧ ⟹ g integrable_on s
unfolding integrable_on_def
using has_integral_eq[of s f g] has_integral_eq **by** blast

lemma has_integral_cong:

assumes $\bigwedge x.\ x \in s \Longrightarrow f\ x = g\ x$
shows $(f\ has\_integral\ i)\ s = (g\ has\_integral\ i)\ s$
by (*metis assms has_integral_eq*)

lemma *integrable_cong*:
assumes $\bigwedge x.\ x \in A \Longrightarrow f\ x = g\ x$
shows $f\ integrable\_on\ A \longleftrightarrow g\ integrable\_on\ A$
using *has_integral_cong* [*OF assms*] **by** *fast*

lemma *integral_cong*:
assumes $\bigwedge x.\ x \in s \Longrightarrow f\ x = g\ x$
shows *integral s f* = *integral s g*
unfolding *integral_def*
by (*metis* (*full_types*, *opaque_lifting*) *assms has_integral_cong integrable_eq*)

lemma *integrable_on_cmult_left_iff* [*simp*]:
assumes $c \neq 0$
shows $(\lambda x.\ of\_real\ c * f\ x)\ integrable\_on\ s \longleftrightarrow f\ integrable\_on\ s$
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then have $(\lambda x.\ of\_real\ (1\ /\ c) * (of\_real\ c * f\ x))\ integrable\_on\ s$
using *integrable_cmul*[*of* $\lambda x.\ of\_real\ c * f\ x\ s\ 1\ /\ of\_real\ c$]
by (*simp add*: *scaleR_conv_of_real*)
then have $(\lambda x.\ (of\_real\ (1\ /\ c) * of\_real\ c * f\ x))\ integrable\_on\ s$
by (*simp add*: *algebra_simps*)
with ‹$c \neq 0$› **show** *?rhs*
by (*metis* (*no_types*, *lifting*) *integrable_eq mult.left_neutral nonzero_divide_eq_eq of_real_1 of_real_mult*)
qed (*blast intro*: *integrable_on_cmult_left*)

lemma *integrable_on_cmult_right*:
fixes *f* :: _ ⇒ *'b* :: {*comm_ring*,*real_algebra_1*,*real_normed_vector*}
assumes *f integrable_on s*
shows $(\lambda x.\ f\ x * of\_real\ c)\ integrable\_on\ s$
using *integrable_on_cmult_left* [*OF assms*] **by** (*simp add*: *mult.commute*)

lemma *integrable_on_cmult_right_iff* [*simp*]:
fixes *f* :: _ ⇒ *'b* :: {*comm_ring*,*real_algebra_1*,*real_normed_vector*}
assumes $c \neq 0$
shows $(\lambda x.\ f\ x * of\_real\ c)\ integrable\_on\ s \longleftrightarrow f\ integrable\_on\ s$
using *integrable_on_cmult_left_iff* [*OF assms*] **by** (*simp add*: *mult.commute*)

lemma *integrable_on_cdivide_iff* [*simp*]:
fixes *f* :: _ ⇒ *'b* :: *real_normed_field*
assumes $c \neq 0$
shows $(\lambda x.\ f\ x\ /\ of\_real\ c)\ integrable\_on\ s \longleftrightarrow f\ integrable\_on\ s$
by (*simp add*: *divide_inverse assms flip*: *of_real_inverse*)

lemma *has_integral_null* [*intro*]: *content*(*cbox a b*) = *0* ⟹ (*f has_integral 0*) (*cbox a b*)
unfolding *has_integral_cbox*
using *eventually_division_filter_tagged_division*[*of cbox a b*]
by (*subst tendsto_cong*[**where** *g*=λ_. *0*]) (*auto elim*: *eventually_mono intro*: *sum_content_null*)

lemma *has_integral_null_real* [*intro*]: *content* {*a*..*b*::*real*} = *0* ⟹ (*f has_integral 0*) {*a*..*b*}
by (*metis box_real*(*2*) *has_integral_null*)

lemma *has_integral_null_eq*[*simp*]: *content* (*cbox a b*) = *0* ⟹ (*f has_integral i*) (*cbox a b*) ⟷ *i* = *0*
by (*auto simp add*: *has_integral_null dest*!: *integral_unique*)

lemma *integral_null* [*simp*]: *content* (*cbox a b*) = *0* ⟹ *integral* (*cbox a b*) *f* = *0*
by (*metis has_integral_null integral_unique*)

lemma *integrable_on_null* [*intro*]: *content* (*cbox a b*) = *0* ⟹ *f integrable_on* (*cbox a b*)
by (*simp add*: *has_integral_integrable*)

lemma *has_integral_empty*[*intro*]: (*f has_integral 0*) {}
by (*meson ex_in_conv has_integral_is_0*)

lemma *has_integral_empty_eq*[*simp*]: (*f has_integral i*) {} ⟷ *i* = *0*
by (*auto simp add*: *has_integral_empty has_integral_unique*)

lemma *integrable_on_empty*[*intro*]: *f integrable_on* {}
unfolding *integrable_on_def* **by** *auto*

lemma *integral_empty*[*simp*]: *integral* {} *f* = *0*
by *blast*

lemma *has_integral_refl*[*intro*]:
fixes *a* :: *'a*::*euclidean_space*
shows (*f has_integral 0*) (*cbox a a*)
and (*f has_integral 0*) {*a*}
proof −
show (*f has_integral 0*) (*cbox a a*)
by (*rule has_integral_null*) *simp*
then show (*f has_integral 0*) {*a*}
by *simp*
qed

lemma *integrable_on_refl*[*intro*]: *f integrable_on cbox a a*
unfolding *integrable_on_def* **by** *auto*

lemma *integral_refl* [*simp*]: *integral* (*cbox a a*) *f* = *0*

by *auto*

lemma *integral_singleton* [*simp*]: *integral* {*a*} *f* = *0*
by *auto*

lemma *integral_blinfun_apply*:
assumes *f integrable_on s*
shows *integral s* (λ*x*. *blinfun_apply h* (*f x*)) = *blinfun_apply h* (*integral s f*)
by (*subst integral_linear*[*symmetric*, *OF assms blinfun.bounded_linear_right*]) (*simp add*: *o_def*)

lemma *blinfun_apply_integral*:
assumes *f integrable_on s*
shows *blinfun_apply* (*integral s f*) *x* = *integral s* (λ*y*. *blinfun_apply* (*f y*) *x*)
by (*metis* (*no_types*, *lifting*) *assms blinfun.prod_left.rep_eq integral_blinfun_apply integral_cong*)

lemma *has_integral_componentwise_iff*:
fixes *f* :: *'a* :: *euclidean_space* ⇒ *'b* :: *euclidean_space*
shows (*f has_integral y*) *A* ⟷ (∀ *b*∈*Basis*. ((λ*x*. *f x* • *b*) *has_integral* (*y* • *b*)) *A*)
proof *safe*
fix *b* :: *'b* **assume** (*f has_integral y*) *A*
from *has_integral_linear*[*OF this*(*1*) *bounded_linear_inner_left*, *of b*]
show ((λ*x*. *f x* • *b*) *has_integral* (*y* • *b*)) *A* **by** (*simp add*: *o_def*)
next
assume (∀ *b*∈*Basis*. ((λ*x*. *f x* • *b*) *has_integral* (*y* • *b*)) *A*)
hence ∀ *b*∈*Basis*. (((λ*x*. *x* ∗*R* *b*) ∘ (λ*x*. *f x* • *b*)) *has_integral* ((*y* • *b*) ∗*R* *b*)) *A*
by (*intro ballI has_integral_linear*) (*simp_all add*: *bounded_linear_scaleR_left*)
hence ((λ*x*. ∑ *b*∈*Basis*. (*f x* • *b*) ∗*R* *b*) *has_integral* (∑ *b*∈*Basis*. (*y* • *b*) ∗*R* *b*)) *A*
by (*intro has_integral_sum*) (*simp_all add*: *o_def*)
thus (*f has_integral y*) *A* **by** (*simp add*: *euclidean_representation*)
qed

lemma *has_integral_componentwise*:
fixes *f* :: *'a* :: *euclidean_space* ⇒ *'b* :: *euclidean_space*
shows (⋀*b*. *b* ∈ *Basis* ⟹ ((λ*x*. *f x* • *b*) *has_integral* (*y* • *b*)) *A*) ⟹ (*f has_integral y*) *A*
by (*subst has_integral_componentwise_iff*) *blast*

lemma *integrable_componentwise_iff*:
fixes *f* :: *'a* :: *euclidean_space* ⇒ *'b* :: *euclidean_space*
shows *f integrable_on A* ⟷ (∀ *b*∈*Basis*. (λ*x*. *f x* • *b*) *integrable_on A*)
proof
assume *f integrable_on A*
then obtain *y* **where** (*f has_integral y*) *A* **by** (*auto simp*: *integrable_on_def*)
hence (∀ *b*∈*Basis*. ((λ*x*. *f x* • *b*) *has_integral* (*y* • *b*)) *A*)
by (*subst* (*asm*) *has_integral_componentwise_iff*)

thus (∀ *b*∈*Basis*. (λ*x*. *f x* ⋅ *b*) *integrable_on A*) **by** (*auto simp*: *integrable_on_def*)
next
assume (∀ *b*∈*Basis*. (λ*x*. *f x* ⋅ *b*) *integrable_on A*)
then obtain *y* **where** ∀ *b*∈*Basis*. ((λ*x*. *f x* ⋅ *b*) *has_integral y b*) *A*
unfolding *integrable_on_def* **by** (*subst* (*asm*) *bchoice_iff*) *blast*
hence ∀ *b*∈*Basis*. (((λ*x*. *x* ∗*R b*) ∘ (λ*x*. *f x* ⋅ *b*)) *has_integral* (*y b* ∗*R b*)) *A*
by (*intro ballI has_integral_linear*) (*simp_all add*: *bounded_linear_scaleR_left*)
hence ((λ*x*. ∑ *b*∈*Basis*. (*f x* ⋅ *b*) ∗*R b*) *has_integral* (∑ *b*∈*Basis*. *y b* ∗*R b*)) *A*
by (*intro has_integral_sum*) (*simp_all add*: *o_def*)
thus *f integrable_on A* **by** (*auto simp*: *integrable_on_def o_def euclidean_representation*)
qed

lemma *integrable_componentwise*:
fixes *f* :: *'a* :: *euclidean_space* ⇒ *'b* :: *euclidean_space*
shows (⋀*b*. *b* ∈ *Basis* ⟹ (λ*x*. *f x* ⋅ *b*) *integrable_on A*) ⟹ *f integrable_on A*
by (*subst integrable_componentwise_iff*) *blast*

lemma *integral_componentwise*:
fixes *f* :: *'a* :: *euclidean_space* ⇒ *'b* :: *euclidean_space*
assumes *f integrable_on A*
shows *integral A f* = (∑ *b*∈*Basis*. *integral A* (λ*x*. (*f x* ⋅ *b*) ∗*R b*))
proof −
from *assms* **have** *integrable*: ∀ *b*∈*Basis*. (λ*x*. *x* ∗*R b*) ∘ (λ*x*. (*f x* ⋅ *b*)) *integrable_on A*
by (*subst* (*asm*) *integrable_componentwise_iff*, *intro integrable_linear ballI*) (*simp_all add*: *bounded_linear_scaleR_left*)
have *integral A f* = *integral A* (λ*x*. ∑ *b*∈*Basis*. (*f x* ⋅ *b*) ∗*R b*)
by (*simp add*: *euclidean_representation*)
also from *integrable* **have** ... = (∑ *a*∈*Basis*. *integral A* (λ*x*. (*f x* ⋅ *a*) ∗*R a*))
by (*subst integral_sum*) (*simp_all add*: *o_def*)
finally show *?thesis* **.**
qed

lemma *integrable_component*:
f integrable_on A ⟹ (λ*x*. *f x* ⋅ (*y* :: *'b* :: *euclidean_space*)) *integrable_on A*
by (*drule integrable_linear*[*OF* _ *bounded_linear_inner_left*[*of y*]]) (*simp add*: *o_def*)

6.14.4 Cauchy-type criterion for integrability

proposition *integrable_Cauchy*:
fixes *f* :: *'n*::*euclidean_space* ⇒ *'a*::{*real_normed_vector*,*complete_space*}
shows *f integrable_on cbox a b* ⟷
(∀ *e*>0. ∃ γ. *gauge* γ ∧
(∀ *D1 D2*. *D1 tagged_division_of* (*cbox a b*) ∧ γ *fine D1* ∧
D2 tagged_division_of (*cbox a b*) ∧ γ *fine D2* ⟶
norm ((∑ (*x*,*K*)∈*D1*. *content K* ∗*R f x*) − (∑ (*x*,*K*)∈*D2*. *content K* ∗*R f x*)) < *e*))
(**is** *?l* = (∀ *e*>0. ∃ γ. *?P e* γ))

proof (*intro iffI allI impI*)
assume *?l*
then obtain *y*
where *y*: $\bigwedge e.\ e > 0 \Longrightarrow$
$\exists \gamma.\ gauge\ \gamma \wedge$
$(\forall \mathcal{D}.\ \mathcal{D}\ tagged\_division\_of\ cbox\ a\ b \wedge \gamma\ fine\ \mathcal{D} \longrightarrow$
$norm\ ((\sum (x,K) \in \mathcal{D}.\ content\ K *_R f\ x) - y) < e)$
by (*auto simp*: *integrable_on_def has_integral*)
show $\exists \gamma.\ ?P\ e\ \gamma$ **if** $e > 0$ **for** *e*
proof −
have $e/2 > 0$ **using** *that* **by** *auto*
with *y* **obtain** γ **where** $gauge\ \gamma$
and γ: $\bigwedge \mathcal{D}.\ \mathcal{D}\ tagged\_division\_of\ cbox\ a\ b \wedge \gamma\ fine\ \mathcal{D} \Longrightarrow$
$norm\ ((\sum (x,K) \in \mathcal{D}.\ content\ K *_R f\ x) - y) < e/2$
by *meson*
show *?thesis*
apply (*rule_tac x*=γ **in** *exI*, *clarsimp simp*: ‹*gauge* γ›)
by (*blast intro*!: γ *dist_triangle_half_l*[**where** *y*=*y*,*unfolded dist_norm*])
qed
next
assume $\forall e{>}0.\ \exists \gamma.\ ?P\ e\ \gamma$
then have $\forall n{::}nat.\ \exists \gamma.\ ?P\ (1\ /\ (n + 1))\ \gamma$
by *auto*
then obtain $\gamma :: nat \Rightarrow 'n \Rightarrow 'n\ set$ **where** γ:
$\bigwedge m.\ gauge\ (\gamma\ m)$
$\bigwedge m\ \mathcal{D}1\ \mathcal{D}2.\ [\![\mathcal{D}1\ tagged\_division\_of\ cbox\ a\ b;$
$\gamma\ m\ fine\ \mathcal{D}1;\ \mathcal{D}2\ tagged\_division\_of\ cbox\ a\ b;\ \gamma\ m\ fine\ \mathcal{D}2]\!]$
$\Longrightarrow norm\ ((\sum (x,K) \in \mathcal{D}1.\ content\ K *_R f\ x) - (\sum (x,K) \in \mathcal{D}2.$
$content\ K *_R f\ x))$
$< 1\ /\ (m + 1)$
by *metis*
have *gauge* $(\lambda x.\ \bigcap \{\gamma\ i\ x\ |i.\ i \in \{0..n\}\})$ **for** *n*
using γ **by** (*intro gauge_Inter*) *auto*
then have $\forall n.\ \exists p.\ p\ tagged\_division\_of\ (cbox\ a\ b) \wedge (\lambda x.\ \bigcap \{\gamma\ i\ x\ |i.\ i \in$
$\{0..n\}\})\ fine\ p$
by (*meson fine_division_exists*)
then obtain *p* **where** *p*: $\bigwedge z.\ p\ z\ tagged\_division\_of\ cbox\ a\ b$
$\bigwedge z.\ (\lambda x.\ \bigcap \{\gamma\ i\ x\ |i.\ i \in \{0..z\}\})\ fine\ p\ z$
by *meson*
have *dp*: $\bigwedge i\ n.\ i{\le}n \Longrightarrow \gamma\ i\ fine\ p\ n$
using *p* **unfolding** *fine_Inter*
using *atLeastAtMost_iff* **by** *blast*
have *Cauchy* $(\lambda n.\ sum\ (\lambda(x,K).\ content\ K *_R (f\ x))\ (p\ n))$
proof (*rule CauchyI*)
fix *e*::*real*
assume $0 < e$
then obtain *N* **where** $N \neq 0$ **and** *N*: $inverse\ (real\ N) < e$
using *real_arch_inverse*[*of e*] **by** *blast*
show $\exists M.\ \forall m{\ge}M.\ \forall n{\ge}M.\ norm\ ((\sum (x,K) \in p\ m.\ content\ K *_R f\ x) -$

```
(∑(x,K) ∈ p n. content K *R f x)) < e
    proof (intro exI allI impI)
      fix m n
      assume mn: N ≤ m N ≤ n
      have norm ((∑(x,K) ∈ p m. content K *R f x) - (∑(x,K) ∈ p n. content
K *R f x)) < 1 / (real N + 1)
        by (simp add: p(1) dp mn γ)
      also have ... < e
        using  N ‹N ≠ 0› ‹0 < e› by (auto simp: field_simps)
      finally show norm ((∑(x,K) ∈ p m. content K *R f x) - (∑(x,K) ∈ p n.
content K *R f x)) < e .
    qed
  qed
  then obtain y where y: ∃no. ∀n≥no. norm ((∑(x,K) ∈ p n. content K *R f
x) - y) < r if r > 0 for r
    by (auto simp: convergent_eq_Cauchy[symmetric] dest: LIMSEQ_D)
  show ?l
    unfolding integrable_on_def has_integral
  proof (rule_tac x=y in exI, clarify)
    fix e :: real
    assume e>0
    then have e2: e/2 > 0 by auto
    then obtain N1::nat where N1: N1 ≠ 0 inverse (real N1) < e/2
      using real_arch_inverse by blast
    obtain N2::nat where N2: ⋀n. n ≥ N2 ⟹ norm ((∑(x,K) ∈ p n. content
K *R f x) - y) < e/2
      using y[OF e2] by metis
    show ∃γ. gauge γ ∧
            (∀D. D tagged_division_of (cbox a b) ∧ γ fine D ⟶
              norm ((∑(x,K) ∈ D. content K *R f x) - y) < e)
    proof (intro exI conjI allI impI)
      show gauge (γ (N1+N2))
        using γ by auto
      show norm ((∑(x,K) ∈ q. content K *R f x) - y) < e
          if q tagged_division_of cbox a b ∧ γ (N1+N2) fine q for q
      proof (rule norm_triangle_half_r)
        have norm ((∑(x,K) ∈ p (N1+N2). content K *R f x) - (∑(x,K) ∈ q.
content K *R f x))
             < 1 / (real (N1+N2) + 1)
          by (rule γ; simp add: dp p that)
        also have ... < e/2
          using N1 ‹0 < e› by (auto simp: field_simps intro: less_le_trans)
        finally show norm ((∑(x,K) ∈ p (N1+N2). content K *R f x) - (∑(x,K)
∈ q. content K *R f x)) < e/2 .
        show norm ((∑(x,K) ∈ p (N1+N2). content K *R f x) - y) < e/2
          using N2 le_add_same_cancel2 by blast
      qed
    qed
  qed
```

qed

6.14.5 Additivity of integral on abutting intervals

lemma *tagged_division_split_left_inj_content*:
assumes $\mathcal{D}$: $\mathcal{D}$ *tagged_division_of S*
and (*x1*, *K1*) $\in$ $\mathcal{D}$ (*x2*, *K2*) $\in$ $\mathcal{D}$ *K1* $\neq$ *K2* *K1* $\cap$ {*x*. *x*•*k* $\leq$ *c*} = *K2* $\cap$ {*x*. *x*•*k* $\leq$ *c*} *k* $\in$ *Basis*
shows *content* (*K1* $\cap$ {*x*. *x*•*k* $\leq$ *c*}) = *0*
proof −
from *tagged_division_ofD*(*4*)[*OF* $\mathcal{D}$ ‹(*x1*, *K1*) $\in$ $\mathcal{D}$›] **obtain** *a b* **where** *K1*: *K1* = *cbox a b*
by *auto*
then have *interior* (*K1* $\cap$ {*x*. *x* • *k* $\leq$ *c*}) = {}
by (*metis tagged_division_split_left_inj assms*)
then show *?thesis*
unfolding *K1 interval_split*[*OF* ‹*k* $\in$ *Basis*›] **by** (*auto simp*: *content_eq_0_interior*)
qed

lemma *tagged_division_split_right_inj_content*:
assumes $\mathcal{D}$: $\mathcal{D}$ *tagged_division_of S*
and (*x1*, *K1*) $\in$ $\mathcal{D}$ (*x2*, *K2*) $\in$ $\mathcal{D}$ *K1* $\neq$ *K2* *K1* $\cap$ {*x*. *x*•*k* $\geq$ *c*} = *K2* $\cap$ {*x*. *x*•*k* $\geq$ *c*} *k* $\in$ *Basis*
shows *content* (*K1* $\cap$ {*x*. *x*•*k* $\geq$ *c*}) = *0*
proof −
from *tagged_division_ofD*(*4*)[*OF* $\mathcal{D}$ ‹(*x1*, *K1*) $\in$ $\mathcal{D}$›] **obtain** *a b* **where** *K1*: *K1* = *cbox a b*
by *auto*
then have *interior* (*K1* $\cap$ {*x*. *c* $\leq$ *x* • *k*}) = {}
by (*metis tagged_division_split_right_inj assms*)
then show *?thesis*
unfolding *K1 interval_split*[*OF* ‹*k* $\in$ *Basis*›]
by (*auto simp*: *content_eq_0_interior*)
qed

proposition *has_integral_split*:
fixes *f* :: *'a*::*euclidean_space* $\Rightarrow$ *'b*::*real_normed_vector*
assumes *fi*: (*f has_integral i*) (*cbox a b* $\cap$ {*x*. *x*•*k* $\leq$ *c*})
and *fj*: (*f has_integral j*) (*cbox a b* $\cap$ {*x*. *x*•*k* $\geq$ *c*})
and *k*: *k* $\in$ *Basis*
shows (*f has_integral* (*i* + *j*)) (*cbox a b*)
unfolding *has_integral*
proof *clarify*
fix *e*::*real*
assume *0* < *e*
then have *e*: *e*/*2* > *0*
by *auto*
obtain $\gamma 1$ **where** $\gamma 1$: *gauge* $\gamma 1$

```
    and γ1norm:
      ⋀𝒟. ⟦𝒟 tagged_division_of cbox a b ∩ {x. x • k ≤ c}; γ1 fine 𝒟⟧
          ⟹ norm ((∑(x,K) ∈ 𝒟. content K *R f x) − i) < e/2
     apply (rule has_integralD[OF fi[unfolded interval_split[OF k]] e])
     apply (simp add: interval_split[symmetric] k)
    done
  obtain γ2 where γ2: gauge γ2
    and γ2norm:
      ⋀𝒟. ⟦𝒟 tagged_division_of cbox a b ∩ {x. c ≤ x • k}; γ2 fine 𝒟⟧
          ⟹ norm ((∑(x, k) ∈ 𝒟. content k *R f x) − j) < e/2
     apply (rule has_integralD[OF fj[unfolded interval_split[OF k]] e])
     apply (simp add: interval_split[symmetric] k)
     done
  let ?γ = λx. if x•k = c then (γ1 x ∩ γ2 x) else ball x |x•k − c| ∩ γ1 x ∩ γ2 x
  have gauge ?γ
    using γ1 γ2 unfolding gauge_def by auto
  then show ∃γ. gauge γ ∧
              (∀𝒟. 𝒟 tagged_division_of cbox a b ∧ γ fine 𝒟 ⟶
                  norm ((∑(x, k)∈𝒟. content k *R f x) − (i + j)) < e)
  proof (rule_tac x=?γ in exI, safe)
    fix p
    assume p: p tagged_division_of (cbox a b) and ?γ fine p
    have ab_eqp: cbox a b = ⋃{K. ∃x. (x, K) ∈ p}
      using p by blast
    have xk_le_c: x•k ≤ c if as: (x,K) ∈ p and K: K ∩ {x. x•k ≤ c} ≠ {} for x
K
    proof (rule ccontr)
      assume **: ¬ x • k ≤ c
      then have K ⊆ ball x |x • k − c|
        using ‹?γ fine p› as by (fastforce simp: not_le algebra_simps)
      with K obtain y where y: y ∈ ball x |x • k − c| y•k ≤ c
        by blast
      then have |x • k − y • k| < |x • k − c|
        using Basis_le_norm[OF k, of x − y]
        by (auto simp add: dist_norm inner_diff_left intro: le_less_trans)
      with y show False
        using ** by (auto simp add: field_simps)
    qed
    have xk_ge_c: x•k ≥ c if as: (x,K) ∈ p and K: K ∩ {x. x•k ≥ c} ≠ {} for x
K
    proof (rule ccontr)
      assume **: ¬ x • k ≥ c
      then have K ⊆ ball x |x • k − c|
        using ‹?γ fine p› as by (fastforce simp: not_le algebra_simps)
      with K obtain y where y: y ∈ ball x |x • k − c| y•k ≥ c
        by blast
      then have |x • k − y • k| < |x • k − c|
        using Basis_le_norm[OF k, of x − y]
        by (auto simp add: dist_norm inner_diff_left intro: le_less_trans)
```

```
  with y show False
    using ∗∗ by (auto simp add: field_simps)
qed
have fin_finite: finite {(x,f K) | x K. (x,K) ∈ s ∧ P x K}
  if finite s for s and f :: 'a set ⇒ 'a set and P :: 'a ⇒ 'a set ⇒ bool
proof −
  from that have finite ((λ(x,K). (x, f K)) ‘ s)
    by auto
  then show ?thesis
    by (rule rev_finite_subset) auto
qed
{ fix G :: 'a set ⇒ 'a set
  fix i :: 'a × 'a set
  assume i ∈ (λ(x, k). (x, G k)) ‘ p − {(x, G k) |x k. (x, k) ∈ p ∧ G k ≠ {}}
  then obtain x K where xk: i = (x, G K)  (x,K) ∈ p
                        (x, G K) ∉ {(x, G K) |x K. (x,K) ∈ p ∧ G K ≠ {}}
    by auto
  have content (G K) = 0
    using xk using content_empty by auto
  then have (λ(x,K). content K ∗R f x) i = 0
    unfolding xk split_conv by auto
} note [simp] = this
have finite p
  using p by blast
let ?M1 = {(x, K ∩ {x. x•k ≤ c}) |x K. (x,K) ∈ p ∧ K ∩ {x. x•k ≤ c} ≠ {}}
have γ1_fine: γ1 fine ?M1
  using ‹?γ fine p› by (fastforce simp: fine_def split: if_split_asm)
have norm ((∑(x, k)∈?M1. content k ∗R f x) − i) < e/2
proof (rule γ1norm [OF tagged_division_ofI γ1_fine])
  show finite ?M1
    by (rule fin_finite) (use p in blast)
  show ⋃{k. ∃x. (x, k) ∈ ?M1} = cbox a b ∩ {x. x•k ≤ c}
    by (auto simp: ab_eqp)

  fix x L
  assume xL: (x, L) ∈ ?M1
  then obtain x' L' where xL': x = x' L = L' ∩ {x. x • k ≤ c}
                             (x', L') ∈ p L' ∩ {x. x • k ≤ c} ≠ {}
    by blast
  then obtain a' b' where ab': L' = cbox a' b'
    using p by blast
  show x ∈ L L ⊆ cbox a b ∩ {x. x • k ≤ c}
    using p xk_le_c xL' by auto
  show ∃ a b. L = cbox a b
    using p xL' ab' by (auto simp add: interval_split[OF k,where c=c])

  fix y R
  assume yR: (y, R) ∈ ?M1
  then obtain y' R' where yR': y = y' R = R' ∩ {x. x • k ≤ c}
```

```
                                (y', R') ∈ p R' ∩ {x. x · k ≤ c} ≠ {}
      by blast
    assume as: (x, L) ≠ (y, R)
    show interior L ∩ interior R = {}
    proof (cases L' = R' ⟶ x' = y')
      case False
      have interior R' = {}
        by (metis (no_types) False Pair_inject inf.idem tagged_division_ofD(5)
[OF p] xL'(3) yR'(3))
      then show ?thesis
        using yR' by simp
    next
      case True
      then have L' ≠ R'
        using as unfolding xL' yR' by auto
      have interior L' ∩ interior R' = {}
        by (metis (no_types) Pair_inject ‹L' ≠ R'› p tagged_division_ofD(5)
xL'(3) yR'(3))
      then show ?thesis
        using xL'(2) yR'(2) by auto
    qed
  qed
  moreover
  let ?M2 = {(x,K ∩ {x. x·k ≥ c}) |x K. (x,K) ∈ p ∧ K ∩ {x. x·k ≥ c} ≠ {}}
  have γ2_fine: γ2 fine ?M2
    using ‹?γ fine p› by (fastforce simp: fine_def split: if_split_asm)
  have norm ((∑(x, k)∈?M2. content k *R f x) - j) < e/2
  proof (rule γ2norm [OF tagged_division_ofI γ2_fine])
    show finite ?M2
      by (rule fin_finite) (use p in blast)
    show ⋃{k. ∃x. (x, k) ∈ ?M2} = cbox a b ∩ {x. x·k ≥ c}
      by (auto simp: ab_eqp)

    fix x L
    assume xL: (x, L) ∈ ?M2
    then obtain x' L' where xL': x = x' L = L' ∩ {x. x · k ≥ c}
                                (x', L') ∈ p L' ∩ {x. x · k ≥ c} ≠ {}
      by blast
    then obtain a' b' where ab': L' = cbox a' b'
      using p by blast
    show x ∈ L L ⊆ cbox a b ∩ {x. x · k ≥ c}
      using p xk_ge_c xL' by auto
    show ∃a b. L = cbox a b
      using p xL' ab' by (auto simp add: interval_split[OF k,where c=c])

    fix y R
    assume yR: (y, R) ∈ ?M2
    then obtain y' R' where yR': y = y' R = R' ∩ {x. x · k ≥ c}
                                (y', R') ∈ p R' ∩ {x. x · k ≥ c} ≠ {}
```

```
      by blast
    assume as: (x, L) ≠ (y, R)
    show interior L ∩ interior R = {}
    proof (cases L' = R' ⟶ x' = y')
      case False
      have interior R' = {}
        by (metis (no_types) False Pair_inject inf.idem tagged_division_ofD(5)
[OF p] xL'(3) yR'(3))
      then show ?thesis
        using yR' by simp
    next
      case True
      then have L' ≠ R'
        using as unfolding xL' yR' by auto
      have interior L' ∩ interior R' = {}
        by (metis (no_types) Pair_inject ‹L' ≠ R'› p tagged_division_ofD(5)
xL'(3) yR'(3))
      then show ?thesis
        using xL'(2) yR'(2) by auto
    qed
  qed
  ultimately
  have norm (((∑(x,K) ∈ ?M1. content K *R f x) − i) + ((∑(x,K) ∈ ?M2.
content K *R f x) − j)) < e/2 + e/2
    using norm_add_less by blast
  moreover have ((∑(x,K) ∈ ?M1. content K *R f x) − i) +
                ((∑(x,K) ∈ ?M2. content K *R f x) − j) =
                (∑(x, ka)∈p. content ka *R f x) − (i + j)
  proof −
    have eq0: ⋀x y. x = (0::real) ⟹ x *R (y::'b) = 0
      by auto
    have cont_eq: ⋀g. (λ(x,l). content l *R f x) ∘ (λ(x,l). (x,g l)) = (λ(x,l).
content (g l) *R f x)
      by auto
    have *: ⋀G :: 'a set ⇒ 'a set.
              (∑(x,K)∈{(x, G K) |x K. (x,K) ∈ p ∧ G K ≠ {}}. content K *R f
x) =
              (∑(x,K)∈(λ(x,K). (x, G K)) ‘ p. content K *R f x)
      by (rule sum.mono_neutral_left) (auto simp: ‹finite p›)
    have ((∑(x, k)∈?M1. content k *R f x) − i) + ((∑(x, k)∈?M2. content k
*R f x) − j) =
      (∑(x, k)∈?M1. content k *R f x) + (∑(x, k)∈?M2. content k *R f x) −
(i + j)
      by auto
    moreover have ... = (∑(x,K) ∈ p. content (K ∩ {x. x • k ≤ c}) *R f x) +
      (∑(x,K) ∈ p. content (K ∩ {x. c ≤ x • k}) *R f x) − (i + j)
      unfolding *
      apply (subst (1 2) sum.reindex_nontrivial)
          apply (auto intro!: k p eq0 tagged_division_split_left_inj_content
```

```
tagged_division_split_right_inj_content
                    simp: cont_eq ‹finite p›)
      done
    moreover have ⋀x. x ∈ p ⟹ (λ(a,B). content (B ∩ {a. a · k ≤ c}) *R f
a) x +
                                (λ(a,B). content (B ∩ {a. c ≤ a · k}) *R f a) x =
                                (λ(a,B). content B *R f a) x
    proof clarify
      fix a B
      assume (a, B) ∈ p
      with p obtain u v where uv: B = cbox u v by blast
      then show content (B ∩ {x. x · k ≤ c}) *R f a + content (B ∩ {x. c ≤ x
· k}) *R f a = content B *R f a
        by (auto simp: scaleR_left_distrib uv content_split[OF k,of u v c])
    qed
    ultimately show ?thesis
      by (auto simp: sum.distrib[symmetric])
    qed
  ultimately show norm ((∑(x, k)∈p. content k *R f x) − (i + j)) < e
    by auto
 qed
qed
```

6.14.6 A sort of converse, integrability on subintervals

```
lemma has_integral_separate_sides:
 fixes f :: 'a::euclidean_space ⇒ 'b::real_normed_vector
 assumes f: (f has_integral i) (cbox a b)
   and e > 0
   and k: k ∈ Basis
 obtains d where gauge d
   ∀ p1 p2. p1 tagged_division_of (cbox a b ∩ {x. x•k ≤ c}) ∧ d fine p1 ∧
      p2 tagged_division_of (cbox a b ∩ {x. x•k ≥ c}) ∧ d fine p2 ⟶
      norm ((sum (λ(x,k). content k *R f x) p1 + sum (λ(x,k). content k *R f x)
p2) − i) < e
proof −
 obtain γ where d: gauge γ
     ⋀p. ⟦p tagged_division_of cbox a b; γ fine p⟧
         ⟹ norm ((∑(x, k)∈p. content k *R f x) − i) < e
   using has_integralD[OF f ‹e > 0›] by metis
 { fix p1 p2
   assume tdiv1: p1 tagged_division_of (cbox a b) ∩ {x. x · k ≤ c} and γ fine
p1
   note p1=tagged_division_ofD[OF this(1)]
   assume tdiv2: p2 tagged_division_of (cbox a b) ∩ {x. c ≤ x · k} and γ fine
p2
   note p2=tagged_division_ofD[OF this(1)]
   note tagged_division_Un_interval[OF tdiv1 tdiv2]
   note p12 = tagged_division_ofD[OF this] this
```

```
  { fix a b
    assume ab: (a, b) ∈ p1 ∩ p2
    have (a, b) ∈ p1
      using ab by auto
    obtain u v where uv: b = cbox u v
      using ‹(a, b) ∈ p1› p1(4) by moura
    have b ⊆ {x. x•k = c}
      using ab p1(3)[of a b] p2(3)[of a b] by fastforce
    moreover
    have interior {x::'a. x • k = c} = {}
    proof (rule ccontr)
      assume ¬ ?thesis
      then obtain x where x: x ∈ interior {x::'a. x•k = c}
        by auto
      then obtain ε where 0 < ε and ε: ball x ε ⊆ {x. x • k = c}
        using mem_interior by metis
      have x: x•k = c
        using x interior_subset by fastforce
      have *: ⋀i. i ∈ Basis ⟹ |(x − (x + (ε/2) *R k)) • i| = (if i = k then ε/2
else 0)
        using ‹0 < ε› k by (auto simp: inner_simps inner_not_same_Basis)
      have (∑ i∈Basis. |(x − (x + (ε/2 ) *R k)) • i|) =
          (∑ i∈Basis. (if i = k then ε/2 else 0))
        using * by (blast intro: sum.cong)
      also have ... < ε
        by (subst sum.delta) (use ‹0 < ε› in auto)
      finally have x + (ε/2) *R k ∈ ball x ε
        unfolding mem_ball dist_norm by(rule le_less_trans[OF norm_le_l1])
      then have x + (ε/2) *R k ∈ {x. x•k = c}
        using ε by auto
      then show False
        using ‹0 < ε› x k by (auto simp: inner_simps)
    qed
    ultimately have content b = 0
      unfolding uv content_eq_0_interior
      using interior_mono by blast
    then have content b *R f a = 0
      by auto
  }
  then have norm ((∑ (x, k)∈p1. content k *R f x) + (∑ (x, k)∈p2. content k
*R f x) − i) =
            norm ((∑ (x, k)∈p1 ∪ p2. content k *R f x) − i)
    by (subst sum.union_inter_neutral) (auto simp: p1 p2)
  also have ... < e
    using d(2) p12 by (simp add: fine_Un k ‹γ fine p1› ‹γ fine p2›)
  finally have norm ((∑ (x, k)∈p1. content k *R f x) + (∑ (x, k)∈p2. content
k *R f x) − i) < e .
 }
 then show ?thesis
```

```
    using d(1) that by auto
qed

lemma integrable_split [intro]:
  fixes f :: 'a::euclidean_space ⇒ 'b::{real_normed_vector,complete_space}
  assumes f: f integrable_on cbox a b
      and k: k ∈ Basis
    shows f integrable_on (cbox a b ∩ {x. x•k ≤ c})   (is ?thesis1)
    and   f integrable_on (cbox a b ∩ {x. x•k ≥ c})   (is ?thesis2)
proof -
  obtain y where y: (f has_integral y) (cbox a b)
    using f by blast
  define a' where a' = (∑ i∈Basis. (if i = k then max (a•k) c else a•i)*R i)
  define b' where b' = (∑ i∈Basis. (if i = k then min (b•k) c else b•i)*R i)
  have ∃ d. gauge d ∧
         (∀ p1 p2. p1 tagged_division_of cbox a b ∩ {x. x • k ≤ c} ∧ d fine p1 ∧
               p2 tagged_division_of cbox a b ∩ {x. x • k ≤ c} ∧ d fine p2 ⟶
                  norm ((∑ (x,K) ∈ p1. content K *R f x) - (∑ (x,K) ∈ p2.
content K *R f x)) < e)
    if e > 0 for e
  proof -
    have e/2 > 0 using that by auto
  with has_integral_separate_sides[OF y this k, of c]
  obtain d
    where gauge d
      and d: ⋀p1 p2. ⟦p1 tagged_division_of cbox a b ∩ {x. x • k ≤ c}; d fine
p1;
                    p2 tagged_division_of cbox a b ∩ {x. c ≤ x • k}; d fine p2⟧
                ⟹ norm ((∑ (x,K)∈p1. content K *R f x) + (∑ (x,K)∈p2. content
K *R f x) - y) < e/2
    by metis
  show ?thesis
    proof (rule_tac x=d in exI, clarsimp simp add: ‹gauge d›)
      fix p1 p2
      assume as: p1 tagged_division_of (cbox a b) ∩ {x. x • k ≤ c} d fine p1
                 p2 tagged_division_of (cbox a b) ∩ {x. x • k ≤ c} d fine p2
      show norm ((∑ (x, k)∈p1. content k *R f x) - (∑ (x, k)∈p2. content k *R
f x)) < e
      proof (rule fine_division_exists[OF ‹gauge d›, of a' b])
        fix p
        assume p tagged_division_of cbox a' b d fine p
        then show ?thesis
          using as norm_triangle_half_l[OF d[of p1 p] d[of p2 p]]
          unfolding interval_split[OF k] b'_def[symmetric] a'_def[symmetric]
          by (auto simp add: algebra_simps)
      qed
    qed
  qed
  with f show ?thesis1
```

```
    by (simp add: interval_split[OF k] integrable_Cauchy)
  have ∃ d. gauge d ∧
          (∀ p1 p2. p1 tagged_division_of cbox a b ∩ {x. x • k ≥ c} ∧ d fine p1 ∧
                    p2 tagged_division_of cbox a b ∩ {x. x • k ≥ c} ∧ d fine p2 ⟶
                       norm ((∑(x,K) ∈ p1. content K *R f x) − (∑(x,K) ∈ p2.
content K *R f x)) < e)
    if e > 0 for e
  proof −
    have e/2 > 0 using that by auto
  with has_integral_separate_sides[OF y this k, of c]
  obtain d
    where gauge d
        and d: ⋀p1 p2. ⟦p1 tagged_division_of cbox a b ∩ {x. x • k ≤ c}; d fine
p1;
                        p2 tagged_division_of cbox a b ∩ {x. c ≤ x • k}; d fine p2⟧
                   ⟹ norm ((∑(x,K)∈p1. content K *R f x) + (∑(x,K)∈p2. content
K *R f x) − y) < e/2
    by metis
  show ?thesis
    proof (rule_tac x=d in exI, clarsimp simp add: ‹gauge d›)
      fix p1 p2
      assume as: p1 tagged_division_of (cbox a b) ∩ {x. x • k ≥ c} d fine p1
                 p2 tagged_division_of (cbox a b) ∩ {x. x • k ≥ c} d fine p2
      show norm ((∑(x, k)∈p1. content k *R f x) − (∑(x, k)∈p2. content k *R
f x)) < e
      proof (rule fine_division_exists[OF ‹gauge d›, of a b'])
        fix p
        assume p tagged_division_of cbox a b' d fine p
        then show ?thesis
          using as norm_triangle_half_l[OF d[of p p1] d[of p p2]]
          unfolding interval_split[OF k] b'_def[symmetric] a'_def[symmetric]
          by (auto simp add: algebra_simps)
      qed
    qed
  qed
  with f show ?thesis2
    by (simp add: interval_split[OF k] integrable_Cauchy)
qed

lemma operative_integralI:
  fixes f :: 'a::euclidean_space ⇒ 'b::banach
  shows operative (lift_option (+)) (Some 0)
    (λi. if f integrable_on i then Some (integral i f) else None)
proof −
  interpret comm_monoid lift_option plus Some (0::'b)
    by (rule comm_monoid_lift_option)
      (rule add.comm_monoid_axioms)
  show ?thesis
  proof
```

fix *a b c*
fix *k* :: *'a*
assume *k*: *k* ∈ *Basis*
show (*if f integrable_on cbox a b then Some* (*integral* (*cbox a b*) *f*) *else None*)
=
lift_option (+) (*if f integrable_on cbox a b* ∩ {*x*. *x* · *k* ≤ *c*} *then Some* (*integral* (*cbox a b* ∩ {*x*. *x* · *k* ≤ *c*}) *f*) *else None*)
(*if f integrable_on cbox a b* ∩ {*x*. *c* ≤ *x* · *k*} *then Some* (*integral* (*cbox a b* ∩ {*x*. *c* ≤ *x* · *k*}) *f*) *else None*)
proof (*cases f integrable_on cbox a b*)
case *True*
with *k* **show** *?thesis*
by (*auto simp*: *integrable_split intro*: *integral_unique* [*OF has_integral_split*[*OF* _ _ *k*]])
next
case *False*
have ¬ (*f integrable_on cbox a b* ∩ {*x*. *x* · *k* ≤ *c*}) ∨ ¬ (*f integrable_on cbox a b* ∩ {*x*. *c* ≤ *x* · *k*})
proof (*rule ccontr*)
assume ¬ *?thesis*
then have *f integrable_on cbox a b*
unfolding *integrable_on_def*
apply (*rule_tac x*=*integral* (*cbox a b* ∩ {*x*. *x* · *k* ≤ *c*}) *f* + *integral* (*cbox a b* ∩ {*x*. *x* · *k* ≥ *c*}) *f* **in** *exI*)
apply (*auto intro*: *has_integral_split*[*OF* _ _ *k*])
done
then show *False*
using *False* **by** *auto*
qed
then show *?thesis*
using *False* **by** *auto*
qed
next
fix *a b* :: *'a*
assume *box a b* = {}
then show (*if f integrable_on cbox a b then Some* (*integral* (*cbox a b*) *f*) *else None*) = *Some 0*
using *has_integral_null_eq*
by (*auto simp*: *integrable_on_null content_eq_0_interior*)
qed
qed

6.14.7 Bounds on the norm of Riemann sums and the integral itself

lemma *dsum_bound*:
assumes *p*: *p division_of* (*cbox a b*)
and *norm c* ≤ *e*
shows *norm* (*sum* (λ*l*. *content l* ∗$_R$ *c*) *p*) ≤ *e* ∗ *content*(*cbox a b*)

proof −
have *sumeq*: $(\sum i{\in}p.\ |content\ i|) = sum\ content\ p$
by *simp*
have *e*: $0 \le e$
using *assms*(2) *norm_ge_zero order_trans* **by** *blast*
have *norm* (*sum* ($\lambda l.$ *content* $l *_R c$) *p*) $\le (\sum i{\in}p.\ norm\ (content\ i *_R c))$
using *norm_sum* **by** *blast*
also have ... $\le e * (\sum i{\in}p.\ |content\ i|)$
by (*simp add*: *sum_distrib_left*[*symmetric*] *mult.commute assms*(2) *mult_right_mono sum_nonneg*)
also have ... $\le e *$ *content* (*cbox a b*)
by (*metis additive_content_division p eq_iff sumeq*)
finally show *?thesis* **.**
qed

lemma *rsum_bound*:
assumes *p*: *p tagged_division_of* (*cbox a b*)
and $\forall x{\in}cbox\ a\ b.\ norm\ (f\ x) \le e$
shows *norm* (*sum* ($\lambda(x,k).$ *content* $k *_R f\ x$) *p*) $\le e *$ *content* (*cbox a b*)
proof (*cases cbox a b* = {})
case *True* **show** *?thesis*
using *p* **unfolding** *True tagged_division_of_trivial* **by** *auto*
next
case *False*
then have *e*: $e \ge 0$
by (*meson ex_in_conv assms*(2) *norm_ge_zero order_trans*)
have *sum_le*: *sum* (*content* ∘ *snd*) *p* ≤ *content* (*cbox a b*)
unfolding *additive_content_tagged_division*[*OF p*, *symmetric*] *split_def*
by (*auto intro*: *eq_refl*)
have *con*: $\bigwedge xk.\ xk \in p \Longrightarrow 0 \le content\ (snd\ xk)$
using *tagged_division_ofD*(4) [*OF p*] *content_pos_le*
by *force*
have *norm* (*sum* ($\lambda(x,k).$ *content* $k *_R f\ x$) *p*) $\le (\sum i{\in}p.\ norm\ (case\ i\ of\ (x,\ k) \Rightarrow content\ k *_R f\ x))$
by (*rule norm_sum*)
also have ... $\le e *$ *content* (*cbox a b*)
proof −
have $\bigwedge xk.\ xk \in p \Longrightarrow norm\ (f\ (fst\ xk)) \le e$
using *assms*(2) *p tag_in_interval* **by** *force*
moreover have $(\sum i{\in}p.\ |content\ (snd\ i)| * e) \le e *$ *content* (*cbox a b*)
unfolding *sum_distrib_right*[*symmetric*]
using *con sum_le* **by** (*auto simp*: *mult.commute intro*: *mult_left_mono* [*OF _ e*])
ultimately show *?thesis*
unfolding *split_def norm_scaleR*
by (*metis* (*no_types*, *lifting*) *mult_left_mono*[*OF _ abs_ge_zero*] *order_trans*[*OF sum_mono*])
qed
finally show *?thesis* **.**

qed

lemma *rsum_diff_bound*:
assumes *p tagged_division_of* (*cbox a b*)
and ∀ *x*∈*cbox a b*. *norm* (*f x* − *g x*) ≤ *e*
shows *norm* (*sum* (λ(*x*,*k*). *content k* ∗$_R$ *f x*) *p* − *sum* (λ(*x*,*k*). *content k* ∗$_R$ *g x*) *p*) ≤
e ∗ *content* (*cbox a b*)
using *order_trans*[*OF* _ *rsum_bound*[*OF assms*]]
by (*simp add*: *split_def scaleR_diff_right sum_subtractf eq_refl*)

lemma *has_integral_bound*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*real_normed_vector*
assumes *0* ≤ *B*
and *f*: (*f has_integral i*) (*cbox a b*)
and ⋀*x*. *x*∈*cbox a b* ⟹ *norm* (*f x*) ≤ *B*
shows *norm i* ≤ *B* ∗ *content* (*cbox a b*)
proof (*rule ccontr*)
assume ¬ *?thesis*
then have *norm i* − *B* ∗ *content* (*cbox a b*) > *0*
by *auto*
with *f*[*unfolded has_integral*]
obtain *γ* **where** *gauge γ* **and** *γ*:
⋀*p*. ⟦*p tagged_division_of cbox a b*; *γ fine p*⟧
⟹ *norm* ((∑ (*x*, *K*)∈*p*. *content K* ∗$_R$ *f x*) − *i*) < *norm i* − *B* ∗ *content* (*cbox a b*)
by *metis*
then obtain *p* **where** *p*: *p tagged_division_of cbox a b* **and** *γ fine p*
using *fine_division_exists* **by** *blast*
have ⋀*s B*. *norm s* ≤ *B* ⟹ ¬ *norm* (*s* − *i*) < *norm i* − *B*
unfolding *not_less*
by (*metis diff_left_mono dist_commute dist_norm norm_triangle_ineq2 order_trans*)
then show *False*
using *γ* [*OF p* ‹*γ fine p*›] *rsum_bound*[*OF p*] *assms* **by** *metis*
qed

corollary *integrable_bound*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*real_normed_vector*
assumes *0* ≤ *B*
and *f integrable_on* (*cbox a b*)
and ⋀*x*. *x*∈*cbox a b* ⟹ *norm* (*f x*) ≤ *B*
shows *norm* (*integral* (*cbox a b*) *f*) ≤ *B* ∗ *content* (*cbox a b*)
by (*metis integrable_integral has_integral_bound assms*)

6.14.8 Similar theorems about relationship among components

lemma *rsum_component_le*:

```
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes p: p tagged_division_of (cbox a b)
      and ⋀x. x ∈ cbox a b ⟹ (f x)•i ≤ (g x)•i
    shows (∑(x, K)∈p. content K *R f x) • i ≤ (∑(x, K)∈p. content K *R g x)
• i
unfolding inner_sum_left
proof (rule sum_mono, clarify)
  fix x K
  assume ab: (x, K) ∈ p
  with p obtain u v where K: K = cbox u v
    by blast
  then show (content K *R f x) • i ≤ (content K *R g x) • i
   by (metis ab assms inner_scaleR_left measure_nonneg mult_left_mono tag_in_interval)
qed

lemma has_integral_component_le:
  fixes f g :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes k: k ∈ Basis
  assumes (f has_integral i) S (g has_integral j) S
    and f_le_g: ⋀x. x ∈ S ⟹ (f x)•k ≤ (g x)•k
  shows i•k ≤ j•k
proof −
  have ik_le_jk: i•k ≤ j•k
    if f_i: (f has_integral i) (cbox a b)
    and g_j: (g has_integral j) (cbox a b)
    and le: ∀x∈cbox a b. (f x)•k ≤ (g x)•k
    for a b i and j :: 'b and f g :: 'a ⇒ 'b
  proof (rule ccontr)
    assume ¬ ?thesis
    then have *: 0 < (i•k − j•k) / 3
      by auto
    obtain γ1 where gauge γ1
      and γ1: ⋀p. ⟦p tagged_division_of cbox a b; γ1 fine p⟧
              ⟹ norm ((∑(x, k)∈p. content k *R f x) − i) < (i • k − j • k) / 3
      using f_i[unfolded has_integral,rule_format,OF *] by fastforce
    obtain γ2 where gauge γ2
      and γ2: ⋀p. ⟦p tagged_division_of cbox a b; γ2 fine p⟧
              ⟹ norm ((∑(x, k)∈p. content k *R g x) − j) < (i • k − j • k) / 3
      using g_j[unfolded has_integral,rule_format,OF *] by fastforce
    obtain p where p: p tagged_division_of cbox a b and γ1 fine p γ2 fine p
      using fine_division_exists[OF gauge_Int[OF ‹gauge γ1› ‹gauge γ2›], of a
b] unfolding fine_Int
      by metis
    then have |((∑(x, k)∈p. content k *R f x) − i) • k| < (i • k − j • k) / 3
        |((∑(x, k)∈p. content k *R g x) − j) • k| < (i • k − j • k) / 3
      using le_less_trans[OF Basis_le_norm[OF k]] k γ1 γ2 by metis+
    then show False
      unfolding inner_simps
      using rsum_component_le[OF p] le
```

```
    by (fastforce simp add: abs_real_def split: if_split_asm)
  qed
  show ?thesis
  proof (cases ∃ a b. S = cbox a b)
    case True
    with ik_le_jk assms show ?thesis
      by auto
  next
    case False
    show ?thesis
    proof (rule ccontr)
      assume ¬ i·k ≤ j·k
      then have ij: (i·k − j·k) / 3 > 0
        by auto
      obtain B1 where 0 < B1
          and B1: ⋀a b. ball 0 B1 ⊆ cbox a b ⟹
                  ∃ z. ((λx. if x ∈ S then f x else 0) has_integral z) (cbox a b) ∧
                     norm (z − i) < (i · k − j · k) / 3
        using has_integral_altD[OF _ False ij] assms by blast
      obtain B2 where 0 < B2
          and B2: ⋀a b. ball 0 B2 ⊆ cbox a b ⟹
                  ∃ z. ((λx. if x ∈ S then g x else 0) has_integral z) (cbox a b) ∧
                     norm (z − j) < (i · k − j · k) / 3
        using has_integral_altD[OF _ False ij] assms by blast
      have bounded (ball 0 B1 ∪ ball (0::'a) B2)
        unfolding bounded_Un by(rule conjI bounded_ball)+
      from bounded_subset_cbox_symmetric[OF this]
      obtain a b::'a where ab: ball 0 B1 ⊆ cbox a b ball 0 B2 ⊆ cbox a b
        by (meson Un_subset_iff)
     then obtain w1 w2 where int_w1: ((λx. if x ∈ S then f x else 0) has_integral
w1) (cbox a b)
                         and norm_w1:  norm (w1 − i) < (i · k − j · k) / 3
                          and int_w2: ((λx. if x ∈ S then g x else 0) has_integral w2)
(cbox a b)
                         and norm_w2: norm (w2 − j) < (i · k − j · k) / 3
        using B1 B2 by blast
      have ∗: ⋀w1 w2 j i::real .|w1 − i| < (i − j) / 3 ⟹ |w2 − j| < (i − j) / 3
⟹ w1 ≤ w2 ⟹ False
        by (simp add: abs_real_def split: if_split_asm)
      have |(w1 − i) · k| < (i · k − j · k) / 3
           |(w2 − j) · k| < (i · k − j · k) / 3
        using Basis_le_norm k le_less_trans norm_w1 norm_w2 by blast+
      moreover
      have w1·k ≤ w2·k
        using ik_le_jk int_w1 int_w2 f_le_g by auto
      ultimately show False
        unfolding inner_simps by(rule ∗)
    qed
  qed
```

qed

lemma *integral_component_le*:
fixes *g f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *k* ∈ *Basis*
and *f integrable_on S g integrable_on S*
and ⋀*x*. *x* ∈ *S* ⟹ (*f x*)•*k* ≤ (*g x*)•*k*
shows (*integral S f*)•*k* ≤ (*integral S g*)•*k*
using *has_integral_component_le assms* **by** *blast*

lemma *has_integral_component_nonneg*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *k* ∈ *Basis*
and (*f has_integral i*) *S*
and ⋀*x*. *x* ∈ *S* ⟹ 0 ≤ (*f x*)•*k*
shows 0 ≤ *i*•*k*
by (*metis* (*no_types*, *lifting*) *assms euclidean_all_zero_iff has_integral_0 has_integral_component_le*)

lemma *integral_component_nonneg*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *k* ∈ *Basis*
and ⋀*x*. *x* ∈ *S* ⟹ 0 ≤ (*f x*)•*k*
shows 0 ≤ (*integral S f*)•*k*
by (*smt* (*verit*, *ccfv_threshold*) *assms eq_integralD euclidean_all_zero_iff has_integral_component_no*

lemma *has_integral_component_neg*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *k* ∈ *Basis*
and (*f has_integral i*) *S*
and ⋀*x*. *x* ∈ *S* ⟹ (*f x*)•*k* ≤ 0
shows *i*•*k* ≤ 0
by (*metis* (*no_types*, *lifting*) *assms has_integral_0 has_integral_component_le inner_zero_left*)

lemma *has_integral_component_lbound*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes (*f has_integral i*) (*cbox a b*)
and ∀ *x*∈*cbox a b*. *B* ≤ *f*(*x*)•*k*
and *k* ∈ *Basis*
shows *B* ∗ *content* (*cbox a b*) ≤ *i*•*k*
using *has_integral_component_le*[*OF assms*(*3*) *has_integral_const assms*(*1*),*of* (∑ *i*∈*Basis*. *B* ∗$_R$ *i*)::*'b*] *assms*(*2*−)
by (*auto simp add*: *field_simps*)

lemma *has_integral_component_ubound*:
fixes *f*::*'a*::*euclidean_space* => *'b*::*euclidean_space*
assumes (*f has_integral i*) (*cbox a b*)
and ∀ *x*∈*cbox a b*. *f x*•*k* ≤ *B*
and *k* ∈ *Basis*

shows $i{\cdot}k \leq B * \mathit{content}\ (\mathit{cbox}\ a\ b)$
using *has_integral_component_le*[*OF assms*(3,1) *has_integral_const*, *of* $\sum i{\in}\mathit{Basis}.\ B *_R i$] *assms*(2−)
by (*auto simp add*: *field_simps*)

lemma *integral_component_lbound*:
fixes $f :: 'a{::}\mathit{euclidean\_space} \Rightarrow 'b{::}\mathit{euclidean\_space}$
assumes *f integrable_on cbox a b*
and $\forall x{\in}\mathit{cbox}\ a\ b.\ B \leq f(x){\cdot}k$
and $k \in \mathit{Basis}$
shows $B * \mathit{content}\ (\mathit{cbox}\ a\ b) \leq (\mathit{integral}(\mathit{cbox}\ a\ b)\ f){\cdot}k$
using *assms has_integral_component_lbound* **by** *blast*

lemma *integral_component_lbound_real*:
assumes *f integrable_on* $\{a :: \mathit{real}..b\}$
and $\forall x{\in}\{a..b\}.\ B \leq f(x){\cdot}k$
and $k \in \mathit{Basis}$
shows $B * \mathit{content}\ \{a..b\} \leq (\mathit{integral}\ \{a..b\}\ f){\cdot}k$
using *assms* **by** (*metis box_real*(2) *integral_component_lbound*)

lemma *integral_component_ubound*:
fixes $f :: 'a{::}\mathit{euclidean\_space} \Rightarrow 'b{::}\mathit{euclidean\_space}$
assumes *f integrable_on cbox a b*
and $\forall x{\in}\mathit{cbox}\ a\ b.\ f\ x{\cdot}k \leq B$
and $k \in \mathit{Basis}$
shows $(\mathit{integral}\ (\mathit{cbox}\ a\ b)\ f){\cdot}k \leq B * \mathit{content}\ (\mathit{cbox}\ a\ b)$
using *assms has_integral_component_ubound* **by** *blast*

lemma *integral_component_ubound_real*:
fixes $f :: \mathit{real} \Rightarrow 'a{::}\mathit{euclidean\_space}$
assumes *f integrable_on* $\{a..b\}$
and $\forall x{\in}\{a..b\}.\ f\ x{\cdot}k \leq B$
and $k \in \mathit{Basis}$
shows $(\mathit{integral}\ \{a..b\}\ f){\cdot}k \leq B * \mathit{content}\ \{a..b\}$
using *assms* **by** (*metis box_real*(2) *integral_component_ubound*)

6.14.9 Uniform limit of integrable functions is integrable

lemma *real_arch_invD*:
$0 < (e{::}\mathit{real}) \Longrightarrow (\exists n{::}\mathit{nat}.\ n \neq 0 \wedge 0 < \mathit{inverse}\ (\mathit{real}\ n) \wedge \mathit{inverse}\ (\mathit{real}\ n) < e)$
by (*subst*(*asm*) *real_arch_inverse*)

lemma *integrable_uniform_limit*:
fixes $f :: 'a{::}\mathit{euclidean\_space} \Rightarrow 'b{::}\mathit{banach}$
assumes $\bigwedge e.\ e > 0 \Longrightarrow \exists g.\ (\forall x{\in}\mathit{cbox}\ a\ b.\ \mathit{norm}\ (f\ x - g\ x) \leq e) \wedge g$ *integrable_on cbox a b*
shows *f integrable_on cbox a b*
proof (*cases content* (*cbox a b*) > 0)

case *False* **then show** *?thesis*
using *has_integral_null* **by** (*simp add*: *content_lt_nz integrable_on_def*)
next
case *True*
have *1 / (real n + 1) > 0* **for** *n*
by *auto*
then have $\exists g.\ (\forall x{\in}cbox\ a\ b.\ norm\ (f\ x - g\ x) \leq 1\ /\ (real\ n + 1)) \wedge g\ integrable\_on\ cbox\ a\ b$ **for** *n*
using *assms* **by** *blast*
then obtain *g* **where** *g_near_f*: $\bigwedge n\ x.\ x \in cbox\ a\ b \Longrightarrow norm\ (f\ x - g\ n\ x) \leq 1\ /\ (real\ n + 1)$
and *int_g*: $\bigwedge n.\ g\ n\ integrable\_on\ cbox\ a\ b$
by *metis*
then obtain *h* **where** *h*: $\bigwedge n.\ (g\ n\ has\_integral\ h\ n)\ (cbox\ a\ b)$
unfolding *integrable_on_def* **by** *metis*
have *Cauchy h*
unfolding *Cauchy_def*
proof *clarify*
fix *e* :: *real*
assume *e>0*
then have *e/4 / content (cbox a b) > 0*
using *True* **by** (*auto simp*: *field_simps*)
then obtain *M* **where** $M \neq 0$ **and** *M*: *1 / (real M) < e/4 / content (cbox a b)*
by (*metis inverse_eq_divide real_arch_inverse*)
show $\exists M.\ \forall m{\geq}M.\ \forall n{\geq}M.\ dist\ (h\ m)\ (h\ n) < e$
proof (*intro exI strip*)
fix *m n*
assume *m*: $M \leq m$ **and** *n*: $M \leq n$
have *e/4>0* **using** ‹*e>0*› **by** *auto*
then obtain *gm gn* **where** *gauge gm gauge gn*
and *gm*: $\bigwedge \mathcal{D}.\ \mathcal{D}\ tagged\_division\_of\ cbox\ a\ b \wedge gm\ fine\ \mathcal{D} \Longrightarrow norm\ ((\sum (x,K) \in \mathcal{D}.\ content\ K *_R g\ m\ x) - h\ m) < e/4$
and *gn*: $\bigwedge \mathcal{D}.\ \mathcal{D}\ tagged\_division\_of\ cbox\ a\ b \wedge gn\ fine\ \mathcal{D} \Longrightarrow norm\ ((\sum (x,K) \in \mathcal{D}.\ content\ K *_R g\ n\ x) - h\ n) < e/4$
using *h*[*unfolded has_integral*] **by** *meson*
then obtain $\mathcal{D}$ **where** $\mathcal{D}$: $\mathcal{D}\ tagged\_division\_of\ cbox\ a\ b\ (\lambda x.\ gm\ x \cap gn\ x)\ fine\ \mathcal{D}$
by (*metis* (*full_types*) *fine_division_exists gauge_Int*)
have *triangle3*: *norm (i1 − i2) < e*
if *no*: *norm(s2 − s1) ≤ e/2 norm (s1 − i1) < e/4 norm (s2 − i2) < e/4* **for** *s1 s2 i1* **and** *i2*::*'b*
proof −
have *norm (i1 − i2) ≤ norm (i1 − s1) + norm (s1 − s2) + norm (s2 − i2)*
using *norm_triangle_ineq*[*of i1 − s1 s1 − i2*]
using *norm_triangle_ineq*[*of s1 − s2 s2 − i2*]
by (*auto simp*: *algebra_simps*)

```
      also have ... < e
        using no by (auto simp: algebra_simps norm_minus_commute)
      finally show ?thesis .
    qed
    have finep: gm fine 𝒟 gn fine 𝒟
      using fine_Int 𝒟 by auto
    have norm_le: norm (g n x - g m x) ≤ 2 / real M if x: x ∈ cbox a b for x
    proof -
      have norm (f x - g n x) + norm (f x - g m x) ≤ 1 / (real n + 1) + 1 /
(real m + 1)
        using g_near_f[OF x, of n] g_near_f[OF x, of m] by simp
      also have ... ≤ 1 / (real M) + 1 / (real M)
        using ‹M ≠ 0› m n by (intro add_mono; force simp: field_split_simps)
      also have ... = 2 / real M
        by auto
      finally show norm (g n x - g m x) ≤ 2 / real M
        using norm_triangle_le[of g n x - f x f x - g m x 2 / real M]
        by (auto simp: algebra_simps simp add: norm_minus_commute)
    qed
    have norm ((∑(x,K) ∈ 𝒟. content K *R g n x) - (∑(x,K) ∈ 𝒟. content K
*R g m x)) ≤ 2 / real M * content (cbox a b)
      by (blast intro: norm_le rsum_diff_bound[OF 𝒟(1), where e=2 / real M])
    also have ... ≤ e/2
      using M True
      by (auto simp: field_simps)
    finally have le_e2: norm ((∑(x,K) ∈ 𝒟. content K *R g n x) - (∑(x,K)
∈ 𝒟. content K *R g m x)) ≤ e/2 .
    then show dist (h m) (h n) < e
      unfolding dist_norm using gm gn 𝒟 finep by (auto intro!: triangle3)
  qed
 qed
 then obtain s where s: h ⟶ s
  using convergent_eq_Cauchy[symmetric] by blast
 show ?thesis
  unfolding integrable_on_def has_integral
 proof (rule_tac x=s in exI, clarify)
  fix e::real
  assume e: 0 < e
  then have e/3 > 0 by auto
  then obtain N1 where N1: ∀n≥N1. norm (h n - s) < e/3
   using LIMSEQ_D [OF s] by metis
  from e True have e/3 / content (cbox a b) > 0
   by (auto simp: field_simps)
  then obtain N2 :: nat
      where N2 ≠ 0 and N2: 1 / (real N2) < e/3 / content (cbox a b)
   by (metis inverse_eq_divide real_arch_inverse)
  obtain g' where gauge g'
        and g': ⋀𝒟. 𝒟 tagged_division_of cbox a b ∧ g' fine 𝒟 ⟹
            norm ((∑(x,K) ∈ 𝒟. content K *R g (N1 + N2) x) - h (N1 +
```

```
N2)) < e/3
      by (metis h has_integral ‹e/3 > 0›)
    have *: norm (sf − s) < e
      if no: norm (sf − sg) ≤ e/3 norm(h − s) < e/3 norm (sg − h) < e/3 for
sf sg h
    proof −
      have norm (sf − s) ≤ norm (sf − sg) + norm (sg − h) + norm (h − s)
        using norm_triangle_ineq[of sf − sg sg − s]
        using norm_triangle_ineq[of sg −  h  h − s]
        by (auto simp: algebra_simps)
      also have ... < e
        using no by (auto simp: algebra_simps norm_minus_commute)
      finally show ?thesis .
    qed
    { fix D
      assume ptag: D tagged_division_of (cbox a b) and g' fine D
      then have norm_less: norm ((∑(x,K) ∈ D. content K *R g (N1 + N2) x)
− h (N1 + N2)) < e/3
        using g' by blast
      have content (cbox a b) < e/3 * (of_nat N2)
        using ‹N2 ≠ 0› N2 using True by (auto simp: field_split_simps)
      moreover have e/3 * of_nat N2 ≤ e/3 * (of_nat (N1 + N2) + 1)
        using ‹e>0› by auto
      ultimately have content (cbox a b) < e/3 * (of_nat (N1 + N2) + 1)
        by linarith
      then have le_e3: 1 / (real (N1 + N2) + 1) * content (cbox a b) ≤ e/3
        unfolding inverse_eq_divide
        by (auto simp: field_simps)
      have ne3: norm (h (N1 + N2) − s) < e/3
        using N1 by auto
      have norm ((∑(x,K) ∈ D. content K *R f x) − (∑(x,K) ∈ D. content K
*R g (N1 + N2) x))
            ≤ 1 / (real (N1 + N2) + 1) * content (cbox a b)
        by (blast intro: g_near_f rsum_diff_bound[OF ptag])
      then have norm ((∑(x,K) ∈ D. content K *R f x) − s) < e
        by (rule *[OF order_trans [OF _ le_e3] ne3 norm_less])
    }
    then show ∃ d. gauge d ∧
          (∀ D. D tagged_division_of cbox a b ∧ d fine D ⟶ norm ((∑(x,K) ∈
D. content K *R f x) − s) < e)
      by (blast intro: g' ‹gauge g'›)
  qed
qed

lemmas integrable_uniform_limit_real = integrable_uniform_limit [where 'a=real,
simplified]
```

6.14.10 Negligible sets

definition *negligible* (*s*:: *'a*::*euclidean_space set*) ⟷
(∀ *a b*. ((*indicator s* :: *'a*⇒*real*) *has_integral 0*) (*cbox a b*))

Negligibility of hyperplane

lemma *content_doublesplit*:
fixes *a* :: *'a*::*euclidean_space*
assumes *0* < *e*
and *k*: *k* ∈ *Basis*
obtains *d* **where** *0* < *d* **and** *content* (*cbox a b* ∩ {*x*. |*x*•*k* − *c*| ≤ *d*}) < *e*
proof *cases*
assume ∗: *a* • *k* ≤ *c* ∧ *c* ≤ *b* • *k* ∧ (∀ *j*∈*Basis*. *a* • *j* ≤ *b* • *j*)
define *a'* **where** *a' d* = (∑ *j*∈*Basis*. (*if j* = *k then max* (*a*•*j*) (*c* − *d*) *else a*•*j*) ∗*R j*) **for** *d*
define *b'* **where** *b' d* = (∑ *j*∈*Basis*. (*if j* = *k then min* (*b*•*j*) (*c* + *d*) *else b*•*j*) ∗*R j*) **for** *d*

have ((λ*d*. ∏ *j*∈*Basis*. (*b' d* − *a' d*) • *j*) ⟶ (∏ *j*∈*Basis*. (*b' 0* − *a' 0*) • *j*)) (*at_right 0*)
by (*auto simp*: *b'_def a'_def intro*!: *tendsto_min tendsto_max tendsto_eq_intros*)
also have (∏ *j*∈*Basis*. (*b' 0* − *a' 0*) • *j*) = *0*
using *k* ∗
by (*intro prod_zero bexI*[*OF* _ *k*])
(*auto simp*: *b'_def a'_def inner_diff inner_sum_left inner_not_same_Basis intro*!: *sum.cong*)
also have ((λ*d*. ∏ *j*∈*Basis*. (*b' d* − *a' d*) • *j*) ⟶ *0*) (*at_right 0*) =
((λ*d*. *content* (*cbox a b* ∩ {*x*. |*x*•*k* − *c*| ≤ *d*})) ⟶ *0*) (*at_right 0*)
proof (*intro tendsto_cong eventually_at_rightI*)
fix *d* :: *real* **assume** *d*: *d* ∈ {*0*<..<*1*}
have *cbox a b* ∩ {*x*. |*x*•*k* − *c*| ≤ *d*} = *cbox* (*a' d*) (*b' d*) **for** *d*
using ∗ *d k* **by** (*auto simp add*: *cbox_def set_eq_iff Int_def ball_conj_distrib abs_diff_le_iff a'_def b'_def*)
moreover have *j* ∈ *Basis* ⟹ *a' d* • *j* ≤ *b' d* • *j* **for** *j*
using ∗ *d k* **by** (*auto simp*: *a'_def b'_def*)
ultimately show (∏ *j*∈*Basis*. (*b' d* − *a' d*) • *j*) = *content* (*cbox a b* ∩ {*x*. |*x*•*k* − *c*| ≤ *d*})
by *simp*
qed *simp*
finally have ((λ*d*. *content* (*cbox a b* ∩ {*x*. |*x* • *k* − *c*| ≤ *d*})) ⟶ *0*) (*at_right 0*) .
from *order_tendstoD*(*2*)[*OF this* ‹*0*<*e*›]
obtain *d'* **where** *0* < *d'* **and** *d'*: ⋀*y*. *y* > *0* ⟹ *y* < *d'* ⟹ *content* (*cbox a b* ∩ {*x*. |*x* • *k* − *c*| ≤ *y*}) < *e*
by (*subst* (*asm*) *eventually_at_right*[*of* _ *1*]) *auto*
show *?thesis*
by (*rule that*[*of d'*/*2*], *insert* ‹*0*<*d'*› *d'*[*of d'*/*2*], *auto*)
next
assume ∗: ¬ (*a* • *k* ≤ *c* ∧ *c* ≤ *b* • *k* ∧ (∀ *j*∈*Basis*. *a* • *j* ≤ *b* • *j*))

```
  then have (∃j∈Basis. b • j < a • j) ∨ (c < a • k ∨ b • k < c)
    by (auto simp: not_le)
  show thesis
  proof cases
    assume ∃j∈Basis. b • j < a • j
    then have [simp]: cbox a b = {}
      using box_ne_empty(1)[of a b] by auto
    show ?thesis
      by (rule that[of 1]) (simp_all add: ‹0<e›)
  next
    assume ¬ (∃j∈Basis. b • j < a • j)
    with ∗ have c < a • k ∨ b • k < c
      by auto
    then show thesis
    proof
      assume c: c < a • k
      moreover have x ∈ cbox a b ⟹ c ≤ x • k for x
        using k c by (auto simp: cbox_def)
      ultimately have cbox a b ∩ {x. |x • k − c| ≤ (a • k − c)/2} = {}
        using k by (auto simp: cbox_def)
      with ‹0<e› c that[of (a • k − c)/2] show ?thesis
        by auto
    next
      assume c: b • k < c
      moreover have x ∈ cbox a b ⟹ x • k ≤ c for x
        using k c by (auto simp: cbox_def)
      ultimately have cbox a b ∩ {x. |x • k − c| ≤ (c − b • k)/2} = {}
        using k by (auto simp: cbox_def)
      with ‹0<e› c that[of (c − b • k)/2] show ?thesis
        by auto
    qed
  qed
qed


proposition negligible_standard_hyperplane[intro]:
  fixes k :: 'a::euclidean_space
  assumes k: k ∈ Basis
  shows negligible {x. x•k = c}
  unfolding negligible_def has_integral
proof clarsimp
  fix a b and e::real assume e > 0
  with k obtain d where 0 < d and d: content (cbox a b ∩ {x. |x • k − c| ≤ d})
< e
    by (metis content_doublesplit)
  let ?i = indicator {x::'a. x•k = c} :: 'a⇒real
  show ∃γ. gauge γ ∧
          (∀D. D tagged_division_of cbox a b ∧ γ fine D ⟶
               |∑(x,K) ∈ D. content K ∗ ?i x| < e)
```

```
  proof (intro exI, safe)
    show gauge (λx. ball x d)
      using ‹0 < d› by blast
  next
    fix 𝒟
    assume p: 𝒟 tagged_division_of (cbox a b) (λx. ball x d) fine 𝒟
    have content L = content (L ∩ {x. |x · k − c| ≤ d})
      if (x, L) ∈ 𝒟 ?i x ≠ 0 for x L
    proof −
      have xk: x·k = c
        using that by (simp add: indicator_def split: if_split_asm)
      have L ⊆ {x. |x · k − c| ≤ d}
      proof
        fix y
        assume y: y ∈ L
        have L ⊆ ball x d
          using p(2) that(1) by auto
        then have norm (x − y) < d
          by (simp add: dist_norm subset_iff y)
        then have |(x − y) · k| < d
          using k norm_bound_Basis_lt by blast
        then show y ∈ {x. |x · k − c| ≤ d}
          unfolding inner_simps xk by auto
      qed
      then show content L = content (L ∩ {x. |x · k − c| ≤ d})
        by (metis inf.orderE)
    qed
    then have ∗: (∑(x,K)∈𝒟. content K ∗ ?i x) = (∑(x,K)∈𝒟. content (K ∩
{x. |x·k − c| ≤ d}) ∗R ?i x)
      by (force simp add: split_paired_all intro!: sum.cong [OF refl])
   note p'= tagged_division_ofD[OF p(1)] and p''=division_of_tagged_division[OF
p(1)]
    have (∑(x,K)∈𝒟. content (K ∩ {x. |x · k − c| ≤ d}) ∗ indicator {x. x · k =
c} x) < e
    proof −
     have (∑(x,K)∈𝒟. content (K ∩ {x. |x · k − c| ≤ d}) ∗ ?i x) ≤ (∑(x,K)∈𝒟.
content (K ∩ {x. |x · k − c| ≤ d}))
       by (force simp add: indicator_def intro!: sum_mono)
      also have ... < e
      proof (subst sum.over_tagged_division_lemma[OF p(1)])
        fix u v::'a
        assume box u v = {}
        then have ∗: content (cbox u v) = 0
          unfolding content_eq_0_interior by simp
        have cbox u v ∩ {x. |x · k − c| ≤ d} ⊆ cbox u v
          by auto
        then have content (cbox u v ∩ {x. |x · k − c| ≤ d}) ≤ content (cbox u v)
          unfolding interval_doublesplit[OF k] by (rule content_subset)
        then show content (cbox u v ∩ {x. |x · k − c| ≤ d}) = 0
```

```
        unfolding ∗ interval_doublesplit[OF k]
        by (blast intro: antisym)
      next
       have (∑ l∈snd ‘ 𝒟. content (l ∩ {x. |x · k − c| ≤ d})) =
        sum content ((λl. l ∩ {x. |x · k − c| ≤ d}) ‘{l∈snd ‘ 𝒟. l ∩ {x. |x · k − c|
≤ d} ≠ {}})
        proof (subst (2) sum.reindex_nontrivial)
         fix x y assume x ∈ {l ∈ snd ‘ 𝒟. l ∩ {x. |x · k − c| ≤ d} ≠ {}} y ∈ {l
∈ snd ‘ 𝒟. l ∩ {x. |x · k − c| ≤ d} ≠ {}}
           x ≠ y and eq: x ∩ {x. |x · k − c| ≤ d} = y ∩ {x. |x · k − c| ≤ d}
         then obtain x′ y′ where (x′, x) ∈ 𝒟 x ∩ {x. |x · k − c| ≤ d} ≠ {} (y′,
y) ∈ 𝒟 y ∩ {x. |x · k − c| ≤ d} ≠ {}
           by (auto)
         from p′(5)[OF ‹(x′, x) ∈ 𝒟› ‹(y′, y) ∈ 𝒟›] ‹x ≠ y› have interior (x ∩ y)
= {}
           by auto
         moreover have interior ((x ∩ {x. |x · k − c| ≤ d}) ∩ (y ∩ {x. |x · k −
c| ≤ d})) ⊆ interior (x ∩ y)
           by (auto intro: interior_mono)
         ultimately have interior (x ∩ {x. |x · k − c| ≤ d}) = {}
           by (auto simp: eq)
         then show content (x ∩ {x. |x · k − c| ≤ d}) = 0
          using p′(4)[OF ‹(x′, x) ∈ 𝒟›] by (auto simp: interval_doublesplit[OF k]
content_eq_0_interior simp del: interior_Int)
        qed (insert p′(1), auto intro!: sum.mono_neutral_right)
        also have ... ≤ norm (∑ l∈(λl. l ∩ {x. |x · k − c| ≤ d}) ‘{l∈snd ‘ 𝒟. l ∩
{x. |x · k − c| ≤ d} ≠ {}}. content l ∗R 1::real)
         by simp
        also have ... ≤ 1 ∗ content (cbox a b ∩ {x. |x · k − c| ≤ d})
         using division_doublesplit[OF p′′ k, unfolded interval_doublesplit[OF k]]
         unfolding interval_doublesplit[OF k] by (intro dsum_bound) auto
        also have ... < e
         using d by simp
        finally show (∑ K∈snd ‘ 𝒟. content (K ∩ {x. |x · k − c| ≤ d})) < e .
      qed
      finally show (∑ (x, K)∈𝒟. content (K ∩ {x. |x · k − c| ≤ d}) ∗ ?i x) < e .
    qed
    then show |∑ (x, K)∈𝒟. content K ∗ ?i x| < e
      unfolding ∗ by (simp add: sum_nonneg split: prod.split)
  qed
qed

corollary negligible_standard_hyperplane_cart:
  fixes k :: 'a::finite
  shows negligible {x. x$k = (0::real)}
  by (simp add: cart_eq_inner_axis negligible_standard_hyperplane)
```

Hence the main theorem about negligible sets

lemma *has_integral_negligible_cbox*:
fixes $f :: 'b::euclidean\_space \Rightarrow 'a::real\_normed\_vector$
assumes *negs*: *negligible S*
and *0*: $\bigwedge x.\ x \notin S \Longrightarrow f\ x = 0$
shows $(f\ has\_integral\ 0)\ (cbox\ a\ b)$
unfolding *has_integral*
proof *clarify*
fix *e*::*real*
assume $e > 0$
then have *nn_gt0*: $e/2\ /\ ((real\ n+1) * (2\ \hat{}\ n)) > 0$ **for** *n*
by *simp*
then have $\exists \gamma.\ gauge\ \gamma \wedge$
$(\forall \mathcal{D}.\ \mathcal{D}\ tagged\_division\_of\ cbox\ a\ b \wedge \gamma\ fine\ \mathcal{D} \longrightarrow$
$|\sum (x,K) \in \mathcal{D}.\ content\ K *_R indicator\ S\ x|$
$< e/2\ /\ ((real\ n + 1) * 2\ \hat{}\ n))$ **for** *n*
using *negs* [*unfolded negligible_def has_integral*] **by** *auto*
then obtain γ **where**
gd: $\bigwedge n.\ gauge\ (\gamma\ n)$
and γ: $\bigwedge n\ \mathcal{D}.\ [\![\mathcal{D}\ tagged\_division\_of\ cbox\ a\ b; \gamma\ n\ fine\ \mathcal{D}]\!]$
$\Longrightarrow |\sum (x,K) \in \mathcal{D}.\ content\ K *_R indicator\ S\ x| < e/2\ /\ ((real\ n +$
$1) * 2\ \hat{}\ n)$
by *metis*
show $\exists \gamma.\ gauge\ \gamma \wedge$
$(\forall \mathcal{D}.\ \mathcal{D}\ tagged\_division\_of\ cbox\ a\ b \wedge \gamma\ fine\ \mathcal{D} \longrightarrow$
$norm\ ((\sum (x,K) \in \mathcal{D}.\ content\ K *_R f\ x) - 0) < e)$
proof (*intro exI*, *safe*)
show $gauge\ (\lambda x.\ \gamma\ (nat\ \lfloor norm\ (f\ x)\rfloor)\ x)$
using *gd* **by** (*auto simp*: *gauge_def*)

show $norm\ ((\sum (x,K) \in \mathcal{D}.\ content\ K *_R f\ x) - 0) < e$
if $\mathcal{D}\ tagged\_division\_of\ (cbox\ a\ b)\ (\lambda x.\ \gamma\ (nat\ \lfloor norm\ (f\ x)\rfloor)\ x)\ fine\ \mathcal{D}$ **for** $\mathcal{D}$
proof (*cases* $\mathcal{D} = \{\}$)
case *True* **with** ‹*0* < *e*› **show** *?thesis* **by** *simp*
next
case *False*
obtain *N* **where** $Max\ ((\lambda(x,\ K).\ norm\ (f\ x))\ `\ \mathcal{D}) \le real\ N$
using *real_arch_simple* **by** *blast*
then have *N*: $\bigwedge x.\ x \in (\lambda(x,\ K).\ norm\ (f\ x))\ `\ \mathcal{D} \Longrightarrow x \le real\ N$
by (*meson Max_ge that*(*1*) *dual_order.trans finite_imageI tagged_division_of_finite*)
have $\forall i.\ \exists q.\ q\ tagged\_division\_of\ (cbox\ a\ b) \wedge (\gamma\ i)\ fine\ q \wedge (\forall (x,K) \in \mathcal{D}.$
$K \subseteq (\gamma\ i)\ x \longrightarrow (x,\ K) \in q)$
by (*auto intro*: *tagged_division_finer*[*OF that*(*1*) *gd*])
from *choice*[*OF this*]
obtain *q* **where** *q*: $\bigwedge n.\ q\ n\ tagged\_division\_of\ cbox\ a\ b$
$\bigwedge n.\ \gamma\ n\ fine\ q\ n$
$\bigwedge n\ x\ K.\ [\![(x,\ K) \in \mathcal{D};\ K \subseteq \gamma\ n\ x]\!] \Longrightarrow (x,\ K) \in q\ n$
by *fastforce*
have *finite* $\mathcal{D}$

```
      using that(1) by blast
    then have sum_le_inc: ⟦finite T; ⋀x y. (x,y) ∈ T ⟹ (0::real) ≤ g(x,y);
                ⋀y. y∈𝒟 ⟹ ∃x. (x,y) ∈ T ∧ f(y) ≤ g(x,y)⟧ ⟹ sum f 𝒟 ≤
sum g T for f g T
      by (rule sum_le_included[of 𝒟 T g snd f]; force)
    have norm (∑(x,K) ∈ 𝒟. content K *R f x) ≤ (∑(x,K) ∈ 𝒟. norm (content
K *R f x))
      unfolding split_def by (rule norm_sum)
    also have ... ≤ (∑(i, j) ∈ Sigma {..N + 1} q.
                  (real i + 1) * (case j of (x, K) ⇒ content K *R indicator S
x))
    proof (rule sum_le_inc, safe)
      show finite (Sigma {..N+1} q)
        by (meson finite_SigmaI finite_atMost tagged_division_of_finite q(1))
    next
      fix x K
      assume xk: (x, K) ∈ 𝒟
      define n where n = nat ⌊norm (f x)⌋
      have *: norm (f x) ∈ (λ(x, K). norm (f x)) ` 𝒟
        using xk by auto
      have nfx: real n ≤ norm (f x) norm (f x) ≤ real n + 1
        unfolding n_def by auto
      then have n ∈ {0..N + 1}
        using N[OF *] by auto
      moreover have K ⊆ γ (nat ⌊norm (f x)⌋) x
        using that(2) xk by auto
      moreover then have (x, K) ∈ q (nat ⌊norm (f x)⌋)
        by (simp add: q(3) xk)
      moreover then have (x, K) ∈ q n
        using n_def by blast
      moreover
      have norm (content K *R f x) ≤ (real n + 1) * (content K * indicator S x)
      proof (cases x ∈ S)
        case False
        then show ?thesis by (simp add: 0)
      next
        case True
        have *: content K ≥ 0
          using tagged_division_ofD(4)[OF that(1) xk] by auto
        moreover have content K * norm (f x) ≤ content K * (real n + 1)
          by (simp add: mult_left_mono nfx(2))
        ultimately show ?thesis
          using nfx True by (auto simp: field_simps)
      qed
      ultimately show ∃y. (y, x, K) ∈ (Sigma {..N + 1} q) ∧ norm (content
K *R f x) ≤
        (real y + 1) * (content K *R indicator S x)
        by force
    qed auto
```

```
    also have ... = (∑ i≤N + 1. ∑ j∈q i. (real i + 1) * (case j of (x, K) ⇒
content K *R indicator S x))
      using q(1) by (intro sum_Sigma_product [symmetric]) auto
    also have ... ≤ (∑ i≤N + 1. (real i + 1) * |∑ (x,K) ∈ q i. content K *R
indicator S x|)
      by (rule sum_mono) (simp add: sum_distrib_left [symmetric])
    also have ... ≤ (∑ i≤N + 1. e/2/2 ^ i)
    proof (rule sum_mono)
      show (real i + 1) * |∑ (x,K) ∈ q i. content K *R indicator S x| ≤ e/2/2
^ i
        if i ∈ {..N + 1} for i
        using γ[of q i i] q by (simp add: divide_simps mult.left_commute)
    qed
    also have ... = e/2 * (∑ i≤N + 1. (1/2) ^ i)
      unfolding sum_distrib_left by (metis divide_inverse inverse_eq_divide
power_one_over)
    also have ... < e/2 * 2
    proof (rule mult_strict_left_mono)
      have sum (power (1/2)) {..N + 1} = sum (power (1/2::real)) {..<N +
2}
        using lessThan_Suc_atMost by auto
      also have ... < 2
        by (auto simp: geometric_sum)
      finally show sum (power (1/2::real)) {..N + 1} < 2 .
    qed (use ‹0 < e› in auto)
    finally  show ?thesis by auto
  qed
 qed
qed


proposition has_integral_negligible:
 fixes f :: 'b::euclidean_space ⇒ 'a::real_normed_vector
 assumes negs: negligible S
   and ⋀x. x ∈ (T - S) ⟹ f x = 0
 shows (f has_integral 0) T
proof (cases ∃ a b. T = cbox a b)
 case True
 then have ((λx. if x ∈ T then f x else 0) has_integral 0) T
   using assms by (auto intro!: has_integral_negligible_cbox)
 then show ?thesis
   by (rule has_integral_eq [rotated]) auto
next
 case False
 let ?f = (λx. if x ∈ T then f x else 0)
 have ((λx. if x ∈ T then f x else 0) has_integral 0) T
   apply (auto simp: False has_integral_alt [of ?f])
   apply (rule_tac x=1 in exI, auto)
  apply (rule_tac x=0 in exI, simp add: has_integral_negligible_cbox [OF negs]
```

assms)
  done
 then show *?thesis*
  by (*rule_tac f=?f* **in** *has_integral_eq*) *auto*
qed

lemma
 assumes *negligible S*
 shows *integrable_negligible*: *f integrable_on S* **and** *integral_negligible*: *integral S f = 0*
 using *has_integral_negligible* [*OF assms*]
 by (*auto simp*: *has_integral_iff*)

lemma *has_integral_spike*:
 fixes *f* :: *'b*::*euclidean_space* ⇒ *'a*::*real_normed_vector*
 assumes *negligible S*
  and *gf*: ⋀*x*. *x* ∈ *T* − *S* ⟹ *g x* = *f x*
  and *fint*: (*f has_integral y*) *T*
 shows (*g has_integral y*) *T*
proof −
 have ∗: (*g has_integral y*) (*cbox a b*)
     if (*f has_integral y*) (*cbox a b*) ∀ *x* ∈ *cbox a b* − *S*. *g x* = *f x* **for** *a b f* **and** *g*:: *'b* ⇒ *'a* **and** *y*
 proof −
  have ((λ*x*. *f x* + (*g x* − *f x*)) *has_integral* (*y* + *0*)) (*cbox a b*)
   using *that* **by** (*intro has_integral_add has_integral_negligible*) (*auto intro*!: ‹*negligible S*›)
  then show *?thesis*
   by *auto*
 qed
 have §: ⋀*a b z*. ⟦⋀*x*. *x* ∈ *T* ∧ *x* ∉ *S* ⟹ *g x* = *f x*;
                ((λ*x*. *if x* ∈ *T then f x else 0*) *has_integral z*) (*cbox a b*)⟧
               ⟹ ((λ*x*. *if x* ∈ *T then g x else 0*) *has_integral z*) (*cbox a b*)
   by (*auto dest*!: ∗[**where** *f*=λ*x*. *if x*∈*T then f x else 0* **and** *g*=λ*x*. *if x* ∈ *T then g x else 0*])
 show *?thesis*
  using *fint gf*
  apply (*subst has_integral_alt*)
  apply (*subst* (*asm*) *has_integral_alt*)
  apply (*auto split*: *if_split_asm*)
   apply (*blast dest*: ∗)
  using § **by** *meson*
qed

lemma *has_integral_spike_eq*:
 assumes *negligible S* **and** ⋀*x*. *x* ∈ *T* − *S* ⟹ *g x* = *f x*
 shows (*f has_integral y*) *T* ⟷ (*g has_integral y*) *T*
 by (*metis assms has_integral_spike*)

lemma *integrable_spike*:
assumes *f integrable_on T negligible S* $\bigwedge x.\ x \in T - S \Longrightarrow g\ x = f\ x$
shows *g integrable_on T*
using *assms* **unfolding** *integrable_on_def* **by** (*blast intro*: *has_integral_spike*)

lemma *integral_spike*:
assumes *negligible S*
and $\bigwedge x.\ x \in T - S \Longrightarrow g\ x = f\ x$
shows *integral T f* = *integral T g*
using *has_integral_spike_eq*[*OF assms*]
by (*auto simp*: *integral_def integrable_on_def*)

6.14.11 Some other trivialities about negligible sets

lemma *negligible_subset*:
assumes *negligible s t* $\subseteq$ *s*
shows *negligible t*
unfolding *negligible_def*
by (*metis* (*no_types*) *Diff_iff assms contra_subsetD has_integral_negligible indicator_simps*(*2*))

lemma *negligible_diff*[*intro?*]:
assumes *negligible s*
shows *negligible* (*s* − *t*)
using *assms* **by** (*meson Diff_subset negligible_subset*)

lemma *negligible_Int*:
assumes *negligible s* ∨ *negligible t*
shows *negligible* (*s* ∩ *t*)
using *assms negligible_subset* **by** *force*

lemma *negligible_Un*:
assumes *negligible S* **and** *T*: *negligible T*
shows *negligible* (*S* ∪ *T*)
proof −
have (*indicat_real* (*S* ∪ *T*) *has_integral 0*) (*cbox a b*)
if *S0*: (*indicat_real S has_integral 0*) (*cbox a b*)
and (*indicat_real T has_integral 0*) (*cbox a b*) **for** *a b*
proof (*subst has_integral_spike_eq*[*OF T*])
show *indicat_real S x* = *indicat_real* (*S* ∪ *T*) *x* **if** *x* ∈ *cbox a b* − *T* **for** *x*
using *that* **by** (*simp add*: *indicator_def*)
show (*indicat_real S has_integral 0*) (*cbox a b*)
by (*simp add*: *S0*)
qed
with *assms* **show** *?thesis*
unfolding *negligible_def* **by** *blast*
qed

lemma *negligible_Un_eq*[*simp*]: *negligible* (*s* ∪ *t*) ⟷ *negligible s* ∧ *negligible t*

using *negligible_Un negligible_subset* **by** *blast*

lemma *negligible_sing*[*intro*]: *negligible* {*a*::'*a*::*euclidean_space*}
using *negligible_standard_hyperplane*[*OF SOME_Basis, of a* · (*SOME i. i* ∈ *Basis*)] *negligible_subset* **by** *blast*

lemma *negligible_insert*[*simp*]: *negligible* (*insert a s*) ⟷ *negligible s*
by (*metis insert_is_Un negligible_Un_eq negligible_sing*)

lemma *negligible_empty*[*iff*]: *negligible* {}
using *negligible_insert* **by** *blast*

Useful in this form for backchaining

lemma *empty_imp_negligible*: *S* = {} ⟹ *negligible S*
by *simp*

lemma *negligible_finite*[*intro*]:
assumes *finite s*
shows *negligible s*
using *assms* **by** (*induct s*) *auto*

lemma *negligible_Union*[*intro*]:
assumes *finite* $\mathcal{T}$
and ⋀*t*. *t* ∈ $\mathcal{T}$ ⟹ *negligible t*
shows *negligible*(⋃ $\mathcal{T}$)
using *assms* **by** *induct auto*

lemma *negligible*: *negligible S* ⟷ (∀ *T*. (*indicat_real S has_integral 0*) *T*)
proof (*intro iffI allI*)
fix *T*
assume *negligible S*
then show (*indicator S has_integral 0*) *T*
by (*meson Diff_iff has_integral_negligible indicator_simps*(*2*))
qed (*simp add*: *negligible_def*)

6.14.12 Finite case of the spike theorem is quite commonly needed

lemma *has_integral_spike_finite*:
assumes *finite S*
and ⋀*x*. *x* ∈ *T* − *S* ⟹ *g x* = *f x*
and (*f has_integral y*) *T*
shows (*g has_integral y*) *T*
using *assms has_integral_spike negligible_finite* **by** *blast*

lemma *has_integral_spike_finite_eq*:
assumes *finite S*
and ⋀*x*. *x* ∈ *T* − *S* ⟹ *g x* = *f x*
shows ((*f has_integral y*) *T* ⟷ (*g has_integral y*) *T*)

by (*metis assms has_integral_spike_finite*)

lemma *integrable_spike_finite*:
assumes *finite S*
and $\bigwedge x.\ x \in T - S \Longrightarrow g\ x = f\ x$
and *f integrable_on T*
shows *g integrable_on T*
using *assms has_integral_spike_finite* **by** *blast*

lemma *has_integral_bound_spike_finite*:
fixes *f* :: *'a::euclidean_space* ⇒ *'b::real_normed_vector*
assumes $0 \le B$ *finite S*
and *f*: (*f has_integral i*) (*cbox a b*)
and *leB*: $\bigwedge x.\ x \in cbox\ a\ b - S \Longrightarrow norm\ (f\ x) \le B$
shows *norm i* ≤ *B* ∗ *content* (*cbox a b*)
proof −
define *g* **where** $g \equiv (\lambda x.\ if\ x \in S\ then\ 0\ else\ f\ x)$
then have $\bigwedge x.\ x \in cbox\ a\ b - S \Longrightarrow norm\ (g\ x) \le B$
using *leB* **by** *simp*
moreover have (*g has_integral i*) (*cbox a b*)
using *has_integral_spike_finite* [*OF* ‹*finite S*› _ *f*]
by (*simp add*: *g_def*)
ultimately show *?thesis*
by (*simp add*: ‹*0* ≤ *B*› *g_def has_integral_bound*)
qed

corollary *has_integral_bound_real*:
fixes *f* :: *real* ⇒ *'b::real_normed_vector*
assumes $0 \le B$ *finite S*
and (*f has_integral i*) {*a..b*}
and $\bigwedge x.\ x \in \{a..b\} - S \Longrightarrow norm\ (f\ x) \le B$
shows *norm i* ≤ *B* ∗ *content* {*a..b*}
by (*metis assms box_real*(*2*) *has_integral_bound_spike_finite*)

6.14.13 In particular, the boundary of an interval is negligible

lemma *negligible_frontier_interval*: *negligible*(*cbox* (*a::'a::euclidean_space*) *b* − *box a b*)
proof −
let *?A* = $\bigcup((\lambda k.\ \{x.\ x \cdot k = a \cdot k\} \cup \{x::'a.\ x \cdot k = b \cdot k\})\ `\ Basis)$
have *negligible ?A*
by (*force simp add*: *negligible_Union*[*OF finite_imageI*])
moreover have *cbox a b* − *box a b* ⊆ *?A*
by (*force simp add*: *mem_box*)
ultimately show *?thesis*
by (*rule negligible_subset*)
qed

lemma *has_integral_spike_interior*:
assumes *f*: (*f has_integral y*) (*cbox a b*) **and** *gf*: $\bigwedge x.\ x \in box\ a\ b \Longrightarrow g\ x = f\ x$
shows (*g has_integral y*) (*cbox a b*)
by (*meson Diff_iff gf has_integral_spike*[*OF negligible_frontier_interval _ f*])

lemma *has_integral_spike_interior_eq*:
assumes $\bigwedge x.\ x \in box\ a\ b \Longrightarrow g\ x = f\ x$
shows (*f has_integral y*) (*cbox a b*) $\longleftrightarrow$ (*g has_integral y*) (*cbox a b*)
by (*metis assms has_integral_spike_interior*)

lemma *integrable_spike_interior*:
assumes $\bigwedge x.\ x \in box\ a\ b \Longrightarrow g\ x = f\ x$
and *f integrable_on cbox a b*
shows *g integrable_on cbox a b*
using *assms has_integral_spike_interior_eq* **by** *blast*

6.14.14 Integrability of continuous functions

lemma *operative_approximableI*:
fixes *f* :: *'b*::*euclidean_space* ⇒ *'a*::*banach*
assumes $0 \le e$
shows *operative conj True* ($\lambda i.\ \exists g.\ (\forall x{\in}i.\ norm\ (f\ x - g\ (x{::}'b)) \le e) \wedge g$ *integrable_on i*)
proof −
interpret *comm_monoid conj True*
by *standard auto*
show *?thesis*
proof (*standard*, *safe*)
fix *a b* :: *'b*
show $\exists g.\ (\forall x{\in}cbox\ a\ b.\ norm\ (f\ x - g\ x) \le e) \wedge g$ *integrable_on cbox a b*
if *box a b* = {} **for** *a b*
using *assms that*
by (*metis content_eq_0_interior integrable_on_null interior_cbox norm_zero right_minus_eq*)
{
fix *c g* **and** *k* :: *'b*
assume *fg*: $\forall x{\in}cbox\ a\ b.\ norm\ (f\ x - g\ x) \le e$ **and** *g*: *g integrable_on cbox a b*
assume *k*: $k \in Basis$
show $\exists g.\ (\forall x{\in}cbox\ a\ b \cap \{x.\ x \cdot k \le c\}.\ norm\ (f\ x - g\ x) \le e) \wedge g$ *integrable_on cbox a b* $\cap \{x.\ x \cdot k \le c\}$
$\exists g.\ (\forall x{\in}cbox\ a\ b \cap \{x.\ c \le x \cdot k\}.\ norm\ (f\ x - g\ x) \le e) \wedge g$ *integrable_on cbox a b* $\cap \{x.\ c \le x \cdot k\}$
using *fg g k* **by** *auto*
}
show $\exists g.\ (\forall x{\in}cbox\ a\ b.\ norm\ (f\ x - g\ x) \le e) \wedge g$ *integrable_on cbox a b*
if *fg1*: $\forall x{\in}cbox\ a\ b \cap \{x.\ x \cdot k \le c\}.\ norm\ (f\ x - g1\ x) \le e$
and *g1*: *g1 integrable_on cbox a b* $\cap \{x.\ x \cdot k \le c\}$
and *fg2*: $\forall x{\in}cbox\ a\ b \cap \{x.\ c \le x \cdot k\}.\ norm\ (f\ x - g2\ x) \le e$

```
      and g2: g2 integrable_on cbox a b ∩ {x. c ≤ x • k}
      and k: k ∈ Basis
    for c k g1 g2
  proof −
    let ?g = λx. if x•k = c then f x else if x•k ≤ c then g1 x else g2 x
    show ∃g. (∀x∈cbox a b. norm (f x − g x) ≤ e) ∧ g integrable_on cbox a b
    proof (intro exI conjI ballI)
      show norm (f x − ?g x) ≤ e if x ∈ cbox a b for x
        by (auto simp: that assms fg1 fg2)
      show ?g integrable_on cbox a b
      proof −
        have ?g integrable_on cbox a b ∩ {x. x • k ≤ c} ?g integrable_on cbox a
b ∩ {x. x • k ≥ c}
          by(rule integrable_spike[OF _ negligible_standard_hyperplane[of k c]],
use k g1 g2 in auto)+
        with has_integral_split[OF _ _ k] show ?thesis
          unfolding integrable_on_def by blast
      qed
    qed
  qed
qed

lemma comm_monoid_set_F_and: comm_monoid_set.F (∧) True f s ⟷ (finite
s ⟶ (∀x∈s. f x))
proof −
  interpret bool: comm_monoid_set ‹(∧)› True ..
  show ?thesis
    by (induction s rule: infinite_finite_induct) auto
qed

lemma approximable_on_division:
  fixes f :: 'b::euclidean_space ⇒ 'a::banach
  assumes 0 ≤ e
    and d: d division_of (cbox a b)
    and f: ∀i∈d. ∃g. (∀x∈i. norm (f x − g x) ≤ e) ∧ g integrable_on i
  obtains g where ∀x∈cbox a b. norm (f x − g x) ≤ e g integrable_on cbox a b
proof −
  interpret operative conj True λi. ∃g. (∀x∈i. norm (f x − g (x::'b)) ≤ e) ∧ g
integrable_on i
    using ‹0 ≤ e› by (rule operative_approximableI)
  from f local.division [OF d] that show thesis
    by auto
qed

lemma integrable_continuous:
  fixes f :: 'b::euclidean_space ⇒ 'a::banach
  assumes continuous_on (cbox a b) f
  shows f integrable_on cbox a b
```

proof (*rule integrable_uniform_limit*)
fix *e* :: *real*
assume *e*: *e* > *0*
then obtain *d* **where** *0* < *d* **and** *d*: ⋀*x x'*. ⟦*x* ∈ *cbox a b*; *x'* ∈ *cbox a b*; *dist x' x* < *d*⟧ ⟹ *dist* (*f x'*) (*f x*) < *e*
using *compact_uniformly_continuous*[*OF assms compact_cbox*] **unfolding** *uniformly_continuous_on_def* **by** *metis*
obtain *p* **where** *ptag*: *p tagged_division_of cbox a b* **and** *finep*: (λ*x*. *ball x d*) *fine p*
using *fine_division_exists*[*OF gauge_ball*[*OF* ‹*0* < *d*›], *of a b*] **.**
have ∗: ∀ *i*∈*snd* ‘ *p*. ∃ *g*. (∀ *x*∈*i*. *norm* (*f x* − *g x*) ≤ *e*) ∧ *g integrable_on i*
proof (*safe*, *unfold snd_conv*)
fix *x l*
assume *as*: (*x*, *l*) ∈ *p*
obtain *a b* **where** *l*: *l* = *cbox a b*
using *as ptag* **by** *blast*
then have *x*: *x* ∈ *cbox a b*
using *as ptag* **by** *auto*
show ∃ *g*. (∀ *x*∈*l*. *norm* (*f x* − *g x*) ≤ *e*) ∧ *g integrable_on l*
proof (*intro exI conjI strip*)
show (λ*y*. *f x*) *integrable_on l*
unfolding *integrable_on_def l* **by** *blast*
next
fix *y*
assume *y*: *y* ∈ *l*
then have *y* ∈ *ball x d*
using *as finep* **by** *fastforce*
then show *norm* (*f y* − *f x*) ≤ *e*
using *d x y as l*
by (*metis dist_commute dist_norm less_imp_le mem_ball ptag subsetCE tagged_division_ofD*(*3*))
qed
qed
from *e* **have** *e* ≥ *0*
by *auto*
from *approximable_on_division*[*OF this division_of_tagged_division*[*OF ptag*] ∗]
show ∃ *g*. (∀ *x*∈*cbox a b*. *norm* (*f x* − *g x*) ≤ *e*) ∧ *g integrable_on cbox a b*
by *metis*
qed

lemma *integrable_continuous_interval*:
fixes *f* :: *'b*::*ordered_euclidean_space* ⇒ *'a*::*banach*
assumes *continuous_on* {*a*..*b*} *f*
shows *f integrable_on* {*a*..*b*}
by (*metis assms integrable_continuous interval_cbox*)

lemmas *integrable_continuous_real* = *integrable_continuous_interval*[**where** *'b*=*real*]

lemma *integrable_continuous_closed_segment*:
fixes *f* :: *real* ⇒ *'a*::*banach*
assumes *continuous_on* (*closed_segment a b*) *f*
shows *f integrable_on* (*closed_segment a b*)
by (*metis assms closed_segment_eq_real_ivl integrable_continuous_interval*)

6.14.15 Specialization of additivity to one dimension

6.14.16 A useful lemma allowing us to factor out the content size

lemma *has_integral_factor_content*:
(*f has_integral i*) (*cbox a b*) ⟷
(∀ *e*>*0*. ∃ *d*. *gauge d* ∧ (∀ *p*. *p tagged_division_of* (*cbox a b*) ∧ *d fine p* ⟶
norm (*sum* (λ(*x*,*k*). *content k* ∗*R f x*) *p* − *i*) ≤ *e* ∗ *content* (*cbox a b*)))
proof (*cases content* (*cbox a b*) = *0*)
case *True*
have ⋀*e p*. *p tagged_division_of cbox a b* ⟹ *norm* ((∑ (*x*, *k*)∈*p*. *content k* ∗*R* *f x*)) ≤ *e* ∗ *content* (*cbox a b*)
unfolding *sum_content_null*[*OF True*] *True* **by** *force*
moreover have *i* = *0*
if ⋀*e*. *e* > *0* ⟹ ∃ *d*. *gauge d* ∧
(∀ *p*. *p tagged_division_of cbox a b* ∧
d fine p ⟶
norm ((∑ (*x*, *k*)∈*p*. *content k* ∗*R f x*) − *i*) ≤ *e* ∗ *content* (*cbox a b*))
using *that* [*of 1*]
by (*force simp add*: *True sum_content_null*[*OF True*] *intro*: *fine_division_exists*[*of* _ *a b*])
ultimately show *?thesis*
unfolding *has_integral_null_eq*[*OF True*]
by (*force simp add*:)
next
case *False*
then have *F*: *0* < *content* (*cbox a b*)
using *zero_less_measure_iff* **by** *blast*
let *?P* = λ*e opp*. ∃ *d*. *gauge d* ∧
(∀ *p*. *p tagged_division_of* (*cbox a b*) ∧ *d fine p* ⟶ *opp* (*norm* ((∑ (*x*, *k*)∈*p*. *content k* ∗*R* *f x*) − *i*)) *e*)
show *?thesis*
proof (*subst has_integral*, *safe*)
fix *e* :: *real*
assume *e*: *e* > *0*
show *?P* (*e* ∗ *content* (*cbox a b*)) (≤) **if** §[*rule_format*]: ∀ *ε*>*0*. *?P ε* (<)
using § [*of e* ∗ *content* (*cbox a b*)]
by (*meson F e less_imp_le mult_pos_pos*)
show *?P e* (<) **if** §[*rule_format*]: ∀ *ε*>*0*. *?P* (*ε* ∗ *content* (*cbox a b*)) (≤)
using § [*of e*/*2* / *content* (*cbox a b*)]
using *F e* **by** (*force simp add*: *algebra_simps*)
qed
qed

lemma *has_integral_factor_content_real*:
(*f has_integral i*) {*a..b*::*real*} ⟷
(∀ *e*>*0*. ∃ *d*. *gauge d* ∧ (∀ *p*. *p tagged_division_of* {*a..b*} ∧ *d fine p* ⟶
norm (*sum* (λ(*x*,*k*). *content k* ∗$_R$ *f x*) *p* − *i*) ≤ *e* ∗ *content* {*a..b*}))
unfolding *box_real*[*symmetric*]
by (*rule has_integral_factor_content*)

6.14.17 Fundamental theorem of calculus

lemma *interval_bounds_real*:
fixes *q b* :: *real*
assumes *a* ≤ *b*
shows *Sup* {*a..b*} = *b*
and *Inf* {*a..b*} = *a*
using *assms* **by** *auto*

theorem *fundamental_theorem_of_calculus*:
fixes *f* :: *real* ⇒ *'a*::*banach*
assumes *a* ≤ *b*
and *vecd*: ⋀*x*. *x* ∈ {*a..b*} ⟹ (*f has_vector_derivative f' x*) (*at x within* {*a..b*})
shows (*f' has_integral* (*f b* − *f a*)) {*a..b*}
unfolding *has_integral_factor_content box_real*[*symmetric*]
proof *safe*
fix *e* :: *real*
assume *e* > *0*
then have ∀ *x*. ∃ *d*>*0*. *x* ∈ {*a..b*} ⟶
(∀ *y*∈{*a..b*}. *norm* (*y*−*x*) < *d* ⟶ *norm* (*f y* − *f x* − (*y*−*x*) ∗$_R$ *f' x*) ≤ *e* ∗ *norm* (*y*−*x*))
using *vecd*[*unfolded has_vector_derivative_def has_derivative_within_alt*] **by** *blast*
then obtain *d* **where** *d*: ⋀*x*. *0* < *d x*
⋀*x y*. ⟦*x* ∈ {*a..b*}; *y* ∈ {*a..b*}; *norm* (*y*−*x*) < *d x*⟧
⟹ *norm* (*f y* − *f x* − (*y*−*x*) ∗$_R$ *f' x*) ≤ *e* ∗ *norm* (*y*−*x*)
by *metis*
show ∃ *d*. *gauge d* ∧ (∀ *p*. *p tagged_division_of* (*cbox a b*) ∧ *d fine p* ⟶
norm ((∑ (*x*, *k*)∈*p*. *content k* ∗$_R$ *f' x*) − (*f b* − *f a*)) ≤ *e* ∗ *content* (*cbox a b*))
proof (*rule exI*, *safe*)
show *gauge* (λ*x*. *ball x* (*d x*))
using *d*(*1*) *gauge_ball_dependent* **by** *blast*
next
fix *p*
assume *ptag*: *p tagged_division_of cbox a b* **and** *finep*: (λ*x*. *ball x* (*d x*)) *fine p*
have *ba*: *b* − *a* = (∑ (*x*,*K*)∈*p*. *Sup K* − *Inf K*) *f b* − *f a* = (∑ (*x*,*K*)∈*p*. *f*(*Sup K*) − *f*(*Inf K*))
using *additive_tagged_division_1*[**where** *f*= λ*x*. *x*] *additive_tagged_division_1*[**where** *f*= *f*]
‹*a* ≤ *b*› *ptag* **by** *auto*

```
    have norm (∑(x, K) ∈ p. (content K *R f' x) − (f (Sup K) − f (Inf K)))
          ≤ (∑n∈p. e * (case n of (x, k) ⇒ Sup k − Inf k))
    proof (rule sum_norm_le,safe)
      fix x K
      assume (x, K) ∈ p
      then have x ∈ K and kab: K ⊆ cbox a b
        using ptag by blast+
      then obtain u v where k: K = cbox u v and x ∈ K and kab: K ⊆ cbox a b
        using ptag ‹(x, K) ∈ p› by auto
      have u ≤ v
        using ‹x ∈ K› unfolding k by auto
      have ball: ∀ y∈K. y ∈ ball x (d x)
        using finep ‹(x, K) ∈ p› by blast
      have u ∈ K v ∈ K
        by (simp_all add: ‹u ≤ v› k)
      have norm ((v − u) *R f' x − (f v − f u)) = norm (f u − f x − (u − x) *R
f' x − (f v − f x − (v − x) *R f' x))
        by (auto simp add: algebra_simps)
      also have ... ≤ norm (f u − f x − (u − x) *R f' x) + norm (f v − f x − (v
− x) *R f' x)
        by (rule norm_triangle_ineq4)
      finally have norm ((v − u) *R f' x − (f v − f u)) ≤
        norm (f u − f x − (u − x) *R f' x) + norm (f v − f x − (v − x) *R f' x) .
      also have ... ≤ e * norm (u − x) + e * norm (v − x)
      proof (rule add_mono)
        show norm (f u − f x − (u − x) *R f' x) ≤ e * norm (u − x)
        proof (rule d)
          show norm (u − x) < d x
            using ‹u ∈ K› ball by (auto simp add: dist_real_def)
        qed (use ‹x ∈ K› ‹u ∈ K› kab in auto)
        show norm (f v − f x − (v − x) *R f' x) ≤ e * norm (v − x)
        proof (rule d)
          show norm (v − x) < d x
            using ‹v ∈ K› ball by (auto simp add: dist_real_def)
        qed (use ‹x ∈ K› ‹v ∈ K› kab in auto)
      qed
      also have ... ≤ e * (Sup K − Inf K)
          using ‹x ∈ K› by (auto simp: k interval_bounds_real[OF ‹u ≤ v›]
field_simps)
      finally show norm (content K *R f' x − (f (Sup K) − f (Inf K))) ≤ e *
(Sup K − Inf K)
        using ‹u ≤ v› by (simp add: k)
    qed
    with ‹a ≤ b› show norm ((∑(x, K)∈p. content K *R f' x) − (f b − f a)) ≤
e * content (cbox a b)
      by (auto simp: ba split_def sum_subtractf [symmetric] sum_distrib_left)
  qed
qed
```

lemma *has_complex_derivative_imp_has_vector_derivative*:
fixes $f :: complex \Rightarrow complex$
assumes $(f\ has\_field\_derivative\ f')\ (at\ (of\_real\ a)\ within\ (cbox\ (of\_real\ x)\ (of\_real\ y)))$
shows $((f\ o\ of\_real)\ has\_vector\_derivative\ f')\ (at\ a\ within\ \{x..y\})$
using *has_derivative_in_compose*[*of of_real of_real a* $\{x..y\}$ *f* $(*)$ f'] *assms*
by (*simp add*: *scaleR_conv_of_real ac_simps has_vector_derivative_def has_field_derivative_def o_def cbox_complex_of_real*)

lemma *ident_has_integral*:
fixes *a*::*real*
assumes $a \leq b$
shows $((\lambda x.\ x)\ has\_integral\ (b^2 - a^2)/2)\ \{a..b\}$
proof −
have $((\lambda x.\ x)\ has\_integral\ inverse\ 2 * b^2 - inverse\ 2 * a^2)\ \{a..b\}$
unfolding *power2_eq_square*
by (*rule fundamental_theorem_of_calculus* [*OF assms*] *derivative_eq_intros* | *simp*)+
then show *?thesis*
by (*simp add*: *field_simps*)
qed

lemma *integral_ident* [*simp*]:
fixes *a*::*real*
assumes $a \leq b$
shows $integral\ \{a..b\}\ (\lambda x.\ x) = (if\ a \leq b\ then\ (b^2 - a^2)/2\ else\ 0)$
by (*metis assms ident_has_integral integral_unique*)

lemma *ident_integrable_on*:
fixes *a*::*real*
shows $(\lambda x.\ x)\ integrable\_on\ \{a..b\}$
using *continuous_on_id integrable_continuous_real* **by** *blast*

lemma *integral_sin* [*simp*]:
fixes *a*::*real*
assumes $a \leq b$ **shows** $integral\ \{a..b\}\ sin = cos\ a - cos\ b$
proof −
have $(sin\ has\_integral\ (-\ cos\ b - -\ cos\ a))\ \{a..b\}$
proof (*rule fundamental_theorem_of_calculus*)
show $((\lambda a.\ -\ cos\ a)\ has\_vector\_derivative\ sin\ x)\ (at\ x\ within\ \{a..b\})$ **for** *x*
unfolding *has_real_derivative_iff_has_vector_derivative* [*symmetric*]
by (*rule derivative_eq_intros* | *force*)+
qed (*use assms* **in** *auto*)
then show *?thesis*
by (*simp add*: *integral_unique*)
qed

lemma *integral_cos* [*simp*]:
fixes *a*::*real*

```
  assumes a ≤ b shows integral {a..b} cos = sin b − sin a
proof −
  have (cos has_integral (sin b − sin a)) {a..b}
  proof (rule fundamental_theorem_of_calculus)
    show (sin has_vector_derivative cos x) (at x within {a..b}) for x
      unfolding has_real_derivative_iff_has_vector_derivative [symmetric]
      by (rule derivative_eq_intros | force)+
  qed (use assms in auto)
  then show ?thesis
    by (simp add: integral_unique)
qed

lemma integral_exp [simp]:
  fixes a::real
  assumes a ≤ b shows integral {a..b} exp = exp b − exp a
 by (meson DERIV_exp assms fundamental_theorem_of_calculus has_real_derivative_iff_has_vector_derivative
      has_vector_derivative_at_within integral_unique)


lemma has_integral_sin_nx: ((λx. sin(real_of_int n ∗ x)) has_integral 0) {−pi..pi}
proof (cases n = 0)
  case False
  have ((λx. sin (n ∗ x)) has_integral (− cos (n ∗ pi)/n − − cos (n ∗ − pi)/n))
{−pi..pi}
  proof (rule fundamental_theorem_of_calculus)
    show ((λx. − cos (n ∗ x) / n) has_vector_derivative sin (n ∗ a)) (at a within
{−pi..pi})
      if a ∈ {−pi..pi} for a :: real
      using that False
      unfolding has_vector_derivative_def
      by (intro derivative_eq_intros | force)+
  qed auto
  then show ?thesis
    by simp
qed auto

lemma integral_sin_nx:
   integral {−pi..pi} (λx. sin(x ∗ real_of_int n)) = 0
  using has_integral_sin_nx [of n] by (force simp: mult.commute)

lemma has_integral_cos_nx:
 ((λx. cos(real_of_int n ∗ x)) has_integral (if n = 0 then 2 ∗ pi else 0)) {−pi..pi}
proof (cases n = 0)
  case True
  then show ?thesis
    using has_integral_const_real [of 1::real −pi pi] by auto
next
  case False
   have ((λx. cos (n ∗ x)) has_integral (sin (n ∗ pi)/n − sin (n ∗ − pi)/n))
```

$\{-pi..pi\}$
proof (*rule fundamental_theorem_of_calculus*)
show ((λ*x*. *sin* (*n* ∗ *x*) / *n*) *has_vector_derivative cos* (*n* ∗ *x*)) (*at x within* $\{-pi..pi\}$)
if $x \in \{-pi..pi\}$
for *x* :: *real*
using *that False*
unfolding *has_vector_derivative_def*
by (*intro derivative_eq_intros* | *force*)+
qed *auto*
with *False* **show** *?thesis*
by (*simp add*: *mult.commute*)
qed

lemma *integral_cos_nx*:
integral $\{-pi..pi\}$ (λ*x*. *cos*(*x* ∗ *real_of_int n*)) = (*if n* = *0 then 2* ∗ *pi else 0*)
using *has_integral_cos_nx* [*of n*] **by** (*force simp*: *mult.commute*)

6.14.18 Taylor series expansion

lemma *mvt_integral*:
fixes *f*::′*a*::*real_normed_vector*⇒′*b*::*banach*
assumes *f*′[*derivative_intros*]:
⋀*x*. *x* ∈ *S* ⟹ (*f has_derivative f*′ *x*) (*at x within S*)
assumes *line_in*: ⋀*t*. *t* ∈ $\{0..1\}$ ⟹ *x* + *t* ∗$_R$ *y* ∈ *S*
shows *f* (*x* + *y*) − *f x* = *integral* $\{0..1\}$ (λ*t*. *f*′ (*x* + *t* ∗$_R$ *y*) *y*)
proof −
from *assms* **have** *subset*: (λ*xa*. *x* + *xa* ∗$_R$ *y*) ‘ $\{0..1\} \subseteq S$ **by** *auto*
note [*derivative_intros*] =
has_derivative_subset[*OF _ subset*]
has_derivative_in_compose[**where** *f*=(λ*xa*. *x* + *xa* ∗$_R$ *y*) **and** *g* = *f*]
note [*continuous_intros*] =
continuous_on_compose2[**where** *f*=(λ*xa*. *x* + *xa* ∗$_R$ *y*)]
continuous_on_subset[*OF _ subset*]
have ⋀*t*. *t* ∈ $\{0..1\}$ ⟹
((λ*t*. *f* (*x* + *t* ∗$_R$ *y*)) *has_vector_derivative f*′ (*x* + *t* ∗$_R$ *y*) *y*)
(*at t within* $\{0..1\}$)
using *assms*
by (*auto simp*: *has_vector_derivative_def*
linear_cmul[*OF has_derivative_linear*[*OF f*′], *symmetric*]
intro!: *derivative_eq_intros*)
from *fundamental_theorem_of_calculus*[*rule_format*, *OF _ this*]
show *?thesis*
by (*auto intro*!: *integral_unique*[*symmetric*])
qed

lemma (**in** *bounded_bilinear*) *sum_prod_derivatives_has_vector_derivative*:
assumes *p*>*0*
and *f0*: *Df 0* = *f*

and *Df*: $\bigwedge m\ t.\ m < p \Longrightarrow a \le t \Longrightarrow t \le b \Longrightarrow$
(*Df m has_vector_derivative Df* (*Suc m*) *t*) (*at t within* {*a..b*})
and *g0*: *Dg 0* = *g*
and *Dg*: $\bigwedge m\ t.\ m < p \Longrightarrow a \le t \Longrightarrow t \le b \Longrightarrow$
(*Dg m has_vector_derivative Dg* (*Suc m*) *t*) (*at t within* {*a..b*})
and *ivl*: $a \le t\ t \le b$
shows ($(\lambda t.\ \sum i{<}p.\ (-1)\hat{}i *_R$ *prod* (*Df i t*) (*Dg* (*p* − *Suc i*) *t*))
has_vector_derivative
prod (*f t*) (*Dg p t*) − $(-1)\hat{}p *_R$ *prod* (*Df p t*) (*g t*))
(*at t within* {*a..b*})
using *assms*
proof *cases*
assume *p*: $p \neq 1$
define p' **where** $p' = p - 2$
from *assms p* **have** p': {..<*p*} = {..*Suc* p'} p = *Suc* (*Suc* p')
by (*auto simp*: p'*_def*)
have $*$: $\bigwedge i.\ i \le p' \Longrightarrow$ *Suc* (*Suc* p' − *i*) = (*Suc* (*Suc* p') − *i*)
by *auto*
let *?f* = $\lambda i.\ (-1)\ \hat{}\ i *_R$ (*prod* (*Df i t*) (*Dg* ((*p* − *i*)) *t*))
have ($\sum i{<}p.\ (-1)\ \hat{}\ i *_R$ (*prod* (*Df i t*) (*Dg* (*Suc* (*p* − *Suc i*)) *t*) +
prod (*Df* (*Suc i*) *t*) (*Dg* (*p* − *Suc i*) *t*))) =
($\sum i{\le}$(*Suc* p'). *?f i* − *?f* (*Suc i*))
by (*auto simp*: *algebra_simps* p'(*2*) *numeral_2_eq_2* $*$ *lessThan_Suc_atMost*)
also note *sum_telescope*
finally
have ($\sum i{<}p.\ (-1)\ \hat{}\ i *_R$ (*prod* (*Df i t*) (*Dg* (*Suc* (*p* − *Suc i*)) *t*) +
prod (*Df* (*Suc i*) *t*) (*Dg* (*p* − *Suc i*) *t*)))
= *prod* (*f t*) (*Dg p t*) − (− *1*) $\hat{}$ *p* $*_R$ *prod* (*Df p t*) (*g t*)
unfolding p'[*symmetric*]
by (*simp add*: *assms*)
thus *?thesis*
using *assms*
by (*auto intro*!: *derivative_eq_intros has_vector_derivative*)
qed (*auto intro*!: *derivative_eq_intros has_vector_derivative*)

lemma
fixes *f*::*real*⇒'*a*::*banach*
assumes *p*>*0*
and *f0*: *Df 0* = *f*
and *Df*: $\bigwedge m\ t.\ m < p \Longrightarrow a \le t \Longrightarrow t \le b \Longrightarrow$
(*Df m has_vector_derivative Df* (*Suc m*) *t*) (*at t within* {*a..b*})
and *ivl*: $a \le b$
defines $i \equiv \lambda x.\ ((b - x)\ \hat{}\ (p - 1)\ /\ fact\ (p - 1)) *_R$ *Df p x*
shows *Taylor_has_integral*:
(*i has_integral f b* − ($\sum i{<}p.\ ((b{-}a)\ \hat{}\ i\ /\ fact\ i) *_R$ *Df i a*)) {*a..b*}
and *Taylor_integral*:
f b = ($\sum i{<}p.\ ((b{-}a)\ \hat{}\ i\ /\ fact\ i) *_R$ *Df i a*) + *integral* {*a..b*} *i*
and *Taylor_integrable*:
i integrable_on {*a..b*}

```
proof goal_cases
  case 1
  interpret bounded_bilinear scaleR::real⇒'a⇒'a
    by (rule bounded_bilinear_scaleR)
  define g where g s = (b − s)^(p − 1)/fact (p − 1) for s
  define Dg where [abs_def]:
    Dg n s = (if n < p then (−1)^n ∗ (b − s)^(p − 1 − n) / fact (p − 1 − n)
else 0) for n s
  have g0: Dg 0 = g
    using ‹p > 0›
    by (auto simp add: Dg_def field_split_simps g_def split: if_split_asm)
  {
    fix m
    assume p > Suc m
    hence p − Suc m = Suc (p − Suc (Suc m))
      by auto
    hence real (p − Suc m) ∗ fact (p − Suc (Suc m)) = fact (p − Suc m)
      by auto
  } note fact_eq = this
  have Dg: ⋀m t. m < p ⟹ a ≤ t ⟹ t ≤ b ⟹
    (Dg m has_vector_derivative Dg (Suc m) t) (at t within {a..b})
    unfolding Dg_def
    by (auto intro!: derivative_eq_intros simp: has_vector_derivative_def fact_eq
field_split_simps)
  let ?sum = λt. ∑i<p. (− 1) ^ i ∗R Dg i t ∗R Df (p − Suc i) t
  from sum_prod_derivatives_has_vector_derivative[of _ Dg _ _ _ Df,
      OF ‹p > 0› g0 Dg f0 Df]
  have deriv: ⋀t. a ≤ t ⟹ t ≤ b ⟹
    (?sum has_vector_derivative
      g t ∗R Df p t − (− 1) ^ p ∗R Dg p t ∗R f t) (at t within {a..b})
    by auto
  from fundamental_theorem_of_calculus[rule_format, OF ‹a ≤ b› deriv]
  have (i has_integral ?sum b − ?sum a) {a..b}
    using atLeastatMost_empty'[simp del]
    by (simp add: i_def g_def Dg_def)
  also
  have one: (− 1) ^ p' ∗ (− 1) ^ p' = (1::real) {..<p} ∩ {i. p = Suc i} = {p −
1} for p'
    using ‹p > 0› by (auto simp: power_mult_distrib)
  then have ?sum b = f b
    using Suc_pred'[OF ‹p > 0›]
    by (simp add: diff_eq_eq Dg_def power_0_left le_Suc_eq if_distrib
      if_distribR sum.If_cases f0)
  also
  have ?sum a = (∑i<p. ((b−a) ^ i / fact i) ∗R Df i a)
  proof (rule sum.reindex_cong)
    have ⋀i. i < p ⟹ ∃j<p. i = p − Suc j
    by (metis Suc_diff_Suc ‹p>0› diff_Suc_less diff_diff_cancel less_or_eq_imp_le)
    then show {..<p} = (λx. p − x − 1) ‘ {..<p}
```

```
    by force
  qed (auto simp add: inj_on_def Dg_def one)
  finally show c: ?case .
  case 2 show ?case using c integral_unique
    by (metis (lifting) add.commute diff_eq_eq integral_unique)
  case 3 show ?case using c by force
qed
```

6.14.19 Only need trivial subintervals if the interval itself is trivial

```
proposition division_of_nontrivial:
  fixes 𝒟 :: 'a::euclidean_space set set
  assumes sdiv: 𝒟 division_of (cbox a b)
    and cont0: content (cbox a b) ≠ 0
  shows {k. k ∈ 𝒟 ∧ content k ≠ 0} division_of (cbox a b)
  using sdiv
proof (induction card 𝒟 arbitrary: 𝒟 rule: less_induct)
  case less
  note 𝒟 = division_ofD[OF less.prems]
  {
    presume ∗: {k ∈ 𝒟. content k ≠ 0} ≠ 𝒟 ⟹ ?case
    then show ?case
      using less.prems by fastforce
  }
  assume noteq: {k ∈ 𝒟. content k ≠ 0} ≠ 𝒟
  then obtain K c d where K ∈ 𝒟 and contk: content K = 0 and keq: K =
cbox c d
    using 𝒟(4) by blast
  then have card 𝒟 > 0
    unfolding card_gt_0_iff using less by auto
  then have card: card (𝒟 − {K}) < card 𝒟
    using less ‹K ∈ 𝒟› by (simp add: 𝒟(1))
  have closed: closed (⋃(𝒟 − {K}))
    using less.prems by auto
  have x islimpt ⋃(𝒟 − {K}) if x ∈ K for x
    unfolding islimpt_approachable
  proof (intro allI impI)
    fix e::real
    assume e > 0
    obtain i where i: c•i = d•i i∈Basis
      using contk 𝒟(3) [OF ‹K ∈ 𝒟›] unfolding box_ne_empty keq
      by (meson content_eq_0 dual_order.antisym)
    then have xi: x•i = d•i
      using ‹x ∈ K› unfolding keq mem_box by (metis antisym)
    define y where y = (∑j∈Basis. (if j = i then if c•i ≤ (a•i + b•i)/2 then c•i
+
      min e (b•i − c•i)/2 else c•i − min e (c•i − a•i)/2 else x•j) ∗R j)
    show ∃x′∈⋃(𝒟 − {K}). x′ ≠ x ∧ dist x′ x < e
```

```
    proof (intro bexI conjI)
      have d ∈ cbox c d
      using 𝒟(3)[OF ‹K ∈ 𝒟›] by (simp add: box_ne_empty(1) keq mem_box(2))
      then have d ∈ cbox a b
        using 𝒟(2)[OF ‹K ∈ 𝒟›] by (auto simp: keq)
      then have di: a • i ≤ d • i ∧ d • i ≤ b • i
        using ‹i ∈ Basis› mem_box(2) by blast
      then have xyi: y•i ≠ x•i
        unfolding y_def i xi using ‹e > 0› cont0 ‹i ∈ Basis›
        by (auto simp: content_eq_0 elim!: ballE[of _ _ i])
      then show y ≠ x
        unfolding euclidean_eq_iff[where 'a='a] using i by auto
      have norm (y−x) ≤ (∑ b∈Basis. |(y − x) • b|)
        by (rule norm_le_l1)
      also have ... = |(y − x) • i| + (∑ b ∈ Basis − {i}. |(y − x) • b|)
        by (meson finite_Basis i(2) sum.remove)
      also have ... <  e + sum (λi. 0) Basis
      proof (rule add_less_le_mono)
        show |(y−x) • i| < e
          using di ‹e > 0› y_def i xi by (auto simp: inner_simps)
        show (∑ i∈Basis − {i}. |(y−x) • i|) ≤ (∑ i∈Basis. 0)
          unfolding y_def by (auto simp: inner_simps)
      qed
      finally have norm (y−x) < e + sum (λi. 0) Basis .
      then show dist y x < e
        unfolding dist_norm by auto
      have y ∉ K
        unfolding keq mem_box using i(1) i(2) xi xyi by fastforce
      moreover have y ∈ ⋃𝒟
        using subsetD[OF 𝒟(2)[OF ‹K ∈ 𝒟›] ‹x ∈ K›] ‹e > 0› di i
        by (auto simp: 𝒟 mem_box y_def field_simps elim!: ballE[of _ _ i])
      ultimately show y ∈ ⋃(𝒟 − {K}) by auto
    qed
  qed
  then have K ⊆ ⋃(𝒟 − {K})
    using closed closed_limpt by blast
  then have ⋃(𝒟 − {K}) = cbox a b
    unfolding 𝒟(6)[symmetric] by auto
  then have 𝒟 − {K} division_of cbox a b
    by (metis Diff_subset less.prems division_of_subset 𝒟(6))
  then have {ka ∈ 𝒟 − {K}. content ka ≠ 0} division_of (cbox a b)
    using card less.hyps by blast
  moreover have {ka ∈ 𝒟 − {K}. content ka ≠ 0} = {K ∈ 𝒟. content K ≠ 0}
    using contk by auto
  ultimately show ?case by auto
qed
```

6.14.20 Integrability on subintervals

lemma *operative_integrableI*:
fixes *f* :: *'b*::*euclidean_space* ⇒ *'a*::*banach*
assumes *0* ≤ *e*
shows *operative conj True* (λ*i*. *f integrable_on i*)
proof −
interpret *comm_monoid conj True*
proof qed
show *?thesis*
proof
show ⋀*a b*. *box a b* = {} ⟹ (*f integrable_on cbox a b*) = *True*
by (*simp add*: *content_eq_0_interior integrable_on_null*)
show ⋀*a b c k*.
k ∈ *Basis* ⟹
(*f integrable_on cbox a b*) ⟷
(*f integrable_on cbox a b* ∩ {*x*. *x* • *k* ≤ *c*} ∧ *f integrable_on cbox a b* ∩ {*x*. *c* ≤ *x* • *k*})
unfolding *integrable_on_def* **by** (*auto intro*!: *has_integral_split*)
qed
qed

lemma *integrable_subinterval*:
fixes *f* :: *'b*::*euclidean_space* ⇒ *'a*::*banach*
assumes *f*: *f integrable_on cbox a b*
and *cd*: *cbox c d* ⊆ *cbox a b*
shows *f integrable_on cbox c d*
proof −
interpret *operative conj True* λ*i*. *f integrable_on i*
using *order_refl* **by** (*rule operative_integrableI*)
show *?thesis*
by (*metis cd division division_of_finite empty f partial_division_extend_1 remove*)
qed

lemma *integrable_subinterval_real*:
fixes *f* :: *real* ⇒ *'a*::*banach*
assumes *f integrable_on* {*a*..*b*}
and {*c*..*d*} ⊆ {*a*..*b*}
shows *f integrable_on* {*c*..*d*}
by (*metis assms box_real*(*2*) *integrable_subinterval*)

6.14.21 Combining adjacent intervals in 1 dimension

lemma *has_integral_combine*:
fixes *a b c* :: *real* **and** *j* :: *'a*::*banach*
assumes *a* ≤ *c*
and *c* ≤ *b*
and *ac*: (*f has_integral i*) {*a*..*c*}
and *cb*: (*f has_integral j*) {*c*..*b*}

```
  shows (f has_integral (i + j)) {a..b}
proof −
  interpret operative_real lift_option plus Some 0
    λi. if f integrable_on i then Some (integral i f) else None
    using operative_integralI by (rule operative_realI)
  from ‹a ≤ c› ‹c ≤ b› ac cb coalesce_less_eq
  have ∗: lift_option (+)
            (if f integrable_on {a..c} then Some (integral {a..c} f) else None)
            (if f integrable_on {c..b} then Some (integral {c..b} f) else None) =
           (if f integrable_on {a..b} then Some (integral {a..b} f) else None)
    by (auto simp: split: if_split_asm)
  then have f integrable_on cbox a b
    using ac cb by (auto split: if_split_asm)
  with ∗ show ?thesis
     using ac cb by (auto simp add: integrable_on_def integral_unique split:
if_split_asm)
qed

lemma integral_combine:
  fixes f :: real ⇒ 'a::banach
  assumes a ≤ c
    and c ≤ b
    and ab: f integrable_on {a..b}
  shows integral {a..c} f + integral {c..b} f = integral {a..b} f
proof −
  have (f has_integral integral {a..c} f) {a..c}
    using ab ‹c ≤ b› integrable_subinterval_real by fastforce
  moreover
  have (f has_integral integral {c..b} f) {c..b}
    using ab ‹a ≤ c› integrable_subinterval_real by fastforce
  ultimately show ?thesis
    by (smt (verit, best) assms has_integral_combine integral_unique)
qed

lemma integrable_combine:
  fixes f :: real ⇒ 'a::banach
  assumes a ≤ c
    and c ≤ b
    and f integrable_on {a..c}
    and f integrable_on {c..b}
  shows f integrable_on {a..b}
  using assms has_integral_combine by blast

lemma integral_minus_sets:
  fixes f::real ⇒ 'a::banach
  shows c ≤ a ⟹ c ≤ b ⟹ f integrable_on {c .. max a b} ⟹
    integral {c .. a} f − integral {c .. b} f =
    (if a ≤ b then − integral {a .. b} f else integral {b .. a} f)
  using integral_combine[of c a b f]  integral_combine[of c b a f]
```

by (*auto simp*: *algebra_simps max_def*)

lemma *integral_minus_sets'*:
fixes *f*::*real* ⇒ *'a*::*banach*
shows $c \geq a \Longrightarrow c \geq b \Longrightarrow f$ *integrable_on* $\{min\ a\ b\ ..\ c\} \Longrightarrow$
integral $\{a\ ..\ c\}$ *f* − *integral* $\{b\ ..\ c\}$ *f* =
(*if* $a \leq b$ *then integral* $\{a\ ..\ b\}$ *f else* − *integral* $\{b\ ..\ a\}$ *f*)
using *integral_combine*[*of b a c f*] *integral_combine*[*of a b c f*]
by (*auto simp*: *algebra_simps min_def*)

6.14.22 Reduce integrability to "local" integrability

lemma *integrable_on_little_subintervals*:
fixes *f* :: *'b*::*euclidean_space* ⇒ *'a*::*banach*
assumes $\forall x{\in}cbox\ a\ b.\ \exists d{>}0.\ \forall u\ v.\ x \in cbox\ u\ v \wedge cbox\ u\ v \subseteq ball\ x\ d \wedge cbox\ u\ v \subseteq cbox\ a\ b \longrightarrow$
f integrable_on cbox u v
shows *f integrable_on cbox a b*
proof −
interpret *operative conj True* λ*i*. *f integrable_on i*
using *order_refl* **by** (*rule operative_integrableI*)
have $\forall x.\ \exists d{>}0.\ x{\in}cbox\ a\ b \longrightarrow (\forall u\ v.\ x \in cbox\ u\ v \wedge cbox\ u\ v \subseteq ball\ x\ d \wedge cbox\ u\ v \subseteq cbox\ a\ b \longrightarrow$
f integrable_on cbox u v)
using *assms* **by** (*metis zero_less_one*)
then obtain *d* **where** *d*: $\bigwedge x.\ 0 < d\ x$
$\bigwedge x\ u\ v.\ [\![x \in cbox\ a\ b;\ x \in cbox\ u\ v;\ cbox\ u\ v \subseteq ball\ x\ (d\ x);\ cbox\ u\ v \subseteq cbox\ a\ b]\!]$
$\Longrightarrow$ *f integrable_on cbox u v*
by *metis*
obtain *p* **where** *p*: *p tagged_division_of cbox a b* (λ*x*. *ball x* (*d x*)) *fine p*
using *fine_division_exists*[*OF gauge_ball_dependent*,*of d a b*] *d*(*1*) **by** *blast*
then have *sndp*: *snd* ‘ *p division_of cbox a b*
by (*metis division_of_tagged_division*)
have *f integrable_on k* **if** (*x*, *k*) ∈ *p* **for** *x k*
using *tagged_division_ofD*(*2*−*4*)[*OF p*(*1*) *that*] *fineD*[*OF p*(*2*) *that*] *d*[*of x*]
by *auto*
then show *?thesis*
unfolding *division* [*symmetric*, *OF sndp*] *comm_monoid_set_F_and*
by *auto*
qed

6.14.23 Second FTC or existence of antiderivative

lemma *integrable_const*[*intro*]: (λ*x*. *c*) *integrable_on cbox a b*
unfolding *integrable_on_def* **by** *blast*

lemma *integral_has_vector_derivative_continuous_at*:
fixes *f* :: *real* ⇒ *'a*::*banach*
assumes *f*: *f integrable_on* $\{a..b\}$

```
    and x: x ∈ {a..b} − S
    and finite S
    and fx: continuous (at x within ({a..b} − S)) f
 shows ((λu. integral {a..u} f) has_vector_derivative f x) (at x within ({a..b} −
S))
proof −
  let ?I = λa b. integral {a..b} f
  { fix e::real
    assume e > 0
   obtain d where d>0 and d: ⋀x'. ⟦x' ∈ {a..b} − S; |x' − x| < d⟧ ⟹ norm(f
x' − f x) ≤ e
      using ‹e>0› fx by (auto simp: continuous_within_eps_delta dist_norm
less_imp_le)
    have norm (integral {a..y} f − integral {a..x} f − (y−x) *R f x) ≤ e * |y −
x| (is ?lhs ≤ ?rhs)
        if y: y ∈ {a..b} − S and yx: |y − x| < d for y
    proof (cases y < x)
      case False
      have f integrable_on {a..y}
        using f y by (simp add: integrable_subinterval_real)
      then have Idiff: ?I a y − ?I a x = ?I x y
        using False x by (simp add: algebra_simps integral_combine)
      have fux_int: ((λu. f u − f x) has_integral integral {x..y} f − (y−x) *R f x)
{x..y}
      proof (rule has_integral_diff)
        show (f has_integral integral {x..y} f) {x..y}
       using x y by (auto intro: integrable_integral [OF integrable_subinterval_real
[OF f]])
        show ((λu. f x) has_integral (y − x) *R f x) {x..y}
          using has_integral_const_real [of f x x y] False by simp
      qed
      have ?lhs ≤ e * content {x..y}
        using yx False d x y ‹e>0› assms
        by (intro has_integral_bound_real[where f=(λu. f u − f x)]) (auto simp:
Idiff fux_int)
      also have ... ≤ ?rhs
        using False by auto
      finally show ?thesis .
    next
      case True
      have f integrable_on {a..x}
        using f x by (simp add: integrable_subinterval_real)
      then have Idiff: ?I a x − ?I a y = ?I y x
        using True x y by (simp add: algebra_simps integral_combine)
      have fux_int: ((λu. f u − f x) has_integral integral {y..x} f − (x − y) *R f
x) {y..x}
      proof (rule has_integral_diff)
        show (f has_integral integral {y..x} f) {y..x}
       using x y by (auto intro: integrable_integral [OF integrable_subinterval_real
```

```
[OF f]])
      show ((λu. f x) has_integral (x − y) ∗R f x) {y..x}
        using has_integral_const_real [of f x y x] True by simp
    qed
     have norm (integral {a..x} f − integral {a..y} f − (x − y) ∗R f x) ≤ e ∗
content {y..x}
      using yx True d x y ‹e>0› assms
      by (intro has_integral_bound_real[where f=(λu. f u − f x)]) (auto simp:
Idiff fux_int)
    also have ... ≤ e ∗ |y − x|
      using True by auto
    finally have norm (integral {a..x} f − integral {a..y} f − (x − y) ∗R f x) ≤
e ∗ |y − x| .
    then show ?thesis
      by (simp add: algebra_simps norm_minus_commute)
  qed
  then have ∃ d>0. ∀ y∈{a..b} − S. |y − x| < d ⟶ norm (integral {a..y} f −
integral {a..x} f − (y−x) ∗R f x) ≤ e ∗ |y − x|
    using ‹d>0› by blast
 }
 then show ?thesis
  by (simp add: has_vector_derivative_def has_derivative_within_alt bounded_linear_scaleR_left)
qed


lemma integral_has_vector_derivative:
 fixes f :: real ⇒ 'a::banach
 assumes continuous_on {a..b} f
   and x ∈ {a..b}
 shows ((λu. integral {a..u} f) has_vector_derivative f(x)) (at x within {a..b})
 using assms integral_has_vector_derivative_continuous_at [OF integrable_continuous_real]
 by (fastforce simp: continuous_on_eq_continuous_within)

lemma integral_has_real_derivative:
 assumes continuous_on {a..b} g
 assumes t ∈ {a..b}
 shows ((λx. integral {a..x} g) has_real_derivative g t) (at t within {a..b})
 using integral_has_vector_derivative[of a b g t] assms
 by (auto simp: has_real_derivative_iff_has_vector_derivative)

lemma antiderivative_continuous:
 fixes q b :: real
 assumes continuous_on {a..b} f
 obtains g where ⋀x. x ∈ {a..b} ⟹ (g has_vector_derivative (f x::_::banach))
(at x within {a..b})
 using integral_has_vector_derivative[OF assms] by auto
```

6.14.24 Combined fundamental theorem of calculus

lemma *antiderivative_integral_continuous*:
fixes $f :: real \Rightarrow 'a::banach$
assumes *continuous_on* $\{a..b\}$ f
obtains g **where** $\forall u{\in}\{a..b\}.\ \forall v \in \{a..b\}.\ u \le v \longrightarrow (f\ has\_integral\ (g\ v - g\ u))\ \{u..v\}$
proof −
obtain g
where g: $\bigwedge x.\ x \in \{a..b\} \Longrightarrow (g\ has\_vector\_derivative\ f\ x)\ (at\ x\ within\ \{a..b\})$
using *antiderivative_continuous*[*OF assms*] **by** *metis*
have $(f\ has\_integral\ g\ v - g\ u)\ \{u..v\}$ **if** $u \in \{a..b\}$ $v \in \{a..b\}$ $u \le v$ **for** $u\ v$
proof −
have $\bigwedge x.\ x \in cbox\ u\ v \Longrightarrow (g\ has\_vector\_derivative\ f\ x)\ (at\ x\ within\ cbox\ u\ v)$
by (*metis atLeastAtMost_iff atLeastatMost_subset_iff box_real*(2) *g has_vector_derivative_within_subset subsetCE that*(1,2))
then show *?thesis*
by (*metis box_real*(2) *that*(3) *fundamental_theorem_of_calculus*)
qed
then show *?thesis*
using *that* **by** *blast*
qed

6.14.25 General "twiddling" for interval-to-interval function image

lemma *has_integral_twiddle*:
assumes $0 < r$
and *hg*: $\bigwedge x.\ h(g\ x) = x$
and *gh*: $\bigwedge x.\ g(h\ x) = x$
and *contg*: $\bigwedge x.\ continuous\ (at\ x)\ g$
and *g*: $\bigwedge u\ v.\ \exists w\ z.\ g\ `\ cbox\ u\ v = cbox\ w\ z$
and *h*: $\bigwedge u\ v.\ \exists w\ z.\ h\ `\ cbox\ u\ v = cbox\ w\ z$
and *r*: $\bigwedge u\ v.\ content(g\ `\ cbox\ u\ v) = r * content\ (cbox\ u\ v)$
and *intfi*: $(f\ has\_integral\ i)\ (cbox\ a\ b)$
shows $((\lambda x.\ f(g\ x))\ has\_integral\ (1\ /\ r)\ *_R\ i)\ (h\ `\ cbox\ a\ b)$
proof (*cases* $cbox\ a\ b = \{\}$)
case *True*
then show *?thesis*
using *intfi* **by** *auto*
next
case *False*
obtain $w\ z$ **where** *wz*: $h\ `\ cbox\ a\ b = cbox\ w\ z$
using *h* **by** *blast*
have *inj*: *inj g inj h*
using *hg gh injI* **by** *metis+*
from *h* **obtain** *ha hb* **where** *h_eq*: $h\ `\ cbox\ a\ b = cbox\ ha\ hb$ **by** *blast*
have $\exists d.\ gauge\ d \wedge (\forall p.\ p\ tagged\_division\_of\ h\ `\ cbox\ a\ b \wedge d\ fine\ p \longrightarrow norm\ ((\sum (x,\ k){\in}p.\ content\ k *_R f\ (g\ x)) - (1\ /\ r) *_R i) < e)$

```
    if e > 0 for e
  proof −
    have e * r > 0 using that ‹0 < r› by simp
    with intfi[unfolded has_integral]
    obtain d where gauge d
              and d: ⋀p. p tagged_division_of cbox a b ∧ d fine p
                     ⟹ norm ((∑(x, k)∈p. content k *R f x) − i) < e * r
      by metis
    define d' where d' x = g −‘ d (g x) for x
    show ?thesis
    proof (rule_tac x=d' in exI, safe)
      show gauge d'
          using ‹gauge d› continuous_open_vimage[OF _ contg] by (auto simp:
gauge_def d'_def)
    next
      fix p
      assume ptag: p tagged_division_of h ‘ cbox a b and finep: d' fine p
      note p = tagged_division_ofD[OF ptag]
      have gab: g y ∈ cbox a b if y ∈ K (x, K) ∈ p for x y K
        by (metis hg inj(2) inj_image_mem_iff p(3) subsetCE that that)
      have gimp: (λ(x,K). (g x, g ‘ K)) ‘ p tagged_division_of (cbox a b) ∧
                 d fine (λ(x, k). (g x, g ‘ k)) ‘ p
        unfolding tagged_division_of
      proof safe
        show finite ((λ(x, k). (g x, g ‘ k)) ‘ p)
          using ptag by auto
        show d fine (λ(x, k). (g x, g ‘ k)) ‘ p
          using finep unfolding fine_def d'_def by auto
      next
        fix x K
        assume xk: (x, K) ∈ p
        show g x ∈ g ‘ K
          using p(2)[OF xk] by auto
        show ∃u v. g ‘ K = cbox u v
          using p(4)[OF xk] using assms(5−6) by auto
        fix x' K' u
        assume xk': (x', K') ∈ p and u: u ∈ interior (g ‘ K) u ∈ interior (g ‘ K')
        have interior K ∩ interior K' ≠ {}
        proof
          assume interior K ∩ interior K' = {}
          moreover have u ∈ g ‘ (interior K ∩ interior K')
            using interior_image_subset[OF ‹inj g› contg] u
            unfolding image_Int[OF inj(1)] by blast
          ultimately show False by blast
        qed
        then have same: (x, K) = (x', K')
          using ptag xk' xk by blast
        then show g x = g x'
          by auto
```

```
      show g u ∈ g ‘ K'if u ∈ K for u
        using that same by auto
      show g u ∈ g ‘ Kif u ∈ K' for u
        using that same by auto
    next
      fix x
      assume x ∈ cbox a b
      then have h x ∈ ⋃{k. ∃x. (x, k) ∈ p}
        using p(6) by auto
      then obtain X y where h x ∈ X (y, X) ∈ p by blast
      then show x ∈ ⋃{k. ∃x. (x, k) ∈ (λ(x, k). (g x, g ‘ k)) ‘ p}
        by clarsimp (metis (no_types, lifting) gh image_eqI pair_imageI)
    qed (use gab in auto)
    have ∗: inj_on (λ(x, k). (g x, g ‘ k)) p
      using inj(1) unfolding inj_on_def by fastforce
    have (∑(x,K)∈(λ(y,L). (g y, g ‘ L)) ‘ p. content K ∗R f x)
        = (∑u∈p. case case u of (x,K) ⇒ (g x, g ‘ K) of (y,L) ⇒ content L ∗R
f y)
      by (metis (mono_tags, lifting) ∗ sum.reindex_cong)
    also have ... = (∑(x,K)∈p. r ∗R content K ∗R f (g x))
      using r by (auto intro!: ∗ sum.cong simp: bij_betw_def dest!: p(4))
    finally
     have (∑(x, K)∈(λ(x,K). (g x, g ‘ K)) ‘ p. content K ∗R f x) − i = r ∗R
(∑(x,K)∈p. content K ∗R f (g x)) − i
      by (simp add: scaleR_right.sum split_def)
    also have ... = r ∗R ((∑(x,K)∈p. content K ∗R f (g x)) − (1 / r) ∗R i)
      using ‹0 < r› by (auto simp: scaleR_diff_right)
    finally show norm ((∑(x,K)∈p. content K ∗R f (g x)) − (1 / r) ∗R i) < e
      using d[OF gimp] ‹0 < r› by auto
  qed
 qed
 then show ?thesis
  by (auto simp: h_eq has_integral)
qed
```

6.14.26 Special case of a basic affine transformation

```
lemma AE_lborel_inner_neq:
 assumes k: k ∈ Basis
 shows AE x in lborel. x • k ≠ c
proof −
 interpret finite_product_sigma_finite λ_. lborel Basis
  proof qed simp
  have emeasure lborel {x∈space lborel. x • k = c}
      = emeasure (ΠM j::'a∈Basis. lborel) (ΠE j∈Basis. if j = k then {c} else
UNIV)
  using k
 by (auto simp add: lborel_eq[where 'a='a] emeasure_distr intro!: arg_cong2[where
f=emeasure])
```

```
      (auto simp: space_PiM PiE_iff extensional_def split: if_split_asm)
  also have ... = (∏ j∈Basis. emeasure lborel (if j = k then {c} else UNIV))
    by (intro measure_times) auto
  also have ... = 0
    by (intro prod_zero bexI[OF _ k]) auto
  finally show ?thesis
    by (subst AE_iff_measurable[OF _ refl]) auto
qed

lemma content_image_stretch_interval:
  fixes m :: 'a::euclidean_space ⇒ real
  defines s f x ≡ (∑ k::'a∈Basis. (f k * (x•k)) *R k)
  shows content (s m ‘ cbox a b) = |∏ k∈Basis. m k| * content (cbox a b)
proof cases
  have s[measurable]: s f ∈ borel →M borel for f
    by (auto simp: s_def[abs_def])
  assume m: ∀ k∈Basis. m k ≠ 0
  then have s_comp_s: s (λk. 1 / m k) ∘ s m = id s m ∘ s (λk. 1 / m k) = id
    by (auto simp: s_def[abs_def] fun_eq_iff euclidean_representation)
  then have inv (s (λk. 1 / m k)) = s m bij (s (λk. 1 / m k))
    by (auto intro: inv_unique_comp o_bij)
  then have eq: s m ‘ cbox a b = s (λk. 1 / m k) −‘ cbox a b
    using bij_vimage_eq_inv_image[OF ‹bij (s (λk. 1 / m k))›, of cbox a b] by
auto
  show ?thesis
    using m unfolding eq measure_def
    by (subst lborel_affine_euclidean[where c=m and t=0])
       (simp_all add: emeasure_density measurable_sets_borel[OF s] abs_prod
nn_integral_cmult
                      s_def[symmetric] emeasure_distr vimage_comp s_comp_s
enn2real_mult prod_nonneg)
next
  assume ¬ (∀ k∈Basis. m k ≠ 0)
  then obtain k where k: k ∈ Basis m k = 0 by auto
  then have [simp]: (∏ k∈Basis. m k) = 0
    by (intro prod_zero) auto
  have emeasure lborel {x∈space lborel. x ∈ s m ‘ cbox a b} = 0
  proof (rule emeasure_eq_0_AE)
    show AE x in lborel. x ∉ s m ‘ cbox a b
      using AE_lborel_inner_neq[OF ‹k∈Basis›]
    proof eventually_elim
      show x • k ≠ 0 ⟹ x ∉ s m ‘ cbox a b for x
        using k by (auto simp: s_def[abs_def] cbox_def)
    qed
  qed
  then show ?thesis
    by (simp add: measure_def)
qed
```

lemma *interval_image_affinity_interval*:
  $\exists u\ v.\ (\lambda x.\ m *_R (x::'a::euclidean\_space) + c)\ `\ cbox\ a\ b = cbox\ u\ v$
  unfolding *image_affinity_cbox*
  by *auto*

lemma *content_image_affinity_cbox*:
  $content((\lambda x::'a::euclidean\_space.\ m *_R x + c)\ `\ cbox\ a\ b) =$
    $|m| \hat{} DIM('a) * content\ (cbox\ a\ b)$ (**is** *?l = ?r*)
proof (*cases cbox a b = {}*)
  case *True* **then show** *?thesis* **by** *simp*
next
  case *False*
  show *?thesis*
  proof (*cases m ≥ 0*)
    case *True*
    with ‹*cbox a b ≠ {}*› **have** *cbox* $(m *_R a + c)\ (m *_R b + c) \neq \{\}$
      by (*simp add*: *box_ne_empty inner_left_distrib mult_left_mono*)
    moreover from *True* **have** $*: \bigwedge i.\ (m *_R b + c) \cdot i - (m *_R a + c) \cdot i = m *_R (b-a) \cdot i$
      by (*simp add*: *inner_simps field_simps*)
    ultimately show *?thesis*
      by (*simp add*: *image_affinity_cbox True content_cbox' prod.distrib inner_diff_left*)
  next
    case *False*
    with ‹*cbox a b ≠ {}*› **have** *cbox* $(m *_R b + c)\ (m *_R a + c) \neq \{\}$
      by (*simp add*: *box_ne_empty inner_left_distrib mult_left_mono*)
    moreover from *False* **have** $*: \bigwedge i.\ (m *_R a + c) \cdot i - (m *_R b + c) \cdot i = (-m) *_R (b-a) \cdot i$
      by (*simp add*: *inner_simps field_simps*)
    ultimately show *?thesis* **using** *False*
      by (*simp add*: *image_affinity_cbox content_cbox'*
        prod.distrib[*symmetric*] *inner_diff_left flip*: *prod_constant*)
  qed
qed

lemma *has_integral_affinity*:
  fixes *a* :: $'a::euclidean\_space$
  assumes (*f has_integral i*) (*cbox a b*)
    and $m \neq 0$
  shows $((\lambda x.\ f(m *_R x + c))\ has\_integral\ (1\ /\ (|m| \hat{} DIM('a))) *_R i)\ ((\lambda x.\ (1\ /\ m) *_R x + -((1\ /\ m) *_R c))\ `\ cbox\ a\ b)$
proof (*rule has_integral_twiddle*)
  show $\exists w\ z.\ (\lambda x.\ (1\ /\ m) *_R x + -((1\ /\ m) *_R c))\ `\ cbox\ u\ v = cbox\ w\ z$
      $\exists w\ z.\ (\lambda x.\ m *_R x + c)\ `\ cbox\ u\ v = cbox\ w\ z$ **for** *u v*
    using *interval_image_affinity_interval* **by** *blast+*
  show $content\ ((\lambda x.\ m *_R x + c)\ `\ cbox\ u\ v) = |m| \hat{} DIM('a) * content\ (cbox\ u\ v)$ **for** *u v*
    using *content_image_affinity_cbox* **by** *blast*

qed (*use assms zero_less_power* **in** ‹*auto simp*: *field_simps*›)

lemma *integrable_affinity*:
assumes *f integrable_on cbox a b*
and $m \neq 0$
shows $(\lambda x.\ f(m *_R x + c))$ *integrable_on* $((\lambda x.\ (1\ /\ m) *_R x + -((1/m) *_R c))$ ‘ *cbox a b*)
using *has_integral_affinity assms*
unfolding *integrable_on_def* **by** *blast*

lemmas *has_integral_affinity01* = *has_integral_affinity* [*of* _ _ 0 1::*real*, *simplified*]

lemma *integrable_on_affinity*:
assumes $m \neq 0$ *f integrable_on* (*cbox a b*)
shows $(\lambda x.\ f\ (m *_R x + c))$ *integrable_on* $((\lambda x.\ (1\ /\ m) *_R x - ((1\ /\ m) *_R c))$ ‘ *cbox a b*)
proof −
from *assms* **obtain** *I* **where** (*f has_integral I*) (*cbox a b*)
by (*auto simp*: *integrable_on_def*)
from *has_integral_affinity*[*OF this assms*(1), *of c*] **show** *?thesis*
by (*auto simp*: *integrable_on_def*)
qed

lemma *has_integral_cmul_iff*:
assumes $c \neq 0$
shows $((\lambda x.\ c *_R f\ x)$ *has_integral* $(c *_R I))\ A \longleftrightarrow (f$ *has_integral* $I)\ A$
using *assms has_integral_cmul*[*of f I A c*]
has_integral_cmul[*of* $\lambda x.\ c *_R f\ x\ c *_R I\ A$ *inverse c*]
by (*auto simp*: *field_simps*)

lemma *has_integral_cmul_iff'*:
assumes $c \neq 0$
shows $((\lambda x.\ c *_R f\ x)$ *has_integral* $I)\ A \longleftrightarrow (f$ *has_integral* $I\ /_R\ c)\ A$
using *assms* **by** (*metis divideR_right has_integral_cmul_iff*)

lemma *has_integral_affinity'*:
fixes $a :: 'a$::*euclidean_space*
assumes (*f has_integral i*) (*cbox a b*) **and** $m > 0$
shows $((\lambda x.\ f(m *_R x + c))$ *has_integral* $(i\ /_R\ m\ \hat{}\ DIM('a)))$
$(cbox\ ((a - c)\ /_R\ m)\ ((b - c)\ /_R\ m))$
proof (*cases cbox a b* = {})
case *True*
hence $(cbox\ ((a - c)\ /_R\ m)\ ((b - c)\ /_R\ m)) = \{\}$
using ‹$m > 0$› **unfolding** *box_eq_empty* **by** (*auto simp*: *algebra_simps*)
with *True* **and** *assms* **show** *?thesis* **by** *simp*
next
case *False*
have $((\lambda x.\ f\ (m *_R x + c))$ *has_integral* $(1\ /\ |m|\ \hat{}\ DIM('a)) *_R i)$

```
        ((λx. (1 / m) *R x + - ((1 / m) *R c)) ' cbox a b)
    using assms by (intro has_integral_affinity) auto
  also have ((λx. (1 / m) *R x + - ((1 / m) *R c)) ' cbox a b) =
            ((λx.  - ((1 / m) *R c) + x) ' (λx. (1 / m) *R x) ' cbox a b)
    by (simp add: image_image algebra_simps)
  also have (λx. (1 / m) *R x) ' cbox a b = cbox ((1 / m) *R a) ((1 / m) *R b)
using ‹m > 0› False
    by (subst image_smult_cbox) simp_all
  also have (λx. - ((1 / m) *R c) + x) ' ... = cbox ((a - c) /R m) ((b - c) /R
m)
   by (subst cbox_translation [symmetric]) (simp add: field_simps vector_add_divide_simps)
  finally show ?thesis using ‹m > 0›
    by (simp add: field_simps)
qed

lemma has_integral_affinity_iff:
  fixes f :: 'a :: euclidean_space ⇒ 'b :: real_normed_vector
  assumes m > 0
  shows   ((λx. f (m *R x + c)) has_integral (I /R m ^ DIM('a)))
              (cbox ((a - c) /R m) ((b - c) /R m)) ⟷
          (f has_integral I) (cbox a b) (is ?lhs = ?rhs)
proof
  assume ?lhs
  from has_integral_affinity'[OF this, of 1 / m -c /R m] and ‹m > 0›
  show ?rhs by (simp add: vector_add_divide_simps) (simp add: field_simps)
next
  assume ?rhs
  from has_integral_affinity'[OF this, of m c] and ‹m > 0›
  show ?lhs by simp
qed
```

6.14.27 Special case of stretching coordinate axes separately

```
lemma has_integral_stretch:
  fixes f :: 'a::euclidean_space ⇒ 'b::real_normed_vector
  assumes (f has_integral i) (cbox a b)
    and ∀k∈Basis. m k ≠ 0
  shows ((λx. f (∑k∈Basis. (m k * (x•k))*R k)) has_integral
         ((1/ |prod m Basis|) *R i)) ((λx. (∑k∈Basis. (1 / m k * (x•k))*R k)) '
cbox a b)
  apply (rule has_integral_twiddle[where f=f])
  unfolding zero_less_abs_iff content_image_stretch_interval
  unfolding image_stretch_interval empty_as_interval euclidean_eq_iff[where
'a='a]
  using assms
  by auto

lemma has_integral_stretch_real:
  fixes f :: real ⇒ 'b::real_normed_vector
```

assumes (*f has_integral i*) {*a*..*b*} **and** *m* ≠ *0*
shows ((λ*x*. *f* (*m* ∗ *x*)) *has_integral* (*1* / |*m*|) ∗$_R$ *i*) ((λ*x*. *x* / *m*) ‘ {*a*..*b*})
using *has_integral_stretch* [*of f i a b* λ*b*. *m*] *assms* **by** *simp*

lemma *integrable_stretch*:
fixes *f* :: ′*a*::*euclidean_space* ⇒ ′*b*::*real_normed_vector*
assumes *f integrable_on cbox a b*
and ∀ *k*∈*Basis*. *m k* ≠ *0*
shows (λ*x*::′*a*. *f* (∑ *k*∈*Basis*. (*m k* ∗ (*x*•*k*))∗$_R$ *k*)) *integrable_on*
((λ*x*. ∑ *k*∈*Basis*. (*1* / *m k* ∗ (*x*•*k*))∗$_R$ *k*) ‘ *cbox a b*)
using *assms* **unfolding** *integrable_on_def*
by (*force dest*: *has_integral_stretch*)

lemma *vec_lambda_eq_sum*:
(χ *k*. *f k* (*x* $ *k*)) = (∑ *k*∈*Basis*. (*f* (*axis_index k*) (*x* · *k*)) ∗$_R$ *k*) (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = (χ *k*. *f k* (*x* · *axis k 1*))
by (*simp add*: *cart_eq_inner_axis*)
also have ... = (∑ *u*∈*UNIV*. *f u* (*x* · *axis u 1*) ∗$_R$ *axis u 1*)
by (*simp add*: *vec_eq_iff axis_def if_distrib cong*: *if_cong*)
also have ... = *?rhs*
by (*simp add*: *Basis_vec_def UNION_singleton_eq_range sum.reindex axis_eq_axis inj_on_def*)
finally show *?thesis* .
qed

lemma *has_integral_stretch_cart*:
fixes *m* :: ′*n*::*finite* ⇒ *real*
assumes *f*: (*f has_integral i*) (*cbox a b*) **and** *m*: ⋀*k*. *m k* ≠ *0*
shows ((λ*x*. *f*(χ *k*. *m k* ∗ *x*$*k*)) *has_integral i* /$_R$ |*prod m UNIV*|)
((λ*x*. χ *k*. *x*$*k* / *m k*) ‘ (*cbox a b*))
proof −
have ∗: ∀ *k*:: *real*^′*n* ∈ *Basis*. *m* (*axis_index k*) ≠ *0*
using *axis_index* **by** (*simp add*: *m*)
have *eqp*: (∏ *k*:: *real*^′*n* ∈ *Basis*. *m* (*axis_index k*)) = *prod m UNIV*
by (*simp add*: *Basis_vec_def UNION_singleton_eq_range prod.reindex axis_eq_axis inj_on_def*)
show *?thesis*
using *has_integral_stretch* [*OF f* ∗] *vec_lambda_eq_sum* [**where** *f*=λ*i x*. *m i* ∗ *x*] *vec_lambda_eq_sum* [**where** *f*=λ*i x*. *x* / *m i*]
by (*simp add*: *field_simps eqp*)
qed

lemma *image_stretch_interval_cart*:
fixes *m* :: ′*n*::*finite* ⇒ *real*
shows (λ*x*. χ *k*. *m k* ∗ *x*$*k*) ‘ *cbox a b* =
(*if cbox a b* = {} *then* {}
else cbox (χ *k*. *min* (*m k* ∗ *a*$*k*) (*m k* ∗ *b*$*k*)) (χ *k*. *max* (*m k* ∗ *a*$*k*) (*m*

$k * b\$k)))$
proof −
have ∗: $(\sum k{\in}Basis.\ min\ (m\ (axis\_index\ k) * (a \cdot k))\ (m\ (axis\_index\ k) * (b \cdot k)) *_R k)$
$= (\chi\ k.\ min\ (m\ k * a\ \$\ k)\ (m\ k * b\ \$\ k))$
$(\sum k{\in}Basis.\ max\ (m\ (axis\_index\ k) * (a \cdot k))\ (m\ (axis\_index\ k) * (b \cdot k)) *_R k)$
$= (\chi\ k.\ max\ (m\ k * a\ \$\ k)\ (m\ k * b\ \$\ k))$
apply (*simp_all add*: *Basis_vec_def cart_eq_inner_axis UNION_singleton_eq_range sum.reindex axis_eq_axis inj_on_def*)
apply (*simp_all add*: *vec_eq_iff axis_def if_distrib cong*: *if_cong*)
done
show *?thesis*
by (*simp add*: *vec_lambda_eq_sum* [**where** $f{=}\lambda i\ x.\ m\ i * x$] *image_stretch_interval eq_cbox* ∗)
qed

6.14.28 even more special cases

lemma *uminus_interval_vector*[*simp*]:
fixes *a b* :: *'a*::*euclidean_space*
shows *uminus ‘ cbox a b* = *cbox* (−*b*) (−*a*)
proof −
have *x* ∈ *uminus ‘ cbox a b* **if** *x* ∈ *cbox* (− *b*) (− *a*) **for** *x*
by (*smt* (*verit*) *add.inverse_inverse image_iff inner_minus_left mem_box*(2) *that*)
then show *?thesis*
by (*auto simp*: *mem_box*)
qed

lemma *has_integral_reflect_lemma*[*intro*]:
assumes (*f has_integral i*) (*cbox a b*)
shows ((λ*x*. *f*(−*x*)) *has_integral i*) (*cbox* (−*b*) (−*a*))
using *has_integral_affinity*[*OF assms*, *of* −*1 0*]
by *auto*

lemma *has_integral_reflect_lemma_real*[*intro*]:
assumes (*f has_integral i*) {*a*..*b*::*real*}
shows ((λ*x*. *f*(−*x*)) *has_integral i*) {−*b* .. −*a*}
by (*metis has_integral_reflect_lemma interval_cbox assms*)

lemma *has_integral_reflect*[*simp*]:
((λ*x*. *f* (−*x*)) *has_integral i*) (*cbox* (−*b*) (−*a*)) ⟷ (*f has_integral i*) (*cbox a b*)
by (*auto dest*: *has_integral_reflect_lemma*)

lemma *has_integral_reflect_real*[*simp*]:
fixes *a b*::*real*
shows ((λ*x*. *f* (−*x*)) *has_integral i*) {−*b*..−*a*} ⟷ (*f has_integral i*) {*a*..*b*}
by (*metis has_integral_reflect interval_cbox*)

lemma *integrable_reflect*[*simp*]: $(\lambda x.\ f(-x))$ *integrable_on cbox* $(-b)$ $(-a) \longleftrightarrow f$ *integrable_on cbox a b*
unfolding *integrable_on_def* **by** *auto*

lemma *integrable_reflect_real*[*simp*]: $(\lambda x.\ f(-x))$ *integrable_on* $\{-b\ ..\ -a\} \longleftrightarrow$ *f integrable_on* $\{a..b::real\}$
unfolding *box_real*[*symmetric*]
by (*rule integrable_reflect*)

lemma *integral_reflect*[*simp*]: *integral* (*cbox* $(-b)$ $(-a)$) $(\lambda x.\ f\ (-x)) =$ *integral* (*cbox a b*) *f*
unfolding *integral_def* **by** *auto*

lemma *integral_reflect_real*[*simp*]: *integral* $\{-b\ ..\ -a\}$ $(\lambda x.\ f\ (-x)) =$ *integral* $\{a..b::real\}$ *f*
unfolding *box_real*[*symmetric*]
by (*rule integral_reflect*)

6.14.29 Stronger form of FCT; quite a tedious proof

lemma *split_minus*[*simp*]: $(\lambda(x,\ k).\ f\ x\ k)\ x - (\lambda(x,\ k).\ g\ x\ k)\ x = (\lambda(x,\ k).\ f\ x\ k - g\ x\ k)\ x$
by (*simp add*: *split_def*)

theorem *fundamental_theorem_of_calculus_interior*:
fixes *f* :: *real* ⇒ *'a*::*real_normed_vector*
assumes $a \le b$
and *contf*: *continuous_on* $\{a..b\}$ *f*
and *derf*: $\bigwedge x.\ x \in \{a <..< b\} \Longrightarrow$ (*f has_vector_derivative f' x*) (*at x*)
shows (*f' has_integral* (*f b* − *f a*)) $\{a..b\}$
proof (*cases* $a = b$)
case *True*
then have ∗: *cbox a b* = $\{b\}$ *f b* − *f a* = *0*
by (*auto simp add*: *order_antisym*)
with *True* **show** *?thesis* **by** *auto*
next
case *False*
with ‹$a \le b$› **have** *ab*: $a < b$ **by** *arith*
show *?thesis*
unfolding *has_integral_factor_content_real*
proof (*intro allI impI*)
fix *e* :: *real*
assume *e*: *e* > *0*
then have *eba8*: (*e* ∗ (*b*−*a*)) / *8* > *0*
using *ab* **by** (*auto simp add*: *field_simps*)
note *derf_exp* = *derf*[*unfolded has_vector_derivative_def has_derivative_at_alt*, *THEN conjunct2*, *rule_format*]
have *bounded*: $\bigwedge x.\ x \in \{a<..<b\} \Longrightarrow$ *bounded_linear* $(\lambda u.\ u *_R f' x)$

by (*simp add*: *bounded_linear_scaleR_left*)
have $\forall x \in box\ a\ b.\ \exists d{>}0.\ \forall y.\ norm\ (y{-}x) < d \longrightarrow norm\ (f\ y - f\ x - (y{-}x) *_R f'\ x) \leq e/2 * norm\ (y{-}x)$
(**is** $\forall x \in box\ a\ b.\ ?Q\ x$) — The explicit quantifier is required by the following step
proof
fix *x* **assume** $x \in box\ a\ b$
with *e* **show** *?Q x*
using *derf_exp* [*of x e/2*] **by** *auto*
qed
then obtain *d* **where** *d*: $\bigwedge x.\ 0 < d\ x$
$\bigwedge x\ y.\ [\![x \in box\ a\ b;\ norm\ (y{-}x) < d\ x]\!] \Longrightarrow norm\ (f\ y - f\ x - (y{-}x) *_R f'\ x) \leq e/2 * norm\ (y{-}x)$
unfolding *bgauge_existence_lemma* **by** *metis*
have *bounded* (*f ‘ cbox a b*)
using *compact_cbox assms* **by** (*auto simp*: *compact_imp_bounded compact_continuous_image*)
then obtain *B*
where $0 < B$ **and** *B*: $\bigwedge x.\ x \in f\ `\ cbox\ a\ b \Longrightarrow norm\ x \leq B$
unfolding *bounded_pos* **by** *metis*
obtain *da* **where** $0 < da$
and *da*: $\bigwedge c.\ [\![a \leq c;\ \{a..c\} \subseteq \{a..b\};\ \{a..c\} \subseteq ball\ a\ da]\!]$
$\Longrightarrow norm\ (content\ \{a..c\} *_R f'\ a - (f\ c - f\ a)) \leq (e * (b{-}a))\ /\ 4$
proof −
have *continuous* (*at a within* {*a..b*}) *f*
using *contf continuous_on_eq_continuous_within* **by** *force*
with *eba8* **obtain** *k* **where** $0 < k$
and *k*: $\bigwedge x.\ [\![x \in \{a..b\};\ 0 < norm\ (x{-}a);\ norm\ (x{-}a) < k]\!] \Longrightarrow norm\ (f\ x - f\ a) < e * (b{-}a)\ /\ 8$
unfolding *continuous_within Lim_within dist_norm* **by** *metis*
obtain *l* **where** *l*: $0 < l$ *norm* $(l *_R f'\ a) \leq e * (b{-}a)\ /\ 8$
proof (*cases* $f'\ a = 0$)
case *True* **with** *ab e that* **show** *?thesis* **by** *auto*
next
case *False*
show *?thesis*
proof
show $norm\ ((e * (b - a)\ /\ 8\ /\ norm\ (f'\ a)) *_R f'\ a) \leq e * (b - a)\ /\ 8$
$0 < e * (b - a)\ /\ 8\ /\ norm\ (f'\ a)$
using *False ab e* **by** (*auto simp add*: *field_simps*)
qed
qed
have $norm\ (content\ \{a..c\} *_R f'\ a - (f\ c - f\ a)) \leq e * (b{-}a)\ /\ 4$
if $a \leq c$ $\{a..c\} \subseteq \{a..b\}$ **and** *bmin*: $\{a..c\} \subseteq ball\ a\ (min\ k\ l)$ **for** *c*
proof −
have *minkl*: $|a - x| < min\ k\ l$ **if** $x \in \{a..c\}$ **for** *x*
using *bmin dist_real_def that* **by** *auto*
then have *lel*: $|c - a| \leq |l|$

```
        using that by force
       have norm ((c − a) *R f′ a − (f c − f a)) ≤ norm ((c − a) *R f′ a) +
norm (f c − f a)
        by (rule norm_triangle_ineq4)
      also have ... ≤ e * (b−a) / 8 + e * (b−a) / 8
      proof (rule add_mono)
        have norm ((c − a) *R f′ a) ≤ norm (l *R f′ a)
          by (auto intro: mult_right_mono [OF lel])
        with l show norm ((c − a) *R f′ a) ≤ e * (b−a) / 8
          by linarith
      next
        have norm (f c − f a) < e * (b−a) / 8
        proof (cases a = c)
          case True then show ?thesis
            using eba8 by auto
        next
          case False show ?thesis
            by (rule k) (use minkl ‹a ≤ c› that False in auto)
        qed
        then show norm (f c − f a) ≤ e * (b−a) / 8 by simp
      qed
      finally show norm (content {a..c} *R f′ a − (f c − f a)) ≤ e * (b−a) / 4
        unfolding content_real[OF ‹a ≤ c›] by auto
    qed
    then show ?thesis
      by (rule_tac da=min k l in that) (auto simp: l ‹0 < k›)
  qed
  obtain db where 0 < db
    and db: ⋀c. ⟦c ≤ b; {c..b} ⊆ {a..b}; {c..b} ⊆ ball b db⟧
                  ⟹ norm (content {c..b} *R f′ b − (f b − f c)) ≤ (e * (b−a))
/ 4
  proof −
    have continuous (at b within {a..b}) f
      using contf continuous_on_eq_continuous_within by force
    with eba8 obtain k
      where 0 < k
        and k: ⋀x. ⟦x ∈ {a..b}; 0 < norm(x−b); norm(x−b) < k⟧
                 ⟹ norm (f b − f x) < e * (b−a) / 8
    unfolding continuous_within Lim_within dist_norm norm_minus_commute
by metis
    obtain l where l: 0 < l norm (l *R f′ b) ≤ (e * (b−a)) / 8
    proof (cases f′ b = 0)
      case True thus ?thesis
        using ab e that by auto
    next
      case False show ?thesis
      proof
        show norm ((e * (b − a) / 8 / norm (f′ b)) *R f′ b) ≤ e * (b − a) / 8
             0 < e * (b − a) / 8 / norm (f′ b)
```

```
        using False ab e by (auto simp add: field_simps)
      qed
    qed
    have norm (content {c..b} *R f' b - (f b - f c)) ≤ e * (b-a) / 4
      if c ≤ b {c..b} ⊆ {a..b} and bmin: {c..b} ⊆ ball b (min k l) for c
    proof -
      have minkl: |b - x| < min k l if x ∈ {c..b} for x
        using bmin dist_real_def that by auto
      then have lel: |b - c| ≤ |l|
        using that by force
       have norm ((b - c) *R f' b - (f b - f c)) ≤ norm ((b - c) *R f' b) +
norm (f b - f c)
        by (rule norm_triangle_ineq4)
      also have ... ≤ e * (b-a) / 8 + e * (b-a) / 8
      proof (rule add_mono)
        have norm ((b - c) *R f' b) ≤ norm (l *R f' b)
          by (auto intro: mult_right_mono [OF lel])
        also have ... ≤ e * (b-a) / 8
          by (rule l)
        finally show norm ((b - c) *R f' b) ≤ e * (b-a) / 8 .
      next
        have norm (f b - f c) < e * (b-a) / 8
        proof (cases b = c)
          case True with eba8 show ?thesis
            by auto
        next
          case False show ?thesis
            by (rule k) (use minkl ‹c ≤ b› that False in auto)
        qed
        then show norm (f b - f c) ≤ e * (b-a) / 8 by simp
      qed
      finally show norm (content {c..b} *R f' b - (f b - f c)) ≤ e * (b-a) / 4
        unfolding content_real[OF ‹c ≤ b›] by auto
    qed
    then show ?thesis
      by (rule_tac db=min k l in that) (auto simp: l ‹0 < k›)
  qed
  let ?d = (λx. ball x (if x=a then da else if x=b then db else d x))
  show ∃ d. gauge d ∧ (∀ p. p tagged_division_of {a..b} ∧ d fine p ⟶
          norm ((∑(x,K)∈p. content K *R f' x) - (f b - f a)) ≤ e * content
{a..b})
  proof (rule exI, safe)
    show gauge ?d
      using ab ‹db > 0› ‹da > 0› d(1) by (auto intro: gauge_ball_dependent)
  next
    fix p
    assume ptag: p tagged_division_of {a..b} and fine: ?d fine p
    let ?A = {t. fst t ∈ {a, b}}
    note p = tagged_division_ofD[OF ptag]
```

have *pA*: $p = (p \cap ?A) \cup (p - ?A)$ *finite* $(p \cap ?A)$ *finite* $(p - ?A)$ $(p \cap ?A) \cap (p - ?A) = \{\}$
using *ptag fine* **by** *auto*
have *le_xz*: $\bigwedge w\ x\ y\ z{::}real.\ y \le z/2 \Longrightarrow w - x \le z/2 \Longrightarrow w + y \le x + z$
by *arith*
have *non*: *False* **if** *xk*: $(x,K) \in p$ **and** $x \neq a\ x \neq b$
and *less*: $e * (Sup\ K - Inf\ K)/2 < norm\ (content\ K *_R f'\ x - (f\ (Sup\ K) - f\ (Inf\ K)))$
for $x\ K$
proof −
obtain $u\ v$ **where** *k*: $K = cbox\ u\ v$
using *p*(4) *xk* **by** *blast*
then have $u \le v$ **and** *uv*: $\{u, v\} \subseteq cbox\ u\ v$
using *p*(2)[*OF xk*] **by** *auto*
then have *result*: $e * (v - u)/2 < norm\ ((v - u) *_R f'\ x - (f\ v - f\ u))$
using *less*[*unfolded k box_real interval_bounds_real content_real*] **by** *auto*
then have $x \in box\ a\ b$
using *p*(2) *p*(3) ‹$x \neq a$› ‹$x \neq b$› *xk* **by** *fastforce*
with *d* **have** ∗: $\bigwedge y.\ norm\ (y-x) < d\ x \Longrightarrow norm\ (f\ y - f\ x - (y-x) *_R f'\ x) \le e/2 * norm\ (y-x)$
by *metis*
have *xd*: $norm\ (u - x) < d\ x$ $norm\ (v - x) < d\ x$
using *fineD*[*OF fine xk*] ‹$x \neq a$› ‹$x \neq b$› *uv*
by (*auto simp add*: *k subset_eq dist_commute dist_real_def*)
have $norm\ ((v - u) *_R f'\ x - (f\ v - f\ u)) = norm\ ((f\ u - f\ x - (u - x) *_R f'\ x) - (f\ v - f\ x - (v - x) *_R f'\ x))$
by (*rule arg_cong*[**where** *f*=*norm*]) (*auto simp*: *scaleR_left.diff*)
also have $\ldots \le e/2 * norm\ (u - x) + e/2 * norm\ (v - x)$
by (*metis norm_triangle_le_diff add_mono* ∗ *xd*)
also have $\ldots \le e/2 * norm\ (v - u)$
using *p*(2)[*OF xk*] **by** (*auto simp add*: *field_simps k*)
also have $\ldots < norm\ ((v - u) *_R f'\ x - (f\ v - f\ u))$
using *result* **by** (*simp add*: ‹$u \le v$›)
finally have $e * (v - u)/2 < e * (v - u)/2$
using *uv* **by** *auto*
then show *False* **by** *auto*
qed
have $norm\ (\sum (x,\ K){\in}p - ?A.\ content\ K *_R f'\ x - (f\ (Sup\ K) - f\ (Inf\ K))) \le (\sum (x,\ K){\in}p - ?A.\ norm\ (content\ K *_R f'\ x - (f\ (Sup\ K) - f\ (Inf\ K))))$
by (*auto intro*: *sum_norm_le*)
also have ... $\le (\sum n{\in}p - ?A.\ e * (case\ n\ of\ (x,\ k) \Rightarrow Sup\ k - Inf\ k)/2)$
using *non* **by** (*fastforce intro*: *sum_mono*)
finally have *I*: $norm\ (\sum (x,\ k){\in}p - ?A.\ content\ k *_R f'\ x - (f\ (Sup\ k) - f\ (Inf\ k))) \le (\sum n{\in}p - ?A.\ e * (case\ n\ of\ (x,\ k) \Rightarrow Sup\ k - Inf\ k))/2$
by (*simp add*: *sum_divide_distrib*)
have *II*: $norm\ (\sum (x,\ k){\in}p \cap ?A.\ content\ k *_R f'\ x - (f\ (Sup\ k) - f\ (Inf$

```
k))) −
          (∑ n∈p ∩ ?A. e * (case n of (x, k) ⇒ Sup k − Inf k))
          ≤ (∑ n∈p − ?A. e * (case n of (x, k) ⇒ Sup k − Inf k))/2
      proof −
        have ge0: 0 ≤ e * (Sup k − Inf k) if xkp: (x, k) ∈ p ∩ ?A for x k
        proof −
          obtain u v where uv: k = cbox u v
            by (meson Int_iff xkp p(4))
          with p that have cbox u v ≠ {}
            by blast
          then show 0 ≤ e * ((Sup k) − (Inf k))
            unfolding uv using e by (auto simp add: field_simps)
        qed
        let ?B = λx. {t ∈ p. fst t = x ∧ content (snd t) ≠ 0}
        let ?C = {t ∈ p. fst t ∈ {a, b} ∧ content (snd t) ≠ 0}
        have norm (∑ (x, k)∈p ∩ {t. fst t ∈ {a, b}}. content k *R f' x − (f (Sup
k) − f (Inf k))) ≤ e * (b−a)/2
        proof −
          have *: ⋀S f e. sum f S = sum f (p ∩ ?C) ⟹ norm (sum f (p ∩ ?C))
≤ e ⟹ norm (sum f S) ≤ e
            by auto
          have 1: content K *R (f' x) − (f ((Sup K)) − f ((Inf K))) = 0
            if (x,K) ∈ p ∩ {t. fst t ∈ {a, b}} − p ∩ ?C for x K
          proof −
            have xk: (x,K) ∈ p and k0: content K = 0
              using that by auto
            then obtain u v where uv: K = cbox u v u = v
              using xk k0 p by fastforce
            then show content K *R (f' x) − (f ((Sup K)) − f ((Inf K))) = 0
              using xk unfolding uv by auto
          qed
         have 2: norm(∑ (x,K)∈p ∩ ?C. content K *R f' x − (f (Sup K) − f (Inf
K))) ≤ e * (b−a)/2
          proof −
            have norm_le: norm (sum f S) ≤ e
             if ⋀x y. ⟦x ∈ S; y ∈ S⟧ ⟹ x = y ⋀x. x ∈ S ⟹ norm (f x) ≤ e e > 0
              for S f and e :: real
            proof (cases S = {})
              case True
              with that show ?thesis by auto
            next
              case False then obtain x where x ∈ S
                by auto
              then have S = {x}
                using that(1) by auto
              then show ?thesis
                using ‹x ∈ S› that(2) by auto
            qed
            have *: p ∩ ?C = ?B a ∪ ?B b
```

```
          by blast
        then have norm (∑(x,K)∈p ∩ ?C. content K *R f' x − (f (Sup K) −
f (Inf K))) =
                  norm (∑(x,K) ∈ ?B a ∪ ?B b. content K *R f' x − (f (Sup
K) − f (Inf K)))
          by simp
        also have ... = norm ((∑(x,K) ∈ ?B a. content K *R f' x − (f (Sup
K) − f (Inf K))) +
                                (∑(x,K) ∈ ?B b. content K *R f' x − (f (Sup K) − f
(Inf K))))
          using p(1) ab e by (subst sum.union_disjoint) auto
        also have ... ≤ e * (b − a) / 4 + e * (b − a) / 4
        proof (rule norm_triangle_le [OF add_mono])
          have pa: ∃ v. k = cbox a v ∧ a ≤ v if (a, k) ∈ p for k
            using p that by fastforce
           show norm (∑(x,K) ∈ ?B a. content K *R f' x − (f (Sup K) − f
(Inf K))) ≤ e * (b − a) / 4
          proof (intro norm_le; clarsimp)
            fix K K'
            assume K: (a, K) ∈ p (a, K') ∈ p and ne0: content K ≠ 0 content
K' ≠ 0
              with pa obtain v v' where v: K = cbox a v a ≤ v and v': K' =
cbox a v' a ≤ v'
              by blast
            let ?v = min v v'
            have box a ?v ⊆ K ∩ K'
              unfolding v v' by (auto simp add: mem_box)
            then have interior (box a (min v v')) ⊆ interior K ∩ interior K'
              using interior_Int interior_mono by blast
            moreover have (a + ?v)/2 ∈ box a ?v
              using ne0  unfolding v v' content_eq_0 not_le
              by (auto simp add: mem_box)
            ultimately have (a + ?v)/2 ∈ interior K ∩ interior K'
              unfolding interior_open[OF open_box] by auto
            then show K = K'
              using p(5)[OF K] by auto
          next
            fix K
            assume K: (a, K) ∈ p and ne0: content K ≠ 0
            show norm (content c *R f' a − (f (Sup c) − f (Inf c))) * 4 ≤ e *
(b−a)
              if (a, c) ∈ p and ne0: content c ≠ 0 for c
            proof −
              obtain v where v: c = cbox a v and a ≤ v
                using pa[OF ‹(a, c) ∈ p›] by metis
              then have a ∈ {a..v} v ≤ b
                using p(3)[OF ‹(a, c) ∈ p›] by auto
              moreover have {a..v} ⊆ ball a da
                 using fineD[OF ‹?d fine p› ‹(a, c) ∈ p›] by (simp add: v split:
```

```
if_split_asm)
              ultimately show ?thesis
                unfolding v interval_bounds_real[OF ‹a ≤ v›] box_real
                using da ‹a ≤ v› by auto
            qed
          qed (use ab e in auto)
        next
          have pb: ∃ v. k = cbox v b ∧ b ≥ v if (b, k) ∈ p for k
            using p that by fastforce
           show norm (∑(x,K) ∈ ?B b. content K *R f' x − (f (Sup K) − f
(Inf K))) ≤ e * (b − a) / 4
          proof (intro norm_le; clarsimp)
            fix K K'
            assume K: (b, K) ∈ p (b, K') ∈ p and ne0: content K ≠ 0 content
K' ≠ 0
              with pb obtain v v' where v: K = cbox v b v ≤ b and v': K' =
cbox v' b v' ≤ b
              by blast
            let ?v = max v v'
            have box ?v b ⊆ K ∩ K'
              unfolding v v' by (auto simp: mem_box)
            then have interior (box (max v v') b) ⊆ interior K ∩ interior K'
              using interior_Int interior_mono by blast
            moreover have ((b + ?v)/2) ∈ box ?v b
                using ne0 unfolding v v' content_eq_0 not_le by (auto simp:
mem_box)
            ultimately have ((b + ?v)/2) ∈ interior K ∩ interior K'
              unfolding interior_open[OF open_box] by auto
            then show K = K'
              using p(5)[OF K] by auto
          next
            fix K
            assume K: (b, K) ∈ p and ne0: content K ≠ 0
            show norm (content c *R f' b − (f (Sup c) − f (Inf c))) * 4 ≤ e *
(b−a)
              if (b, c) ∈ p and ne0: content c ≠ 0 for c
            proof −
            obtain v where v: c = cbox v b and v ≤ b
              using ‹(b, c) ∈ p› pb by blast
            then have v ≥ ab ∈ {v.. b}
              using p(3)[OF ‹(b, c) ∈ p›] by auto
            moreover have {v..b} ⊆ ball b db
               using fineD[OF ‹?d fine p› ‹(b, c) ∈ p›] box_real(2) v False by
force
            ultimately show ?thesis
              using db v by auto
            qed
          qed (use ab e in auto)
        qed
```

```
          also have ... = e * (b−a)/2
            by simp
          finally show norm (∑(x,k)∈p ∩ ?C.
                    content k *R f' x − (f (Sup k) − f (Inf k))) ≤ e * (b−a)/2 .
        qed
        show norm (∑(x, k)∈p ∩ ?A. content k *R f' x − (f ((Sup k)) − f ((Inf
k)))) ≤ e * (b−a)/2
          apply (rule * [OF sum.mono_neutral_right[OF pA(2)]])
          using 1 2 by (auto simp: split_paired_all)
      qed
      also have ... = (∑n∈p. e * (case n of (x, k) ⇒ Sup k − Inf k))/2
        unfolding sum_distrib_left[symmetric]
        by (subst additive_tagged_division_1[OF ‹a ≤ b› ptag]) auto
      finally have norm_le: norm (∑(x,K)∈p ∩ {t. fst t ∈ {a, b}}. content K
*R f' x − (f (Sup K) − f (Inf K)))
          ≤ (∑n∈p. e * (case n of (x, K) ⇒ Sup K − Inf K))/2 .
      have le2: ⋀x s1 s2::real. 0 ≤ s1 ⟹ x ≤ (s1 + s2)/2 ⟹ x − s1 ≤ s2/2
        by auto
      show ?thesis
        apply (rule le2 [OF sum_nonneg])
        using ge0 apply force
          by (metis (no_types, lifting) Diff_Diff_Int Diff_subset norm_le p(1)
sum.subset_diff)
    qed
    note * = additive_tagged_division_1[OF assms(1) ptag, symmetric]
    have norm (∑(x,K)∈p ∩ ?A ∪ (p − ?A). content K *R f' x − (f (Sup K)
− f (Inf K)))
          ≤ e * (∑(x,K)∈p ∩ ?A ∪ (p − ?A). Sup K − Inf K)
      unfolding sum_distrib_left
      unfolding sum.union_disjoint[OF pA(2−)]
      using le_xz norm_triangle_le I II by blast
    then
    show norm ((∑(x,K)∈p. content K *R f' x) − (f b − f a)) ≤ e * content
{a..b}
    by (simp only: content_real[OF ‹a ≤ b›] *[of λx. x] *[of f] sum_subtractf[symmetric]
split_minus pA(1) [symmetric])
  qed
 qed
qed
```

6.14.30 Stronger form with finite number of exceptional points

```
lemma fundamental_theorem_of_calculus_interior_strong:
 fixes f :: real ⇒ 'a::banach
 assumes finite S
   and a ≤ b ⋀x. x ∈ {a <..< b} − S ⟹ (f has_vector_derivative f'(x)) (at x)
   and continuous_on {a .. b} f
 shows (f' has_integral (f b − f a)) {a .. b}
  using assms
```

proof (*induction arbitrary*: *a b*)
case *empty*
then show *?case*
using *fundamental_theorem_of_calculus_interior* **by** *force*
next
case (*insert x S*)
show *?case*
proof (*cases x* ∈ {*a*<..<*b*})
case *False* **then show** *?thesis*
using *insert* **by** *blast*
next
case *True* **then have** *a* < *x x* < *b*
by *auto*
have (*f'* *has_integral f x* − *f a*) {*a..x*} (*f'* *has_integral f b* − *f x*) {*x..b*}
using ‹*continuous_on* {*a..b*} *f*› ‹*a* < *x*› ‹*x* < *b*› *continuous_on_subset* **by** (*force simp*: *intro*!: *insert*)+
then have (*f'* *has_integral f x* − *f a* + (*f b* − *f x*)) {*a..b*}
using ‹*a* < *x*› ‹*x* < *b*› *has_integral_combine less_imp_le* **by** *blast*
then show *?thesis*
by *simp*
qed
qed

corollary *fundamental_theorem_of_calculus_strong*:
fixes *f* :: *real* ⇒ *'a*::*banach*
assumes *finite S*
and *a* ≤ *b*
and *vec*: ⋀*x*. *x* ∈ {*a..b*} − *S* ⟹ (*f* *has_vector_derivative f'*(*x*)) (*at x*)
and *continuous_on* {*a..b*} *f*
shows (*f'* *has_integral* (*f b* − *f a*)) {*a..b*}
by (*rule fundamental_theorem_of_calculus_interior_strong* [*OF* ‹*finite S*›]) (*force simp*: *assms*)+

proposition *indefinite_integral_continuous_left*:
fixes *f*:: *real* ⇒ *'a*::*banach*
assumes *intf*: *f integrable_on* {*a..b*} **and** *a* < *c c* ≤ *b e* > *0*
obtains *d* **where** *d* > *0*
and ∀ *t*. *c* − *d* < *t* ∧ *t* ≤ *c* ⟶ *norm* (*integral* {*a..c*} *f* − *integral* {*a..t*} *f*) < *e*
proof −
obtain *w* **where** *w* > *0* **and** *w*: ⋀*t*. ⟦*c* − *w* < *t*; *t* < *c*⟧ ⟹ *norm* (*f c*) ∗ *norm*(*c* − *t*) < *e*/*3*
proof (*cases f c* = *0*)
case *False*
hence *e3*: *0* < *e*/*3* / *norm* (*f c*) **using** ‹*e*>*0*› **by** *simp*
moreover have *norm* (*f c*) ∗ *norm* (*c* − *t*) < *e*/*3*
if *t* < *c* **and** *c* − *e*/*3* / *norm* (*f c*) < *t* **for** *t*
unfolding *real_norm_def*
by (*smt* (*verit*) *False divide_right_mono nonzero_mult_div_cancel_left*

```
norm_eq_zero norm_ge_zero that)
    ultimately show ?thesis
      using that by auto
  qed (use ‹e > 0› in auto)

  let ?SUM = λp. (∑(x,K) ∈ p. content K *R f x)
  have e3: e/3 > 0
    using ‹e > 0› by auto
  have f integrable_on {a..c}
    using ‹a < c› ‹c ≤ b› by (auto intro: integrable_subinterval_real[OF intf])
  then obtain d1 where gauge d1 and d1:
     ⋀p. ⟦p tagged_division_of {a..c}; d1 fine p⟧ ⟹ norm (?SUM p − integral
{a..c} f) < e/3
    using integrable_integral has_integral_real e3 by metis
  define d where [abs_def]: d x = ball x w ∩ d1 x for x
  have gauge d
    unfolding d_def using ‹w > 0› ‹gauge d1› by auto
  then obtain k where 0 < k and k: ball c k ⊆ d c
    by (meson gauge_def open_contains_ball)

  let ?d = min k (c − a)/2
  show thesis
  proof (intro that[of ?d] allI impI, safe)
    show ?d > 0
      using ‹0 < k› ‹a < c› by auto
  next
    fix t
    assume t: c − ?d < t t ≤ c
    show norm (integral ({a..c}) f − integral ({a..t}) f) < e
    proof (cases t < c)
      case False with ‹t ≤ c› show ?thesis
        by (simp add: ‹e > 0›)
    next
      case True
      have f integrable_on {a..t}
        using ‹t < c› ‹c ≤ b› by (auto intro: integrable_subinterval_real[OF intf])
      then obtain d2 where d2: gauge d2
        ⋀p. p tagged_division_of {a..t} ∧ d2 fine p ⟹ norm (?SUM p − integral
{a..t} f) < e/3
        using integrable_integral has_integral_real e3 by metis
      define d3 where d3 x = (if x ≤ t then d1 x ∩ d2 x else d1 x) for x
      have gauge d3
        using ‹gauge d1› ‹gauge d2› unfolding d3_def gauge_def by auto
      then obtain p where ptag: p tagged_division_of {a..t} and pfine: d3 fine p
        by (metis box_real(2) fine_division_exists)
      note p' = tagged_division_ofD[OF ptag]
      have pt: (x,K)∈p ⟹ x ≤ t for x K
        by (meson atLeastAtMost_iff p'(2) p'(3) subsetCE)
      with pfine have d2 fine p
```

```
    unfolding fine_def d3_def by fastforce
  then have d2_fin: norm (?SUM p − integral {a..t} f) < e/3
    using d2(2) ptag by auto
  have eqs: {a..c} ∩ {x. x ≤ t} = {a..t} {a..c} ∩ {x. x ≥ t} = {t..c}
    using t by (auto simp add: field_simps)
  have p ∪ {(c, {t..c})} tagged_division_of {a..c}
  proof (intro tagged_division_Un_interval_real)
    show {(c, {t..c})} tagged_division_of {a..c} ∩ {x. t ≤ x • 1}
      using ‹t ≤ c› by (auto simp: eqs tagged_division_of_self_real)
  qed (auto simp: eqs ptag)
  moreover
  have d1 fine p ∪ {(c, {t..c})}
    unfolding fine_def
  proof safe
    fix x K y
    assume (x,K) ∈ p and y ∈ K then show y ∈ d1 x
      by (metis Int_iff d3_def subsetD fineD pfine)
  next
    fix x assume x ∈ {t..c}
    then have dist c x < k
      using t(1) by (auto simp add: field_simps dist_real_def)
    with k show x ∈ d1 c
      unfolding d_def by auto
  qed
  ultimately have d1_fin: norm (?SUM(p ∪ {(c, {t..c})}) − integral {a..c}
f) < e/3
    using d1 by metis
  have SUMEQ: ?SUM(p ∪ {(c, {t..c})}) = (c − t) *R f c + ?SUM p
  proof −
    have ?SUM(p ∪ {(c, {t..c})}) = (content{t..c} *R f c) + ?SUM p
    proof (subst sum.union_disjoint)
      show p ∩ {(c, {t..c})} = {}
        using ‹t < c› pt by force
    qed (use p'(1) in auto)
    also have ... = (c − t) *R f c + ?SUM p
      using ‹t ≤ c› by auto
    finally show ?thesis .
  qed
  have c − k < t
    using ‹k>0› t(1) by (auto simp add: field_simps)
  moreover have k ≤ w
  proof (rule ccontr)
    assume ¬ k ≤ w
    then have c + (k + w) / 2 ∉ d c
      by (auto simp add: field_simps not_le not_less dist_real_def d_def)
    then have c + (k + w) / 2 ∉ ball c k
      using k by blast
    then show False
      using ‹0 < w› ‹¬ k ≤ w› dist_real_def by auto
```

```
      qed
      ultimately have cwt: c − w < t
        by (auto simp add: field_simps)
      have eq: integral {a..c} f − integral {a..t} f = −(((c − t) *R f c + ?SUM
p) −
              integral {a..c} f) + (?SUM p − integral {a..t} f) + (c − t) *R f c
        by auto
      have norm (integral {a..c} f − integral {a..t} f) < e/3 + e/3 + e/3
        unfolding eq
      proof (intro norm_triangle_lt add_strict_mono)
        show norm (− ((c − t) *R f c + ?SUM p − integral {a..c} f)) < e/3
          by (metis SUMEQ d1_fin norm_minus_cancel)
        show norm (?SUM p − integral {a..t} f) < e/3
          using d2_fin by blast
        show norm ((c − t) *R f c) < e/3
          using w cwt ‹t < c› by simp (simp add: field_simps)
      qed
      then show ?thesis by simp
    qed
  qed
qed

lemma indefinite_integral_continuous_right:
  fixes f :: real ⇒ 'a::banach
  assumes f integrable_on {a..b}
    and a ≤ c
    and c < b
    and e > 0
  obtains d where 0 < d
    and ∀ t. c ≤ t ∧ t < c + d ⟶ norm (integral {a..c} f − integral {a..t} f) <
e
proof −
  have intm: (λx. f (− x)) integrable_on {−b .. −a} − b < − c − c ≤ − a
    using assms by auto
  from indefinite_integral_continuous_left[OF intm ‹e>0›]
  obtain d where 0 < d
    and d: ⋀t. ⟦− c − d < t; t ≤ −c⟧
           ⟹ norm (integral {− b..− c} (λx. f (−x)) − integral {− b..t} (λx. f
(−x))) < e
    by metis
  let ?d = min d (b − c)
  show ?thesis
  proof (intro that[of ?d] allI impI)
    show 0 < ?d
      using ‹0 < d› ‹c < b› by auto
    fix t :: real
    assume t: c ≤ t ∧ t < c + ?d
    have ∗: integral {a..c} f = integral {a..b} f − integral {c..b} f
            integral {a..t} f = integral {a..b} f − integral {t..b} f
```

```
    using assms t by (auto simp: algebra_simps integral_combine)
   have (− c) − d < (− t) − t ≤ − c
    using t by auto
   from d[OF this] show norm (integral {a..c} f − integral {a..t} f) < e
    by (auto simp add: algebra_simps norm_minus_commute ∗)
  qed
qed

lemma indefinite_integral_continuous_1:
  fixes f :: real ⇒ 'a::banach
  assumes f integrable_on {a..b}
  shows continuous_on {a..b} (λx. integral {a..x} f)
proof −
  have ∃ d>0. ∀ x'∈{a..b}. dist x' x < d ⟶ dist (integral {a..x'} f) (integral
{a..x} f) < e
     if x: x ∈ {a..b} and e > 0 for x e :: real
  proof (cases a = b)
   case True
   with that show ?thesis by force
  next
   case False
   with x have a < b by force
   with x consider x = a | x = b | a < x x < b
    by force
   then show ?thesis
   proof cases
    case 1 then show ?thesis
    by (force simp: dist_norm algebra_simps intro: indefinite_integral_continuous_right
[OF assms _ ‹a < b› ‹e > 0›])
   next
    case 2 then show ?thesis
      by (force simp: dist_norm norm_minus_commute algebra_simps intro:
indefinite_integral_continuous_left [OF assms ‹a < b› _ ‹e > 0›])
   next
    case 3
    obtain d1 where 0 < d1
      and d1: ⋀t. ⟦x − d1 < t; t ≤ x⟧ ⟹ norm (integral {a..x} f − integral
{a..t} f) < e
      using 3 by (auto intro: indefinite_integral_continuous_left [OF assms ‹a
< x› _ ‹e > 0›])
    obtain d2 where 0 < d2
      and d2: ⋀t. ⟦x ≤ t; t < x + d2⟧ ⟹ norm (integral {a..x} f − integral
{a..t} f) < e
      using 3 by (auto intro: indefinite_integral_continuous_right [OF assms _
‹x < b› ‹e > 0›])
    show ?thesis
    proof (intro exI ballI conjI impI)
     show 0 < min d1 d2
      using ‹0 < d1› ‹0 < d2› by simp
```

```
      show dist (integral {a..y} f) (integral {a..x} f) < e
        if dist y x < min d1 d2 for y
          by (smt (verit) d1 d2 dist_norm dist_real_def norm_minus_commute
that)
    qed
  qed
 qed
 then show ?thesis
  by (auto simp: continuous_on_iff)
qed

lemma indefinite_integral_continuous_1':
 fixes f::real ⇒ 'a::banach
 assumes f integrable_on {a..b}
 shows continuous_on {a..b} (λx. integral {x..b} f)
proof −
 have integral {a..b} f − integral {a..x} f = integral {x..b} f if x ∈ {a..b} for x
  using integral_combine[OF _ _ assms, of x] that
  by (auto simp: algebra_simps)
 with _ show ?thesis
  by (rule continuous_on_eq) (auto intro!: continuous_intros indefinite_integral_continuous_1
assms)
qed

theorem integral_has_vector_derivative':
 fixes f :: real ⇒ 'b::banach
 assumes continuous_on {a..b} f
  and x ∈ {a..b}
 shows ((λu. integral {u..b} f) has_vector_derivative − f x) (at x within {a..b})
proof −
 have ∗: integral {x..b} f = integral {a .. b} f − integral {a .. x} f if a ≤ x x ≤
b for x
  using integral_combine[of a x b for x, OF that integrable_continuous_real[OF
assms(1)]]
  by (simp add: algebra_simps)
 show ?thesis
  using ‹x ∈ _› ∗
  by (rule has_vector_derivative_transform)
   (auto intro!: derivative_eq_intros assms integral_has_vector_derivative)
qed

lemma integral_has_real_derivative':
 assumes continuous_on {a..b} g
 assumes t ∈ {a..b}
 shows ((λx. integral {x..b} g) has_real_derivative −g t) (at t within {a..b})
 using integral_has_vector_derivative'[OF assms]
 by (auto simp: has_real_derivative_iff_has_vector_derivative)
```

6.14.31 This doesn't directly involve integration, but that gives an easy proof

lemma *has_derivative_zero_unique_strong_interval*:
fixes f :: *real* ⇒ *'a*::*banach*
assumes *finite k*
and *contf*: *continuous_on* {*a*..*b*} *f*
and $f\ a = y$
and *fder*: $\bigwedge x.\ x \in \{a..b\} - k \Longrightarrow (f\ has\_derivative\ (\lambda h.\ 0))\ (at\ x\ within\ \{a..b\})$
and x: $x \in \{a..b\}$
shows $f\ x = y$
proof −
have $a \leq b\ a \leq x$
using *assms* **by** *auto*
have $((\lambda x.\ 0::'a)\ has\_integral\ f\ x - f\ a)\ \{a..x\}$
proof (*rule fundamental_theorem_of_calculus_interior_strong*[*OF* ‹*finite k*› ‹*a* ≤ *x*›]; *clarify?*)
have {*a*..*x*} ⊆ {*a*..*b*}
using *x* **by** *auto*
then show *continuous_on* {*a*..*x*} *f*
by (*rule continuous_on_subset*[*OF contf*])
show (*f has_vector_derivative 0*) (*at z*) **if** *z*: $z \in \{a<..<x\}$ **and** *notin*: $z \notin k$ **for** *z*
unfolding *has_vector_derivative_def*
proof (*simp add*: *at_within_open*[*OF z*, *symmetric*])
show (*f has_derivative* (λ*x*. *0*)) (*at z within* {*a*<..<*x*})
by (*rule has_derivative_subset* [*OF fder*]) (*use x z notin* **in** *auto*)
qed
qed
from *has_integral_unique*[*OF has_integral_0 this*]
show *?thesis*
unfolding *assms* **by** *auto*
qed

6.14.32 Generalize a bit to any convex set

lemma *has_derivative_zero_unique_strong_convex*:
fixes f :: *'a*::*euclidean_space* ⇒ *'b*::*banach*
assumes *convex S finite K*
and *contf*: *continuous_on S f*
and $c \in S\ f\ c = y$
and *derf*: $\bigwedge x.\ x \in S - K \Longrightarrow (f\ has\_derivative\ (\lambda h.\ 0))\ (at\ x\ within\ S)$
and $x \in S$
shows $f\ x = y$
proof (*cases* $x = c$)
case *True* **with** ‹$f\ c = y$› **show** *?thesis*
by *blast*
next
case *False*
let *?φ* = $\lambda u.\ (1 - u) *_R c + u *_R x$

```
  have contf': continuous_on {0 ..1} (f ∘ ?φ)
  proof (rule continuous_intros continuous_on_subset[OF contf])+
    show (λu. (1 − u) *R c + u *R x) ‘ {0..1} ⊆ S
      using ‹convex S› ‹x ∈ S› ‹c ∈ S› by (auto simp add: convex_alt alge-
bra_simps)
  qed
  have t = u if ?φ t = ?φ u for t u
  proof −
    from that have (t − u) *R x = (t − u) *R c
      by (auto simp add: algebra_simps)
    then show ?thesis
      using ‹x ≠ c› by auto
  qed
  then have eq: (SOME t. ?φ t = ?φ u) = u for u
    by blast
  then have (?φ −‘ K) ⊆ (λz. SOME t. ?φ t = z) ‘ K
    by (clarsimp simp: image_iff) (metis (no_types) eq)
  then have fin: finite (?φ −‘ K)
    by (rule finite_surj[OF ‹finite K›])

  have derf': ((λu. f (?φ u)) has_derivative (λh. 0)) (at t within {0..1})
              if t ∈ {0..1} − {t. ?φ t ∈ K} for t
  proof −
    have df: (f has_derivative (λh. 0)) (at (?φ t) within ?φ ‘ {0..1})
      using ‹convex S› ‹x ∈ S› ‹c ∈ S› that
      by (auto simp add: convex_alt algebra_simps intro: has_derivative_subset
[OF derf])
    have (f ∘ ?φ has_derivative (λx. 0) ∘ (λz. (0 − z *R c) + z *R x)) (at t within
{0..1})
      by (rule derivative_eq_intros df | simp)+
    then show ?thesis
      unfolding o_def .
  qed
  have (f ∘ ?φ) 1 = y
    apply (rule has_derivative_zero_unique_strong_interval[OF fin contf'])
    unfolding o_def using ‹f c = y› derf' by auto
  then show ?thesis
    by auto
qed
```

Also to any open connected set with finite set of exceptions. Could generalize to locally convex set with limpt-free set of exceptions.

```
lemma has_derivative_zero_unique_strong_connected:
  fixes f :: 'a::euclidean_space ⇒ 'b::banach
  assumes connected S
    and open S
    and finite K
    and contf: continuous_on S f
    and c ∈ S
```

```
    and f c = y
    and derf: ⋀x. x ∈ S − K ⟹ (f has_derivative (λh. 0)) (at x within S)
    and x ∈ S
  shows f x = y
proof −
  have ∃ e>0. ball x e ⊆ (S ∩ f −‘ {f x}) if x ∈ S for x
  proof −
    obtain e where 0 < e and e: ball x e ⊆ S
      using ‹x ∈ S› ‹open S› open_contains_ball by blast
    have ball x e ⊆ {u ∈ S. f u ∈ {f x}}
    proof safe
      fix y
      assume y: y ∈ ball x e
      then show y ∈ S
        using e by auto
      show f y = f x
      proof (rule has_derivative_zero_unique_strong_convex[OF convex_ball ‹finite K›])
        show continuous_on (ball x e) f
          using contf continuous_on_subset e by blast
        show (f has_derivative (λh. 0)) (at u within ball x e)
            if u ∈ ball x e − K for u
          by (metis Diff_iff contra_subsetD derf e has_derivative_subset that)
      qed (use y e ‹0 < e› in auto)
    qed
    then show ∃ e>0. ball x e ⊆ (S ∩ f −‘ {f x})
      using ‹0 < e› by blast
  qed
  then have openin (top_of_set S) (S ∩ f −‘ {y})
    by (auto intro!: open_openin_trans[OF ‹open S›] simp: open_contains_ball)
  moreover have closedin (top_of_set S) (S ∩ f −‘ {y})
    by (force intro!: continuous_closedin_preimage [OF contf])
  ultimately have (S ∩ f −‘ {y}) = {} ∨ (S ∩ f −‘ {y}) = S
    using ‹connected S› by (simp add: connected_clopen)
  then show ?thesis
    using ‹x ∈ S› ‹f c = y› ‹c ∈ S› by auto
qed

lemma has_derivative_zero_connected_constant_on:
  fixes f :: 'a::euclidean_space ⇒ 'b::banach
  assumes connected S open S finite K continuous_on S f
      and ∀ x∈S−K. (f has_derivative (λh. 0)) (at x within S)
  shows   f constant_on S
  by (smt (verit, best) assms constant_on_def has_derivative_zero_unique_strong_connected)

lemma DERIV_zero_connected_constant_on:
  fixes f :: 'a::{real_normed_field,euclidean_space} ⇒ 'a
  assumes ∗: connected S open S finite K continuous_on S f
      and 0: ∀ x∈S−K. DERIV f x :> 0
```

shows *f constant_on S*
using *has_derivative_zero_connected_constant_on* [*OF* ∗] *0*
by (*metis has_derivative_at_withinI has_field_derivative_def lambda_zero*)

lemma *DERIV_zero_connected_constant*:
fixes *f* :: *'a*::{*real_normed_field*,*euclidean_space*} ⇒ *'a*
assumes *connected S* **and** *open S* **and** *finite K* **and** *continuous_on S f*
and ∀ *x*∈*S*−*K*. *DERIV f x* :> *0*
obtains *c* **where** ⋀*x*. *x* ∈ *S* ⟹ *f*(*x*) = *c*
by (*metis DERIV_zero_connected_constant_on* [*OF assms*] *constant_on_def*)

lemma *has_field_derivative_0_imp_constant_on*:
fixes *f* :: *'a*::{*real_normed_field*,*euclidean_space*} ⇒ *'a*
assumes ⋀*z*. *z* ∈ *S* ⟹ (*f has_field_derivative 0*) (*at z*) **and** *S*: *connected S open S*
shows *f constant_on S*
using *DERIV_zero_connected_constant_on* [**where** *K*=*Basis*]
by (*metis DERIV_isCont Diff_iff assms continuous_at_imp_continuous_on eucl.finite_Basis*)

6.14.33 Integrating characteristic function of an interval

lemma *has_integral_restrict_open_subinterval*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*banach*
assumes *intf*: (*f has_integral i*) (*cbox c d*)
and *cb*: *cbox c d* ⊆ *cbox a b*
shows ((λ*x*. *if x* ∈ *box c d then f x else 0*) *has_integral i*) (*cbox a b*)
proof (*cases cbox c d* = {})
case *True*
then have *box c d* = {}
by (*metis bot.extremum_uniqueI box_subset_cbox*)
then show *?thesis*
using *True intf* **by** *auto*
next
case *False*
then obtain *p* **where** *pdiv*: *p division_of cbox a b* **and** *inp*: *cbox c d* ∈ *p*
using *cb partial_division_extend_1* **by** *blast*
define *g* **where** [*abs_def*]: *g x* = (*if x* ∈*box c d then f x else 0*) **for** *x*
interpret *operative lift_option plus Some* (*0* :: *'b*)
λ*i*. *if g integrable_on i then Some* (*integral i g*) *else None*
by (*fact operative_integralI*)
note *operat* = *division* [*OF pdiv*, *symmetric*]
show *?thesis*
proof (*cases* (*g has_integral i*) (*cbox a b*))
case *True* **then show** *?thesis*
by (*simp add*: *g_def*)
next
case *False*
have *iterate*:*F* (λ*i*. *if g integrable_on i then Some* (*integral i g*) *else None*) (*p*

```
− {cbox c d}) = Some 0
    proof (intro neutral ballI)
      fix x
      assume x: x ∈ p − {cbox c d}
      then have x ∈ p
        by auto
      then obtain u v where uv: x = cbox u v
        using pdiv by blast
      have interior x ∩ interior (cbox c d) = {}
        using pdiv inp x by blast
      then have (g has_integral 0) x
        unfolding uv using has_integral_spike_interior[where f=λx. 0]
      by (metis (no_types, lifting) disjoint_iff_not_equal g_def has_integral_0_eq
interior_cbox)
     then show (if g integrable_on x then Some (integral x g) else None) = Some
0
       by auto
    qed
    interpret comm_monoid_set lift_option plus Some (0 :: 'b)
    by (intro comm_monoid_set.intro comm_monoid_lift_option add.comm_monoid_axioms)
    have intg: g integrable_on cbox c d
      using integrable_spike_interior[where f=f]
      by (meson g_def has_integral_integrable intf)
    moreover have integral (cbox c d) g = i
      by (meson g_def has_integral_iff has_integral_spike_interior intf)
    ultimately have F (λA. if g integrable_on A then Some (integral A g) else
None) p = Some i
       by (metis (full_types, lifting) division_of_finite inp iterate pdiv remove
right_neutral)
    with False show ?thesis
      by (metis integrable_integral not_None_eq operat option.inject)
  qed
qed


lemma has_integral_restrict_closed_subinterval:
  fixes f :: 'a::euclidean_space ⇒ 'b::banach
  assumes (f has_integral i) (cbox c d)
    and cbox c d ⊆ cbox a b
  shows ((λx. if x ∈ cbox c d then f x else 0) has_integral i) (cbox a b)
proof −
  note has_integral_restrict_open_subinterval[OF assms]
  note ∗ = has_integral_spike[OF negligible_frontier_interval _ this]
  show ?thesis
    by (rule ∗[of c d]) (use box_subset_cbox[of c d] in auto)
qed

lemma has_integral_restrict_closed_subintervals_eq:
  fixes f :: 'a::euclidean_space ⇒ 'b::banach
```

assumes *cbox c d* ⊆ *cbox a b*
shows ((λ*x*. *if x* ∈ *cbox c d then f x else 0*) *has_integral i*) (*cbox a b*) ⟷ (*f has_integral i*) (*cbox c d*)
(**is** *?l* = *?r*)
proof (*cases cbox c d* = {})
case *False*
let *?g* = λ*x*. *if x* ∈ *cbox c d then f x else 0*
show *?thesis*
proof
assume *?l*
then have *?g integrable_on cbox c d*
using *assms has_integral_integrable integrable_subinterval* **by** *blast*
then have *f integrable_on cbox c d*
by (*rule integrable_eq*) *auto*
moreover then have *i* = *integral* (*cbox c d*) *f*
by (*meson* ‹((λ*x*. *if x* ∈ *cbox c d then f x else 0*) *has_integral i*) (*cbox a b*)› *assms has_integral_restrict_closed_subinterval has_integral_unique integrable_integral*)
ultimately show *?r* **by** *auto*
next
assume *?r* **then show** *?l*
by (*rule has_integral_restrict_closed_subinterval*[*OF* _ *assms*])
qed
qed *auto*

Hence we can apply the limit process uniformly to all integrals.

lemma *has_integral'*:
fixes *f* :: *'n*::*euclidean_space* ⇒ *'a*::*banach*
shows (*f has_integral i*) *S* ⟷
(∀ *e*>0. ∃ *B*>0. ∀ *a b*. *ball 0 B* ⊆ *cbox a b* ⟶
(∃ *z*. ((λ*x*. *if x* ∈ *S then f*(*x*) *else 0*) *has_integral z*) (*cbox a b*) ∧ *norm*(*z* − *i*) < *e*))
(**is** *?l* ⟷ (∀ *e*>0. *?r e*))
proof (*cases* ∃ *a b*. *S* = *cbox a b*)
case *False* **then show** *?thesis*
by (*simp add*: *has_integral_alt*)
next
case *True*
then obtain *a b* **where** *S*: *S* = *cbox a b*
by *blast*
obtain *B* **where** 0 < *B* **and** *B*: ⋀*x*. *x* ∈ *cbox a b* ⟹ *norm x* ≤ *B*
using *bounded_cbox*[*unfolded bounded_pos*] **by** *blast*
show *?thesis*
proof *safe*
fix *e* :: *real*
assume *?l* **and** *e* > 0
have ((λ*x*. *if x* ∈ *S then f x else 0*) *has_integral i*) (*cbox c d*)
if *ball 0* (*B*+1) ⊆ *cbox c d* **for** *c d*
unfolding *S* **using** *B that*

```
      by (force intro: ‹?l›[unfolded S] has_integral_restrict_closed_subinterval)
    then show ?r e
      by (meson ‹0 < B› ‹0 < e› add_pos_pos le_less_trans zero_less_one
norm_pths(2))
  next
    assume as: ∀ e>0. ?r e
    then obtain C
      where C: ⋀a b. ball 0 C ⊆ cbox a b ⟹
            ∃ z. ((λx. if x ∈ S then f x else 0) has_integral z) (cbox a b)
      by (meson zero_less_one)
    define c :: 'n where c = (∑ i∈Basis. (− max B C) *R i)
    define d :: 'n where d = (∑ i∈Basis. max B C *R i)
    have c · i ≤ x · i ∧ x · i ≤ d · i if norm x ≤ B i ∈ Basis for x i
      using that and Basis_le_norm[OF ‹i∈Basis›, of x]
      by (auto simp add: field_simps sum_negf c_def d_def)
    then have c_d: cbox a b ⊆ cbox c d
      by (meson B mem_box(2) subsetI)
    have c · i ≤ x · i ∧ x · i ≤ d · i
      if x: norm (0 − x) < C and i: i ∈ Basis for x i
      using Basis_le_norm[OF i, of x] x i by (auto simp: sum_negf c_def d_def)
      then have ball 0 C ⊆ cbox c d
        by (auto simp: mem_box dist_norm)
    with C obtain y where y: (f has_integral y) (cbox a b)
      using c_d has_integral_restrict_closed_subintervals_eq S by blast
    have y = i
    proof (rule ccontr)
      assume y ≠ i
      then have 0 < norm (y − i)
        by auto
      from as[rule_format,OF this]
      obtain C where C: ⋀a b. ball 0 C ⊆ cbox a b ⟹
          ∃ z. ((λx. if x ∈ S then f x else 0) has_integral z) (cbox a b) ∧ norm (z−i)
< norm (y−i)
        by auto
      define c :: 'n where c = (∑ i∈Basis. (− max B C) *R i)
      define d :: 'n where d = (∑ i∈Basis. max B C *R i)
      have c · i ≤ x · i ∧ x · i ≤ d · i
        if norm x ≤ B and i ∈ Basis for x i
        using that Basis_le_norm[of i x] by (auto simp add: field_simps sum_negf
c_def d_def)
        then have c_d: cbox a b ⊆ cbox c d
        by (simp add: B mem_box(2) subset_eq)
      have c · i ≤ x · i ∧ x · i ≤ d · i if norm (0 − x) < C and i ∈ Basis for x i
        using Basis_le_norm[of i x] that by (auto simp: sum_negf c_def d_def)
      then have ball 0 C ⊆ cbox c d
        by (auto simp: mem_box dist_norm)
      with C obtain z where z: (f has_integral z) (cbox a b) norm (z−i) < norm
(y−i)
        using has_integral_restrict_closed_subintervals_eq[OF c_d] S by blast
```

```
    moreover then have z = y
      by (blast intro: has_integral_unique[OF _ y])
    ultimately show False
      by auto
  qed
  then show ?l
    using y by (auto simp: S)
 qed
qed

lemma has_integral_le:
 fixes f :: 'n::euclidean_space ⇒ real
 assumes fg: (f has_integral i) S (g has_integral j) S
   and le: ⋀x. x ∈ S ⟹ f x ≤ g x
 shows i ≤ j
 using has_integral_component_le[OF _ fg, of 1] le by auto

lemma integral_le:
 fixes f :: 'n::euclidean_space ⇒ real
 assumes f integrable_on S
   and g integrable_on S
   and ⋀x. x ∈ S ⟹ f x ≤ g x
 shows integral S f ≤ integral S g
 by (meson assms has_integral_le integrable_integral)

lemma has_integral_nonneg:
 fixes f :: 'n::euclidean_space ⇒ real
 assumes (f has_integral i) S and ⋀x. x ∈ S ⟹ 0 ≤ f x
 shows 0 ≤ i
 using assms has_integral_0 has_integral_le by blast

lemma integral_nonneg:
 fixes f :: 'n::euclidean_space ⇒ real
 assumes f: f integrable_on S and 0: ⋀x. x ∈ S ⟹ 0 ≤ f x
 shows 0 ≤ integral S f
 by (rule has_integral_nonneg[OF f[unfolded has_integral_integral] 0])
```

Hence a general restriction property.

```
lemma has_integral_restrict [simp]:
 fixes f :: 'a :: euclidean_space ⇒ 'b :: banach
 assumes S ⊆ T
 shows ((λx. if x ∈ S then f x else 0) has_integral i) T ⟷ (f has_integral i) S
proof −
 have ∗: ⋀x. (if x ∈ T then if x ∈ S then f x else 0 else 0) = (if x∈S then f x
else 0)
   using assms by auto
 show ?thesis
   apply (subst(2) has_integral')
   apply (subst has_integral')
```

```
    apply (simp add: *)
   done
qed

corollary has_integral_restrict_UNIV:
  fixes f :: 'n::euclidean_space ⇒ 'a::banach
  shows ((λx. if x ∈ s then f x else 0) has_integral i) UNIV ⟷ (f has_integral
i) s
  by auto

lemma has_integral_restrict_Int:
  fixes f :: 'a :: euclidean_space ⇒ 'b :: banach
  shows ((λx. if x ∈ S then f x else 0) has_integral i) T ⟷ (f has_integral i) (S
∩ T)
proof −
  have ((λx. if x ∈ T then if x ∈ S then f x else 0 else 0) has_integral i) UNIV =
        ((λx. if x ∈ S ∩ T then f x else 0) has_integral i) UNIV
    by (rule has_integral_cong) auto
  then show ?thesis
    using has_integral_restrict_UNIV by fastforce
qed

lemma integral_restrict_Int:
  fixes f :: 'a :: euclidean_space ⇒ 'b :: banach
  shows integral T (λx. if x ∈ S then f x else 0) = integral (S ∩ T) f
  by (metis (no_types, lifting) has_integral_cong has_integral_restrict_Int inte-
grable_integral integral_unique not_integrable_integral)

lemma integrable_restrict_Int:
  fixes f :: 'a :: euclidean_space ⇒ 'b :: banach
  shows (λx. if x ∈ S then f x else 0) integrable_on T ⟷ f integrable_on (S ∩
T)
  using has_integral_restrict_Int by fastforce

lemma has_integral_on_superset:
  fixes f :: 'n::euclidean_space ⇒ 'a::banach
  assumes (f has_integral i) S
      and ⋀x. x ∉ S ⟹ f x = 0
      and S ⊆ T
    shows (f has_integral i) T
  by (smt (verit, ccfv_SIG) assms has_integral_cong has_integral_restrict)

lemma integrable_on_superset:
  fixes f :: 'n::euclidean_space ⇒ 'a::banach
  assumes f integrable_on S and ⋀x. x ∉ S ⟹ f x = 0 and S ⊆ t
  shows f integrable_on t
 by (meson assms has_integral_on_superset integrable_integral integrable_on_def)

lemma integral_subset_negligible:
```

fixes $f :: 'a :: euclidean\_space \Rightarrow 'b :: banach$
assumes $S \subseteq T$ *negligible* $(T - S)$
shows *integral* $S\ f =$ *integral* $T\ f$
proof −
have *integral* $T\ f =$ *integral* $T\ (\lambda x.\ if\ x \in S\ then\ f\ x\ else\ 0)$
by (*rule integral_spike*[*of* $T - S$]) (*use assms* **in** *auto*)
also have $\ldots =$ *integral* $(S \cap T)\ f$
by (*subst integral_restrict_Int*) *auto*
also have $S \cap T = S$ **using** *assms* **by** *auto*
finally show *?thesis* **..**
qed

lemma *integral_restrict_UNIV*:
fixes $f :: 'n::euclidean\_space \Rightarrow 'a::banach$
shows *integral UNIV* $(\lambda x.\ if\ x \in S\ then\ f\ x\ else\ 0) =$ *integral* $S\ f$
by (*simp add*: *integral_restrict_Int*)

lemma *integrable_restrict_UNIV*:
fixes $f :: 'n::euclidean\_space \Rightarrow 'a::banach$
shows $(\lambda x.\ if\ x \in s\ then\ f\ x\ else\ 0)$ *integrable_on UNIV* $\longleftrightarrow$ f *integrable_on* s
unfolding *integrable_on_def*
by *auto*

lemma *has_integral_subset_component_le*:
fixes $f :: 'n::euclidean\_space \Rightarrow 'm::euclidean\_space$
assumes k: $k \in Basis$
and $S \subseteq T$ $(f$ *has_integral* $i)\ S$ $(f$ *has_integral* $j)\ T$ $\bigwedge x.\ x \in T \Longrightarrow 0 \le f(x) \cdot k$
shows $i \cdot k \le j \cdot k$
proof −
have §: $((\lambda x.\ if\ x \in S\ then\ f\ x\ else\ 0)$ *has_integral* $i)$ *UNIV*
$((\lambda x.\ if\ x \in T\ then\ f\ x\ else\ 0)$ *has_integral* $j)$ *UNIV*
by (*simp_all add*: *assms*)
show *?thesis*
using *assms* **by** (*force intro*!: *has_integral_component_le*[*OF* k §])
qed

6.14.34 Integrals on set differences

lemma *has_integral_setdiff*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::banach$
assumes S: $(f$ *has_integral* $i)\ S$ **and** T: $(f$ *has_integral* $j)\ T$
and *neg*: *negligible* $(T - S)$
shows $(f$ *has_integral* $(i - j))\ (S - T)$
proof −
show *?thesis*
unfolding *has_integral_restrict_UNIV* [*symmetric*, *of* f]
proof (*rule has_integral_spike* [*OF neg*])
have *eq*: $(\lambda x.\ (if\ x \in S\ then\ f\ x\ else\ 0) - (if\ x \in T\ then\ f\ x\ else\ 0)) =$
$(\lambda x.\ if\ x \in T - S\ then - f\ x\ else\ if\ x \in S - T\ then\ f\ x\ else\ 0)$

```
    by (force simp add: )
  have ((λx. if x ∈ S then f x else 0) has_integral i) UNIV
    ((λx. if x ∈ T then f x else 0) has_integral j) UNIV
    using S T has_integral_restrict_UNIV by auto
  from has_integral_diff [OF this]
  show ((λx. if x ∈ T − S then − f x else if x ∈ S − T then f x else 0)
            has_integral i−j) UNIV
    by (simp add: eq)
 qed force
qed

lemma integral_setdiff:
 fixes f :: 'a::euclidean_space ⇒ 'b::banach
 assumes f integrable_on S f integrable_on T negligible(T − S)
 shows integral (S − T) f = integral S f − integral T f
 by (rule integral_unique) (simp add: assms has_integral_setdiff integrable_integral)

lemma integrable_setdiff:
 fixes f :: 'a::euclidean_space ⇒ 'b::banach
 assumes (f has_integral i) S (f has_integral j) T negligible (T − S)
 shows f integrable_on (S − T)
 using has_integral_setdiff [OF assms]
 by (simp add: has_integral_iff)

lemma negligible_setdiff [simp]: T ⊆ S ⟹ negligible (T − S)
 by (metis Diff_eq_empty_iff negligible_empty)

lemma negligible_on_intervals: negligible s ⟷ (∀ a b. negligible(s ∩ cbox a b))
(is ?l ⟷ ?r)
proof
 assume R: ?r
 show ?l
   unfolding negligible_def
   by (metis Diff_iff Int_iff R has_integral_negligible indicator_simps(2))
qed (simp add: negligible_Int)

lemma negligible_translation:
 assumes negligible S
   shows negligible ((+) c ` S)
proof −
 have inj: inj ((+) c)
   by simp
 show ?thesis
 using assms
 proof (clarsimp simp: negligible_def)
   fix a b
   assume ∀ x y. (indicator S has_integral 0) (cbox x y)
   then have ∗: (indicator S has_integral 0) (cbox (a−c) (b−c))
     by (meson Diff_iff assms has_integral_negligible indicator_simps(2))
```

```
    have eq: indicator ((+) c ‘ S) = (λx. indicator S (x − c))
      by (force simp add: indicator_def)
    show (indicator ((+) c ‘ S) has_integral 0) (cbox a b)
      using has_integral_affinity [OF ∗, of 1 −c]
            cbox_translation [of c −c+a −c+b]
      by (simp add: eq) (simp add: ac_simps)
  qed
qed

lemma negligible_translation_rev:
  assumes negligible ((+) c ‘ S)
  shows negligible S
  by (metis negligible_translation [OF assms, of −c] translation_galois)

lemma negligible_atLeastAtMostI: b ≤ a ⟹ negligible {a..(b::real)}
  using negligible_insert by fastforce

lemma has_integral_spike_set_eq:
  fixes f :: 'n::euclidean_space ⇒ 'a::banach
  assumes negligible {x ∈ S − T. f x ≠ 0} negligible {x ∈ T − S. f x ≠ 0}
  shows (f has_integral y) S ⟷ (f has_integral y) T
proof −
  have ((λx. if x ∈ S then f x else 0) has_integral y) UNIV =
        ((λx. if x ∈ T then f x else 0) has_integral y) UNIV
  proof (rule has_integral_spike_eq)
    show negligible ({x ∈ S − T. f x ≠ 0} ∪ {x ∈ T − S. f x ≠ 0})
      by (rule negligible_Un [OF assms])
  qed auto
  then show ?thesis
    by (simp add: has_integral_restrict_UNIV)
qed

corollary integral_spike_set:
  fixes f :: 'n::euclidean_space ⇒ 'a::banach
  assumes negligible {x ∈ S − T. f x ≠ 0} negligible {x ∈ T − S. f x ≠ 0}
  shows integral S f = integral T f
  using has_integral_spike_set_eq [OF assms]
  by (metis eq_integralD integral_unique)

lemma integrable_spike_set:
  fixes f :: 'n::euclidean_space ⇒ 'a::banach
  assumes f: f integrable_on S and neg: negligible {x ∈ S − T. f x ≠ 0} negligible
{x ∈ T − S. f x ≠ 0}
  shows f integrable_on T
  using has_integral_spike_set_eq [OF neg] f by blast

lemma integrable_spike_set_eq:
  fixes f :: 'n::euclidean_space ⇒ 'a::banach
  assumes negligible ((S − T) ∪ (T − S))
```

shows *f integrable_on S* $\longleftrightarrow$ *f integrable_on T*
by (*blast intro*: *integrable_spike_set assms negligible_subset*)

lemma *integrable_on_insert_iff*: *f integrable_on* (*insert x X*) $\longleftrightarrow$ *f integrable_on X*
for *f*::_ $\Rightarrow$ *'a*::*banach*
by (*rule integrable_spike_set_eq*) (*auto simp*: *insert_Diff_if*)

lemma *has_integral_interior*:
fixes *f* :: *'a* :: *euclidean_space* $\Rightarrow$ *'b* :: *banach*
shows *negligible*(*frontier S*) $\Longrightarrow$ (*f has_integral y*) (*interior S*) $\longleftrightarrow$ (*f has_integral y*) *S*
by (*rule has_integral_spike_set_eq* [*OF empty_imp_negligible negligible_subset*])
(*use interior_subset* **in** ‹*auto simp*: *frontier_def closure_def*›)

lemma *has_integral_closure*:
fixes *f* :: *'a* :: *euclidean_space* $\Rightarrow$ *'b* :: *banach*
shows *negligible*(*frontier S*) $\Longrightarrow$ (*f has_integral y*) (*closure S*) $\longleftrightarrow$ (*f has_integral y*) *S*
by (*rule has_integral_spike_set_eq* [*OF negligible_subset empty_imp_negligible*])
(*auto simp*: *closure_Un_frontier*)

lemma *has_integral_open_interval*:
fixes *f* :: *'a* :: *euclidean_space* $\Rightarrow$ *'b* :: *banach*
shows (*f has_integral y*) (*box a b*) $\longleftrightarrow$ (*f has_integral y*) (*cbox a b*)
unfolding *interior_cbox* [*symmetric*]
by (*metis frontier_cbox has_integral_interior negligible_frontier_interval*)

lemma *integrable_on_open_interval*:
fixes *f* :: *'a* :: *euclidean_space* $\Rightarrow$ *'b* :: *banach*
shows *f integrable_on box a b* $\longleftrightarrow$ *f integrable_on cbox a b*
by (*simp add*: *has_integral_open_interval integrable_on_def*)

lemma *integral_open_interval*:
fixes *f* :: *'a* :: *euclidean_space* $\Rightarrow$ *'b* :: *banach*
shows *integral*(*box a b*) *f* = *integral*(*cbox a b*) *f*
by (*metis has_integral_integrable_integral has_integral_open_interval not_integrable_integral*)

lemma *integrable_on_open_interval_real*:
fixes *f* :: *real* $\Rightarrow$ *'b* :: *banach*
shows *f integrable_on* {*a*<..<*b*} $\longleftrightarrow$ *f integrable_on* {*a*..*b*}
using *integrable_on_open_interval*[*of f a b*] **by** *simp*

lemma *integral_open_interval_real*:
integral {*a*..*b*} (*f* :: *real* $\Rightarrow$ *'a* :: *banach*) = *integral* {*a*<..<(*b*::*real*)} *f*
using *integral_open_interval*[*of a b f*] **by** *simp*

lemma *has_integral_Icc_iff_Ioo*:
fixes *f* :: *real* $\Rightarrow$ *'a* :: *banach*

shows (*f has_integral I*) {*a*..*b*} ⟷ (*f has_integral I*) {*a*<..<*b*}
by (*metis box_real*(1) *cbox_interval has_integral_open_interval*)

lemma *integrable_on_Icc_iff_Ioo*:
fixes *f* :: *real* ⇒ *'a* :: *banach*
shows *f integrable_on* {*a*..*b*} ⟷ *f integrable_on* {*a*<..<*b*}
using *has_integral_Icc_iff_Ioo* **by** *blast*

6.14.35 More lemmas that are useful later

lemma *has_integral_subset_le*:
fixes *f* :: *'n*::*euclidean_space* ⇒ *real*
assumes *s* ⊆ *t*
and (*f has_integral i*) *s*
and (*f has_integral j*) *t*
and ∀ *x*∈*t*. *0* ≤ *f x*
shows *i* ≤ *j*
using *assms has_integral_subset_component_le*[*OF _ assms*(1), *of 1 f i j*]
by *auto*

lemma *integral_subset_component_le*:
fixes *f* :: *'n*::*euclidean_space* ⇒ *'m*::*euclidean_space*
assumes *k* ∈ *Basis*
and *s* ⊆ *t*
and *f integrable_on s*
and *f integrable_on t*
and ∀ *x* ∈ *t*. *0* ≤ *f x* · *k*
shows (*integral s f*)·*k* ≤ (*integral t f*)·*k*
by (*meson assms has_integral_subset_component_le integrable_integral*)

lemma *integral_subset_le*:
fixes *f* :: *'n*::*euclidean_space* ⇒ *real*
assumes *s* ⊆ *t*
and *f integrable_on s*
and *f integrable_on t*
and ∀ *x* ∈ *t*. *0* ≤ *f x*
shows *integral s f* ≤ *integral t f*
using *assms has_integral_subset_le* **by** *blast*

lemma *has_integral_alt'*:
fixes *f* :: *'n*::*euclidean_space* ⇒ *'a*::*banach*
shows (*f has_integral i*) *s* ⟷
(∀ *a b*. (λ*x*. *if x* ∈ *s then f x else 0*) *integrable_on cbox a b*) ∧
(∀ *e*>*0*. ∃ *B*>*0*. ∀ *a b*. *ball 0 B* ⊆ *cbox a b* ⟶
norm (*integral* (*cbox a b*) (λ*x*. *if x* ∈ *s then f x else 0*) − *i*) < *e*)
(**is** *?l* = *?r*)
proof
assume *rhs*: *?r*
show *?l*

proof (*subst has_integral'*, *intro allI impI*)
fix *e*::*real*
assume *e* > *0*
from *rhs*[*THEN conjunct2*,*rule_format*,*OF this*]
show ∃ *B*>*0*. ∀ *a b*. *ball 0 B* ⊆ *cbox a b* ⟶
(∃ *z*. ((λ*x*. *if x* ∈ *s then f x else 0*) *has_integral z*)
(*cbox a b*) ∧ *norm* (*z* − *i*) < *e*)
by (*simp add*: *has_integral_iff rhs*)
qed
next
let *?*Φ = λ*e a b*. ∃ *z*. ((λ*x*. *if x* ∈ *s then f x else 0*) *has_integral z*) (*cbox a b*) ∧ *norm* (*z* − *i*) < *e*
assume *?l*
then have *lhs*: ∃ *B*>*0*. ∀ *a b*. *ball 0 B* ⊆ *cbox a b* ⟶ *?*Φ *e a b* **if** *e* > *0* **for** *e*
using *that has_integral'*[*of f*] **by** *auto*
let *?f* = λ*x*. *if x* ∈ *s then f x else 0*
show *?r*
proof (*intro conjI allI impI*)
fix *a b* :: *'n*
from *lhs*[*OF zero_less_one*]
obtain *B* **where** *0* < *B* **and** *B*: ⋀*a b*. *ball 0 B* ⊆ *cbox a b* ⟹ *?*Φ *1 a b*
by *blast*
let *?a* = ∑ *i*∈*Basis*. *min* (*a*•*i*) (−*B*) ∗*R* *i*::*'n*
let *?b* = ∑ *i*∈*Basis*. *max* (*b*•*i*) *B* ∗*R* *i*::*'n*
show *?f integrable_on cbox a b*
proof (*rule integrable_subinterval*[*of _ ?a ?b*])
have *?a* · *i* ≤ *x* · *i* ∧ *x* · *i* ≤ *?b* · *i* **if** *norm* (*0* − *x*) < *B i* ∈ *Basis* **for** *x i*
using *Basis_le_norm*[*of i x*] *that* **by** (*auto simp add*:*field_simps*)
then have *ball 0 B* ⊆ *cbox ?a ?b*
by (*auto simp*: *mem_box dist_norm*)
then show *?f integrable_on cbox ?a ?b*
unfolding *integrable_on_def* **using** *B* **by** *blast*
show *cbox a b* ⊆ *cbox ?a ?b*
by (*force simp*: *mem_box*)
qed
fix *e* :: *real*
assume *e* > *0*
with *lhs* **show** ∃ *B*>*0*. ∀ *a b*. *ball 0 B* ⊆ *cbox a b* ⟶
norm (*integral* (*cbox a b*) (λ*x*. *if x* ∈ *s then f x else 0*) − *i*) < *e*
by (*metis* (*no_types*, *lifting*) *has_integral_integrable_integral*)
qed
qed

6.14.36 Continuity of the integral (for a 1-dimensional interval)

lemma *integrable_alt*:
fixes *f* :: *'n*::*euclidean_space* ⇒ *'a*::*banach*
shows *f integrable_on s* ⟷

```
    (∀ a b. (λx. if x ∈ s then f x else 0) integrable_on cbox a b) ∧
    (∀ e>0. ∃ B>0. ∀ a b c d. ball 0 B ⊆ cbox a b ∧ ball 0 B ⊆ cbox c d ⟶
    norm (integral (cbox a b) (λx. if x ∈ s then f x else 0) −
      integral (cbox c d)  (λx. if x ∈ s then f x else 0)) < e)
  (is ?l = ?r)
proof
  let ?F = λx. if x ∈ s then f x else 0
  assume ?l
  then obtain y where intF: ⋀a b. ?F integrable_on cbox a b
        and y: ⋀e. 0 < e ⟹
           ∃ B>0. ∀ a b. ball 0 B ⊆ cbox a b ⟶ norm (integral (cbox a b) ?F −
y) < e
    unfolding integrable_on_def has_integral_alt'[of f] by auto
  show ?r
  proof (intro conjI allI impI intF)
    fix e::real
    assume e > 0
    then have e/2 > 0
      by auto
    obtain B where 0 < B
      and B: ⋀a b. ball 0 B ⊆ cbox a b ⟹ norm (integral (cbox a b) ?F − y) <
e/2
      using ‹0 < e/2› y by blast
    show ∃ B>0. ∀ a b c d. ball 0 B ⊆ cbox a b ∧ ball 0 B ⊆ cbox c d ⟶
              norm (integral (cbox a b) ?F − integral (cbox c d) ?F) < e
    proof (intro conjI exI impI allI, rule ‹0 < B›)
      fix a b c d::'n
      assume sub: ball 0 B ⊆ cbox a b ∧ ball 0 B ⊆ cbox c d
      show norm (integral (cbox a b) ?F − integral (cbox c d) ?F) < e
        using sub by (auto intro: norm_triangle_half_l dest: B)
    qed
  qed
next
  let ?F = λx. if x ∈ s then f x else 0
  assume rhs: ?r
  let ?cube = λn. cbox (∑ i∈Basis. − real n *R i::'n) (∑ i∈Basis. real n *R i)
  have Cauchy (λn. integral (?cube n) ?F)
    unfolding Cauchy_def
  proof (intro allI impI)
    fix e::real
    assume e > 0
    with rhs obtain B where 0 < B
      and B: ⋀a b c d. ball 0 B ⊆ cbox a b ∧ ball 0 B ⊆ cbox c d
                  ⟹ norm (integral (cbox a b) ?F − integral (cbox c d) ?F) < e
      by blast
    obtain N where N: B ≤ real N
      using real_arch_simple by blast
    have ball 0 B ⊆ ?cube n if n: n ≥ N for n
    proof −
```

```
    have sum ((*R) (- real n)) Basis · i ≤ x · i ∧
          x · i ≤ sum ((*R) (real n)) Basis · i
      if norm x < B i ∈ Basis for x i::'n
        using Basis_le_norm[of i x] n N that by (auto simp add: field_simps
sum_negf)
    then show ?thesis
      by (auto simp: mem_box dist_norm)
  qed
  then show ∃ M. ∀ m≥M. ∀ n≥M. dist (integral (?cube m) ?F) (integral (?cube
n) ?F) < e
    by (fastforce simp add: dist_norm intro!: B)
 qed
 then obtain i where i: (λn. integral (?cube n) ?F) ⟶ i
  using convergent_eq_Cauchy by blast
 have ∃ B>0. ∀ a b. ball 0 B ⊆ cbox a b ⟶ norm (integral (cbox a b) ?F − i)
< e
  if e > 0 for e
 proof −
  have ∗: e/2 > 0 using that by auto
  then obtain N where N: ⋀n. N ≤ n ⟹ norm (i − integral (?cube n) ?F)
< e/2
    using i[THEN LIMSEQ_D, simplified norm_minus_commute] by meson
  obtain B where 0 < B
    and B: ⋀a b c d. ⟦ball 0 B ⊆ cbox a b; ball 0 B ⊆ cbox c d⟧ ⟹
             norm (integral (cbox a b) ?F − integral (cbox c d) ?F) < e/2
    using rhs ∗ by meson
  let ?B = max (real N) B
  show ?thesis
  proof (intro exI conjI allI impI)
    show 0 < ?B
      using ‹B > 0› by auto
    fix a b :: 'n
    assume ball 0 ?B ⊆ cbox a b
    moreover obtain n where n: max (real N) B ≤ real n
      using real_arch_simple by blast
    moreover have ball 0 B ⊆ ?cube n
    proof
      fix x :: 'n
      assume x: x ∈ ball 0 B
      have ⟦norm (0 − x) < B; i ∈ Basis⟧
          ⟹ sum ((*R) (−n)) Basis · i≤ x · i ∧ x · i ≤ sum ((*R) n) Basis · i
for i
       using Basis_le_norm[of i x] n by (auto simp add: field_simps sum_negf)
      then show x ∈ ?cube n
       using x by (auto simp: mem_box dist_norm)
    qed
    ultimately show norm (integral (cbox a b) ?F − i) < e
      using norm_triangle_half_l [OF B N] by force
  qed
```

```
  qed
  then show ?l unfolding integrable_on_def has_integral_alt'[of f]
    using rhs by blast
qed

lemma integrable_altD:
  fixes f :: 'n::euclidean_space ⇒ 'a::banach
  assumes f integrable_on s
  shows ⋀a b. (λx. if x ∈ s then f x else 0) integrable_on cbox a b
    and ⋀e. e > 0 ⟹ ∃B>0. ∀a b c d. ball 0 B ⊆ cbox a b ∧ ball 0 B ⊆ cbox c
d ⟶
      norm (integral (cbox a b) (λx. if x ∈ s then f x else 0) − integral (cbox c d)
(λx. if x ∈ s then f x else 0)) < e
  using assms[unfolded integrable_alt[of f]] by auto

lemma integrable_alt_subset:
  fixes f :: 'a::euclidean_space ⇒ 'b::banach
  shows
     f integrable_on S ⟷
      (∀ a b. (λx. if x ∈ S then f x else 0) integrable_on cbox a b) ∧
      (∀ e>0. ∃ B>0. ∀ a b c d.
                    ball 0 B ⊆ cbox a b ∧ cbox a b ⊆ cbox c d
                    ⟶ norm(integral (cbox a b) (λx. if x ∈ S then f x else 0) −
                          integral (cbox c d) (λx. if x ∈ S then f x else 0)) < e)
      (is _ = ?rhs)
proof −
  let ?g = λx. if x ∈ S then f x else 0
  have f integrable_on S ⟷
       (∀ a b. ?g integrable_on cbox a b) ∧
       (∀ e>0. ∃ B>0. ∀ a b c d. ball 0 B ⊆ cbox a b ∧ ball 0 B ⊆ cbox c d ⟶
         norm (integral (cbox a b) ?g − integral (cbox c d)  ?g) < e)
    by (rule integrable_alt)
  also have ... = ?rhs
  proof −
    { fix e :: real
      assume e: ⋀e. e>0 ⟹ ∃B>0. ∀a b c d. ball 0 B ⊆ cbox a b ∧ cbox a b ⊆
cbox c d ⟶
                                  norm (integral (cbox a b) ?g − integral (cbox c d) ?g)
< e
       and e > 0
      obtain B where B > 0
        and B: ⋀a b c d. ⟦ball 0 B ⊆ cbox a b; cbox a b ⊆ cbox c d⟧ ⟹
                       norm (integral (cbox a b) ?g − integral (cbox c d) ?g) < e/2
        using ‹e > 0› e [of e/2] by force
      have ∃B>0. ∀a b c d.
             ball 0 B ⊆ cbox a b ∧ ball 0 B ⊆ cbox c d ⟶
             norm (integral (cbox a b) ?g − integral (cbox c d) ?g) < e
      proof (intro exI allI conjI impI)
        fix a b c d :: 'a
```

```
      let ?α = ∑ i∈Basis. max (a • i) (c • i) *R i
      let ?β = ∑ i∈Basis. min (b • i) (d • i) *R i
      show norm (integral (cbox a b) ?g − integral (cbox c d) ?g) < e
        if ball: ball 0 B ⊆ cbox a b ∧ ball 0 B ⊆ cbox c d
      proof −
       have B': norm (integral (cbox a b ∩ cbox c d) ?g − integral (cbox x y) ?g)
< e/2
          if cbox a b ∩ cbox c d ⊆ cbox x y for x y
         using B [of ?α ?β x y] ball that by (simp add: Int_interval [symmetric])
         show ?thesis
          using B' [of a b] B' [of c d] norm_triangle_half_r by blast
       qed
     qed (use ‹B > 0› in auto)}
   then show ?thesis
    by force
  qed
  finally show ?thesis .
qed

lemma integrable_on_subcbox:
  fixes f :: 'n::euclidean_space ⇒ 'a::banach
  assumes intf: f integrable_on S
    and sub: cbox a b ⊆ S
  shows f integrable_on cbox a b
proof −
  have (λx. if x ∈ S then f x else 0) integrable_on cbox a b
    by (simp add: intf integrable_altD(1))
then show ?thesis
   by (metis (mono_tags) sub integrable_restrict_Int le_inf_iff order_refl sub-
set_antisym)
qed
```

6.14.37 A straddling criterion for integrability

```
lemma integrable_straddle_interval:
  fixes f :: 'n::euclidean_space ⇒ real
  assumes ⋀e. e>0 ⟹ ∃ g h i j. (g has_integral i) (cbox a b) ∧ (h has_integral
j) (cbox a b) ∧
                              |i − j| < e ∧ (∀ x∈cbox a b. (g x) ≤ f x ∧ f x ≤ h x)
  shows f integrable_on cbox a b
proof −
  have ∃ d. gauge d ∧
          (∀ p1 p2. p1 tagged_division_of cbox a b ∧ d fine p1 ∧
                    p2 tagged_division_of cbox a b ∧ d fine p2 ⟶
                    |(∑ (x,K)∈p1. content K *R f x) − (∑ (x,K)∈p2. content K *R
f x)| < e)
    if e > 0 for e
  proof −
    have e: e/3 > 0
```

```
    using that by auto
  then obtain g h i j where ij: |i − j| < e/3
        and (g has_integral i) (cbox a b)
        and (h has_integral j) (cbox a b)
        and fgh: ⋀x. x ∈ cbox a b ⟹ g x ≤ f x ∧ f x ≤ h x
    using assms real_norm_def by metis
  then obtain d1 d2 where gauge d1 gauge d2
        and d1: ⋀p. ⟦p tagged_division_of cbox a b; d1 fine p⟧ ⟹
                  |(∑(x,K)∈p. content K *R g x) − i| < e/3
        and d2: ⋀p. ⟦p tagged_division_of cbox a b; d2 fine p⟧ ⟹
                  |(∑(x,K) ∈ p. content K *R h x) − j| < e/3
    by (metis e has_integral real_norm_def)
  have |(∑(x,K) ∈ p1. content K *R f x) − (∑(x,K) ∈ p2. content K *R f x)|
< e
    if p1: p1 tagged_division_of cbox a b and 11: d1 fine p1 and 21: d2 fine p1
     and p2: p2 tagged_division_of cbox a b and 12: d1 fine p2 and 22: d2 fine
p2 for p1 p2
  proof −
    have *: ⋀g1 g2 h1 h2 f1 f2.
              ⟦|g2 − i| < e/3; |g1 − i| < e/3; |h2 − j| < e/3; |h1 − j| < e/3;
               g1 − h2 ≤ f1 − f2; f1 − f2 ≤ h1 − g2⟧
              ⟹ |f1 − f2| < e
      using ‹e > 0› ij by arith
    have 0: (∑(x, k)∈p1. content k *R f x) − (∑(x, k)∈p1. content k *R g x)
≥ 0
          0 ≤ (∑(x, k)∈p2. content k *R h x) − (∑(x, k)∈p2. content k *R f x)
          (∑(x, k)∈p2. content k *R f x) − (∑(x, k)∈p2. content k *R g x) ≥ 0
          0 ≤ (∑(x, k)∈p1. content k *R h x) − (∑(x, k)∈p1. content k *R f x)
      unfolding sum_subtractf[symmetric]
       apply (auto intro!: sum_nonneg)
       apply (meson fgh measure_nonneg mult_left_mono tag_in_interval that
sum_nonneg)+
      done
    show ?thesis
    proof (rule *)
      show |(∑(x,K) ∈ p2. content K *R g x) − i| < e/3
        by (rule d1[OF p2 12])
      show |(∑(x,K) ∈ p1. content K *R g x) − i| < e/3
        by (rule d1[OF p1 11])
      show |(∑(x,K) ∈ p2. content K *R h x) − j| < e/3
        by (rule d2[OF p2 22])
      show |(∑(x,K) ∈ p1. content K *R h x) − j| < e/3
        by (rule d2[OF p1 21])
    qed (use 0 in auto)
  qed
  then show ?thesis
    by (rule_tac x=λx. d1 x ∩ d2 x in exI)
      (auto simp: fine_Int intro: ‹gauge d1› ‹gauge d2› d1 d2)
qed
```

```
  then show ?thesis
    by (simp add: integrable_Cauchy)
qed

lemma integrable_straddle:
  fixes f :: 'n::euclidean_space ⇒ real
  assumes ⋀e. e>0 ⟹ ∃g h i j. (g has_integral i) s ∧ (h has_integral j) s ∧
                    |i − j| < e ∧ (∀x∈s. g x ≤ f x ∧ f x ≤ h x)
  shows f integrable_on s
proof −
  let ?fs = (λx. if x ∈ s then f x else 0)
  have ?fs integrable_on cbox a b for a b
  proof (rule integrable_straddle_interval)
    fix e::real
    assume e > 0
    then have ∗: e/4 > 0
      by auto
   with assms obtain g h i j where g: (g has_integral i) s and h: (h has_integral
j) s
                  and ij: |i − j| < e/4
                  and fgh: ⋀x. x ∈ s ⟹ g x ≤ f x ∧ f x ≤ h x
      by metis
    let ?gs = (λx. if x ∈ s then g x else 0)
    let ?hs = (λx. if x ∈ s then h x else 0)
    obtain Bg where Bg: ⋀a b. ball 0 Bg ⊆ cbox a b ⟹ |integral (cbox a b) ?gs
− i| < e/4
             and int_g: ⋀a b. ?gs integrable_on cbox a b
      using g ∗ unfolding has_integral_alt' real_norm_def by meson
    obtain Bh where
        Bh: ⋀a b. ball 0 Bh ⊆ cbox a b ⟹ |integral (cbox a b) ?hs − j| < e/4
       and int_h: ⋀a b. ?hs integrable_on cbox a b
      using h ∗ unfolding has_integral_alt' real_norm_def by meson
    define c where c = (∑i∈Basis. min (a•i) (− (max Bg Bh)) ∗R i)
    define d where d = (∑i∈Basis. max (b•i) (max Bg Bh) ∗R i)
    have ⟦norm (0 − x) < Bg; i ∈ Basis⟧ ⟹ c • i ≤ x • i ∧ x • i ≤ d • i for x i
      using Basis_le_norm[of i x] unfolding c_def d_def by auto
    then have ballBg: ball 0 Bg ⊆ cbox c d
      by (auto simp: mem_box dist_norm)
    have ⟦norm (0 − x) < Bh; i ∈ Basis⟧ ⟹ c • i ≤ x • i ∧ x • i ≤ d • i for x i
      using Basis_le_norm[of i x] unfolding c_def d_def by auto
    then have ballBh: ball 0 Bh ⊆ cbox c d
      by (auto simp: mem_box dist_norm)
    have ab_cd: cbox a b ⊆ cbox c d
      by (auto simp: c_def d_def subset_box_imp)
   have ∗∗: ⋀ch cg ag ah::real. ⟦|ah − ag| ≤ |ch − cg|; |cg − i| < e/4; |ch − j|
< e/4⟧
       ⟹ |ag − ah| < e
      using ij by arith
    show ∃g h i j. (g has_integral i) (cbox a b) ∧ (h has_integral j) (cbox a b) ∧
```

```
|i − j| < e ∧
        (∀x∈cbox a b. g x ≤ (if x ∈ s then f x else 0) ∧
                      (if x ∈ s then f x else 0) ≤ h x)
   proof (intro exI ballI conjI)
     have eq: ⋀x f g. (if x ∈ s then f x else 0) − (if x ∈ s then g x else 0) =
                      (if x ∈ s then f x − g x else (0::real))
       by auto
     have int_hg: (λx. if x ∈ s then h x − g x else 0) integrable_on cbox a b
                  (λx. if x ∈ s then h x − g x else 0) integrable_on cbox c d
         by (metis (no_types) integrable_diff g h has_integral_integrable integrable_altD(1))+
     show (?gs has_integral integral (cbox a b) ?gs) (cbox a b)
          (?hs has_integral integral (cbox a b) ?hs) (cbox a b)
       by (intro integrable_integral int_g int_h)+
     then have integral (cbox a b) ?gs ≤ integral (cbox a b) ?hs
       using fgh by (force intro: has_integral_le)
     then have 0 ≤ integral (cbox a b) ?hs − integral (cbox a b) ?gs
       by simp
     then have |integral (cbox a b) ?hs − integral (cbox a b) ?gs|
             ≤ |integral (cbox c d) ?hs − integral (cbox c d) ?gs|
       apply (simp add: integral_diff [symmetric] int_g int_h)
        apply (subst abs_of_nonneg[OF integral_nonneg[OF integrable_diff, OF int_h int_g]])
          using fgh apply (force simp: eq intro!: integral_subset_le [OF ab_cd int_hg])+
       done
     then show |integral (cbox a b) ?gs − integral (cbox a b) ?hs| < e
       using ∗∗ Bg ballBg Bh ballBh by blast
     show ⋀x. x ∈ cbox a b ⟹ ?gs x ≤ ?fs x ⋀x. x ∈ cbox a b ⟹ ?fs x ≤ ?hs x
       using fgh by auto
   qed
 qed
 then have int_f: ?fs integrable_on cbox a b for a b
   by simp
 have ∃B>0. ∀a b c d.
               ball 0 B ⊆ cbox a b ∧ ball 0 B ⊆ cbox c d ⟶
               abs (integral (cbox a b) ?fs − integral (cbox c d) ?fs) < e
     if 0 < e for e
 proof −
   have ∗: e/3 > 0
     using that by auto
  with assms obtain g h i j where g: (g has_integral i) s and h: (h has_integral j) s
                and ij: |i − j| < e/3
                and fgh: ⋀x. x ∈ s ⟹ g x ≤ f x ∧ f x ≤ h x
     by metis
   let ?gs = (λx. if x ∈ s then g x else 0)
   let ?hs = (λx. if x ∈ s then h x else 0)
   obtain Bg where Bg > 0
```

```
          and Bg: ⋀a b. ball 0 Bg ⊆ cbox a b ⟹ |integral (cbox a b) ?gs − i| <
e/3
          and int_g: ⋀a b. ?gs integrable_on cbox a b
    using g ∗ unfolding has_integral_alt' real_norm_def by meson
  obtain Bh where Bh > 0
          and Bh: ⋀a b. ball 0 Bh ⊆ cbox a b ⟹ |integral (cbox a b) ?hs − j|
< e/3
          and int_h: ⋀a b. ?hs integrable_on cbox a b
    using h ∗ unfolding has_integral_alt' real_norm_def by meson
  { fix a b c d :: 'n
    assume as: ball 0 (max Bg Bh) ⊆ cbox a b ball 0 (max Bg Bh) ⊆ cbox c d
    have ∗∗: ball 0 Bg ⊆ ball (0::'n) (max Bg Bh) ball 0 Bh ⊆ ball (0::'n) (max
Bg Bh)
      by auto
    have ∗: ⋀ga gc ha hc fa fc. ⟦|ga − i| < e/3; |gc − i| < e/3; |ha − j| < e/3;
                  |hc − j| < e/3; ga ≤ fa; fa ≤ ha; gc ≤ fc; fc ≤ hc⟧ ⟹
      |fa − fc| < e
      using ij by arith
    have abs (integral (cbox a b) (λx. if x ∈ s then f x else 0) − integral (cbox c
d)
      (λx. if x ∈ s then f x else 0)) < e
    proof (rule ∗)
      show |integral (cbox a b) ?gs − i| < e/3
        using ∗∗ Bg as by blast
      show |integral (cbox c d) ?gs − i| < e/3
        using ∗∗ Bg as by blast
      show |integral (cbox a b) ?hs − j| < e/3
        using ∗∗ Bh as by blast
      show |integral (cbox c d) ?hs − j| < e/3
        using ∗∗ Bh as by blast
    qed (use int_f int_g int_h fgh in ‹simp_all add: integral_le›)
  }
  then show ?thesis
    apply (rule_tac x=max Bg Bh in exI)
      using ‹Bg > 0› by auto
 qed
 then show ?thesis
   unfolding integrable_alt[of f] real_norm_def by (blast intro: int_f)
qed
```

6.14.38 Adding integrals over several sets

```
lemma has_integral_Un:
 fixes f :: 'n::euclidean_space ⇒ 'a::banach
 assumes f: (f has_integral i) S (f has_integral j) T
   and neg: negligible (S ∩ T)
 shows (f has_integral (i + j)) (S ∪ T)
 unfolding has_integral_restrict_UNIV[symmetric, of f]
proof (rule has_integral_spike[OF neg])
```

let *?f* = $\lambda x.$ (*if* $x \in S$ *then* *f x* *else* *0*) + (*if* $x \in T$ *then* *f x* *else* *0*)
show (*?f has_integral i* + *j*) *UNIV*
by (*simp add*: *f has_integral_add*)
qed *auto*

lemma *integral_Un* [*simp*]:
fixes *f* :: *'n*::*euclidean_space* ⇒ *'a*::*banach*
assumes *f integrable_on S f integrable_on T negligible* (*S* ∩ *T*)
shows *integral* (*S* ∪ *T*) *f* = *integral S f* + *integral T f*
by (*simp add*: *has_integral_Un assms integrable_integral integral_unique*)

lemma *integrable_Un*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b* :: *banach*
assumes *negligible* (*A* ∩ *B*) *f integrable_on A f integrable_on B*
shows *f integrable_on* (*A* ∪ *B*)
proof −
from *assms* **obtain** *y z* **where** (*f has_integral y*) *A* (*f has_integral z*) *B*
by (*auto simp*: *integrable_on_def*)
from *has_integral_Un*[*OF this assms*(*1*)] **show** *?thesis* **by** (*auto simp*: *integrable_on_def*)
qed

lemma *integrable_Un'*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b* :: *banach*
assumes *f integrable_on A f integrable_on B negligible* (*A* ∩ *B*) *C* = *A* ∪ *B*
shows *f integrable_on C*
using *integrable_Un*[*of A B f*] *assms* **by** *simp*

lemma *has_integral_UN*:
fixes *f* :: *'n*::*euclidean_space* ⇒ *'a*::*banach*
assumes *finite I*
and *int*: ⋀*i*. *i* ∈ *I* ⟹ (*f has_integral* (*g i*)) (*T i*)
and *neg*: *pairwise* (λ*i i'*. *negligible* (*T i* ∩ *T i'*)) *I*
shows (*f has_integral* (*sum g I*)) (⋃*i*∈*I*. *T i*)
proof −
let *?U* = ((λ(*a*,*b*). *T a* ∩ *T b*) ' {(*a*,*b*). *a* ∈ *I* ∧ *b* ∈ *I*−{*a*}})
have ((λ*x*. *if x* ∈ (⋃*i*∈*I*. *T i*) *then f x else 0*) *has_integral sum g I*) *UNIV*
proof (*rule has_integral_spike*)
show *negligible* (⋃ *?U*)
proof (*rule negligible_Union*)
have *finite* (*I* × *I*)
by (*simp add*: ‹*finite I*›)
moreover have {(*a*,*b*). *a* ∈ *I* ∧ *b* ∈ *I*−{*a*}} ⊆ *I* × *I*
by *auto*
ultimately show *finite ?U*
by (*simp add*: *finite_subset*)
show ⋀*t*. *t* ∈ *?U* ⟹ *negligible t*
using *neg* **unfolding** *pairwise_def* **by** *auto*
qed

```
  next
    show (if x ∈ (⋃ i∈I. 𝒯 i) then f x else 0) = (∑ i∈I. if x ∈ 𝒯 i then f x else 0)
      if x ∈ UNIV − (⋃ ?𝒰) for x
    proof clarsimp
      fix i assume i: i ∈ I x ∈ 𝒯 i
      then have ∀ j∈I. x ∈ 𝒯 j ⟷ j = i
        using that by blast
      with i show f x = (∑ i∈I. if x ∈ 𝒯 i then f x else 0)
        by (simp add: sum.delta[OF ‹finite I›])
    qed
  next
    show ((λx. (∑ i∈I. if x ∈ 𝒯 i then f x else 0)) has_integral sum g I) UNIV
      using int by (simp add: has_integral_restrict_UNIV has_integral_sum [OF
‹finite I›])
  qed
  then show ?thesis
    using has_integral_restrict_UNIV by blast
qed

lemma has_integral_Union:
  fixes f :: 'n::euclidean_space ⇒ 'a::banach
  assumes finite 𝒯
    and ⋀S. S ∈ 𝒯 ⟹ (f has_integral (i S)) S
    and pairwise (λS S'. negligible (S ∩ S')) 𝒯
  shows (f has_integral (sum i 𝒯)) (⋃𝒯)
proof −
  have (f has_integral (sum i 𝒯)) (⋃ S∈𝒯. S)
    by (intro has_integral_UN assms)
  then show ?thesis
    by force
qed
```

In particular adding integrals over a division, maybe not of an interval.

```
lemma has_integral_combine_division:
  fixes f :: 'n::euclidean_space ⇒ 'a::banach
  assumes 𝒟 division_of S
    and ⋀k. k ∈ 𝒟 ⟹ (f has_integral (i k)) k
  shows (f has_integral (sum i 𝒟)) S
proof −
  note 𝒟 = division_ofD[OF assms(1)]
  have neg: negligible (S ∩ s') if S ∈ 𝒟 s' ∈ 𝒟 S ≠ s' for S s'
  proof −
    obtain a c b 𝒟 where obt: S = cbox a b s' = cbox c 𝒟
      by (meson ‹S ∈ 𝒟› ‹s' ∈ 𝒟› 𝒟(4))
    from 𝒟(5)[OF that] show ?thesis
      unfolding obt interior_cbox
      by (metis (no_types, lifting) Diff_empty Int_interval box_Int_box negligible_frontier_interval)
  qed
```

show *?thesis*
unfolding $\mathcal{D}(6)$[*symmetric*]
by (*auto intro*: $\mathcal{D}$ *neg assms has_integral_Union pairwiseI*)
qed

lemma *integral_combine_division_bottomup*:
fixes f :: *'n::euclidean_space* ⇒ *'a::banach*
assumes $\mathcal{D}$ *division_of S* $\bigwedge k.\ k \in \mathcal{D} \Longrightarrow f$ *integrable_on k*
shows *integral S f* = *sum* ($\lambda i.$ *integral i f*) $\mathcal{D}$
by (*meson assms integral_unique has_integral_combine_division has_integral_integrable_integral*)

lemma *has_integral_combine_division_topdown*:
fixes f :: *'n::euclidean_space* ⇒ *'a::banach*
assumes *f*: *f integrable_on S*
and $\mathcal{D}$: $\mathcal{D}$ *division_of K*
and $K \subseteq S$
shows (*f has_integral* (*sum* ($\lambda i.$ *integral i f*) $\mathcal{D}$)) *K*
proof −
have *f integrable_on L* **if** $L \in \mathcal{D}$ **for** *L*
by (*smt* (*verit*, *best*) *assms cbox_division_memE f integrable_on_subcbox subset_trans that*)
then show *?thesis*
by (*meson* $\mathcal{D}$ *has_integral_combine_division has_integral_integrable_integral*)
qed

lemma *integral_combine_division_topdown*:
fixes f :: *'n::euclidean_space* ⇒ *'a::banach*
assumes *f integrable_on S*
and $\mathcal{D}$ *division_of S*
shows *integral S f* = *sum* ($\lambda i.$ *integral i f*) $\mathcal{D}$
using *assms has_integral_combine_division_topdown* **by** *blast*

lemma *integrable_combine_division*:
fixes f :: *'n::euclidean_space* ⇒ *'a::banach*
assumes $\mathcal{D}$: $\mathcal{D}$ *division_of S*
and *f*: $\bigwedge i.\ i \in \mathcal{D} \Longrightarrow f$ *integrable_on i*
shows *f integrable_on S*
using *f* **unfolding** *integrable_on_def* **by** (*metis has_integral_combine_division*[*OF* $\mathcal{D}$])

lemma *integrable_on_subdivision*:
fixes f :: *'n::euclidean_space* ⇒ *'a::banach*
assumes $\mathcal{D}$: $\mathcal{D}$ *division_of i*
and *f*: *f integrable_on S*
and $i \subseteq S$
shows *f integrable_on i*
proof −
have *f integrable_on i* **if** $i \in \mathcal{D}$ **for** *i*
by (*smt* (*verit*, *best*) *assms cbox_division_memE f integrable_on_subcbox or-*

der_trans that)
then show *?thesis*
using $\mathcal{D}$ *integrable_combine_division* **by** *blast*
qed

6.14.39 Also tagged divisions

lemma *has_integral_combine_tagged_division*:
fixes *f* :: *'n::euclidean_space* ⇒ *'a::banach*
assumes *p tagged_division_of S*
and ⋀*x k*. (*x*,*k*) ∈ *p* ⟹ (*f has_integral* (*i k*)) *k*
shows (*f has_integral* (∑(*x*,*k*)∈*p*. *i k*)) *S*
proof −
have ∗: (*f has_integral* (∑ *k*∈*snd*‘*p*. *integral k f*)) *S*
by (*smt* (*verit*, *del_insts*) *assms division_of_tagged_division has_integral_combine_division has_integral_iff imageE prod.collapse*)
also have (∑ *k*∈*snd*‘*p*. *integral k f*) = (∑(*x*, *k*)∈*p*. *integral k f*)
by (*intro sum.over_tagged_division_lemma*[*OF assms*(*1*), *symmetric*] *integral_null*)
(*simp add*: *content_eq_0_interior*)
finally show *?thesis*
using *assms* **by** (*auto simp add*: *has_integral_iff intro*!: *sum.cong*)
qed

lemma *integral_combine_tagged_division_bottomup*:
fixes *f* :: *'n::euclidean_space* ⇒ *'a::banach*
assumes *p*: *p tagged_division_of* (*cbox a b*)
and *f*: ⋀*x k*. (*x*,*k*)∈*p* ⟹ *f integrable_on k*
shows *integral* (*cbox a b*) *f* = *sum* (λ(*x*,*k*). *integral k f*) *p*
by (*simp add*: *has_integral_combine_tagged_division*[*OF p*] *integral_unique f integrable_integral*)

lemma *has_integral_combine_tagged_division_topdown*:
fixes *f* :: *'n::euclidean_space* ⇒ *'a::banach*
assumes *f*: *f integrable_on cbox a b*
and *p*: *p tagged_division_of* (*cbox a b*)
shows (*f has_integral* (*sum* (λ(*x*,*K*). *integral K f*) *p*)) (*cbox a b*)
proof −
have (*f has_integral integral K f*) *K* **if** (*x*,*K*) ∈ *p* **for** *x K*
by (*metis assms integrable_integral integrable_on_subcbox tagged_division_ofD*(*3*,*4*) *that*)
then show *?thesis*
by (*simp add*: *has_integral_combine_tagged_division p*)
qed

lemma *integral_combine_tagged_division_topdown*:
fixes *f* :: *'n::euclidean_space* ⇒ *'a::banach*
assumes *f integrable_on cbox a b*
and *p tagged_division_of* (*cbox a b*)

shows *integral* (*cbox a b*) *f* = *sum* (λ(*x*,*k*). *integral k f*) *p*
using *assms* **by** (*auto intro*: *integral_unique* [*OF has_integral_combine_tagged_division_topdown*])

6.14.40 Henstock's lemma

lemma *Henstock_lemma_part1*:
fixes *f* :: *'n*::*euclidean_space* ⇒ *'a*::*banach*
assumes *intf*: *f integrable_on cbox a b*
and *e* > *0*
and *gauge d*
and *less_e*: ⋀*p*. ⟦*p tagged_division_of* (*cbox a b*); *d fine p*⟧ ⟹
norm (*sum* (λ(*x*,*K*). *content K* ∗*R f x*) *p* − *integral*(*cbox a b*) *f*)
< *e*
and *p*: *p tagged_partial_division_of* (*cbox a b*) *d fine p*
shows *norm* (*sum* (λ(*x*,*K*). *content K* ∗*R f x* − *integral K f*) *p*) ≤ *e* (**is** *?lhs* ≤
e)
proof (*rule field_le_epsilon*)
fix *k* :: *real*
assume *k* > *0*
let *?SUM* = λ*p*. (∑ (*x*,*K*) ∈ *p*. *content K* ∗*R f x*)
note *p'* = *tagged_partial_division_ofD*[*OF p*(*1*)]
have ⋃(*snd* ‘ *p*) ⊆ *cbox a b*
using *p'*(*3*) **by** *fastforce*
then obtain *q* **where** *q*: *snd* ‘ *p* ⊆ *q* **and** *qdiv*: *q division_of cbox a b*
by (*meson p*(*1*) *partial_division_extend_interval partial_division_of_tagged_division*)
note *q'* = *division_ofD*[*OF qdiv*]
define *r* **where** *r* = *q* − *snd* ‘ *p*
have *snd* ‘ *p* ∩ *r* = {}
unfolding *r_def* **by** *auto*
have *finite r*
using *q'* **unfolding** *r_def* **by** *auto*
have ∃ *p*. *p tagged_division_of i* ∧ *d fine p* ∧
norm (*?SUM p* − *integral i f*) < *k* / (*real* (*card r*) + *1*)
if *i*∈*r* **for** *i*
proof −
have *gt0*: *k* / (*real* (*card r*) + *1*) > *0* **using** ‹*k* > *0*› **by** *simp*
have *i*: *i* ∈ *q*
using *that* **unfolding** *r_def* **by** *auto*
then obtain *u v* **where** *uv*: *i* = *cbox u v*
using *q'*(*4*) **by** *blast*
then have *cbox u v* ⊆ *cbox a b*
using *i q'*(*2*) **by** *auto*
then have *f integrable_on cbox u v*
by (*rule integrable_subinterval*[*OF intf*])
with *integrable_integral*[*OF this*, *unfolded has_integral*[*of f*]]
obtain *dd* **where** *gauge dd* **and** *dd*:
⋀*D*. ⟦*D tagged_division_of cbox u v*; *dd fine D*⟧ ⟹
norm (*?SUM D* − *integral* (*cbox u v*) *f*) < *k* / (*real* (*card r*) + *1*)
using *gt0* **by** *auto*

```
    with gauge_Int[OF ‹gauge d› ‹gauge dd›]
    obtain qq where qq: qq tagged_division_of cbox u v (λx. d x ∩ dd x) fine qq
      using fine_division_exists by blast
    with dd[of qq]  show ?thesis
      by (auto simp: fine_Int uv)
  qed
  then obtain qq where qq: ⋀i. i ∈ r ⟹ qq i tagged_division_of i ∧
      d fine qq i ∧ norm (?SUM (qq i) − integral i f) < k / (real (card r) + 1)
    by metis

  let ?p = p ∪ ⋃(qq ‘ r)
  have norm (?SUM ?p − integral (cbox a b) f) < e
  proof (rule less_e)
    show d fine ?p
      by (metis (mono_tags, opaque_lifting) qq fine_Un fine_Union imageE p(2))
    note ptag = tagged_partial_division_of_Union_self[OF p(1)]
    have p ∪ ⋃(qq ‘ r) tagged_division_of ⋃(snd ‘ p) ∪ ⋃r
    proof (rule tagged_division_Un[OF ptag tagged_division_Union [OF ‹finite
r›]])
      show ⋀i. i ∈ r ⟹ qq i tagged_division_of i
        using qq by auto
      show ⋀i1 i2. ⟦i1 ∈ r; i2 ∈ r; i1 ≠ i2⟧ ⟹ interior i1 ∩ interior i2 = {}
        by (simp add: q′(5) r_def)
      show interior (⋃(snd ‘ p)) ∩ interior (⋃r) = {}
      proof (rule Int_interior_Union_intervals [OF ‹finite r›])
        show open (interior (⋃(snd ‘ p)))
          by blast
        show ⋀T. T ∈ r ⟹ ∃a b. T = cbox a b
          by (simp add: q′(4) r_def)
        have interior T ∩ interior (⋃(snd ‘ p)) = {} if T ∈ r for T
        proof (rule Int_interior_Union_intervals)
          show ⋀U. U ∈ snd ‘ p ⟹ ∃a b. U = cbox a b
            using q q′(4) by blast
          show ⋀U. U ∈ snd ‘ p ⟹ interior T ∩ interior U = {}
            by (metis DiffE q q′(5) r_def subsetD that)
        qed (use p′ in auto)
        then show ⋀T. T ∈ r ⟹ interior (⋃(snd ‘ p)) ∩ interior T = {}
          by (metis Int_commute)
      qed
    qed
    moreover have ⋃(snd ‘ p) ∪ ⋃r = cbox a b and {qq i |i. i ∈ r} = qq ‘ r
      using qdiv q unfolding Union_Un_distrib[symmetric] r_def by auto
    ultimately show ?p tagged_division_of (cbox a b)
      by fastforce
  qed
  then have norm (?SUM p + (?SUM (⋃(qq ‘ r))) − integral (cbox a b) f) < e
  proof (subst sum.union_inter_neutral[symmetric, OF ‹finite p›], safe)
    show content L *R f x = 0 if (x, L) ∈ p (x, L) ∈ qq K K ∈ r for x K L
    proof −
```

```
      obtain u v where uv: L = cbox u v
        using ‹(x,L) ∈ p› p'(4) by blast
      have L ⊆ K
        using  qq[OF that(3)] tagged_division_ofD(3) ‹(x,L) ∈ qq K› by metis
      have L ∈ snd ‘ p
        using ‹(x,L) ∈ p› image_iff by fastforce
      then have L ∈ q K ∈ q L ≠ K
        using that q(1) unfolding r_def by auto
      with q'(5) show content L *R f x = 0
      by (metis ‹L ⊆ K› content_eq_0_interior inf.orderE interior_Int scaleR_eq_0_iff
uv)
    qed
    show finite (⋃(qq ‘ r))
      by (meson finite_UN qq ‹finite r› tagged_division_of_finite)
  qed
  moreover have content M *R f x = 0
      if x: (x,M) ∈ qq K (x,M) ∈ qq L and KL: qq K ≠ qq L and r: K ∈ r L ∈ r
    for x M K L
  proof −
    note kl = tagged_division_ofD(3,4)[OF qq[THEN conjunct1]]
    obtain u v where uv: M = cbox u v
      using ‹(x, M) ∈ qq L› ‹L ∈ r› kl(2) by blast
    have empty: interior (K ∩ L) = {}
      by (metis DiffD1 interior_Int q'(5) r_def KL r)
    with that kl show content M *R f x = 0
      by (metis content_eq_0_interior dual_order.refl inf.orderE inf_mono inte-
rior_mono
                scaleR_eq_0_iff subset_empty uv x)
  qed
  ultimately have norm (?SUM p + sum ?SUM (qq ‘ r) − integral (cbox a b) f)
< e
    apply (subst (asm) sum.Union_comp)
    using qq by (force simp: split_paired_all)+
  moreover have content M *R f x = 0
      if K ∈ r L ∈ r K ≠ L qq K = qq L (x, M) ∈ qq K for K L x M
    using tagged_division_ofD(6) qq that by (metis (no_types, lifting))
  ultimately have less_e: norm (?SUM p + sum (?SUM ∘ qq) r − integral (cbox
a b) f) < e
  proof (subst (asm) sum.reindex_nontrivial [OF ‹finite r›])
    qed (auto simp: split_paired_all sum.neutral)
  have norm_le: norm (cp − ip) ≤ e + k
              if norm ((cp + cr) − i) < e norm (cr − ir) < k ip + ir = i
              for ir ip i cr cp::'a
    using norm_triangle_le[of cp + cr − i − (cr − ir)] that
    unfolding that(3)[symmetric] norm_minus_cancel
    by (auto simp add: algebra_simps)

  have ?lhs =  norm (?SUM p − (∑(x, k)∈p. integral k f))
    unfolding split_def sum_subtractf ..
```

```
  also have ... ≤ e + k
  proof (rule norm_le[OF less_e])
    have lessk: k * real (card r) / (1 + real (card r)) < k
      using ‹k>0› by (auto simp add: field_simps)
    have norm (sum (?SUM ∘ qq) r - (∑k∈r. integral k f)) ≤ (∑x∈r. k / (real
(card r) + 1))
     unfolding sum_subtractf[symmetric] by (force dest: qq intro!: sum_norm_le)
    also have ... < k
      by (simp add: lessk add.commute mult.commute)
    finally show norm (sum (?SUM ∘ qq) r - (∑k∈r. integral k f)) < k .
  next
    from q(1) have [simp]: snd ` p ∪ q = q by auto
    have integral l f = 0
      if inp: (x, l) ∈ p (y, m) ∈ p and ne: (x, l) ≠ (y, m) and l = m for x l y m
    proof -
      obtain u v where uv: l = cbox u v
        using inp p'(4) by blast
      then show ?thesis
        using uv that p
        by (metis content_eq_0_interior dual_order.refl inf.orderE integral_null
ne tagged_partial_division_ofD(5))
    qed
    then have (∑(x, K)∈p. integral K f) = (∑K∈snd ` p. integral K f)
      apply (subst sum.reindex_nontrivial [OF ‹finite p›])
      unfolding split_paired_all split_def by auto
    then show (∑(x, k)∈p. integral k f) + (∑k∈r. integral k f) = integral (cbox
a b) f
      unfolding integral_combine_division_topdown[OF intf qdiv] r_def
      using q'(1) p'(1) sum.union_disjoint [of snd ` p q - snd ` p, symmetric]
        by simp
  qed
  finally show ?lhs ≤ e + k .
qed

lemma Henstock_lemma_part2:
  fixes f :: 'm::euclidean_space ⇒ 'n::euclidean_space
  assumes fed: f integrable_on cbox a b e > 0 gauge d
    and less_e: ⋀D. ⟦D tagged_division_of (cbox a b); d fine D⟧ ⟹
                  norm (sum (λ(x,k). content k *R f x) D - integral (cbox a b) f)
< e
    and tag: p tagged_partial_division_of (cbox a b)
    and d fine p
  shows sum (λ(x,k). norm (content k *R f x - integral k f)) p ≤ 2 * real
(DIM('n)) * e
proof -
  have finite p
    using tag tagged_partial_division_ofD by blast
  then show ?thesis
    unfolding split_def
```

```
  proof (rule sum_norm_allsubsets_bound)
    fix Q
    assume Q: Q ⊆ p
    then have fine: d fine Q
      by (simp add: ‹d fine p› fine_subset)
    show norm (∑ x∈Q. content (snd x) *R f (fst x) − integral (snd x) f) ≤ e
      apply (rule Henstock_lemma_part1[OF fed less_e, unfolded split_def])
      using Q tag tagged_partial_division_subset by (force simp add: fine)+
  qed
qed

lemma Henstock_lemma:
  fixes f :: 'm::euclidean_space ⇒ 'n::euclidean_space
  assumes intf: f integrable_on cbox a b
    and e > 0
  obtains γ where gauge γ
    and ⋀p. ⟦p tagged_partial_division_of (cbox a b); γ fine p⟧ ⟹
          sum (λ(x,k). norm(content k *R f x − integral k f)) p < e
proof −
  have *: e/(2 * (real DIM('n) + 1)) > 0 using ‹e > 0› by simp
  with integrable_integral[OF intf, unfolded has_integral]
  obtain γ where gauge γ
    and γ: ⋀𝒟. ⟦𝒟 tagged_division_of cbox a b; γ fine 𝒟⟧ ⟹
        norm ((∑ (x,K)∈𝒟. content K *R f x) − integral (cbox a b) f)
        < e/(2 * (real DIM('n) + 1))
    by metis
  show thesis
  proof (rule that [OF ‹gauge γ›])
    fix p
    assume p: p tagged_partial_division_of cbox a b γ fine p
    have (∑ (x,K)∈p. norm (content K *R f x − integral K f))
        ≤ 2 * real DIM('n) * (e/(2 * (real DIM('n) + 1)))
      using Henstock_lemma_part2[OF intf * ‹gauge γ› γ p] by metis
    also have ... < e
      using ‹e > 0› by (auto simp add: field_simps)
    finally
    show (∑ (x,K)∈p. norm (content K *R f x − integral K f)) < e .
  qed
qed
```

6.14.41 Monotone convergence (bounded interval first)

```
lemma bounded_increasing_convergent:
  fixes f :: nat ⇒ real
  shows ⟦bounded (range f); ⋀n. f n ≤ f (Suc n)⟧ ⟹ ∃l. f ——→ l
  using Bseq_mono_convergent[of f] incseq_Suc_iff[of f]
  by (auto simp: image_def Bseq_eq_bounded convergent_def incseq_def)

lemma monotone_convergence_interval:
```

fixes f :: $nat \Rightarrow 'n$::$euclidean\_space \Rightarrow real$
assumes $intf$: $\bigwedge k.\ (f\ k)\ integrable\_on\ cbox\ a\ b$
and le: $\bigwedge k\ x.\ x \in cbox\ a\ b \Longrightarrow (f\ k\ x) \leq f\ (Suc\ k)\ x$
and fg: $\bigwedge x.\ x \in cbox\ a\ b \Longrightarrow ((\lambda k.\ f\ k\ x) \longrightarrow g\ x)\ sequentially$
and bou: $bounded\ (range\ (\lambda k.\ integral\ (cbox\ a\ b)\ (f\ k)))$
shows $g\ integrable\_on\ cbox\ a\ b \wedge ((\lambda k.\ integral\ (cbox\ a\ b)\ (f\ k)) \longrightarrow integral\ (cbox\ a\ b)\ g)\ sequentially$
proof $(cases\ content\ (cbox\ a\ b) = 0)$
case $True$ **then show** $?thesis$
by $auto$
next
case $False$
have $fg1$: $(f\ k\ x) \leq (g\ x)$ **if** x: $x \in cbox\ a\ b$ **for** $x\ k$
proof −
have $\forall_F\ j\ in\ sequentially.\ f\ k\ x \leq f\ j\ x$
proof $(rule\ eventually\_sequentiallyI\ [of\ k])$
show $\bigwedge j.\ k \leq j \Longrightarrow f\ k\ x \leq f\ j\ x$
using $le\ x$ **by** $(force\ intro: transitive\_stepwise\_le)$
qed
then show $f\ k\ x \leq g\ x$
using $tendsto\_lowerbound\ [OF\ fg]\ x\ trivial\_limit\_sequentially$ **by** $blast$
qed
have int_inc: $\bigwedge n.\ integral\ (cbox\ a\ b)\ (f\ n) \leq integral\ (cbox\ a\ b)\ (f\ (Suc\ n))$
by $(metis\ integral\_le\ intf\ le)$
then obtain i **where** i: $(\lambda k.\ integral\ (cbox\ a\ b)\ (f\ k)) \longrightarrow i$
using $bounded\_increasing\_convergent\ bou$ **by** $blast$
have $\bigwedge k.\ \forall_F\ x\ in\ sequentially.\ integral\ (cbox\ a\ b)\ (f\ k) \leq integral\ (cbox\ a\ b)\ (f\ x)$
unfolding $eventually\_sequentially$
by $(force\ intro: transitive\_stepwise\_le\ int\_inc)$
then have i': $\bigwedge k.\ (integral(cbox\ a\ b)\ (f\ k)) \leq i$
using $tendsto\_le\ [OF\ trivial\_limit\_sequentially\ i]$ **by** $blast$
have $(g\ has\_integral\ i)\ (cbox\ a\ b)$
unfolding $has\_integral\ real\_norm\_def$
proof $clarify$
fix e::$real$
assume e: $e > 0$
have $\bigwedge k.\ (\exists \gamma.\ gauge\ \gamma \wedge (\forall \mathcal{D}.\ \mathcal{D}\ tagged\_division\_of\ (cbox\ a\ b) \wedge \gamma\ fine\ \mathcal{D} \longrightarrow$
$abs\ ((\sum (x,K) \in \mathcal{D}.\ content\ K *_R f\ k\ x) - integral\ (cbox\ a\ b)\ (f\ k)) < e/2\ \hat{}\ (k + 2)))$
using $intf\ e$ **by** $(auto\ simp: has\_integral\_integral\ has\_integral)$
then obtain c **where** c: $\bigwedge x.\ gauge\ (c\ x)$
$\bigwedge x\ \mathcal{D}.\ [\![\mathcal{D}\ tagged\_division\_of\ cbox\ a\ b;\ c\ x\ fine\ \mathcal{D}]\!] \Longrightarrow$
$abs\ ((\sum (u,K) \in \mathcal{D}.\ content\ K *_R f\ x\ u) - integral\ (cbox\ a\ b)\ (f\ x))$
$< e/2\ \hat{}\ (x + 2)$
by $metis$

have $\exists r.\ \forall k \geq r.\ 0 \leq i - (integral\ (cbox\ a\ b)\ (f\ k)) \wedge i - (integral\ (cbox\ a\ b)$

```
(f k)) < e/4
  proof −
    have e/4 > 0
      using e by auto
    show ?thesis
      using LIMSEQ_D [OF i ‹e/4 > 0›] i' by auto
  qed
  then obtain r where r: ⋀k. r ≤ k ⟹ 0 ≤ i − integral (cbox a b) (f k)
                         ⋀k. r ≤ k ⟹ i − integral (cbox a b) (f k) < e/4
    by metis
 have ∃n≥r. ∀k≥n. 0 ≤ (g x) − (f k x) ∧ (g x) − (f k x) < e/(4 ∗ content(cbox
a b))
    if x ∈ cbox a b for x
  proof −
    have e/(4 ∗ content (cbox a b)) > 0
      by (simp add: False content_lt_nz e)
    with fg that LIMSEQ_D
    obtain N where ∀n≥N. norm (f n x − g x) < e/(4 ∗ content (cbox a b))
      by metis
    then show ∃n≥r. ∀k≥n. 0 ≤ g x − f k x ∧ g x − f k x < e/(4 ∗ content
(cbox a b))
      apply (rule_tac x=N + r in exI)
      using fg1[OF that] by (auto simp add: field_simps)
  qed
  then obtain m where r_le_m: ⋀x. x ∈ cbox a b ⟹ r ≤ m x
    and m: ⋀x k. ⟦x ∈ cbox a b; m x ≤ k⟧
                 ⟹ 0 ≤ g x − f k x ∧ g x − f k x < e/(4 ∗ content (cbox a b))
    by metis
  define d where d x = c (m x) x for x
  show ∃γ. gauge γ ∧
          (∀D. D tagged_division_of cbox a b ∧
               γ fine D ⟶ abs ((∑(x,K)∈D. content K ∗R g x) − i) < e)
  proof (rule exI, safe)
    show gauge d
      using c(1) unfolding gauge_def d_def by auto
  next
    fix D
    assume ptag: D tagged_division_of (cbox a b) and d fine D
    note p'=tagged_division_ofD[OF ptag]
    obtain s where s: ⋀x. x ∈ D ⟹ m (fst x) ≤ s
    by (metis finite_imageI finite_nat_set_iff_bounded_le p'(1) rev_image_eqI)
    have ∗: |a − d| < e if |a − b| ≤ e/4 |b − c| < e/2 |c − d| < e/4 for a b c d
      using that norm_triangle_lt[of a − b b − c 3∗ e/4]
        norm_triangle_lt[of a − b + (b − c) c − d e]
      by (auto simp add: algebra_simps)
    show |(∑(x, k)∈D. content k ∗R g x) − i| < e
    proof (rule ∗)
      have |(∑(x,K)∈D. content K ∗R g x) − (∑(x,K)∈D. content K ∗R f (m
x) x)|
```

≤ (∑ i∈𝒟. |(case i of (x, K) ⇒ content K *R g x) − (case i of (x, K) ⇒ content K *R f (m x) x)|)
by (metis (mono_tags) sum_subtractf sum_abs)
also have ... ≤ (∑ (x, k)∈𝒟. content k * (e/(4 * content (cbox a b))))
proof (rule sum_mono, simp add: split_paired_all)
fix x K
assume xk: (x,K) ∈ 𝒟
with ptag have x: x ∈ cbox a b
by blast
then have abs (content K * (g x − f (m x) x)) ≤ content K * (e/(4 * content (cbox a b)))
by (metis m[OF x] mult_nonneg_nonneg abs_of_nonneg less_eq_real_def measure_nonneg mult_left_mono order_refl)
then show |content K * g x − content K * f (m x) x| ≤ content K * e/(4 * content (cbox a b))
by (simp add: algebra_simps)
qed
also have ... = (e/(4 * content (cbox a b))) * (∑ (x, k)∈𝒟. content k)
by (simp add: sum_distrib_left sum_divide_distrib split_def mult.commute)
also have ... ≤ e/4
by (metis False additive_content_tagged_division [OF ptag] nonzero_mult_divide_mult_cancel_r order_refl times_divide_eq_left)
finally show |(∑ (x,K)∈𝒟. content K *R g x) − (∑ (x,K)∈𝒟. content K *R f (m x) x)| ≤ e/4 .

next
have norm ((∑ (x,K)∈𝒟. content K *R f (m x) x) − (∑ (x,K)∈𝒟. integral K (f (m x))))
≤ norm (∑ j = 0..s. ∑ (x,K)∈{xk ∈ 𝒟. m (fst xk) = j}. content K *R f (m x) x − integral K (f (m x)))
apply (subst sum.group)
using s by (auto simp: sum_subtractf split_def p'(1))
also have ... < e/2
proof −
have norm (∑ j = 0..s. ∑ (x, k)∈{xk ∈ 𝒟. m (fst xk) = j}. content k *R f (m x) x − integral k (f (m x)))
≤ (∑ i = 0..s. e/2 ^ (i + 2))
proof (rule sum_norm_le)
fix t
assume t ∈ {0..s}
have norm (∑ (x,k)∈{xk ∈ 𝒟. m (fst xk) = t}. content k *R f (m x) x − integral k (f (m x))) =
norm (∑ (x,k)∈{xk ∈ 𝒟. m (fst xk) = t}. content k *R f t x − integral k (f t))
by (force intro!: sum.cong arg_cong[where f=norm])
also have ... ≤ e/2 ^ (t + 2)
proof (rule Henstock_lemma_part1 [OF intf])
show {xk ∈ 𝒟. m (fst xk) = t} tagged_partial_division_of cbox a b
proof (rule tagged_partial_division_subset[of 𝒟])

```
        show 𝒟 tagged_partial_division_of cbox a b
          using ptag tagged_division_of_def by blast
      qed auto
      show c t fine {xk ∈ 𝒟. m (fst xk) = t}
        using ‹d fine 𝒟› by (auto simp: fine_def d_def)
    qed (use c e in auto)
    finally show norm (∑(x,K)∈{xk ∈ 𝒟. m (fst xk) = t}. content K *R f
(m x) x −
                integral K (f (m x))) ≤ e/2 ^ (t + 2) .
  qed
  also have ... = (e/2/2) * (∑ i = 0..s. (1/2) ^ i)
    by (simp add: sum_distrib_left field_simps)
  also have ... < e/2
    by (simp add: sum_gp mult_strict_left_mono[OF _ e])
  finally show norm (∑ j = 0..s. ∑(x, k)∈{xk ∈ 𝒟.
    m (fst xk) = j}. content k *R f (m x) x − integral k (f (m x))) < e/2 .
 qed
 finally show |(∑(x,K)∈𝒟. content K *R f (m x) x) − (∑(x,K)∈𝒟. integral
K (f (m x)))| < e/2
  by simp
next
 have comb: integral (cbox a b) (f y) = (∑(x, k)∈𝒟. integral k (f y)) for y
  using integral_combine_tagged_division_topdown[OF intf ptag] by metis
 have f_le: ⋀y m n. ⟦y ∈ cbox a b; n≥m⟧ ⟹ f m y ≤ f n y
  using le by (auto intro: transitive_stepwise_le)
 have (∑(x, k)∈𝒟. integral k (f r)) ≤ (∑(x, K)∈𝒟. integral K (f (m x)))
 proof (rule sum_mono, simp add: split_paired_all)
  fix x K
  assume xK: (x, K) ∈ 𝒟
  show integral K (f r) ≤ integral K (f (m x))
  proof (rule integral_le)
   show f r integrable_on K
    by (metis integrable_on_subcbox intf p'(3) p'(4) xK)
   show f (m x) integrable_on K
      by (metis elementary_interval integrable_on_subdivision intf p'(3)
p'(4) xK)
   show f r y ≤ f (m x) y if y ∈ K for y
    using that r_le_m[of x] p'(2−3)[OF xK] f_le by auto
  qed
 qed
 moreover have (∑(x, K)∈𝒟. integral K (f (m x))) ≤ (∑(x, k)∈𝒟. integral
k (f s))
  proof (rule sum_mono, simp add: split_paired_all)
   fix x K
   assume xK: (x, K) ∈ 𝒟
   show integral K (f (m x)) ≤ integral K (f s)
   proof (rule integral_le)
    show f (m x) integrable_on K
      by (metis elementary_interval integrable_on_subdivision intf p'(3)
```

```
p'(4) xK)
          show f s integrable_on K
            by (metis integrable_on_subcbox intf p'(3) p'(4) xK)
          show f (m x) y ≤ f s y if y ∈ K for y
            using that s xK f_le p'(3) by fastforce
        qed
      qed
      moreover have 0 ≤ i − integral (cbox a b) (f r) i − integral (cbox a b) (f
r) < e/4
        using r by auto
      ultimately show |(∑(x,K)∈𝒟. integral K (f (m x))) − i| < e/4
        using comb i'[of s] by auto
    qed
  qed
 qed
 with i integral_unique show ?thesis
  by blast
qed

lemma monotone_convergence_increasing:
 fixes f :: nat ⇒ 'n::euclidean_space ⇒ real
 assumes int_f: ⋀k. (f k) integrable_on S
  and ⋀k x. x ∈ S ⟹ (f k x) ≤ (f (Suc k) x)
  and fg: ⋀x. x ∈ S ⟹ ((λk. f k x) ⟶ g x) sequentially
  and bou: bounded (range (λk. integral S (f k)))
 shows g integrable_on S ∧ ((λk. integral S (f k)) ⟶ integral S g) sequentially
proof −
 have lem: g integrable_on S ∧ ((λk. integral S (f k)) ⟶ integral S g) sequen-
tially
   if f0: ⋀k x. x ∈ S ⟹ 0 ≤ f k x
   and int_f: ⋀k. (f k) integrable_on S
   and le: ⋀k x. x ∈ S ⟹ f k x ≤ f (Suc k) x
   and lim: ⋀x. x ∈ S ⟹ ((λk. f k x) ⟶ g x) sequentially
   and bou: bounded (range(λk. integral S (f k)))
   for f :: nat ⇒ 'n::euclidean_space ⇒ real and g S
 proof −
   have fg: (f k x) ≤ (g x) if x ∈ S for x k
   proof −
     have ⋀xa. k ≤ xa ⟹ f k x ≤ f xa x
       using le  by (force intro: transitive_stepwise_le that)
     then show ?thesis
       using tendsto_lowerbound [OF lim [OF that]] eventually_sequentiallyI by
force
   qed
   obtain i where i: (λk. integral S (f k)) ⟶ i
     using bounded_increasing_convergent [OF bou] le int_f integral_le by blast
   have i': (integral S (f k)) ≤ i for k
   proof −
     have ⋀k. ⋀x. x ∈ S ⟹ ∀ n≥k. f k x ≤ f n x
```

```
      using le  by (force intro: transitive_stepwise_le)
    then show ?thesis
    using tendsto_lowerbound [OF i eventually_sequentiallyI trivial_limit_sequentially]
      by (meson int_f integral_le)
  qed
  let ?f = (λk x. if x ∈ S then f k x else 0)
  let ?g = (λx. if x ∈ S then g x else 0)
  have int: ?f k integrable_on cbox a b for a b k
    by (simp add: int_f integrable_altD(1))
  have int': ⋀k a b. f k integrable_on cbox a b ∩ S
    using int by (simp add: Int_commute integrable_restrict_Int)
  have g: ?g integrable_on cbox a b ∧
        (λk. integral (cbox a b) (?f k)) ⟶ integral (cbox a b) ?g for a b
  proof (rule monotone_convergence_interval)
    have norm (integral (cbox a b) (?f k)) ≤ norm (integral S (f k)) for k
    proof −
      have 0 ≤ integral (cbox a b) (?f k)
        by (metis (no_types) integral_nonneg Int_iff f0 inf_commute integral_restrict_Int int')
      moreover have 0 ≤ integral S (f k)
        by (simp add: integral_nonneg f0 int_f)
      moreover have integral (S ∩ cbox a b) (f k) ≤ integral S (f k)
        by (metis f0 inf_commute int' int_f integral_subset_le le_inf_iff order_refl)
      ultimately show ?thesis
        by (simp add: integral_restrict_Int)
    qed
    moreover obtain B where ⋀x. x ∈ range (λk. integral S (f k)) ⟹ norm x ≤ B
      using bou unfolding bounded_iff by blast
    ultimately show bounded (range (λk. integral (cbox a b) (?f k)))
      unfolding bounded_iff by (blast intro: order_trans)
  qed (use int le lim in auto)
  moreover have ∃B>0. ∀ a b. ball 0 B ⊆ cbox a b ⟶ norm (integral (cbox a b) ?g − i) < e
    if 0 < e for e
  proof −
    have e/4>0
      using that by auto
    with LIMSEQ_D [OF i] obtain N where N: ⋀n. n ≥ N ⟹ norm (integral S (f n) − i) < e/4
      by metis
    with int_f[of N, unfolded has_integral_integral has_integral_alt'[of f N]]
    obtain B where 0 < B and B:
      ⋀a b. ball 0 B ⊆ cbox a b ⟹ norm (integral (cbox a b) (?f N) − integral S (f N)) < e/4
      by (meson ‹0 < e/4›)
    have norm (integral (cbox a b) ?g − i) < e if ab: ball 0 B ⊆ cbox a b for a b
    proof −
```

```
      obtain M where M: ⋀n. n ≥ M ⟹ abs (integral (cbox a b) (?f n) −
integral (cbox a b) ?g) < e/2
       using ‹e > 0› g by (fastforce simp add: dest!: LIMSEQ_D [where r =
e/2])
     have ∗: ⋀α β g. ⟦|α − i| < e/2; |β − g| < e/2; α ≤ β; β ≤ i⟧ ⟹ |g − i|
< e
       unfolding real_inner_1_right by arith
     show norm (integral (cbox a b) ?g − i) < e
       unfolding real_norm_def
     proof (rule ∗)
       show |integral (cbox a b) (?f N) − i| < e/2
       proof (rule abs_triangle_half_l)
         show |integral (cbox a b) (?f N) − integral S (f N)| < e/2/2
           using B[OF ab] by simp
         show abs (i − integral S (f N)) < e/2/2
           using N by (simp add: abs_minus_commute)
       qed
       show |integral (cbox a b) (?f (M + N)) − integral (cbox a b) ?g| < e/2
         by (metis le_add1 M[of M + N])
       show integral (cbox a b) (?f N) ≤ integral (cbox a b) (?f (M + N))
       proof (intro ballI integral_le[OF int int])
         fix x assume x ∈ cbox a b
         have (f m x) ≤ (f n x) if x ∈ S n ≥ m for m n
         proof (rule transitive_stepwise_le [OF ‹n ≥ m› order_refl])
           show ⋀u y z. ⟦f u x ≤ f y x; f y x ≤ f z x⟧ ⟹ f u x ≤ f z x
             using dual_order.trans by blast
         qed (simp add: le ‹x ∈ S›)
         then show (?f N)x ≤ (?f (M+N))x
           by auto
       qed
       have integral (cbox a b ∩ S) (f (M + N)) ≤ integral S (f (M + N))
         by (metis Int_lower1 f0 inf_commute int' int_f integral_subset_le)
       then have integral (cbox a b) (?f (M + N)) ≤ integral S (f (M + N))
         by (metis (no_types) inf_commute integral_restrict_Int)
       also have ... ≤ i
         using i'[of M + N] by auto
       finally show integral (cbox a b) (?f (M + N)) ≤ i .
     qed
   qed
   then show ?thesis
     using ‹0 < B› by blast
 qed
 ultimately have (g has_integral i) S
   unfolding has_integral_alt' by auto
 then show ?thesis
   using has_integral_integrable_integral i integral_unique by metis
qed

have sub: ⋀k. integral S (λx. f k x − f 0 x) = integral S (f k) − integral S (f 0)
```

```
    by (simp add: integral_diff int_f)
  have *: ⋀x m n. x ∈ S ⟹ n≥m ⟹ f m x ≤ f n x
    using assms(2) by (force intro: transitive_stepwise_le)
  have gf: (λx. g x − f 0 x) integrable_on S ∧ ((λk. integral S (λx. f (Suc k) x −
f 0 x)) ⟶
    integral S (λx. g x − f 0 x)) sequentially
  proof (rule lem)
    show ⋀k. (λx. f (Suc k) x − f 0 x) integrable_on S
      by (simp add: integrable_diff int_f)
    show (λk. f (Suc k) x − f 0 x) ⟶ g x − f 0 x if x ∈ S for x
    proof −
      have (λn. f (Suc n) x) ⟶ g x
        using LIMSEQ_ignore_initial_segment[OF fg[OF ‹x ∈ S›], of 1] by simp
      then show ?thesis
        by (simp add: tendsto_diff)
    qed
    show bounded (range (λk. integral S (λx. f (Suc k) x − f 0 x)))
    proof −
      obtain B where B: ⋀k. norm (integral S (f k)) ≤ B
        using  bou by (auto simp: bounded_iff)
      then have norm (integral S (λx. f (Suc k) x − f 0 x))
            ≤ B + norm (integral S (f 0)) for k
        unfolding sub by (meson add_le_cancel_right norm_triangle_le_diff)
      then show ?thesis
        unfolding bounded_iff by blast
    qed
  qed (use * in auto)
  then have (λx. integral S (λxa. f (Suc x) xa − f 0 xa) + integral S (f 0))
          ⟶ integral S (λx. g x − f 0 x) + integral S (f 0)
    by (auto simp add: tendsto_add)
  moreover have (λx. g x − f 0 x + f 0 x) integrable_on S
    using gf integrable_add int_f [of 0] by metis
  ultimately show ?thesis
    by (simp add: integral_diff int_f LIMSEQ_imp_Suc sub)
qed

lemma has_integral_monotone_convergence_increasing:
  fixes f :: nat ⇒ 'a::euclidean_space ⇒ real
  assumes f: ⋀k. (f k has_integral x k) s
  assumes ⋀k x. x ∈ s ⟹ f k x ≤ f (Suc k) x
  assumes ⋀x. x ∈ s ⟹ (λk. f k x) ⟶ g x
  assumes x ⟶ x'
  shows (g has_integral x') s
proof −
  have x_eq: x = (λi. integral s (f i))
    by (simp add: integral_unique[OF f])
  then have x: range(λk. integral s (f k)) = range x
    by auto
  have *: g integrable_on s ∧ (λk. integral s (f k)) ⟶ integral s g
```

```
  proof (intro monotone_convergence_increasing allI ballI assms)
    show bounded (range(λk. integral s (f k)))
      using x convergent_imp_bounded assms by metis
  qed (use f in auto)
  then have integral s g = x'
    by (intro LIMSEQ_unique[OF _ ‹x ⟶ x'›]) (simp add: x_eq)
  with ∗ show ?thesis
    by (simp add: has_integral_integral)
qed

lemma monotone_convergence_decreasing:
  fixes f :: nat ⇒ 'n::euclidean_space ⇒ real
  assumes intf: ⋀k. (f k) integrable_on S
    and le: ⋀k x. x ∈ S ⟹ f (Suc k) x ≤ f k x
    and fg: ⋀x. x ∈ S ⟹ ((λk. f k x) ⟶ g x) sequentially
    and bou: bounded (range(λk. integral S (f k)))
  shows g integrable_on S ∧ (λk. integral S (f k)) ⟶ integral S g
proof −
  have ∗: range(λk. integral S (λx. − f k x)) = (∗R) (− 1) ‘ (range(λk. integral S
(f k)))
    by force
  have (λx. − g x) integrable_on S ∧ (λk. integral S (λx. − f k x)) ⟶ integral
S (λx. − g x)
  proof (rule monotone_convergence_increasing)
    show ⋀k. (λx. − f k x) integrable_on S
      by (blast intro: integrable_neg intf)
    show ⋀k x. x ∈ S ⟹ − f k x ≤ − f (Suc k) x
      by (simp add: le)
    show ⋀x. x ∈ S ⟹ (λk. − f k x) ⟶ − g x
      by (simp add: fg tendsto_minus)
    show bounded (range(λk. integral S (λx. − f k x)))
      using ∗ bou bounded_scaling by auto
  qed
  then show ?thesis
    by (force dest: integrable_neg tendsto_minus)
qed

lemma integral_norm_bound_integral:
  fixes f :: 'n::euclidean_space ⇒ 'a::banach
  assumes int_f: f integrable_on S
    and int_g: g integrable_on S
    and le_g: ⋀x. x ∈ S ⟹ norm (f x) ≤ g x
  shows norm (integral S f) ≤ integral S g
proof −
  have norm: norm η ≤ y + e
    if norm ζ ≤ x and |x − y| < e/2 and norm (ζ − η) < e/2
    for e x y and ζ η :: 'a
  proof −
    have norm (η − ζ) < e/2
```

```
    by (metis norm_minus_commute that(3))
  moreover have x ≤ y + e/2
    using that(2) by linarith
  ultimately show ?thesis
    using that(1) le_less_trans[OF norm_triangle_sub[of η ζ]] by (auto simp:
less_imp_le)
 qed
 have lem: norm (integral(cbox a b) f) ≤ integral (cbox a b) g
  if f: f integrable_on cbox a b
  and g: g integrable_on cbox a b
  and nle: ⋀x. x ∈ cbox a b ⟹ norm (f x) ≤ g x
  for f :: 'n ⇒ 'a and g a b
 proof (rule field_le_epsilon)
  fix e :: real
  assume e > 0
  then have e: e/2 > 0
    by auto
  with integrable_integral[OF f,unfolded has_integral[of f]]
  obtain γ where γ: gauge γ
          ⋀D. D tagged_division_of cbox a b ∧ γ fine D
        ⟹ norm ((∑(x, k)∈D. content k *R f x) − integral (cbox a b) f) < e/2
    by meson
  moreover
  from integrable_integral[OF g,unfolded has_integral[of g]] e
  obtain δ where δ: gauge δ
          ⋀D. D tagged_division_of cbox a b ∧ δ fine D
        ⟹ norm ((∑(x, k)∈D. content k *R g x) − integral (cbox a b) g) < e/2
    by meson
  ultimately have gauge (λx. γ x ∩ δ x)
    using gauge_Int by blast
  with fine_division_exists obtain D
    where p: D tagged_division_of cbox a b (λx. γ x ∩ δ x) fine D
    by metis
  have γ fine D δ fine D
    using fine_Int p(2) by blast+
  show norm (integral (cbox a b) f) ≤ integral (cbox a b) g + e
  proof (rule norm)
    have norm (content K *R f x) ≤ content K *R g x if (x, K) ∈ D for x K
    proof−
      have K: x ∈ K K ⊆ cbox a b
        using ‹(x, K) ∈ D› p(1) by blast+
      obtain u v where K = cbox u v
        using ‹(x, K) ∈ D› p(1) by blast
      moreover have content K * norm (f x) ≤ content K * g x
        by (meson K(1) K(2) content_pos_le mult_left_mono nle subsetD)
      then show ?thesis
        by simp
    qed
    then show norm (∑(x, k)∈D. content k *R f x) ≤ (∑(x, k)∈D. content k
```

```
*R g x)
      by (simp add: sum_norm_le split_def)
    show norm ((∑(x, k)∈𝒟. content k *R f x) − integral (cbox a b) f) < e/2
      using ‹γ fine 𝒟› γ p(1) by simp
    show |(∑(x, k)∈𝒟. content k *R g x) − integral (cbox a b) g| < e/2
      using ‹δ fine 𝒟› δ p(1) by simp
  qed
 qed
 show ?thesis
 proof (rule field_le_epsilon)
  fix e :: real
  assume e > 0
  then have e: e/2 > 0
   by auto
  let ?f = (λx. if x ∈ S then f x else 0)
  let ?g = (λx. if x ∈ S then g x else 0)
  have f: ?f integrable_on cbox a b and g: ?g integrable_on cbox a b for a b
   using int_f int_g integrable_altD by auto
  obtain Bf where 0 < Bf
   and Bf: ⋀a b. ball 0 Bf ⊆ cbox a b ⟹
    ∃z. (?f has_integral z) (cbox a b) ∧ norm (z − integral S f) < e/2
   using integrable_integral [OF int_f,unfolded has_integral'[of f]] e that by
blast
  obtain Bg where 0 < Bg
   and Bg: ⋀a b. ball 0 Bg ⊆ cbox a b ⟹
    ∃z. (?g has_integral z) (cbox a b) ∧ norm (z − integral S g) < e/2
   using integrable_integral [OF int_g,unfolded has_integral'[of g]] e that by
blast
  obtain a b::'n where ab: ball 0 Bf ∪ ball 0 Bg ⊆ cbox a b
   using ball_max_Un by (metis bounded_ball bounded_subset_cbox_symmetric)
  have ball 0 Bf ⊆ cbox a b
   using ab by auto
  with Bf obtain z where int_fz: (?f has_integral z) (cbox a b) and z: norm
(z − integral S f) < e/2
   by meson
  have ball 0 Bg ⊆ cbox a b
   using ab by auto
 with Bg obtain w where int_gw: (?g has_integral w) (cbox a b) and w: norm
(w − integral S g) < e/2
   by meson
  show norm (integral S f) ≤ integral S g + e
  proof (rule norm)
   show norm (integral (cbox a b) ?f) ≤ integral (cbox a b) ?g
    by (simp add: le_g lem[OF f g, of a b])
   show |integral (cbox a b) ?g − integral S g| < e/2
    using int_gw integral_unique w by auto
   show norm (integral (cbox a b) ?f − integral S f) < e/2
    using int_fz integral_unique z by blast
  qed
```

qed
qed

lemma *continuous_on_imp_absolutely_integrable_on*:
fixes f::*real* ⇒ 'a::*banach*
shows *continuous_on* {*a*..*b*} f ⟹
norm (*integral* {*a*..*b*} f) ≤ *integral* {*a*..*b*} (λ*x*. *norm* (f *x*))
by (*intro integral_norm_bound_integral integrable_continuous_real continuous_on_norm*) *auto*

lemma *integral_bound*:
fixes f::*real* ⇒ 'a::*banach*
assumes *a* ≤ *b*
assumes *continuous_on* {*a* .. *b*} f
assumes ⋀*t*. *t* ∈ {*a* .. *b*} ⟹ *norm* (f *t*) ≤ *B*
shows *norm* (*integral* {*a* .. *b*} f) ≤ *B* ∗ (*b* − *a*)
proof −
note *continuous_on_imp_absolutely_integrable_on*[*OF assms*(*2*)]
also have *integral* {*a*..*b*} (λ*x*. *norm* (f *x*)) ≤ *integral* {*a*..*b*} (λ_. *B*)
by (*rule integral_le*)
(*auto intro*!: *integrable_continuous_real continuous_intros assms*)
also have ... = *B* ∗ (*b* − *a*) **using** *assms* **by** *simp*
finally show *?thesis* **.**
qed

lemma *integral_norm_bound_integral_component*:
fixes f :: 'n::*euclidean_space* ⇒ 'a::*banach*
fixes g :: 'n ⇒ 'b::*euclidean_space*
assumes f: f *integrable_on S* **and** g: g *integrable_on S*
and *fg*: ⋀*x*. *x* ∈ *S* ⟹ *norm*(f *x*) ≤ (g *x*)•*k*
shows *norm* (*integral S* f) ≤ (*integral S* g)•*k*
proof −
have *norm* (*integral S* f) ≤ *integral S* ((λ*x*. *x* • *k*) ∘ g)
using *integral_norm_bound_integral*[*OF f integrable_linear*[*OF g*]]
by (*simp add*: *bounded_linear_inner_left fg*)
then show *?thesis*
unfolding *o_def integral_component_eq*[*OF g*] **.**
qed

lemma *has_integral_norm_bound_integral_component*:
fixes f :: 'n::*euclidean_space* ⇒ 'a::*banach*
fixes g :: 'n ⇒ 'b::*euclidean_space*
assumes (f *has_integral i*) *S* **and** (g *has_integral j*) *S*
and ⋀*x*. *x* ∈ *S* ⟹ *norm* (f *x*) ≤ (g *x*)•*k*
shows *norm i* ≤ *j*•*k*
by (*metis assms has_integral_integrable integral_norm_bound_integral_component integral_unique*)

```
lemma uniformly_convergent_improper_integral:
  fixes f :: 'b ⇒ real ⇒ 'a :: {banach}
  assumes deriv: ⋀x. x ≥ a ⟹ (G has_field_derivative g x) (at x within {a..})
  assumes integrable: ⋀a' b x. x ∈ A ⟹ a' ≥ a ⟹ b ≥ a' ⟹ f x integrable_on
{a'..b}
  assumes G: convergent G
  assumes le: ⋀y x. y ∈ A ⟹ x ≥ a ⟹ norm (f y x) ≤ g x
  shows   uniformly_convergent_on A (λb x. integral {a..b} (f x))
proof (intro Cauchy_uniformly_convergent uniformly_Cauchy_onI', goal_cases)
  case (1 ε)
  from G have Cauchy G
    by (auto intro!: convergent_Cauchy)
  with 1 obtain M where M: dist (G (real m)) (G (real n)) < ε if m ≥ M n ≥
M for m n
    by (force simp: Cauchy_def)
  define M' where M' = max (nat ⌈a⌉) M

  show ?case
  proof (rule exI[of _ M'], safe, goal_cases)
    case (1 x m n)
    have M': M' ≥ a M' ≥ M unfolding M'_def by linarith+
    have int_g: (g has_integral (G (real n) − G (real m))) {real m..real n}
      using 1 M' by (intro fundamental_theorem_of_calculus)
                    (auto simp: has_real_derivative_iff_has_vector_derivative
[symmetric]
                          intro!: DERIV_subset[OF deriv])
    have int_f: f x integrable_on {a'..real n} if a' ≥ a for a'
      using that 1 by (cases a' ≤ real n) (auto intro: integrable)

    have dist (integral {a..real m} (f x)) (integral {a..real n} (f x)) =
          norm (integral {a..real n} (f x) − integral {a..real m} (f x))
      by (simp add: dist_norm norm_minus_commute)
    also have integral {a..real m} (f x) + integral {real m..real n} (f x) =
                integral {a..real n} (f x)
      using M' and 1 by (intro integral_combine int_f) auto
    hence integral {a..real n} (f x) − integral {a..real m} (f x) =
            integral {real m..real n} (f x)
      by (simp add: algebra_simps)
    also have norm ... ≤ integral {real m..real n} g
      using le 1 M' int_f int_g by (intro integral_norm_bound_integral) auto
    also from int_g have integral {real m..real n} g = G (real n) − G (real m)
      by (simp add: has_integral_iff)
    also have ... ≤ dist (G m) (G n)
      by (simp add: dist_norm)
    also from 1 and M' have ... < ε
      by (intro M) auto
    finally show ?case .
  qed
qed
```

lemma *uniformly_convergent_improper_integral'*:
fixes $f :: 'b \Rightarrow real \Rightarrow 'a :: \{banach, real\_normed\_algebra\}$
assumes *deriv*: $\bigwedge x.\ x \geq a \Longrightarrow (G\ has\_field\_derivative\ g\ x)\ (at\ x\ within\ \{a..\})$
assumes *integrable*: $\bigwedge a'\ b\ x.\ x \in A \Longrightarrow a' \geq a \Longrightarrow b \geq a' \Longrightarrow f\ x\ integrable\_on\ \{a'..b\}$
assumes *G*: *convergent G*
assumes *le*: *eventually* $(\lambda x.\ \forall y{\in}A.\ norm\ (f\ y\ x) \leq g\ x)\ at\_top$
shows *uniformly_convergent_on A* $(\lambda b\ x.\ integral\ \{a..b\}\ (f\ x))$
proof −
from *le* **obtain** a'' **where** *le*: $\bigwedge y\ x.\ y \in A \Longrightarrow x \geq a'' \Longrightarrow norm\ (f\ y\ x) \leq g\ x$
by (*auto simp*: *eventually_at_top_linorder*)
define a' **where** $a' = max\ a\ a''$

have *uniformly_convergent_on A* $(\lambda b\ x.\ integral\ \{a'..real\ b\}\ (f\ x))$
proof (*rule uniformly_convergent_improper_integral*)
fix *t* **assume** *t*: $t \geq a'$
hence $(G\ has\_field\_derivative\ g\ t)\ (at\ t\ within\ \{a..\})$
by (*intro deriv*) (*auto simp*: a'_def)
moreover have $\{a'..\} \subseteq \{a..\}$ **unfolding** a'_def **by** *auto*
ultimately show $(G\ has\_field\_derivative\ g\ t)\ (at\ t\ within\ \{a'..\})$
by (*rule DERIV_subset*)
qed (*insert le*, *auto simp*: a'_def *intro*: *integrable G*)
hence *uniformly_convergent_on A* $(\lambda b\ x.\ integral\ \{a..a'\}\ (f\ x) + integral\ \{a'..real\ b\}\ (f\ x))$
(**is** *?P*) **by** (*intro uniformly_convergent_add*) *auto*
also have *eventually* $(\lambda x.\ \forall y{\in}A.\ integral\ \{a..a'\}\ (f\ y) + integral\ \{a'..x\}\ (f\ y) = integral\ \{a..x\}\ (f\ y))\ sequentially$
by (*intro eventually_mono* [*OF eventually_ge_at_top*[*of nat* $\lceil a' \rceil$]] *ballI integral_combine*)
(*auto simp*: a'_def *intro*: *integrable*)
hence *?P* $\longleftrightarrow$ *?thesis*
by (*intro uniformly_convergent_cong*) *simp_all*
finally show *?thesis* **.**
qed

6.14.42 differentiation under the integral sign

lemma *integral_continuous_on_param*:
fixes $f :: 'a :: topological\_space \Rightarrow 'b :: euclidean\_space \Rightarrow 'c :: banach$
assumes *cont_fx*: *continuous_on* $(U \times cbox\ a\ b)\ (\lambda(x,\ t).\ f\ x\ t)$
shows *continuous_on U* $(\lambda x.\ integral\ (cbox\ a\ b)\ (f\ x))$
proof *cases*
assume $content\ (cbox\ a\ b) \neq 0$
then have *ne*: $cbox\ a\ b \neq \{\}$ **by** *auto*

note [*continuous_intros*] =
continuous_on_compose2[*OF cont_fx*, **where** $f{=}\lambda y.\ Pair\ x\ y$ **for** *x*,
unfolded split_beta fst_conv snd_conv]

```
  show ?thesis
    unfolding continuous_on_def
  proof (intro strip tendstoI)
    fix e'::real and x
    assume e' > 0
    define e where e = e' / (content (cbox a b) + 1)
     have e > 0 using ‹e' > 0› by (auto simp: e_def intro!: divide_pos_pos
add_nonneg_pos)
    assume x ∈ U
    from continuous_on_prod_compactE[OF cont_fx compact_cbox ‹x ∈ U› ‹0
< e›]
    obtain X0 where X0: x ∈ X0 open X0
      and fx_bound: ⋀y t. y ∈ X0 ∩ U ⟹ t ∈ cbox a b ⟹ norm (f y t − f x t)
≤ e
      unfolding split_beta fst_conv snd_conv dist_norm
      by metis
    have ∀F y in at x within U. y ∈ X0 ∩ U
      using X0(1) X0(2) eventually_at_topological by auto
   then show ∀F y in at x within U. dist (integral (cbox a b) (f y)) (integral (cbox
a b) (f x)) < e'
    proof eventually_elim
      case (elim y)
      have dist (integral (cbox a b) (f y)) (integral (cbox a b) (f x)) =
        norm (integral (cbox a b) (λt. f y t − f x t))
        using elim ‹x ∈ U›
        unfolding dist_norm
        by (subst integral_diff)
           (auto intro!: integrable_continuous continuous_intros)
      also have ... ≤ e * content (cbox a b)
        using elim ‹x ∈ U›
        by (intro integrable_bound)
           (auto intro!: fx_bound ‹x ∈ U › less_imp_le[OF ‹0 < e›]
              integrable_continuous continuous_intros)
      also have ... < e'
        using ‹0 < e'› ‹e > 0›
        by (auto simp: e_def field_split_simps)
      finally show dist (integral (cbox a b) (f y)) (integral (cbox a b) (f x)) < e' .
    qed
  qed
qed (auto intro!: continuous_on_const)

lemma leibniz_rule:
  fixes f::'a::banach ⇒ 'b::euclidean_space ⇒ 'c::banach
  assumes fx: ⋀x t. x ∈ U ⟹ t ∈ cbox a b ⟹
    ((λx. f x t) has_derivative blinfun_apply (fx x t)) (at x within U)
  assumes integrable_f2: ⋀x. x ∈ U ⟹ f x integrable_on cbox a b
  assumes cont_fx: continuous_on (U × (cbox a b)) (λ(x, t). fx x t)
  assumes [intro]: x0 ∈ U
```

```
  assumes convex U
  shows
    ((λx. integral (cbox a b) (f x)) has_derivative integral (cbox a b) (fx x0)) (at x0
within U)
    (is (?F has_derivative ?dF) _)
proof cases
  assume content (cbox a b) ≠ 0
  then have ne: cbox a b ≠ {} by auto
  note [continuous_intros] =
    continuous_on_compose2[OF cont_fx, where f=λy. Pair x y for x,
      unfolded split_beta fst_conv snd_conv]
  show ?thesis
  proof (intro has_derivativeI bounded_linear_scaleR_left tendstoI, fold norm_conv_dist)
    have cont_f1: ⋀t. t ∈ cbox a b ⟹ continuous_on U (λx. f x t)
    by (auto simp: continuous_on_eq_continuous_within intro!: has_derivative_continuous
fx)
    note [continuous_intros] = continuous_on_compose2[OF cont_f1]
    fix e'::real
    assume e' > 0
    define e where e = e' / (content (cbox a b) + 1)
     have e > 0 using ‹e' > 0› by (auto simp: e_def intro!: divide_pos_pos
add_nonneg_pos)
    from continuous_on_prod_compactE[OF cont_fx compact_cbox ‹x0 ∈ U› ‹e
> 0›]
    obtain X0 where X0: x0 ∈ X0 open X0
     and fx_bound: ⋀x t. x ∈ X0 ∩ U ⟹ t ∈ cbox a b ⟹ norm (fx x t − fx x0
t) ≤ e
      unfolding split_beta fst_conv snd_conv
      by (metis dist_norm)

    note eventually_closed_segment[OF ‹open X0› ‹x0 ∈ X0›, of U]
    moreover
    have ∀F x in at x0 within U. x ∈ X0
      using ‹open X0› ‹x0 ∈ X0› eventually_at_topological by blast
    moreover have ∀F x in at x0 within U. x ≠ x0
      by (auto simp: eventually_at_filter)
    moreover have ∀F x in at x0 within U. x ∈ U
      by (auto simp: eventually_at_filter)
    ultimately
   show ∀F x in at x0 within U. norm ((?F x − ?F x0 − ?dF (x − x0)) /R norm
(x − x0)) < e'
    proof eventually_elim
      case (elim x)
      from elim have 0 < norm (x − x0) by simp
      have closed_segment x0 x ⊆ U
        by (simp add: assms closed_segment_subset elim(4))
      from elim have [intro]: x ∈ U by auto
      have ?F x − ?F x0 − ?dF (x − x0) =
        integral (cbox a b) (λy. f x y − f x0 y − fx x0 y (x − x0))
```

```
      (is _ = ?id)
      using ‹x ≠ x0›
      by (subst blinfun_apply_integral integral_diff,
          auto intro!: integrable_diff integrable_f2 continuous_intros
            intro: integrable_continuous)+
    also
    {
      fix t assume t: t ∈ (cbox a b)
      then have deriv:
        ((λx. f x t) has_derivative (fx y t)) (at y within X0 ∩ U)
        if y ∈ X0 ∩ U for y
        using fx has_derivative_subset that by fastforce
      have seg: ⋀t. t ∈ {0..1} ⟹ x0 + t *R (x − x0) ∈ X0 ∩ U
        using ‹closed_segment x0 x ⊆ U›
          ‹closed_segment x0 x ⊆ X0›
        by (force simp: closed_segment_def algebra_simps)
      have ⋀x. x ∈ X0 ∩ U ⟹ onorm (blinfun_apply (fx x t) − (fx x0 t)) ≤ e
        using fx_bound t
      by (auto simp add: norm_blinfun_def fun_diff_def blinfun.bilinear_simps[symmetric])
      from differentiable_bound_linearization[OF seg deriv this] X0
      have norm (f x t − f x0 t − fx x0 t (x − x0)) ≤ e * norm (x − x0)
        by (auto simp add: ac_simps)
    }
    then have norm ?id ≤ integral (cbox a b) (λ_. e * norm (x − x0))
      by (intro integral_norm_bound_integral)
        (auto intro!: continuous_intros integrable_diff integrable_f2
          intro: integrable_continuous)
    also have ... = content (cbox a b) * e * norm (x − x0)
      by simp
    also have ... < e' * norm (x − x0)
    proof (intro mult_strict_right_mono[OF _ ‹0 < norm (x − x0)›])
      show content (cbox a b) * e < e'
        using ‹e' > 0› by (simp add: divide_simps e_def not_less)
    qed
    finally have norm (?F x − ?F x0 − ?dF (x − x0)) < e' * norm (x − x0) .
    then show ?case
      by (auto simp: divide_simps)
  qed
 qed (rule blinfun.bounded_linear_right)
qed (auto intro!: derivative_eq_intros simp: blinfun.bilinear_simps)

lemma has_vector_derivative_eq_has_derivative_blinfun:
 (f has_vector_derivative f') (at x within U) ⟷
   (f has_derivative blinfun_scaleR_left f') (at x within U)
 by (simp add: has_vector_derivative_def)

lemma leibniz_rule_vector_derivative:
 fixes f::real ⇒ 'b::euclidean_space ⇒ 'c::banach
 assumes fx: ⋀x t. x ∈ U ⟹ t ∈ cbox a b ⟹
```

```
    ((λx. f x t) has_vector_derivative (fx x t)) (at x within U)
  assumes integrable_f2: ⋀x. x ∈ U ⟹ (f x) integrable_on cbox a b
  assumes cont_fx: continuous_on (U × cbox a b) (λ(x, t). fx x t)
  assumes U: x0 ∈ U convex U
  shows ((λx. integral (cbox a b) (f x)) has_vector_derivative integral (cbox a b)
(fx x0))
    (at x0 within U)
proof −
  note [continuous_intros] =
    continuous_on_compose2[OF cont_fx, where f=λy. Pair x y for x,
      unfolded split_beta fst_conv snd_conv]
  show ?thesis
    unfolding has_vector_derivative_eq_has_derivative_blinfun
  proof (rule has_derivative_eq_rhs [OF leibniz_rule[OF _ integrable_f2 _ U]])
    show continuous_on (U × cbox a b) (λ(x, t). blinfun_scaleR_left (fx x t))
      using cont_fx by (auto simp: split_beta intro!: continuous_intros)
    show blinfun_apply (integral (cbox a b) (λt. blinfun_scaleR_left (fx x0 t))) =
        blinfun_apply (blinfun_scaleR_left (integral (cbox a b) (fx x0)))
    by (subst integral_linear[symmetric])
      (auto simp: has_vector_derivative_def o_def
        intro!: integrable_continuous U continuous_intros bounded_linear_intros)
    qed (use fx in ‹auto simp: has_vector_derivative_def›)
qed

lemma has_field_derivative_eq_has_derivative_blinfun:
 (f has_field_derivative f') (at x within U) ⟷ (f has_derivative blinfun_mult_right
f') (at x within U)
  by (simp add: has_field_derivative_def)

lemma leibniz_rule_field_derivative:
  fixes f::'a::{real_normed_field, banach} ⇒ 'b::euclidean_space ⇒ 'a
  assumes fx: ⋀x t. x ∈ U ⟹ t ∈ cbox a b ⟹ ((λx. f x t) has_field_derivative
fx x t) (at x within U)
  assumes integrable_f2: ⋀x. x ∈ U ⟹ (f x) integrable_on cbox a b
  assumes cont_fx: continuous_on (U × (cbox a b)) (λ(x, t). fx x t)
  assumes U: x0 ∈ U convex U
  shows ((λx. integral (cbox a b) (f x)) has_field_derivative integral (cbox a b) (fx
x0)) (at x0 within U)
proof −
  note [continuous_intros] =
    continuous_on_compose2[OF cont_fx, where f=λy. Pair x y for x,
      unfolded split_beta fst_conv snd_conv]
  have ∗: blinfun_mult_right (integral (cbox a b) (fx x0)) =
    integral (cbox a b) (λt. blinfun_mult_right (fx x0 t))
    by (subst integral_linear[symmetric])
      (auto simp: has_vector_derivative_def o_def
        intro!: integrable_continuous U continuous_intros bounded_linear_intros)
  show ?thesis
    unfolding has_field_derivative_eq_has_derivative_blinfun
```

proof (*rule has_derivative_eq_rhs* [*OF leibniz_rule*[*OF* _ *integrable_f2* _ *U*, **where** *fx*=λ*x t. blinfun_mult_right* (*fx x t*)]])
show *continuous_on* (*U* × *cbox a b*) (λ(*x*, *t*). *blinfun_mult_right* (*fx x t*))
using *cont_fx* **by** (*auto simp*: *split_beta intro*!: *continuous_intros*)
show *blinfun_apply* (*integral* (*cbox a b*) (λ*t. blinfun_mult_right* (*fx x0 t*))) =
blinfun_apply (*blinfun_mult_right* (*integral* (*cbox a b*) (*fx x0*)))
by (*subst integral_linear*[*symmetric*])
(*auto simp*: *has_vector_derivative_def o_def*
intro!: *integrable_continuous U continuous_intros bounded_linear_intros*)
qed (*use fx* **in** ‹*auto simp*: *has_field_derivative_def*›)
qed

lemma *leibniz_rule_field_differentiable*:
fixes *f*::′*a*::{*real_normed_field*, *banach*} ⇒ ′*b*::*euclidean_space* ⇒ ′*a*
assumes ⋀*x t. x* ∈ *U* ⟹ *t* ∈ *cbox a b* ⟹ ((λ*x. f x t*) *has_field_derivative fx x t*) (*at x within U*)
assumes ⋀*x. x* ∈ *U* ⟹ (*f x*) *integrable_on cbox a b*
assumes *continuous_on* (*U* × (*cbox a b*)) (λ(*x*, *t*). *fx x t*)
assumes *x0* ∈ *U convex U*
shows (λ*x. integral* (*cbox a b*) (*f x*)) *field_differentiable at x0 within U*
using *leibniz_rule_field_derivative*[*OF assms*] **by** (*auto simp*: *field_differentiable_def*)

6.14.43 Exchange uniform limit and integral

lemma *uniform_limit_integral_cbox*:
fixes *f*::′*a* ⇒ ′*b*::*euclidean_space* ⇒ ′*c*::*banach*
assumes *u*: *uniform_limit* (*cbox a b*) *f g F*
assumes *c*: ⋀*n. continuous_on* (*cbox a b*) (*f n*)
assumes [*simp*]: *F* ≠ *bot*
obtains *I J* **where**
⋀*n*. (*f n has_integral I n*) (*cbox a b*)
(*g has_integral J*) (*cbox a b*)
(*I* ⟶ *J*) *F*
proof −
have *fi*[*simp*]: *f n integrable_on* (*cbox a b*) **for** *n*
by (*auto intro*!: *integrable_continuous assms*)
then obtain *I* **where** *I*: ⋀*n*. (*f n has_integral I n*) (*cbox a b*)
unfolding *integrable_on_def* **by** *metis*

moreover
have *gi*[*simp*]: *g integrable_on* (*cbox a b*)
by (*auto intro*!: *integrable_continuous uniform_limit_theorem*[*OF* _ *u*] *eventuallyI c*)
then obtain *J* **where** *J*: (*g has_integral J*) (*cbox a b*)
by *blast*

moreover
have (*I* ⟶ *J*) *F*
proof *cases*

```
    assume content (cbox a b) = 0
    hence I = (λ_. 0) J = 0
      by (auto intro!: has_integral_unique I J)
    thus ?thesis by simp
  next
    assume content_nonzero: content (cbox a b) ≠ 0
    show ?thesis
    proof (rule tendstoI)
      fix e::real
      assume e > 0
      define e' where e' = e/2
      with ‹e > 0› have e' > 0 by simp
     then have ∀ F n in F. ∀ x∈cbox a b. norm (f n x − g x) < e' / content (cbox
a b)
          using u content_nonzero by (auto simp: uniform_limit_iff dist_norm
zero_less_measure_iff)
      then show ∀ F n in F. dist (I n) J < e
      proof eventually_elim
        case (elim n)
        have I n = integral (cbox a b) (f n)
            J = integral (cbox a b) g
          using I[of n] J by (simp_all add: integral_unique)
        then have dist (I n) J = norm (integral (cbox a b) (λx. f n x − g x))
          by (simp add: integral_diff dist_norm)
        also have ... ≤ integral (cbox a b) (λx. (e' / content (cbox a b)))
          using elim
          by (intro integral_norm_bound_integral) (auto intro!: integrable_diff)
        also have ... < e
          using ‹0 < e› by (simp add: e'_def)
        finally show ?case .
      qed
    qed
  qed
  ultimately show ?thesis ..
qed

lemma uniform_limit_integral:
  fixes f::'a ⇒ 'b::ordered_euclidean_space ⇒ 'c::banach
  assumes u: uniform_limit {a..b} f g F
  assumes c: ⋀n. continuous_on {a..b} (f n)
  assumes [simp]: F ≠ bot
  obtains I J where
    ⋀n. (f n has_integral I n) {a..b}
    (g has_integral J) {a..b}
    (I ⟶ J) F
  by (metis interval_cbox assms uniform_limit_integral_cbox)
```

6.14.44 Integration by parts

lemma *integration_by_parts_interior_strong*:
fixes *prod* :: _ ⇒ _ ⇒ 'b :: *banach*
assumes *bilinear*: *bounded_bilinear* (*prod*)
assumes *s*: *finite s* **and** *le*: $a \leq b$
assumes *cont* [*continuous_intros*]: *continuous_on* {*a..b*} *f continuous_on* {*a..b*} *g*
assumes *deriv*: $\bigwedge x.\ x \in \{a<..<b\} - s \Longrightarrow (f \text{ has\_vector\_derivative } f'\ x)\ (\text{at } x)$
$\bigwedge x.\ x \in \{a<..<b\} - s \Longrightarrow (g \text{ has\_vector\_derivative } g'\ x)\ (\text{at } x)$
assumes *int*: $((\lambda x.\ prod\ (f\ x)\ (g'\ x)) \text{ has\_integral}$
$(prod\ (f\ b)\ (g\ b) - prod\ (f\ a)\ (g\ a) - y))\ \{a..b\}$
shows $((\lambda x.\ prod\ (f'\ x)\ (g\ x)) \text{ has\_integral } y)\ \{a..b\}$
proof −
interpret *bounded_bilinear prod* **by** *fact*
have $((\lambda x.\ prod\ (f\ x)\ (g'\ x) + prod\ (f'\ x)\ (g\ x)) \text{ has\_integral}$
$(prod\ (f\ b)\ (g\ b) - prod\ (f\ a)\ (g\ a)))\ \{a..b\}$
using *deriv* **by** (*intro fundamental_theorem_of_calculus_interior_strong*[*OF s le*])
(*auto intro*!: *continuous_intros continuous_on has_vector_derivative*)
from *has_integral_diff*[*OF this int*] **show** *?thesis* **by** (*simp add*: *algebra_simps*)
qed

lemma *integration_by_parts_interior*:
fixes *prod* :: _ ⇒ _ ⇒ 'b :: *banach*
assumes *bounded_bilinear* (*prod*) $a \leq b$
continuous_on {*a..b*} *f continuous_on* {*a..b*} *g*
assumes $\bigwedge x.\ x \in \{a<..<b\} \Longrightarrow (f \text{ has\_vector\_derivative } f'\ x)\ (\text{at } x)$
$\bigwedge x.\ x \in \{a<..<b\} \Longrightarrow (g \text{ has\_vector\_derivative } g'\ x)\ (\text{at } x)$
assumes $((\lambda x.\ prod\ (f\ x)\ (g'\ x)) \text{ has\_integral } (prod\ (f\ b)\ (g\ b) - prod\ (f\ a)\ (g\ a) - y))\ \{a..b\}$
shows $((\lambda x.\ prod\ (f'\ x)\ (g\ x)) \text{ has\_integral } y)\ \{a..b\}$
by (*rule integration_by_parts_interior_strong*[*of* _ {} _ _ *f g f' g'*]) (*use assms* **in** *simp_all*)

lemma *integration_by_parts*:
fixes *prod* :: _ ⇒ _ ⇒ 'b :: *banach*
assumes *bounded_bilinear* (*prod*) $a \leq b$
continuous_on {*a..b*} *f continuous_on* {*a..b*} *g*
assumes $\bigwedge x.\ x \in \{a..b\} \Longrightarrow (f \text{ has\_vector\_derivative } f'\ x)\ (\text{at } x)$
$\bigwedge x.\ x \in \{a..b\} \Longrightarrow (g \text{ has\_vector\_derivative } g'\ x)\ (\text{at } x)$
assumes $((\lambda x.\ prod\ (f\ x)\ (g'\ x)) \text{ has\_integral } (prod\ (f\ b)\ (g\ b) - prod\ (f\ a)\ (g\ a) - y))\ \{a..b\}$
shows $((\lambda x.\ prod\ (f'\ x)\ (g\ x)) \text{ has\_integral } y)\ \{a..b\}$
by (*rule integration_by_parts_interior*[*of* _ _ _ *f g f' g'*]) (*use assms* **in** *simp_all*)

lemma *integrable_by_parts_interior_strong*:
fixes *prod* :: _ ⇒ _ ⇒ 'b :: *banach*
assumes *bilinear*: *bounded_bilinear* (*prod*)

assumes *s*: *finite s* **and** *le*: $a \leq b$
assumes *cont* [*continuous_intros*]: *continuous_on* $\{a..b\}$ *f continuous_on* $\{a..b\}$ *g*
assumes *deriv*: $\bigwedge x.\ x \in \{a<..<b\} - s \Longrightarrow (f \text{ has\_vector\_derivative } f'\ x)\ (\text{at } x)$
$\bigwedge x.\ x \in \{a<..<b\} - s \Longrightarrow (g \text{ has\_vector\_derivative } g'\ x)\ (\text{at } x)$
assumes *int*: $(\lambda x.\ \text{prod}\ (f\ x)\ (g'\ x))$ *integrable_on* $\{a..b\}$
shows $(\lambda x.\ \text{prod}\ (f'\ x)\ (g\ x))$ *integrable_on* $\{a..b\}$
proof −
from *int* **obtain** *I* **where** $((\lambda x.\ \text{prod}\ (f\ x)\ (g'\ x))$ *has_integral I*$)\ \{a..b\}$
unfolding *integrable_on_def* **by** *blast*
hence $((\lambda x.\ \text{prod}\ (f\ x)\ (g'\ x))$ *has_integral* $(\text{prod}\ (f\ b)\ (g\ b) - \text{prod}\ (f\ a)\ (g\ a) -$
$(\text{prod}\ (f\ b)\ (g\ b) - \text{prod}\ (f\ a)\ (g\ a) - I)))\ \{a..b\}$ **by** *simp*
from *integration_by_parts_interior_strong*[*OF assms*(1−7) *this*]
show *?thesis* **unfolding** *integrable_on_def* **by** *blast*
qed

lemma *integrable_by_parts_interior*:
fixes *prod* :: _ ⇒ _ ⇒ *'b* :: *banach*
assumes *bounded_bilinear* (*prod*) $a \leq b$
continuous_on $\{a..b\}$ *f continuous_on* $\{a..b\}$ *g*
assumes $\bigwedge x.\ x \in \{a<..<b\} \Longrightarrow (f \text{ has\_vector\_derivative } f'\ x)\ (\text{at } x)$
$\bigwedge x.\ x \in \{a<..<b\} \Longrightarrow (g \text{ has\_vector\_derivative } g'\ x)\ (\text{at } x)$
assumes $(\lambda x.\ \text{prod}\ (f\ x)\ (g'\ x))$ *integrable_on* $\{a..b\}$
shows $(\lambda x.\ \text{prod}\ (f'\ x)\ (g\ x))$ *integrable_on* $\{a..b\}$
by (*rule integrable_by_parts_interior_strong*[*of* _ {} _ _ *f g f' g'*]) (*use assms* **in** *simp_all*)

lemma *integrable_by_parts*:
fixes *prod* :: _ ⇒ _ ⇒ *'b* :: *banach*
assumes *bounded_bilinear* (*prod*) $a \leq b$
continuous_on $\{a..b\}$ *f continuous_on* $\{a..b\}$ *g*
assumes $\bigwedge x.\ x \in \{a..b\} \Longrightarrow (f \text{ has\_vector\_derivative } f'\ x)\ (\text{at } x)$
$\bigwedge x.\ x \in \{a..b\} \Longrightarrow (g \text{ has\_vector\_derivative } g'\ x)\ (\text{at } x)$
assumes $(\lambda x.\ \text{prod}\ (f\ x)\ (g'\ x))$ *integrable_on* $\{a..b\}$
shows $(\lambda x.\ \text{prod}\ (f'\ x)\ (g\ x))$ *integrable_on* $\{a..b\}$
by (*rule integrable_by_parts_interior_strong*[*of* _ {} _ _ *f g f' g'*]) (*use assms* **in** *simp_all*)

6.14.45 Integration by substitution

lemma *has_integral_substitution_general*:
fixes *f* :: *real* ⇒ *'a*::*euclidean_space* **and** *g* :: *real* ⇒ *real*
assumes *s*: *finite s* **and** *le*: $a \leq b$
and *subset*: *g* ‘ $\{a..b\} \subseteq \{c..d\}$
and *f* [*continuous_intros*]: *continuous_on* $\{c..d\}$ *f*
and *g* [*continuous_intros*]: *continuous_on* $\{a..b\}$ *g*
and *deriv* [*derivative_intros*]:
$\bigwedge x.\ x \in \{a..b\} - s \Longrightarrow (g \text{ has\_field\_derivative } g'\ x)\ (\text{at } x \text{ within } \{a..b\})$
shows $((\lambda x.\ g'\ x *_R f\ (g\ x))$ *has_integral* (*integral* $\{g\ a..g\ b\}\ f$ − *integral* $\{g$

```
b..g a} f)) {a..b}
proof −
  let ?F = λx. integral {c..g x} f
  have cont_int: continuous_on {a..b} ?F
   by (rule continuous_on_compose2[OF _ g subset] indefinite_integral_continuous_1
         f integrable_continuous_real)+
  have deriv: (((λx. integral {c..x} f) ∘ g) has_vector_derivative g′ x ∗R f (g x))
                (at x within {a..b}) if x ∈ {a..b} − s for x
  proof (rule has_vector_derivative_eq_rhs [OF vector_diff_chain_within refl])
    show (g has_vector_derivative g′ x) (at x within {a..b})
      using deriv has_real_derivative_iff_has_vector_derivative that by blast
    show ((λx. integral {c..x} f) has_vector_derivative f (g x))
         (at (g x) within g ‘ {a..b})
      using that le subset
     by (blast intro: has_vector_derivative_within_subset integral_has_vector_derivative
f)
  qed
  have deriv: (?F has_vector_derivative g′ x ∗R f (g x))
                (at x) if x ∈ {a..b} − (s ∪ {a,b}) for x
    using deriv[of x] that by (simp add: at_within_Icc_at o_def)
  have ((λx. g′ x ∗R f (g x)) has_integral (?F b − ?F a)) {a..b}
    using le cont_int s deriv cont_int
     by (intro fundamental_theorem_of_calculus_interior_strong[of s ∪ {a,b}])
simp_all
  also
  from subset have g x ∈ {c..d} if x ∈ {a..b} for x using that by blast
  from this[of a] this[of b] le have cd: c ≤ g a g b ≤ d c ≤ g b g a ≤ d by auto
  have integral {c..g b} f − integral {c..g a} f = integral {g a..g b} f − integral
{g b..g a} f
  proof cases
    assume g a ≤ g b
    note le = le this
    from cd have integral {c..g a} f + integral {g a..g b} f = integral {c..g b} f
     by (intro integral_combine integrable_continuous_real continuous_on_subset[OF
f] le) simp_all
    with le show ?thesis
      by (cases g a = g b) (simp_all add: algebra_simps)
  next
    assume less: ¬g a ≤ g b
    then have g a ≥ g b by simp
    note le = le this
    from cd have integral {c..g b} f + integral {g b..g a} f = integral {c..g a} f
     by (intro integral_combine integrable_continuous_real continuous_on_subset[OF
f] le) simp_all
    with less show ?thesis
      by (simp_all add: algebra_simps)
  qed
  finally show ?thesis .
qed
```

lemma *has_integral_substitution_strong*:
fixes $f :: real \Rightarrow 'a::euclidean\_space$ **and** $g :: real \Rightarrow real$
assumes *s*: *finite s* **and** *le*: $a \le b$ $g\ a \le g\ b$
and *subset*: $g\ `\ \{a..b\} \subseteq \{c..d\}$
and f [*continuous_intros*]: *continuous_on* $\{c..d\}$ f
and g [*continuous_intros*]: *continuous_on* $\{a..b\}$ g
and *deriv* [*derivative_intros*]:
$\bigwedge x.\ x \in \{a..b\} - s \Longrightarrow (g\ has\_field\_derivative\ g'\ x)\ (at\ x\ within\ \{a..b\})$
shows $((\lambda x.\ g'\ x *_R f\ (g\ x))\ has\_integral\ (integral\ \{g\ a..g\ b\}\ f))\ \{a..b\}$
using *has_integral_substitution_general*[*OF s le*(*1*) *subset f g deriv*] *le*(*2*)
by (*cases g a = g b*) *auto*

lemma *has_integral_substitution*:
fixes $f :: real \Rightarrow 'a::euclidean\_space$ **and** $g :: real \Rightarrow real$
assumes $a \le b$ $g\ a \le g\ b$ $g\ `\ \{a..b\} \subseteq \{c..d\}$
and *continuous_on* $\{c..d\}$ f
and $\bigwedge x.\ x \in \{a..b\} \Longrightarrow (g\ has\_field\_derivative\ g'\ x)\ (at\ x\ within\ \{a..b\})$
shows $((\lambda x.\ g'\ x *_R f\ (g\ x))\ has\_integral\ (integral\ \{g\ a..g\ b\}\ f))\ \{a..b\}$
by (*intro has_integral_substitution_strong*[*of* {} *a b g c d*] *assms*)
(*auto intro*: *DERIV_continuous_on assms*)

lemma *integral_shift*:
fixes $f :: real \Rightarrow 'a::euclidean\_space$
assumes *cont*: *continuous_on* $\{a + c..b + c\}$ f
shows $integral\ \{a..b\}\ (f \circ (\lambda x.\ x + c)) = integral\ \{a + c..b + c\}\ f$
proof (*cases* $a \le b$)
case *True*
have $((\lambda x.\ 1 *_R f\ (x + c))\ has\_integral\ integral\ \{a+c..b+c\}\ f)\ \{a..b\}$
using *True cont*
by (*intro has_integral_substitution*[**where** $c = a + c$ **and** $d = b + c$])
(*auto intro*!: *derivative_eq_intros*)
thus *?thesis* **by** (*simp add*: *has_integral_iff o_def*)
qed *auto*

6.14.46 Compute a double integral using iterated integrals and switching the order of integration

lemma *continuous_on_imp_integrable_on_Pair1*:
fixes $f :: \_ \Rightarrow 'b::banach$
assumes *con*: *continuous_on* (*cbox* (*a*,*c*) (*b*,*d*)) f **and** x: $x \in cbox\ a\ b$
shows $(\lambda y.\ f\ (x, y))\ integrable\_on\ (cbox\ c\ d)$
proof −
have $f \circ (\lambda y.\ (x, y))\ integrable\_on\ (cbox\ c\ d)$
proof (*intro integrable_continuous continuous_on_compose* [*OF* _ *continuous_on_subset* [*OF con*]])
show *continuous_on* (*cbox c d*) (*Pair x*)
by (*simp add*: *continuous_on_Pair*)
show *Pair x* ‘ *cbox c d* ⊆ *cbox* (*a*,*c*) (*b*,*d*)

```
    using x by blast
  qed
  then show ?thesis
    by (simp add: o_def)
qed

lemma integral_integrable_2dim:
  fixes f :: ('a::euclidean_space * 'b::euclidean_space) ⇒ 'c::banach
  assumes continuous_on (cbox (a,c) (b,d)) f
    shows (λx. integral (cbox c d) (λy. f (x,y))) integrable_on cbox a b
proof (cases content(cbox c d) = 0)
case True
  then show ?thesis
    by (simp add: True integrable_const)
next
  case False
  have uc: uniformly_continuous_on (cbox (a,c) (b,d)) f
    by (simp add: assms compact_cbox compact_uniformly_continuous)
  { fix x::'a and e::real
    assume x: x ∈ cbox a b and e: 0 < e
    then have e2_gt: 0 < e/2 / content (cbox c d) and e2_less: e/2 / content
(cbox c d) * content (cbox c d) < e
      by (auto simp: False content_lt_nz e)
    then obtain dd
    where dd: ⋀x x'. ⟦x∈cbox (a, c) (b, d); x'∈cbox (a, c) (b, d); norm (x' − x)
< dd⟧
                    ⟹ norm (f x' − f x) ≤ e/(2 * content (cbox c d))  dd>0
      using uc [unfolded uniformly_continuous_on_def, THEN spec, of e/(2 *
content (cbox c d))]
      by (auto simp: dist_norm intro: less_imp_le)
   have ∃ delta>0. ∀ x'∈cbox a b. norm (x' − x) < delta ⟶ norm (integral (cbox
c d) (λu. f (x', u) − f (x, u))) < e
      using dd e2_gt assms x
      apply (rule_tac x=dd in exI)
      apply clarify
      apply (rule le_less_trans [OF integrable_bound e2_less])
     apply (auto intro: integrable_diff continuous_on_imp_integrable_on_Pair1)
      done
  } note * = this
  show ?thesis
  proof (rule integrable_continuous)
    show continuous_on (cbox a b) (λx. integral (cbox c d) (λy. f (x, y)))
      by (simp add: * continuous_on_iff dist_norm integral_diff [symmetric]
continuous_on_imp_integrable_on_Pair1 [OF assms])
  qed
qed

lemma integral_split:
  fixes f :: 'a::euclidean_space ⇒ 'b::{real_normed_vector,complete_space}
```

assumes *f*: *f integrable_on* (*cbox a b*)
and *k*: *k* ∈ *Basis*
shows *integral* (*cbox a b*) *f* =
integral (*cbox a b* ∩ {*x*. *x*•*k* ≤ *c*}) *f* +
integral (*cbox a b* ∩ {*x*. *x*•*k* ≥ *c*}) *f*
using *k f*
by (*auto simp*: *has_integral_integral intro*: *integral_unique* [*OF has_integral_split*])

lemma *integral_swap_operativeI*:
fixes *f* :: ('*a*::*euclidean_space* ∗ '*b*::*euclidean_space*) ⇒ '*c*::*banach*
assumes *f*: *continuous_on s f* **and** *e*: *0* < *e*
shows *operative conj True*
(λ*k*. ∀ *a b c d*.
cbox (*a*,*c*) (*b*,*d*) ⊆ *k* ∧ *cbox* (*a*,*c*) (*b*,*d*) ⊆ *s*
⟶ *norm*(*integral* (*cbox* (*a*,*c*) (*b*,*d*)) *f* −
integral (*cbox a b*) (λ*x*. *integral* (*cbox c d*) (λ*y*. *f*((*x*,*y*)))))
≤ *e* ∗ *content* (*cbox* (*a*,*c*) (*b*,*d*)))
proof (*standard*, *auto*)
fix *a*::'*a* **and** *c*::'*b* **and** *b*::'*a* **and** *d*::'*b* **and** *u*::'*a* **and** *v*::'*a* **and** *w*::'*b* **and** *z*::'*b*
assume ∗: *box* (*a*, *c*) (*b*, *d*) = {}
and *cb1*: *cbox* (*u*, *w*) (*v*, *z*) ⊆ *cbox* (*a*, *c*) (*b*, *d*)
and *cb2*: *cbox* (*u*, *w*) (*v*, *z*) ⊆ *s*
then have *c0*: *content* (*cbox* (*a*, *c*) (*b*, *d*)) = *0*
using ∗ **unfolding** *content_eq_0_interior* **by** *simp*
have *c0'*: *content* (*cbox* (*u*, *w*) (*v*, *z*)) = *0*
by (*fact content_0_subset* [*OF c0 cb1*])
show *norm* (*integral* (*cbox* (*u*,*w*) (*v*,*z*)) *f* − *integral* (*cbox u v*) (λ*x*. *integral* (*cbox w z*) (λ*y*. *f* (*x*, *y*))))
≤ *e* ∗ *content* (*cbox* (*u*,*w*) (*v*,*z*))
using *content_cbox_pair_eq0_D* [*OF c0'*]
by (*force simp add*: *c0'*)
next
fix *a*::'*a* **and** *c*::'*b* **and** *b*::'*a* **and** *d*::'*b*
and *M*::*real* **and** *i*::'*a* **and** *j*::'*b*
and *u*::'*a* **and** *v*::'*a* **and** *w*::'*b* **and** *z*::'*b*
assume *ij*: (*i*,*j*) ∈ *Basis*
and *n1*: ∀ *a'* *b'* *c'* *d'*.
cbox (*a'*,*c'*) (*b'*,*d'*) ⊆ *cbox* (*a*,*c*) (*b*,*d*) ∧
cbox (*a'*,*c'*) (*b'*,*d'*) ⊆ {*x*. *x* • (*i*,*j*) ≤ *M*} ∧ *cbox* (*a'*,*c'*) (*b'*,*d'*) ⊆ *s* ⟶
norm (*integral* (*cbox* (*a'*,*c'*) (*b'*,*d'*)) *f* − *integral* (*cbox a' b'*) (λ*x*. *integral* (*cbox c' d'*) (λ*y*. *f* (*x*,*y*))))
≤ *e* ∗ *content* (*cbox* (*a'*,*c'*) (*b'*,*d'*))
and *n2*: ∀ *a'* *b'* *c'* *d'*.
cbox (*a'*,*c'*) (*b'*,*d'*) ⊆ *cbox* (*a*,*c*) (*b*,*d*) ∧
cbox (*a'*,*c'*) (*b'*,*d'*) ⊆ {*x*. *M* ≤ *x* • (*i*,*j*)} ∧ *cbox* (*a'*,*c'*) (*b'*,*d'*) ⊆ *s* ⟶
norm (*integral* (*cbox* (*a'*,*c'*) (*b'*,*d'*)) *f* − *integral* (*cbox a' b'*) (λ*x*. *integral* (*cbox c' d'*) (λ*y*. *f* (*x*,*y*))))
≤ *e* ∗ *content* (*cbox* (*a'*,*c'*) (*b'*,*d'*))
and *subs*: *cbox* (*u*,*w*) (*v*,*z*) ⊆ *cbox* (*a*,*c*) (*b*,*d*) *cbox* (*u*,*w*) (*v*,*z*) ⊆ *s*

```
  have fcont: continuous_on (cbox (u, w) (v, z)) f
    using assms(1) continuous_on_subset  subs(2) by blast
  then have fint: f integrable_on cbox (u, w) (v, z)
    by (metis integrable_continuous)
  consider i ∈ Basis j=0 | j ∈ Basis i=0  using ij
    by (auto simp: Euclidean_Space.Basis_prod_def)
 then show norm (integral (cbox (u,w) (v,z)) f − integral (cbox u v) (λx. integral
(cbox w z) (λy. f (x,y))))
           ≤ e * content (cbox (u,w) (v,z)) (is ?normle)
  proof cases
    case 1
    then have e: e * content (cbox (u, w) (v, z)) =
                  e * (content (cbox u v ∩ {x. x • i ≤ M}) * content (cbox w z)) +
                  e * (content (cbox u v ∩ {x. M ≤ x • i}) * content (cbox w z))
      by (simp add: content_split [where c=M] content_Pair algebra_simps)
    have ∗: integral (cbox u v) (λx. integral (cbox w z) (λy. f (x, y))) =
                 integral (cbox u v ∩ {x. x • i ≤ M}) (λx. integral (cbox w z) (λy. f
(x, y))) +
                 integral (cbox u v ∩ {x. M ≤ x • i}) (λx. integral (cbox w z) (λy. f
(x, y)))
      using 1 f subs integral_integrable_2dim continuous_on_subset
      by (blast intro: integral_split)
    show ?normle
      apply (rule norm_diff2 [OF integral_split [where c=M, OF fint ij] ∗ e])
      using 1 subs
    apply (simp_all add: cbox_Pair_eq setcomp_dot1 [of λu. M≤u] setcomp_dot1
[of λu. u≤M] Sigma_Int_Paircomp1)
      apply (simp_all add: interval_split ij flip: cbox_Pair_eq content_Pair)
      apply (force simp flip: interval_split intro!: n1 [rule_format])
      apply (force simp flip: interval_split intro!: n2 [rule_format])
      done
  next
    case 2
    then have e: e * content (cbox (u, w) (v, z)) =
                  e * (content (cbox u v) * content (cbox w z ∩ {x. x • j ≤ M})) +
                  e * (content (cbox u v) * content (cbox w z ∩ {x. M ≤ x • j}))
      by (simp add: content_split [where c=M] content_Pair algebra_simps)
    have (λx. integral (cbox w z ∩ {x. x • j ≤ M}) (λy. f (x, y))) integrable_on
cbox u v
         (λx. integral (cbox w z ∩ {x. M ≤ x • j}) (λy. f (x, y))) integrable_on cbox
u v
      using 2 subs
      apply (simp_all add: interval_split)
       apply (rule integral_integrable_2dim [OF continuous_on_subset [OF f]];
auto simp: cbox_Pair_eq interval_split [symmetric])+
      done
    with 2 have ∗: integral (cbox u v) (λx. integral (cbox w z) (λy. f (x, y))) =
                   integral (cbox u v) (λx. integral (cbox w z ∩ {x. x • j ≤ M}) (λy. f
(x, y))) +
```

```
                integral (cbox u v) (λx. integral (cbox w z ∩ {x. M ≤ x • j}) (λy. f
(x, y)))
      by (simp add: integral_add [symmetric] integral_split [symmetric]
                    continuous_on_imp_integrable_on_Pair1 [OF fcont] cong:
integral_cong)
    show ?normle
      apply (rule norm_diff2 [OF integral_split [where c=M, OF fint ij] * e])
      using 2 subs
    apply (simp_all add: cbox_Pair_eq setcomp_dot2 [of λu. M≤u] setcomp_dot2
[of λu. u≤M] Sigma_Int_Paircomp2)
      apply (simp_all add: interval_split ij flip: cbox_Pair_eq content_Pair)
      apply (force simp flip: interval_split intro!: n1 [rule_format])
      apply (force simp flip: interval_split intro!: n2 [rule_format])
      done
  qed
qed

lemma integral_Pair_const:
    integral (cbox (a,c) (b,d)) (λx. k) =
     integral (cbox a b) (λx. integral (cbox c d) (λy. k))
  by (simp add: content_Pair)

lemma integral_prod_continuous:
  fixes f :: ('a::euclidean_space * 'b::euclidean_space) ⇒ 'c::banach
  assumes continuous_on (cbox (a, c) (b, d)) f (is continuous_on ?CBOX f)
    shows integral (cbox (a, c) (b, d)) f = integral (cbox a b) (λx. integral (cbox c
d) (λy. f (x, y)))
proof (cases content ?CBOX = 0)
  case True
  then show ?thesis
    by (auto simp: content_Pair)
next
  case False
  then have cbp: content ?CBOX > 0
    using content_lt_nz by blast
  have norm (integral ?CBOX f − integral (cbox a b) (λx. integral (cbox c d) (λy.
f (x,y)))) = 0
  proof (rule dense_eq0_I, simp)
    fix e :: real
    assume 0 < e
    with ‹content ?CBOX > 0› have 0 < e/content ?CBOX
      by simp
    have f_int_acbd: f integrable_on ?CBOX
      by (rule integrable_continuous [OF assms])
    { fix p
      assume p: p division_of ?CBOX
      then have finite p
        by blast
      define e' where e' = e/content ?CBOX
```

with ‹$0 < e$› ‹$0 < e$/*content ?CBOX*›
have $0 < e'$
by *simp*
interpret *operative conj True*
$\lambda k.\ \forall\, a'\ b'\ c'\ d'.$
$cbox\ (a',\ c')\ (b',\ d') \subseteq k \wedge cbox\ (a',\ c')\ (b',\ d') \subseteq$ *?CBOX*
$\longrightarrow$ *norm* (*integral* (*cbox* $(a',\ c')\ (b',\ d')$) $f -$
integral (*cbox* $a'\ b'$) ($\lambda x.$ *integral* (*cbox* $c'\ d'$) ($\lambda y.\ f\ ((x,\ y)))))$
$\leq e' *$ *content* (*cbox* $(a',\ c')\ (b',\ d')$)
using *assms* ‹$0 < e'$› **by** (*rule integral_swap_operativeI*)
have *norm* (*integral ?CBOX f* $-$ *integral* (*cbox a b*) ($\lambda x.$ *integral* (*cbox c d*) ($\lambda y.\ f\ (x,\ y)$)))
$\leq e' *$ *content ?CBOX*
if $\bigwedge t\ u\ v\ w\ z.\ t \in p \Longrightarrow cbox\ (u,\ w)\ (v,\ z) \subseteq t \Longrightarrow cbox\ (u,\ w)\ (v,\ z) \subseteq$ *?CBOX*
$\Longrightarrow$ *norm* (*integral* (*cbox* $(u,\ w)\ (v,\ z)$) $f -$
integral (*cbox u v*) ($\lambda x.$ *integral* (*cbox w z*) ($\lambda y.\ f\ (x,\ y)$)))
$\leq e' *$ *content* (*cbox* $(u,\ w)\ (v,\ z)$))
using *that division* [*of p* $(a,\ c)\ (b,\ d)$] *p* ‹*finite p*› **by** (*auto simp add*: *comm_monoid_set_F_and*)
with *False* **have** *norm* (*integral ?CBOX f* $-$ *integral* (*cbox a b*) ($\lambda x.$ *integral* (*cbox c d*) ($\lambda y.\ f\ (x,\ y)$)))
$\leq e$
if $\bigwedge t\ u\ v\ w\ z.\ t \in p \Longrightarrow cbox\ (u,\ w)\ (v,\ z) \subseteq t \Longrightarrow cbox\ (u,\ w)\ (v,\ z) \subseteq$ *?CBOX*
$\Longrightarrow$ *norm* (*integral* (*cbox* $(u,\ w)\ (v,\ z)$) $f -$
integral (*cbox u v*) ($\lambda x.$ *integral* (*cbox w z*) ($\lambda y.\ f\ (x,\ y)$)))
$\leq e *$ *content* (*cbox* $(u,\ w)\ (v,\ z)$) / *content ?CBOX*
using *that* **by** (*simp add*: *e'_def*)
} note *op_acbd* = *this*
{ fix *k*::*real* **and** $\mathcal{D}$ **and** *u*::*'a* **and** *v w* **and** *z*::*'b* **and** *t1 t2 l*
assume *k*: $0 < k$
and *nf*: $\bigwedge x\ y\ u\ v.$
$[\![x \in cbox\ a\ b;\ y \in cbox\ c\ d;\ u \in cbox\ a\ b;\ v \in cbox\ c\ d;\ norm\ (u-x,\ v-y) < k]\!]$
$\Longrightarrow$ *norm* $(f(u,v) - f(x,y)) < e/(2 *$ (*content ?CBOX*))
and *p_acbd*: $\mathcal{D}$ *tagged_division_of cbox* $(a,c)\ (b,d)$
and *fine*: ($\lambda x.$ *ball x k*) *fine* $\mathcal{D}$ $((t1,t2),\ l) \in \mathcal{D}$
and *uwvz_sub*: *cbox* $(u,w)\ (v,z) \subseteq l$ *cbox* $(u,w)\ (v,z) \subseteq$ *cbox* $(a,c)\ (b,d)$
have *t*: *t1* $\in$ *cbox a b t2* $\in$ *cbox c d*
by (*meson fine p_acbd cbox_Pair_iff tag_in_interval*)+
have *ls*: $l \subseteq$ *ball* $(t1,t2)\ k$
using *fine* **by** (*simp add*: *fine_def Ball_def*)
{ fix *x1 x2*
assume *xuvwz*: *x1* $\in$ *cbox u v x2* $\in$ *cbox w z*
then have *x*: *x1* $\in$ *cbox a b x2* $\in$ *cbox c d*
using *uwvz_sub* **by** *auto*
have *norm* $(x1 - t1,\ x2 - t2) =$ *norm* $(t1 - x1,\ t2 - x2)$
by (*simp add*: *norm_Pair norm_minus_commute*)

```
    also have norm (t1 − x1, t2 − x2) < k
      using xuvwz ls uwvz_sub unfolding ball_def
      by (force simp add: cbox_Pair_eq dist_norm )
    finally have norm (f (x1,x2) − f (t1,t2)) ≤ e/content ?CBOX/2
      using nf [OF t x] by simp
  } note nf' = this
  have f_int_uwvz: f integrable_on cbox (u,w) (v,z)
    using f_int_acbd uwvz_sub integrable_on_subcbox by blast
  have f_int_uv: ⋀x. x ∈ cbox u v ⟹ (λy. f (x,y)) integrable_on cbox w z
    using assms continuous_on_subset uwvz_sub
    by (blast intro: continuous_on_imp_integrable_on_Pair1)
  have 1: norm (integral (cbox (u,w) (v,z)) f − integral (cbox (u,w) (v,z)) (λx.
f (t1,t2)))
        ≤ e * content (cbox (u,w) (v,z)) / content ?CBOX/2
    using cbp ‹0 < e/content ?CBOX› nf'
    apply (simp only: integral_diff [symmetric] f_int_uwvz integrable_const)
     apply (auto simp: integrable_diff f_int_uwvz integrable_const intro: or-
der_trans [OF integrable_bound [of e/content ?CBOX/2]])
    done
  have int_integrable: (λx. integral (cbox w z) (λy. f (x, y))) integrable_on
cbox u v
    using assms integral_integrable_2dim continuous_on_subset uwvz_sub(2)
by blast
  have normint_wz:
    ⋀x. x ∈ cbox u v ⟹
        norm (integral (cbox w z) (λy. f (x, y)) − integral (cbox w z) (λy. f
(t1, t2)))
        ≤ e * content (cbox w z) / content (cbox (a, c) (b, d))/2
    using cbp ‹0 < e/content ?CBOX› nf'
    apply (simp only: integral_diff [symmetric] f_int_uv integrable_const)
     apply (auto simp: integrable_diff f_int_uv integrable_const intro: or-
der_trans [OF integrable_bound [of e/content ?CBOX/2]])
    done
  have norm (integral (cbox u v)
          (λx. integral (cbox w z) (λy. f (x,y)) − integral (cbox w z) (λy. f
(t1,t2))))
      ≤ e * content (cbox w z) / content ?CBOX/2 * content (cbox u v)
    using cbp ‹0 < e/content ?CBOX›
    apply (intro integrable_bound [OF _ _ normint_wz])
     apply (auto simp: field_split_simps integrable_diff int_integrable inte-
grable_const)
    done
  also have ... ≤ e * content (cbox (u,w) (v,z)) / content ?CBOX/2
    by (simp add: content_Pair field_split_simps)
  finally have 2: norm (integral (cbox u v) (λx. integral (cbox w z) (λy. f
(x,y))) −
                integral (cbox u v) (λx. integral (cbox w z) (λy. f (t1,t2))))
        ≤ e * content (cbox (u,w) (v,z)) / content ?CBOX/2
    by (simp only: integral_diff [symmetric] int_integrable integrable_const)
```

```
    have norm_xx: ⟦x' = y'; norm(x - x') ≤ e/2; norm(y - y') ≤ e/2⟧ ⟹
norm(x - y) ≤ e for x::'c and y x' y' e
      using norm_triangle_mono [of x-y' e/2 y'-y e/2] field_sum_of_halves
      by (simp add: norm_minus_commute)
    have norm (integral (cbox (u,w) (v,z)) f - integral (cbox u v) (λx. integral
(cbox w z) (λy. f (x,y))))
          ≤ e * content (cbox (u,w) (v,z)) / content ?CBOX
      by (rule norm_xx [OF integral_Pair_const 1 2])
  } note * = this
  have norm (integral ?CBOX f - integral (cbox a b) (λx. integral (cbox c d)
(λy. f (x,y)))) ≤ e
    if ∀x∈?CBOX. ∀x'∈?CBOX. norm (x' - x) < k ⟶ norm (f x' - f x) < e
/(2 * content (?CBOX)) 0 < k for k
  proof -
    obtain p where ptag: p tagged_division_of cbox (a, c) (b, d)
                and fine: (λx. ball x k) fine p
      using fine_division_exists ‹0 < k› by blast
    show ?thesis
      using that fine ptag ‹0 < k›
      by (auto simp: * intro: op_acbd [OF division_of_tagged_division [OF ptag]])
  qed
  then show norm (integral ?CBOX f - integral (cbox a b) (λx. integral (cbox
c d) (λy. f (x,y)))) ≤ e
    using compact_uniformly_continuous [OF assms compact_cbox]
    apply (simp add: uniformly_continuous_on_def dist_norm)
    apply (drule_tac x=e/2 / content?CBOX in spec)
    using cbp ‹0 < e› by (auto simp: zero_less_mult_iff)
 qed
 then show ?thesis
  by simp
qed

lemma integral_swap_2dim:
 fixes f :: ['a::euclidean_space, 'b::euclidean_space] ⇒ 'c::banach
 assumes continuous_on (cbox (a,c) (b,d)) (λ(x,y). f x y)
   shows integral (cbox (a, c) (b, d)) (λ(x, y). f x y) = integral (cbox (c, a) (d,
b)) (λ(x, y). f y x)
proof -
 have ((λ(x, y). f x y) has_integral integral (cbox (c, a) (d, b)) (λ(x, y). f y x))
(prod.swap ` (cbox (c, a) (d, b)))
 proof (rule has_integral_twiddle [of 1 prod.swap prod.swap λ(x,y). f y x integral
(cbox (c, a) (d, b)) (λ(x, y). f y x), simplified])
   show ⋀u v. content (prod.swap ` cbox u v) = content (cbox u v)
     by (metis content_Pair mult.commute old.prod.exhaust swap_cbox_Pair)
   show ((λ(x, y). f y x) has_integral integral (cbox (c, a) (d, b)) (λ(x, y). f y x))
(cbox (c, a) (d, b))
    by (simp add: assms integrable_continuous integrable_integral swap_continuous)
 qed (use isCont_swap in ‹fastforce+›)
then show ?thesis
```

by *force*
qed

theorem *integral_swap_continuous*:
fixes f :: [$'a$::*euclidean_space*, $'b$::*euclidean_space*] ⇒ $'c$::*banach*
assumes *continuous_on* (*cbox* (*a*,*c*) (*b*,*d*)) (λ(*x*,*y*). *f x y*)
shows *integral* (*cbox a b*) (λ*x*. *integral* (*cbox c d*) (*f x*)) =
integral (*cbox c d*) (λ*y*. *integral* (*cbox a b*) (λ*x*. *f x y*))
proof −
have *integral* (*cbox a b*) (λ*x*. *integral* (*cbox c d*) (*f x*)) = *integral* (*cbox* (*a*, *c*) (*b*, *d*)) (λ(*x*, *y*). *f x y*)
using *integral_prod_continuous* [*OF assms*] **by** *auto*
also have ... = *integral* (*cbox* (*c*, *a*) (*d*, *b*)) (λ(*x*, *y*). *f y x*)
by (*rule integral_swap_2dim* [*OF assms*])
also have ... = *integral* (*cbox c d*) (λ*y*. *integral* (*cbox a b*) (λ*x*. *f x y*))
using *integral_prod_continuous* [*OF swap_continuous*] *assms*
by *auto*
finally show *?thesis* **.**
qed

6.14.47 Definite integrals for exponential and power function

lemma *has_integral_exp_minus_to_infinity*:
assumes *a*: *a* > *0*
shows ((λ*x*::*real*. *exp* (−*a*∗*x*)) *has_integral exp* (−*a*∗*c*)/*a*) {*c*..}
proof −
define *f* **where** *f* = (λ*k x*. *if x* ∈ {*c*..*real k*} *then exp* (−*a*∗*x*) *else* *0*)
{
fix *k* :: *nat* **assume** *k*: *of_nat k* ≥ *c*
from *k a*
have ((λ*x*. *exp* (−*a*∗*x*)) *has_integral* (−*exp* (−*a*∗*real k*)/*a* − (−*exp* (−*a*∗*c*)/*a*))) {*c*..*real k*}
by (*intro fundamental_theorem_of_calculus*)
(*auto intro*!: *derivative_eq_intros*
simp: *has_real_derivative_iff_has_vector_derivative* [*symmetric*])
hence (*f k has_integral* (*exp* (−*a*∗*c*)/*a* − *exp* (−*a*∗*real k*)/*a*)) {*c*..} **unfolding** *f_def*
by (*subst has_integral_restrict*) *simp_all*
} **note** *has_integral_f* = *this*

have [*simp*]: *f k* = (λ_. *0*) **if** *of_nat k* < *c* **for** *k* **using** *that* **by** (*auto simp*: *fun_eq_iff f_def*)
have *integral_f*: *integral* {*c*..} (*f k*) =
(*if real k* ≥ *c then exp* (−*a*∗*c*)/*a* − *exp* (−*a*∗*real k*)/*a else* *0*)
for *k* **using** *integral_unique*[*OF has_integral_f*[*of k*]] **by** *simp*

have *A*: (λ*x*. *exp* (−*a*∗*x*)) *integrable_on* {*c*..} ∧
(λ*k*. *integral* {*c*..} (*f k*)) ⟶ *integral* {*c*..} (λ*x*. *exp* (−*a*∗*x*))
proof (*intro monotone_convergence_increasing allI ballI*)

```
    fix k ::nat
    have (λx. exp (−a∗x)) integrable_on {c..of_real k}
     unfolding f_def by (auto intro!: continuous_intros integrable_continuous_real)
    hence  (f k) integrable_on {c..of_real k}
      by (rule integrable_eq) (simp add: f_def)
    then show f k integrable_on {c..}
      by (rule integrable_on_superset) (auto simp: f_def)
  next
    fix x assume x: x ∈ {c..}
   have sequentially ≤ principal {nat ⌈x⌉..} unfolding at_top_def by (simp add:
Inf_lower)
    also have {nat ⌈x⌉..} ⊆ {k. x ≤ real k} by auto
    also have inf (principal ...) (principal {k. ¬x ≤ real k}) =
                principal ({k. x ≤ real k} ∩ {k. ¬x ≤ real k})
      by simp
    also have {k. x ≤ real k} ∩ {k. ¬x ≤ real k} = {} by blast
    finally have inf sequentially (principal {k. ¬x ≤ real k}) = bot
      by (simp add: inf.coboundedI1 bot_unique)
    with x show (λk. f k x) ⟶ exp (−a∗x) unfolding f_def
      by (intro filterlim_If) auto
  next
    have |integral {c..} (f k)| ≤ exp (−a∗c)/a for k
    proof (cases c > of_nat k)
      case False
     hence abs (integral {c..} (f k)) = abs (exp (− (a ∗ c)) / a − exp (− (a ∗ real
k)) / a)
        by (simp add: integral_f)
      also have abs (exp (− (a ∗ c)) / a − exp (− (a ∗ real k)) / a) =
                  exp (− (a ∗ c)) / a − exp (− (a ∗ real k)) / a
        using False a by (intro abs_of_nonneg) (simp_all add: field_simps)
      also have ... ≤ exp (− a ∗ c) / a using a by simp
      finally show ?thesis .
    qed (insert a, simp_all add: integral_f)
    thus bounded (range(λk. integral {c..} (f k)))
      by (intro boundedI[of _ exp (−a∗c)/a]) auto
  qed (auto simp: f_def)
  have (λk. exp (−a∗c)/a − exp (−a ∗ of_nat k)/a) ⟶ exp (−a∗c)/a − 0/a
    by (intro tendsto_intros filterlim_compose[OF exp_at_bot]
      filterlim_tendsto_neg_mult_at_bot[OF tendsto_const] filterlim_real_sequentially)+
      (use a in simp_all)
  moreover
   from eventually_gt_at_top[of nat ⌈c⌉] have eventually (λk. of_nat k > c)
sequentially
    by eventually_elim linarith
  hence eventually (λk. exp (−a∗c)/a − exp (−a ∗ of_nat k)/a = integral {c..}
(f k)) sequentially
    by eventually_elim (simp add: integral_f)
  ultimately have (λk. integral {c..} (f k)) ⟶ exp (−a∗c)/a − 0/a
    by (rule Lim_transform_eventually)
```

```
  from LIMSEQ_unique[OF conjunct2[OF A] this]
  have integral {c..} (λx. exp (−a∗x)) = exp (−a∗c)/a by simp
  with conjunct1[OF A] show ?thesis
    by (simp add: has_integral_integral)
qed

lemma integrable_on_exp_minus_to_infinity: a > 0 ⟹ (λx. exp (−a∗x) :: real)
integrable_on {c..}
 using has_integral_exp_minus_to_infinity[of a c] unfolding integrable_on_def
by blast

lemma has_integral_powr_from_0:
  assumes a: a > (−1::real) and c: c ≥ 0
  shows   ((λx. x powr a) has_integral (c powr (a+1) / (a+1))) {0..c}
proof (cases c = 0)
  case False
  define f where f = (λk x. if x ∈ {inverse (of_nat (Suc k))..c} then x powr a
else 0)
  define F where F = (λk. if inverse (of_nat (Suc k)) ≤ c then
                              c powr (a+1)/(a+1) − inverse (real (Suc k)) powr
(a+1)/(a+1) else 0)
  have has_integral_f: (f k has_integral F k) {0..c} for k::nat
  proof (cases inverse (of_nat (Suc k)) ≤ c)
    case True
    have x: x > 0 if x ≥ inverse (1 + real k) for x
      by (smt (verit) that inverse_Suc of_nat_Suc)
    hence ((λx. x powr a) has_integral c powr (a + 1) / (a + 1) −
              inverse (real (Suc k)) powr (a + 1) / (a + 1)) {inverse (real (Suc
k))..c}
      using True a by (intro fundamental_theorem_of_calculus)
      (auto intro!: derivative_eq_intros continuous_on_powr' continuous_on_const
          simp: has_real_derivative_iff_has_vector_derivative [symmetric])
   with True show ?thesis unfolding f_def F_def by (subst has_integral_restrict)
simp_all
  next
    case False
    thus ?thesis unfolding f_def F_def
      by (subst has_integral_restrict) auto
  qed
  then have integral_f: integral {0..c} (f k) = F k for k
    by blast

  have A: (λx. x powr a) integrable_on {0..c} ∧
         (λk. integral {0..c} (f k)) ⟶ integral {0..c} (λx. x powr a)
  proof (intro monotone_convergence_increasing ballI allI)
    fix k from has_integral_f[of k] show f k integrable_on {0..c}
      by (auto simp: integrable_on_def)
  next
    fix k :: nat and x :: real
```

```
  {
    assume x: inverse (real (Suc k)) ≤ x
    then have inverse (real (Suc (Suc k))) ≤ x
      using dual_order.trans by fastforce
  }
  thus f k x ≤ f (Suc k) x by (auto simp: f_def simp del: of_nat_Suc)
next
  fix x assume x: x ∈ {0..c}
  show (λk. f k x) ⟶ x powr a
  proof (cases x = 0)
    case False
    with x have x > 0 by simp
    from order_tendstoD(2)[OF LIMSEQ_inverse_real_of_nat this]
      have eventually (λk. x powr a = f k x) sequentially
      by eventually_elim (insert x, simp add: f_def)
    moreover have (λ_. x powr a) ⟶ x powr a by simp
    ultimately show ?thesis by (blast intro: Lim_transform_eventually)
  qed (simp_all add: f_def)
next
  {
    fix k
    from a have F k ≤ c powr (a + 1) / (a + 1)
      by (auto simp add: F_def divide_simps)
    also from a have F k ≥ 0
    by (auto simp: F_def divide_simps simp del: of_nat_Suc intro!: powr_mono2)
    hence F k = abs (F k) by simp
    finally have abs (F k) ≤  c powr (a + 1) / (a + 1) .
  }
  thus bounded (range(λk. integral {0..c} (f k)))
    by (intro boundedI[of _ c powr (a+1) / (a+1)]) (auto simp: integral_f)
qed

from False c have c > 0 by simp
from order_tendstoD(2)[OF LIMSEQ_inverse_real_of_nat this]
  have eventually (λk. c powr (a + 1) / (a + 1) − inverse (real (Suc k)) powr
(a+1) / (a+1) =
        integral {0..c} (f k)) sequentially
  by eventually_elim (simp add: integral_f F_def)
moreover have (λk. c powr (a + 1) / (a + 1) − inverse (real (Suc k)) powr
(a + 1) / (a + 1))
              ⟶ c powr (a + 1) / (a + 1) − 0 powr (a + 1) / (a + 1)
  using a by (intro tendsto_intros LIMSEQ_inverse_real_of_nat) auto
hence (λk. c powr (a + 1) / (a + 1) − inverse (real (Suc k)) powr (a + 1) /
(a + 1))
      ⟶ c powr (a + 1) / (a + 1) by simp
ultimately have (λk. integral {0..c} (f k)) ⟶ c powr (a+1) / (a+1)
  by (blast intro: Lim_transform_eventually)
with A have integral {0..c} (λx. x powr a) = c powr (a+1) / (a+1)
  by (blast intro: LIMSEQ_unique)
```

with *A* **show** *?thesis* **by** (*simp add*: *has_integral_integral*)
qed (*simp_all add*: *has_integral_refl*)

lemma *integrable_on_powr_from_0*:
assumes *a*: $a > (-1::real)$ **and** *c*: $c \geq 0$
shows $(\lambda x.\ x\ powr\ a)$ *integrable_on* $\{0..c\}$
using *has_integral_powr_from_0*[*OF assms*] **unfolding** *integrable_on_def* **by** *blast*

lemma *has_integral_powr_to_inf*:
fixes *a e* :: *real*
assumes $e < -1$ $a > 0$
shows $((\lambda x.\ x\ powr\ e)\ has\_integral\ -(a\ powr\ (e + 1)) / (e + 1))\ \{a..\}$
proof −
define *f* :: *nat* ⇒ *real* ⇒ *real* **where** $f = (\lambda n\ x.\ if\ x \in \{a..n\}\ then\ x\ powr\ e\ else\ 0)$
define *F* **where** $F = (\lambda x.\ x\ powr\ (e + 1) / (e + 1))$

have *has_integral_f*: $(f\ n\ has\_integral\ (F\ n - F\ a))\ \{a..\}$
if *n*: $n \geq a$ **for** *n* :: *nat*
proof −
from *n assms* **have** $((\lambda x.\ x\ powr\ e)\ has\_integral\ (F\ n - F\ a))\ \{a..n\}$
by (*intro fundamental_theorem_of_calculus*) (*auto intro*!: *derivative_eq_intros*
simp: *has_real_derivative_iff_has_vector_derivative* [*symmetric*] *F_def*)
hence $(f\ n\ has\_integral\ (F\ n - F\ a))\ \{a..n\}$
by (*rule has_integral_eq* [*rotated*]) (*simp add*: *f_def*)
thus $(f\ n\ has\_integral\ (F\ n - F\ a))\ \{a..\}$
by (*rule has_integral_on_superset*) (*auto simp*: *f_def*)
qed
have *integral_f*: *integral* $\{a..\}$ $(f\ n) = (if\ n \geq a\ then\ F\ n - F\ a\ else\ 0)$ **for** *n* :: *nat*
proof (*cases* $n \geq a$)
case *True*
with *has_integral_f*[*OF this*] **show** *?thesis* **by** (*simp add*: *integral_unique*)
next
case *False*
have $(f\ n\ has\_integral\ 0)\ \{a\}$ **by** (*rule has_integral_refl*)
hence $(f\ n\ has\_integral\ 0)\ \{a..\}$
using *False f_def* **by** *force*
with *False* **show** *?thesis* **by** (*simp add*: *integral_unique*)
qed

have ∗: $(\lambda x.\ x\ powr\ e)\ integrable\_on\ \{a..\} \wedge$
$(\lambda n.\ integral\ \{a..\}\ (f\ n)) \longrightarrow integral\ \{a..\}\ (\lambda x.\ x\ powr\ e)$
proof (*intro monotone_convergence_increasing allI ballI*)
fix *n* :: *nat*
from *assms* **have** $(\lambda x.\ x\ powr\ e)\ integrable\_on\ \{a..n\}$
by (*auto intro*!: *integrable_continuous_real continuous_intros*)
hence *f n integrable_on* $\{a..n\}$

```
    by (rule integrable_eq) (auto simp: f_def)
  thus f n integrable_on {a..}
    by (rule integrable_on_superset) (auto simp: f_def)
 next
  fix n :: nat and x :: real
  show f n x ≤ f (Suc n) x by (auto simp: f_def)
 next
  fix x :: real assume x: x ∈ {a..}
  from filterlim_real_sequentially
   have eventually (λn. real n ≥ x) at_top
   by (simp add: filterlim_at_top)
  with x have eventually (λn. f n x = x powr e) at_top
   by (auto elim!: eventually_mono simp: f_def)
  thus (λn. f n x) ⟶ x powr e by (simp add: tendsto_eventually)
 next
  have norm (integral {a..} (f n)) ≤ -F a for n :: nat
  proof (cases n ≥ a)
   case True
   with assms have a powr (e + 1) ≥ n powr (e + 1)
    by (intro powr_mono2') simp_all
   with assms show ?thesis by (auto simp: divide_simps F_def integral_f)
  qed (use assms in ‹simp add: integral_f F_def field_split_simps›)
  thus bounded (range(λk. integral {a..} (f k)))
   unfolding bounded_iff by (intro exI[of _ -F a]) auto
 qed

 from filterlim_real_sequentially
  have eventually (λn. real n ≥ a) at_top
  by (simp add: filterlim_at_top)
 hence eventually (λn. F n - F a = integral {a..} (f n)) at_top
  by eventually_elim (simp add: integral_f)
 moreover have (λn. F n - F a) ⟶ 0 / (e + 1) - F a unfolding F_def
  by (insert assms, (rule tendsto_intros filterlim_compose[OF tendsto_neg_powr]
       filterlim_ident filterlim_real_sequentially | simp)+)
 hence (λn. F n - F a) ⟶ -F a by simp
 ultimately have (λn. integral {a..} (f n)) ⟶ -F a by (blast intro: Lim_transform_eventually)
 then have integral {a..} (λx. x powr e) = -F a
  using * LIMSEQ_unique by blast
 with * show ?thesis
  by (simp add: has_integral_integral F_def)
qed

lemma has_integral_inverse_power_to_inf:
 fixes a :: real and n :: nat
 assumes n > 1 a > 0
 shows   ((λx. 1 / x ^ n) has_integral 1 / (real (n - 1) * a ^ (n - 1))) {a..}
proof -
 from assms have real_of_int (-int n) < -1 by simp
 note has_integral_powr_to_inf[OF this ‹a > 0›]
```

also have $-$ (*a powr* (*real_of_int* ($-$ *int n*) + *1*)) / (*real_of_int* ($-$ *int n*) + *1*) =
1 / (*real* (*n* $-$ *1*) $*$ *a powr* (*real* (*n* $-$ *1*))) **using** *assms*
by (*simp add*: *field_split_simps powr_add* [*symmetric*] *of_nat_diff*)
also from *assms* **have** *a powr* (*real* (*n* $-$ *1*)) = *a* ^ (*n* $-$ *1*)
by (*intro powr_realpow*)
finally show *?thesis*
by (*rule has_integral_eq* [*rotated*])
(*insert assms*, *simp_all add*: *powr_minus powr_realpow field_split_simps*)
qed

Adaption to ordered Euclidean spaces and the Cartesian Euclidean space

lemma *integral_component_eq_cart*[*simp*]:
fixes *f* :: *'n*::*euclidean_space* ⇒ *real*^*'m*
assumes *f integrable_on s*
shows *integral s* (λ*x*. *f x* $ *k*) = *integral s f* $ *k*
using *integral_linear*[*OF assms*(*1*) *bounded_linear_vec_nth*,*unfolded o_def*] .

lemma *content_closed_interval*:
fixes *a* :: *'a*::*ordered_euclidean_space*
assumes *a* ≤ *b*
shows *content* {*a*..*b*} = (∏ *i*∈*Basis*. *b*•*i* $-$ *a*•*i*)
using *content_cbox*[*of a b*] *assms* **by** (*simp add*: *cbox_interval eucl_le*[**where** *'a*=*'a*])

lemma *integrable_const_ivl*[*intro*]:
fixes *a*::*'a*::*ordered_euclidean_space*
shows (λ*x*. *c*) *integrable_on* {*a*..*b*}
unfolding *cbox_interval*[*symmetric*] **by** (*rule integrable_const*)

lemma *integrable_on_subinterval*:
fixes *f* :: *'n*::*ordered_euclidean_space* ⇒ *'a*::*banach*
assumes *f integrable_on S* {*a*..*b*} ⊆ *S*
shows *f integrable_on* {*a*..*b*}
using *integrable_on_subcbox*[*of f S a b*] *assms* **by** (*simp add*: *cbox_interval*)

end

6.15 Radon-Nikodým Derivative

theory *Radon_Nikodym*
imports *Bochner_Integration*
begin

definition *diff_measure* :: *'a measure* ⇒ *'a measure* ⇒ *'a measure*
where

diff_measure M N = measure_of (space M) (sets M) (λA. emeasure M A − emeasure N A)

lemma
shows *space_diff_measure*[*simp*]: *space* (*diff_measure M N*) = *space M*
and *sets_diff_measure*[*simp*]: *sets* (*diff_measure M N*) = *sets M*
by (*auto simp*: *diff_measure_def*)

lemma *emeasure_diff_measure*:
assumes *fin*: *finite_measure M finite_measure N* **and** *sets_eq*: *sets M* = *sets N*
assumes *pos*: ⋀*A*. *A* ∈ *sets M* ⟹ *emeasure N A* ≤ *emeasure M A* **and** *A*: *A* ∈ *sets M*
shows *emeasure* (*diff_measure M N*) *A* = *emeasure M A* − *emeasure N A* (**is** _ = *?μ A*)
unfolding *diff_measure_def*
proof (*rule emeasure_measure_of_sigma*)
show *sigma_algebra* (*space M*) (*sets M*) **..**
show *positive* (*sets M*) *?μ*
using *pos* **by** (*simp add*: *positive_def*)
show *countably_additive* (*sets M*) *?μ*
proof (*rule countably_additiveI*)
fix *A* :: *nat* ⇒ _ **assume** *A*: *range A* ⊆ *sets M* **and** *disjoint_family A*
then have *suminf*:
(∑ *i*. *emeasure M* (*A i*)) = *emeasure M* (⋃ *i*. *A i*)
(∑ *i*. *emeasure N* (*A i*)) = *emeasure N* (⋃ *i*. *A i*)
by (*simp_all add*: *suminf_emeasure sets_eq*)
with *A* **have** (∑ *i*. *emeasure M* (*A i*) − *emeasure N* (*A i*)) =
(∑ *i*. *emeasure M* (*A i*)) − (∑ *i*. *emeasure N* (*A i*))
using *fin pos*[*of A* _]
by (*intro ennreal_suminf_minus*)
(*auto simp*: *sets_eq finite_measure.emeasure_eq_measure suminf_emeasure*)
then show (∑ *i*. *emeasure M* (*A i*) − *emeasure N* (*A i*)) =
emeasure M (⋃ *i*. *A i*) − *emeasure N* (⋃ *i*. *A i*)
by (*simp add*: *suminf*)
qed
qed *fact*

An equivalent characterization of sigma-finite spaces is the existence of integrable positive functions (or, still equivalently, the existence of a probability measure which is in the same measure class as the original measure).

proposition (**in** *sigma_finite_measure*) *obtain_positive_integrable_function*:
obtains *f*::*'a* ⇒ *real* **where**
f ∈ *borel_measurable M*
⋀*x*. *f x* > *0*
⋀*x*. *f x* ≤ *1*
integrable M f
proof −
obtain *A* :: *nat* ⇒ *'a set* **where** *range A* ⊆ *sets M* (⋃ *i*. *A i*) = *space M* ⋀*i*. *emeasure M* (*A i*) ≠ ∞

```
    using sigma_finite by auto
  then have [measurable]:A n ∈ sets M for n by auto
  define g where g = (λx. (∑ n. (1/2)^(Suc n) * indicator (A n) x / (1+ measure
M (A n))))
  have [measurable]: g ∈ borel_measurable M unfolding g_def by auto
  have *: summable (λn. (1/2)^(Suc n) * indicator (A n) x / (1+ measure M (A
n))) for x
    apply (rule summable_comparison_test'[of λn. (1/2)^(Suc n) 0])
     using power_half_series summable_def by (auto simp add: indicator_def
divide_simps)
  have g x ≤ (∑ n. (1/2)^(Suc n)) for x unfolding g_def
    apply (rule suminf_le) using * power_half_series summable_def by (auto
simp add: indicator_def divide_simps)
  then have g_le_1: g x ≤ 1 for x using power_half_series sums_unique by
fastforce

  have g_pos: g x > 0 if x ∈ space M for x
  unfolding g_def proof (subst suminf_pos_iff[OF *[of x]], auto)
   obtain i where x ∈ A i using ‹(⋃ i. A i) = space M› ‹x ∈ space M› by auto
  then have 0 < (1 / 2) ^ Suc i * indicator (A i) x / (1 + Sigma_Algebra.measure
M (A i))
     unfolding indicator_def apply (auto simp add: divide_simps) using mea-
sure_nonneg[of M A i]
     by (auto, meson add_nonneg_nonneg linorder_not_le mult_nonneg_nonneg
zero_le_numeral zero_le_one zero_le_power)
  then show ∃ i. 0 < (1 / 2) ^ i * indicator (A i) x / (2 + 2 * Sigma_Algebra.measure
M (A i))
     by auto
  qed

  have integrable M g
  unfolding g_def proof (rule integrable_suminf)
    fix n
  show integrable M (λx. (1 / 2) ^ Suc n * indicator (A n) x / (1 + Sigma_Algebra.measure
M (A n)))
     using ‹emeasure M (A n) ≠ ∞›
    by (auto intro!: integrable_mult_right integrable_divide_zero integrable_real_indicator
simp add: top.not_eq_extremum)
  next
    show AE x in M. summable (λn. norm ((1 / 2) ^ Suc n * indicator (A n) x
/ (1 + Sigma_Algebra.measure M (A n))))
     using * by auto
    show summable (λn. (∫ x. norm ((1 / 2) ^ Suc n * indicator (A n) x / (1 +
Sigma_Algebra.measure M (A n))) ∂M))
     apply (rule summable_comparison_test'[of λn. (1/2)^(Suc n) 0], auto)
     using power_half_series summable_def apply auto[1]
       apply (auto simp add: field_split_simps) using measure_nonneg[of M]
not_less by fastforce
  qed
```

define f **where** $f = (\lambda x.\ \mathit{if}\ x \in \mathit{space}\ M\ \mathit{then}\ g\ x\ \mathit{else}\ 1)$
have $f\ x > 0$ **for** x **unfolding** *f_def* **using** *g_pos* **by** *auto*
moreover have $f\ x \leq 1$ **for** x **unfolding** *f_def* **using** *g_le_1* **by** *auto*
moreover have [*measurable*]: $f \in \mathit{borel\_measurable}\ M$ **unfolding** *f_def* **by** *auto*
moreover have *integrable M f*
apply (*subst integrable_cong*[*of _ _ _ g*]) **unfolding** *f_def* **using** ‹*integrable M g*› **by** *auto*
ultimately show $(\bigwedge f.\ f \in \mathit{borel\_measurable}\ M \Longrightarrow (\bigwedge x.\ 0 < f\ x) \Longrightarrow (\bigwedge x.\ f\ x \leq 1) \Longrightarrow \mathit{integrable}\ M\ f \Longrightarrow \mathit{thesis}) \Longrightarrow \mathit{thesis}$
by (*meson that*)
qed

lemma (**in** *sigma_finite_measure*) *Ex_finite_integrable_function*:
$\exists h \in \mathit{borel\_measurable}\ M.\ \mathit{integral}^N\ M\ h \neq \infty \wedge (\forall x \in \mathit{space}\ M.\ 0 < h\ x \wedge h\ x < \infty)$
proof −
obtain $A :: \mathit{nat} \Rightarrow 'a\ \mathit{set}$ **where**
range[*measurable*]: *range A* ⊆ *sets M* **and**
space: $(\bigcup i.\ A\ i) = \mathit{space}\ M$ **and**
measure: $\bigwedge i.\ \mathit{emeasure}\ M\ (A\ i) \neq \infty$ **and**
disjoint: *disjoint_family A*
using *sigma_finite_disjoint* **by** *blast*
let $?B = \lambda i.\ 2\hat{}\mathit{Suc}\ i * \mathit{emeasure}\ M\ (A\ i)$
have [*measurable*]: $\bigwedge i.\ A\ i \in \mathit{sets}\ M$
using *range* **by** *fastforce+*
have $\forall i.\ \exists x.\ 0 < x \wedge x < \mathit{inverse}\ (?B\ i)$
proof
fix *i* **show** $\exists x.\ 0 < x \wedge x < \mathit{inverse}\ (?B\ i)$
using *measure*[*of i*]
by (*auto intro*!: *dense simp*: *ennreal_inverse_positive ennreal_mult_eq_top_iff power_eq_top_ennreal*)
qed
from *choice*[*OF this*] **obtain** *n* **where** *n*: $\bigwedge i.\ 0 < n\ i$
$\bigwedge i.\ n\ i < \mathit{inverse}\ (2\hat{}\mathit{Suc}\ i * \mathit{emeasure}\ M\ (A\ i))$ **by** *auto*
{ fix *i* **have** $0 \leq n\ i$ **using** *n*(*1*)[*of i*] **by** *auto* **} note** *pos* = *this*
let $?h = \lambda x.\ \sum i.\ n\ i * \mathit{indicator}\ (A\ i)\ x$
show *?thesis*
proof (*safe intro*!: *bexI*[*of _ ?h*] *del*: *notI*)
have $\mathit{integral}^N\ M\ ?h = (\sum i.\ n\ i * \mathit{emeasure}\ M\ (A\ i))$ **using** *pos*
by (*simp add*: *nn_integral_suminf nn_integral_cmult_indicator*)
also have $\ldots \leq (\sum i.\ \mathit{ennreal}\ ((1/2)\hat{}\mathit{Suc}\ i))$
proof (*intro suminf_le allI*)
fix *N*
have $n\ N * \mathit{emeasure}\ M\ (A\ N) \leq \mathit{inverse}\ (2\hat{}\mathit{Suc}\ N * \mathit{emeasure}\ M\ (A\ N)) * \mathit{emeasure}\ M\ (A\ N)$
using *n*[*of N*] **by** (*intro mult_right_mono*) *auto*
also have $\ldots = (1/2)\hat{}\mathit{Suc}\ N * (\mathit{inverse}\ (\mathit{emeasure}\ M\ (A\ N)) * \mathit{emeasure}\ M\ (A\ N))$

```
      using measure[of N]
    by (simp add: ennreal_inverse_power divide_ennreal_def ennreal_inverse_mult
                  power_eq_top_ennreal less_top[symmetric] mult_ac
                del: power_Suc)
    also have ... ≤ inverse (ennreal 2) ^ Suc N
      using measure[of N]
      by (cases emeasure M (A N); cases emeasure M (A N) = 0)
        (auto simp: inverse_ennreal ennreal_mult[symmetric] divide_ennreal_def
simp del: power_Suc)
    also have ... = ennreal (inverse 2 ^ Suc N)
      by (subst ennreal_power[symmetric], simp) (simp add: inverse_ennreal)
    finally show n N * emeasure M (A N) ≤ ennreal ((1/2)^Suc N)
      by simp
  qed auto
  also have ... < top
    unfolding less_top[symmetric]
    by (rule ennreal_suminf_neq_top)
      (auto simp: summable_geometric summable_Suc_iff simp del: power_Suc)
  finally show integral^N M ?h ≠ ∞
    by (auto simp: top_unique)
next
  { fix x assume x ∈ space M
    then obtain i where x ∈ A i using space[symmetric] by auto
    with disjoint n have ?h x = n i
      by (auto intro!: suminf_cmult_indicator intro: less_imp_le)
  then show 0 < ?h x and ?h x < ∞ using n[of i] by (auto simp: less_top[symmetric])
}
  note pos = this
qed measurable
qed
```

6.15.1 Absolutely continuous

```
definition absolutely_continuous :: 'a measure ⇒ 'a measure ⇒ bool where
  absolutely_continuous M N ⟷ null_sets M ⊆ null_sets N

lemma absolutely_continuousI_count_space: absolutely_continuous (count_space
A) M
  unfolding absolutely_continuous_def by (auto simp: null_sets_count_space)

lemma absolutely_continuousI_density:
  f ∈ borel_measurable M ⟹ absolutely_continuous M (density M f)
  by (force simp add: absolutely_continuous_def null_sets_density_iff dest: AE_not_in)

lemma absolutely_continuousI_point_measure_finite:
  (⋀x. ⟦ x ∈ A ; f x ≤ 0 ⟧ ⟹ g x ≤ 0) ⟹ absolutely_continuous (point_measure
A f) (point_measure A g)
  unfolding absolutely_continuous_def by (force simp: null_sets_point_measure_iff)
```

lemma *absolutely_continuousD*:
absolutely_continuous M N $\Longrightarrow$ *A* $\in$ *sets M* $\Longrightarrow$ *emeasure M A* = *0* $\Longrightarrow$ *emeasure N A* = *0*
by (*auto simp*: *absolutely_continuous_def null_sets_def*)

lemma *absolutely_continuous_AE*:
assumes *sets_eq*: *sets M'* = *sets M*
and *absolutely_continuous M M' AE x in M*. *P x*
shows *AE x in M'*. *P x*
proof −
from ‹*AE x in M*. *P x*› **obtain** *N* **where** *N*: *N* $\in$ *null_sets M* {*x*∈*space M*. ¬ *P x*} $\subseteq$ *N*
unfolding *eventually_ae_filter* **by** *auto*
show *AE x in M'*. *P x*
proof (*rule AE_I'*)
show {*x*∈*space M'*. ¬ *P x*} $\subseteq$ *N* **using** *sets_eq_imp_space_eq*[*OF sets_eq*] *N*(*2*) **by** *simp*
from ‹*absolutely_continuous M M'*› **show** *N* $\in$ *null_sets M'*
using *N* **unfolding** *absolutely_continuous_def sets_eq null_sets_def* **by** *auto*
qed
qed

6.15.2 Existence of the Radon-Nikodym derivative

proposition
(**in** *finite_measure*) *Radon_Nikodym_finite_measure*:
assumes *finite_measure N* **and** *sets_eq*[*simp*]: *sets N* = *sets M*
assumes *absolutely_continuous M N*
shows ∃*f* $\in$ *borel_measurable M*. *density M f* = *N*
proof −
interpret *N*: *finite_measure N* **by** *fact*
define *G* **where** *G* = {*g* $\in$ *borel_measurable M*. ∀ *A*∈*sets M*. (∫ $^+$*x*. *g x* ∗ *indicator A x* ∂*M*) $\leq$ *N A*}
have [*measurable_dest*]: *f* $\in$ *G* $\Longrightarrow$ *f* $\in$ *borel_measurable M*
and *G_D*: ⋀*A*. *f* $\in$ *G* $\Longrightarrow$ *A* $\in$ *sets M* $\Longrightarrow$ (∫ $^+$*x*. *f x* ∗ *indicator A x* ∂*M*) $\leq$ *N A* **for** *f*
by (*auto simp*: *G_def*)
note *this*[*measurable_dest*]
have (λ*x*. *0*) $\in$ *G* **unfolding** *G_def* **by** *auto*
hence *G* ≠ {} **by** *auto*
{ fix *f g* **assume** *f*[*measurable*]: *f* $\in$ *G* **and** *g*[*measurable*]: *g* $\in$ *G*
have (λ*x*. *max* (*g x*) (*f x*)) $\in$ *G* (**is** *?max* $\in$ *G*) **unfolding** *G_def*
proof *safe*
let *?A* = {*x* $\in$ *space M*. *f x* $\leq$ *g x*}
have *?A* $\in$ *sets M* **using** *f g* **unfolding** *G_def* **by** *auto*
fix *A* **assume** [*measurable*]: *A* $\in$ *sets M*
have *union*: ((*?A* ∩ *A*) ∪ ((*space M* − *?A*) ∩ *A*)) = *A*
using *sets.sets_into_space*[*OF* ‹*A* $\in$ *sets M*›] **by** *auto*

```
    have ⋀x. x ∈ space M ⟹ max (g x) (f x) * indicator A x =
      g x * indicator (?A ∩ A) x + f x * indicator ((space M - ?A) ∩ A) x
      by (auto simp: indicator_def max_def)
    hence (∫⁺x. max (g x) (f x) * indicator A x ∂M) =
      (∫⁺x. g x * indicator (?A ∩ A) x ∂M) +
      (∫⁺x. f x * indicator ((space M - ?A) ∩ A) x ∂M)
      by (auto cong: nn_integral_cong intro!: nn_integral_add)
    also have ... ≤ N (?A ∩ A) + N ((space M - ?A) ∩ A)
      using f g unfolding G_def by (auto intro!: add_mono)
    also have ... = N A
      using union by (subst plus_emeasure) auto
    finally show (∫⁺x. max (g x) (f x) * indicator A x ∂M) ≤ N A .
  qed auto }
 note max_in_G = this
 { fix f assume  incseq f and f: ⋀i. f i ∈ G
  then have [measurable]: ⋀i. f i ∈ borel_measurable M by (auto simp: G_def)
  have (λx. SUP i. f i x) ∈ G unfolding G_def
  proof safe
    show (λx. SUP i. f i x) ∈ borel_measurable M by measurable
  next
    fix A assume A ∈ sets M
    have (∫⁺x. (SUP i. f i x) * indicator A x ∂M) =
      (∫⁺x. (SUP i. f i x * indicator A x) ∂M)
      by (intro nn_integral_cong) (simp split: split_indicator)
    also have ... = (SUP i. (∫⁺x. f i x * indicator A x ∂M))
      using ‹incseq f› f ‹A ∈ sets M›
      by (intro nn_integral_monotone_convergence_SUP)
         (auto simp: G_def incseq_Suc_iff le_fun_def split: split_indicator)
    finally show (∫⁺x. (SUP i. f i x) * indicator A x ∂M) ≤ N A
      using f ‹A ∈ sets M› by (auto intro!: SUP_least simp: G_D)
  qed }
 note SUP_in_G = this
 let ?y = SUP g ∈ G. integral^N M g
 have y_le: ?y ≤ N (space M) unfolding G_def
 proof (safe intro!: SUP_least)
   fix g assume ∀A∈sets M. (∫⁺x. g x * indicator A x ∂M) ≤ N A
   from this[THEN bspec, OF sets.top] show integral^N M g ≤ N (space M)
     by (simp cong: nn_integral_cong)
 qed
 from ennreal_SUP_countable_SUP [OF ‹G ≠ {}›, of integral^N M]
 obtain ys :: nat ⇒ ennreal
   where ys: range ys ⊆ integral^N M ‘ G ∧ Sup (integral^N M ‘ G) = Sup (range
ys)
   by auto
 then have ∀n. ∃g. g∈G ∧ integral^N M g = ys n
 proof safe
   fix n assume range ys ⊆ integral^N M ‘ G
   hence ys n ∈ integral^N M ‘ G by auto
   thus ∃g. g∈G ∧ integral^N M g = ys n by auto
```

```
  qed
  from choice[OF this] obtain gs where ⋀i. gs i ∈ G ⋀n. integral^N M (gs n) =
ys n by auto
  hence y_eq: ?y = (SUP i. integral^N M (gs i)) using ys by auto
  let ?g = λi x. Max ((λn. gs n x) ` {..i})
  define f where [abs_def]: f x = (SUP i. ?g i x) for x
  let ?F = λA x. f x * indicator A x
  have gs_not_empty: ⋀i x. (λn. gs n x) ` {..i} ≠ {} by auto
  { fix i have ?g i ∈ G
    proof (induct i)
      case 0 thus ?case by simp fact
    next
      case (Suc i)
      with Suc gs_not_empty ‹gs (Suc i) ∈ G› show ?case
        by (auto simp add: atMost_Suc intro!: max_in_G)
    qed }
  note g_in_G = this
  have incseq ?g using gs_not_empty
    by (auto intro!: incseq_SucI le_funI simp add: atMost_Suc)

  from SUP_in_G[OF this g_in_G] have [measurable]: f ∈ G unfolding f_def
.
  then have [measurable]: f ∈ borel_measurable M unfolding G_def by auto

  have integral^N M f = (SUP i. integral^N M (?g i)) unfolding f_def
   using g_in_G ‹incseq ?g› by (auto intro!: nn_integral_monotone_convergence_SUP
simp: G_def)
  also have ... = ?y
  proof (rule antisym)
    show (SUP i. integral^N M (?g i)) ≤ ?y
      using g_in_G by (auto intro: SUP_mono)
    show ?y ≤ (SUP i. integral^N M (?g i)) unfolding y_eq
      by (auto intro!: SUP_mono nn_integral_mono Max_ge)
  qed
  finally have int_f_eq_y: integral^N M f = ?y .

  have upper_bound: ∀ A∈sets M. N A ≤ density M f A
  proof (rule ccontr)
    assume ¬ ?thesis
    then obtain A where A[measurable]: A ∈ sets M and f_less_N: density M f
A < N A
      by (auto simp: not_le)
    then have pos_A: 0 < M A
      using ‹absolutely_continuous M N›[THEN absolutely_continuousD, OF A]
      by (auto simp: zero_less_iff_neq_zero)

    define b where b = (N A - density M f A) / M A / 2
    with f_less_N pos_A have 0 < b b ≠ top
    by (auto intro!: diff_gr0_ennreal simp: zero_less_iff_neq_zero diff_eq_0_iff_ennreal
```

ennreal_divide_eq_top_iff)

let *?f* = λ*x*. *f x* + *b*
have *nn_integral M f* ≠ *top*
using ‹*f* ∈ *G*›[*THEN G_D*, *of space M*] **by** (*auto simp*: *top_unique cong*: *nn_integral_cong*)
with ‹*b* ≠ *top*› **interpret** *Mf*: *finite_measure density M ?f*
by (*intro finite_measureI*)
(*auto simp*: *field_simps mult_indicator_subset ennreal_mult_eq_top_iff emeasure_density nn_integral_cmult_indicator nn_integral_add cong*: *nn_integral_cong*)

from *unsigned_Hahn_decomposition*[*of density M ?f N A*]
obtain *Y* **where** [*measurable*]: *Y* ∈ *sets M* **and** [*simp*]: *Y* ⊆ *A*
and *Y1*: ⋀*C*. *C* ∈ *sets M* ⟹ *C* ⊆ *Y* ⟹ *density M ?f C* ≤ *N C*
and *Y2*: ⋀*C*. *C* ∈ *sets M* ⟹ *C* ⊆ *A* ⟹ *C* ∩ *Y* = {} ⟹ *N C* ≤ *density M ?f C*
by *auto*

let *?f′* = λ*x*. *f x* + *b* ∗ *indicator Y x*
have *M Y* ≠ *0*
proof
assume *M Y* = *0*
then have *N Y* = *0*
using ‹*absolutely_continuous M N*›[*THEN absolutely_continuousD*, *of Y*] **by** *auto*
then have *N A* = *N* (*A* − *Y*)
by (*subst emeasure_Diff*) *auto*
also have ... ≤ *density M ?f* (*A* − *Y*)
by (*rule Y2*) *auto*
also have ... ≤ *density M ?f A* − *density M ?f Y*
by (*subst emeasure_Diff*) *auto*
also have ... ≤ *density M ?f A* − *0*
by (*intro ennreal_minus_mono*) *auto*
also have *density M ?f A* = *b* ∗ *M A* + *density M f A*
by (*simp add*: *emeasure_density field_simps mult_indicator_subset nn_integral_add nn_integral_cmult_indicator*)
also have ... < *N A*
using *f_less_N pos_A*
by (*cases density M f A*; *cases M A*; *cases N A*)
(*auto simp*: *b_def ennreal_less_iff ennreal_minus divide_ennreal ennreal_numeral*[*symmetric*] *ennreal_plus*[*symmetric*] *ennreal_mult*[*symmetric*] *field_simps simp del*: *ennreal_numeral ennreal_plus*)
finally show *False*
by *simp*
qed
then have *nn_integral M f* < *nn_integral M ?f′*
using ‹*0* < *b*› ‹*nn_integral M f* ≠ *top*›

by (*simp add*: *nn_integral_add nn_integral_cmult_indicator ennreal_zero_less_mult_iff zero_less_iff_neq_zero*)
moreover
have *?f′* ∈ *G*
unfolding *G_def*
proof *safe*
fix *X* **assume** [*measurable*]: *X* ∈ *sets M*
have (∫⁺ *x*. *?f′ x* ∗ *indicator X x* ∂*M*) = *density M f* (*X* − *Y*) + *density M ?f* (*X* ∩ *Y*)
by (*auto simp add*: *emeasure_density nn_integral_add*[*symmetric*] *split*: *split_indicator intro*!: *nn_integral_cong*)
also have ... ≤ *N* (*X* − *Y*) + *N* (*X* ∩ *Y*)
using *G_D*[*OF* ‹*f* ∈ *G*›] **by** (*intro add_mono Y1*) (*auto simp*: *emeasure_density*)
also have ... = *N X*
by (*subst plus_emeasure*) (*auto intro*!: *arg_cong2*[**where** *f*=*emeasure*])
finally show (∫⁺ *x*. *?f′ x* ∗ *indicator X x* ∂*M*) ≤ *N X* **.**
qed *simp*
then have *nn_integral M ?f′* ≤ *?y*
by (*rule SUP_upper*)
ultimately show *False*
by (*simp add*: *int_f_eq_y*)
qed
show *?thesis*
proof (*intro bexI*[*of* _ *f*] *measure_eqI conjI antisym*)
fix *A* **assume** *A* ∈ *sets* (*density M f*) **then show** *emeasure* (*density M f*) *A* ≤ *emeasure N A*
by (*auto simp*: *emeasure_density intro*!: *G_D*[*OF* ‹*f* ∈ *G*›])
next
fix *A* **assume** *A*: *A* ∈ *sets* (*density M f*) **then show** *emeasure N A* ≤ *emeasure* (*density M f*) *A*
using *upper_bound* **by** *auto*
qed *auto*
qed

lemma (**in** *finite_measure*) *split_space_into_finite_sets_and_rest*:
assumes *ac*: *absolutely_continuous M N* **and** *sets_eq*[*simp*]: *sets N* = *sets M*
shows ∃ *B*::*nat*⇒′*a set*. *disjoint_family B* ∧ *range B* ⊆ *sets M* ∧ (∀ *i*. *N* (*B i*) ≠ ∞) ∧
(∀ *A*∈*sets M*. *A* ∩ (⋃ *i*. *B i*) = {} ⟶ (*emeasure M A* = *0* ∧ *N A* = *0*) ∨ (*emeasure M A* > *0* ∧ *N A* = ∞))
proof −
let *?Q* = {*Q*∈*sets M*. *N Q* ≠ ∞}
let *?a* = *SUP Q*∈*?Q*. *emeasure M Q*
have {} ∈ *?Q* **by** *auto*
then have *Q_not_empty*: *?Q* ≠ {} **by** *blast*
have *?a* ≤ *emeasure M* (*space M*) **using** *sets.sets_into_space*
by (*auto intro*!: *SUP_least emeasure_mono*)
then have *?a* ≠ ∞

```
    using finite_emeasure_space
     by (auto simp: less_top[symmetric] top_unique simp del: SUP_eq_top_iff
Sup_eq_top_iff)
  from ennreal_SUP_countable_SUP [OF Q_not_empty, of emeasure M]
  obtain Q'' where range Q'' ⊆ emeasure M ` ?Q and a: ?a = (SUP i::nat. Q''
i)
    by auto
  then have ∀ i. ∃ Q'. Q'' i = emeasure M Q' ∧ Q' ∈ ?Q by auto
  from choice[OF this] obtain Q' where Q': ⋀i. Q'' i = emeasure M (Q' i) ⋀i.
Q' i ∈ ?Q
    by auto
  then have a_Lim: ?a = (SUP i. emeasure M (Q' i)) using a by simp
  let ?O = λn. ⋃ i≤n. Q' i
  have Union: (SUP i. emeasure M (?O i)) = emeasure M (⋃ i. ?O i)
  proof (rule SUP_emeasure_incseq[of ?O])
    show range ?O ⊆ sets M using Q' by auto
    show incseq ?O by (fastforce intro!: incseq_SucI)
  qed
  have Q'_sets[measurable]: ⋀i. Q' i ∈ sets M using Q' by auto
  have O_sets: ⋀i. ?O i ∈ sets M using Q' by auto
  then have O_in_G: ⋀i. ?O i ∈ ?Q
  proof (safe del: notI)
    fix i have Q' ` {..i} ⊆ sets M using Q' by auto
    then have N (?O i) ≤ (∑ i≤i. N (Q' i))
      by (simp add: emeasure_subadditive_finite)
    also have ... < ∞ using Q' by (simp add: less_top)
    finally show N (?O i) ≠ ∞ by simp
  qed auto
  have O_mono: ⋀n. ?O n ⊆ ?O (Suc n) by fastforce
  have a_eq: ?a = emeasure M (⋃ i. ?O i) unfolding Union[symmetric]
  proof (rule antisym)
    show ?a ≤ (SUP i. emeasure M (?O i)) unfolding a_Lim
      using Q' by (auto intro!: SUP_mono emeasure_mono)
    show (SUP i. emeasure M (?O i)) ≤ ?a
    proof (safe intro!: Sup_mono, unfold bex_simps)
      fix i
      have ∗: (⋃ (Q' ` {..i})) = ?O i by auto
      then show ∃ x. (x ∈ sets M ∧ N x ≠ ∞) ∧
        emeasure M (⋃ (Q' ` {..i})) ≤ emeasure M x
        using O_in_G[of i] by (auto intro!: exI[of _ ?O i])
    qed
  qed
  let ?O_0 = (⋃ i. ?O i)
  have ?O_0 ∈ sets M using Q' by auto
  have disjointed Q' i ∈ sets M for i
    using sets.range_disjointed_sets[of Q' M] using Q'_sets by (auto simp: sub-
set_eq)
  note Q_sets = this
  show ?thesis
```

```
  proof (intro bexI exI conjI ballI impI allI)
    show disjoint_family (disjointed Q')
      by (rule disjoint_family_disjointed)
    show range (disjointed Q') ⊆ sets M
      using Q'_sets by (intro sets.range_disjointed_sets) auto
    { fix A assume A: A ∈ sets M A ∩ (⋃ i. disjointed Q' i) = {}
      then have A1: A ∩ (⋃ i. Q' i) = {}
        unfolding UN_disjointed_eq by auto
      show emeasure M A = 0 ∧ N A = 0 ∨ 0 < emeasure M A ∧ N A = ∞
      proof (rule disjCI, simp)
        assume *: emeasure M A = 0 ∨ N A ≠ top
        show emeasure M A = 0 ∧ N A = 0
        proof (cases emeasure M A = 0)
          case True
          with ac A have N A = 0
            unfolding absolutely_continuous_def by auto
          with True show ?thesis by simp
        next
          case False
          with * have N A ≠ ∞ by auto
          with A have emeasure M ?O_0 + emeasure M A = emeasure M (?O_0
∪ A)
            using Q' A1 by (auto intro!: plus_emeasure simp: set_eq_iff)
          also have ... = (SUP i. emeasure M (?O i ∪ A))
        proof (rule SUP_emeasure_incseq[of λi. ?O i ∪ A, symmetric, simplified])
            show range (λi. ?O i ∪ A) ⊆ sets M
              using ‹N A ≠ ∞› O_sets A by auto
          qed (fastforce intro!: incseq_SucI)
          also have ... ≤ ?a
          proof (safe intro!: SUP_least)
            fix i have ?O i ∪ A ∈ ?Q
            proof (safe del: notI)
              show ?O i ∪ A ∈ sets M using O_sets A by auto
              from O_in_G[of i] have N (?O i ∪ A) ≤ N (?O i) + N A
                using emeasure_subadditive[of ?O i N A] A O_sets by auto
              with O_in_G[of i] show N (?O i ∪ A) ≠ ∞
                using ‹N A ≠ ∞› by (auto simp: top_unique)
            qed
            then show emeasure M (?O i ∪ A) ≤ ?a by (rule SUP_upper)
          qed
          finally have emeasure M A = 0
            unfolding a_eq using measure_nonneg[of M A] by (simp add: emea-
sure_eq_measure)
          with ‹emeasure M A ≠ 0› show ?thesis by auto
        qed
      qed }
    { fix i
      have N (disjointed Q' i) ≤ N (Q' i)
        by (auto intro!: emeasure_mono simp: disjointed_def)
```

```
    then show N (disjointed Q' i) ≠ ∞
      using Q'(2)[of i] by (auto simp: top_unique) }
  qed
qed

proposition (in finite_measure) Radon_Nikodym_finite_measure_infinite:
  assumes absolutely_continuous M N and sets_eq: sets N = sets M
  shows ∃f∈borel_measurable M. density M f = N
proof −
  from split_space_into_finite_sets_and_rest[OF assms]
  obtain Q :: nat ⇒ 'a set
    where Q: disjoint_family Q range Q ⊆ sets M
    and in_Q0: ⋀A. A ∈ sets M ⟹ A ∩ (⋃i. Q i) = {} ⟹ emeasure M A =
0 ∧ N A = 0 ∨ 0 < emeasure M A ∧ N A = ∞
    and Q_fin: ⋀i. N (Q i) ≠ ∞ by force
  from Q have Q_sets: ⋀i. Q i ∈ sets M by auto
  let ?N = λi. density N (indicator (Q i)) and ?M = λi. density M (indicator
(Q i))
  have ∀i. ∃f∈borel_measurable (?M i). density (?M i) f = ?N i
  proof (intro allI finite_measure.Radon_Nikodym_finite_measure)
    fix i
    from Q show finite_measure (?M i)
      by (auto intro!: finite_measureI cong: nn_integral_cong
            simp add: emeasure_density subset_eq sets_eq)
    from Q have emeasure (?N i) (space N) = emeasure N (Q i)
    by (simp add: sets_eq[symmetric] emeasure_density subset_eq cong: nn_integral_cong)
    with Q_fin show finite_measure (?N i)
      by (auto intro!: finite_measureI)
    show sets (?N i) = sets (?M i) by (simp add: sets_eq)
    have [measurable]: ⋀A. A ∈ sets M ⟹ A ∈ sets N by (simp add: sets_eq)
    show absolutely_continuous (?M i) (?N i)
      using ‹absolutely_continuous M N› ‹Q i ∈ sets M›
      by (auto simp: absolutely_continuous_def null_sets_density_iff sets_eq
            intro!: absolutely_continuous_AE[OF sets_eq])
  qed
  from choice[OF this[unfolded Bex_def]]
  obtain f where borel: ⋀i. f i ∈ borel_measurable M ⋀i x. 0 ≤ f i x
    and f_density: ⋀i. density (?M i) (f i) = ?N i
    by force
  { fix A i assume A: A ∈ sets M
    with Q borel have (∫⁺x. f i x * indicator (Q i ∩ A) x ∂M) = emeasure
(density (?M i) (f i)) A
      by (auto simp add: emeasure_density nn_integral_density subset_eq
            intro!: nn_integral_cong split: split_indicator)
    also have ... = emeasure N (Q i ∩ A)
      using A Q by (simp add: f_density emeasure_restricted subset_eq sets_eq)
    finally have emeasure N (Q i ∩ A) = (∫⁺x. f i x * indicator (Q i ∩ A) x
∂M) .. }
  note integral_eq = this
```

let *?f* = $\lambda x.$ ($\sum i.$ *f i x* $*$ *indicator* (*Q i*) *x*) + ∞ $*$ *indicator* (*space M* − ($\bigcup i.$ *Q i*)) *x*
show *?thesis*
proof (*safe intro*!: *bexI*[*of* _ *?f*])
show *?f* ∈ *borel_measurable M* **using** *borel Q_sets*
by (*auto intro*!: *measurable_If*)
show *density M ?f* = *N*
proof (*rule measure_eqI*)
fix *A* **assume** *A* ∈ *sets* (*density M ?f*)
then have *A* ∈ *sets M* **by** *simp*
have *Qi*: $\bigwedge i.$ *Q i* ∈ *sets M* **using** *Q* **by** *auto*
have [*intro*,*simp*]: $\bigwedge i.$ ($\lambda x.$ *f i x* $*$ *indicator* (*Q i* ∩ *A*) *x*) ∈ *borel_measurable M*
$\bigwedge i.$ *AE x in M*. *0* ≤ *f i x* $*$ *indicator* (*Q i* ∩ *A*) *x*
using *borel Qi* ‹*A* ∈ *sets M*› **by** *auto*
have ($\int^+ x.$ *?f x* $*$ *indicator A x* ∂*M*) = ($\int^+ x.$ ($\sum i.$ *f i x* $*$ *indicator* (*Q i* ∩ *A*) *x*) + ∞ $*$ *indicator* ((*space M* − ($\bigcup i.$ *Q i*)) ∩ *A*) *x* ∂*M*)
using *borel* **by** (*intro nn_integral_cong*) (*auto simp*: *indicator_def*)
also have ... = ($\int^+ x.$ ($\sum i.$ *f i x* $*$ *indicator* (*Q i* ∩ *A*) *x*) ∂*M*) + ∞ $*$ *emeasure M* ((*space M* − ($\bigcup i.$ *Q i*)) ∩ *A*)
using *borel Qi* ‹*A* ∈ *sets M*›
by (*subst nn_integral_add*)
(*auto simp add*: *nn_integral_cmult_indicator sets.Int intro*!: *suminf_0_le*)
also have ... = ($\sum i.$ *N* (*Q i* ∩ *A*)) + ∞ $*$ *emeasure M* ((*space M* − ($\bigcup i.$ *Q i*)) ∩ *A*)
by (*subst integral_eq*[*OF* ‹*A* ∈ *sets M*›], *subst nn_integral_suminf*) *auto*
finally have ($\int^+ x.$ *?f x* $*$ *indicator A x* ∂*M*) = ($\sum i.$ *N* (*Q i* ∩ *A*)) + ∞ $*$ *emeasure M* ((*space M* − ($\bigcup i.$ *Q i*)) ∩ *A*) **.**
moreover have ($\sum i.$ *N* (*Q i* ∩ *A*)) = *N* (($\bigcup i.$ *Q i*) ∩ *A*)
using *Q Q_sets* ‹*A* ∈ *sets M*›
by (*subst suminf_emeasure*) (*auto simp*: *disjoint_family_on_def sets_eq*)
moreover
have (*space M* − ($\bigcup x.$ *Q x*)) ∩ *A* ∩ ($\bigcup x.$ *Q x*) = {}
by *auto*
then have ∞ $*$ *emeasure M* ((*space M* − ($\bigcup i.$ *Q i*)) ∩ *A*) = *N* ((*space M* − ($\bigcup i.$ *Q i*)) ∩ *A*)
using *in_Q0*[*of* (*space M* − ($\bigcup i.$ *Q i*)) ∩ *A*] ‹*A* ∈ *sets M*› *Q* **by** (*auto simp*: *ennreal_top_mult*)
moreover have (*space M* − ($\bigcup i.$ *Q i*)) ∩ *A* ∈ *sets M* (($\bigcup i.$ *Q i*) ∩ *A*) ∈ *sets M*
using *Q_sets* ‹*A* ∈ *sets M*› **by** *auto*
moreover have (($\bigcup i.$ *Q i*) ∩ *A*) ∪ ((*space M* − ($\bigcup i.$ *Q i*)) ∩ *A*) = *A* (($\bigcup i.$ *Q i*) ∩ *A*) ∩ ((*space M* − ($\bigcup i.$ *Q i*)) ∩ *A*) = {}
using ‹*A* ∈ *sets M*› *sets.sets_into_space* **by** *auto*
ultimately have *N A* = ($\int^+ x.$ *?f x* $*$ *indicator A x* ∂*M*)
using *plus_emeasure*[*of* ($\bigcup i.$ *Q i*) ∩ *A N* (*space M* − ($\bigcup i.$ *Q i*)) ∩ *A*] **by** (*simp add*: *sets_eq*)
with ‹*A* ∈ *sets M*› *borel Q* **show** *emeasure* (*density M ?f*) *A* = *N A*
by (*auto simp*: *subset_eq emeasure_density*)

```
    qed (simp add: sets_eq)
  qed
qed

theorem (in sigma_finite_measure) Radon_Nikodym:
  assumes ac: absolutely_continuous M N assumes sets_eq: sets N = sets M
  shows ∃f ∈ borel_measurable M. density M f = N
proof −
  from Ex_finite_integrable_function
  obtain h where finite: integral^N M h ≠ ∞ and
    borel: h ∈ borel_measurable M and
    nn: ⋀x. 0 ≤ h x and
    pos: ⋀x. x ∈ space M ⟹ 0 < h x and
    ⋀x. x ∈ space M ⟹ h x < ∞ by auto
  let ?T = λA. (∫⁺x. h x * indicator A x ∂M)
  let ?MT = density M h
  from borel finite nn interpret T: finite_measure ?MT
   by (auto intro!: finite_measureI cong: nn_integral_cong simp: emeasure_density)
  have absolutely_continuous ?MT N sets N = sets ?MT
  proof (unfold absolutely_continuous_def, safe)
    fix A assume A ∈ null_sets ?MT
    with borel have A ∈ sets M AE x in M. x ∈ A ⟶ h x ≤ 0
      by (auto simp add: null_sets_density_iff)
    with pos sets.sets_into_space have AE x in M. x ∉ A
      by (elim eventually_mono) (auto simp: not_le[symmetric])
    then have A ∈ null_sets M
      using ‹A ∈ sets M› by (simp add: AE_iff_null_sets)
    with ac show A ∈ null_sets N
      by (auto simp: absolutely_continuous_def)
  qed (auto simp add: sets_eq)
  from T.Radon_Nikodym_finite_measure_infinite[OF this]
  obtain f where f_borel: f ∈ borel_measurable M density ?MT f = N by auto
  with nn borel show ?thesis
    by (auto intro!: bexI[of _ λx. h x * f x] simp: density_density_eq)
qed
```

6.15.3 Uniqueness of densities

```
lemma finite_density_unique:
  assumes borel: f ∈ borel_measurable M g ∈ borel_measurable M
  assumes pos: AE x in M. 0 ≤ f x AE x in M. 0 ≤ g x
  and fin: integral^N M f ≠ ∞
  shows density M f = density M g ⟷ (AE x in M. f x = g x)
proof (intro iffI ballI)
  fix A assume eq: AE x in M. f x = g x
  with borel show density M f = density M g
    by (auto intro: density_cong)
next
  let ?P = λf A. ∫⁺ x. f x * indicator A x ∂M
```

assume *density M f = density M g*
with *borel* **have** *eq*: $\forall A \in sets\ M.\ ?P\ f\ A = ?P\ g\ A$
by (*simp add*: *emeasure_density*[*symmetric*])
from *this*[*THEN bspec*, *OF sets.top*] *fin*
have *g_fin*: $integral^N\ M\ g \neq \infty$ **by** (*simp cong*: *nn_integral_cong*)
{ fix *f g* **assume** *borel*: *f ∈ borel_measurable M g ∈ borel_measurable M*
and *pos*: *AE x in M. 0 ≤ f x AE x in M. 0 ≤ g x*
and *g_fin*: $integral^N\ M\ g \neq \infty$ **and** *eq*: $\forall A \in sets\ M.\ ?P\ f\ A = ?P\ g\ A$
let *?N* = {*x∈space M. g x < f x*}
have *N*: *?N ∈ sets M* **using** *borel* **by** *simp*
have *?P g ?N ≤ integral*^{N} *M g* **using** *pos*
by (*intro nn_integral_mono_AE*) (*auto split*: *split_indicator*)
then have *Pg_fin*: *?P g ?N ≠ ∞* **using** *g_fin* **by** (*auto simp*: *top_unique*)
have *?P* (*λx. (f x − g x)*) *?N* = ($\int^+ x.$ *f x ∗ indicator ?N x − g x ∗ indicator ?N x ∂M*)
by (*auto intro*!: *nn_integral_cong simp*: *indicator_def*)
also have *... = ?P f ?N − ?P g ?N*
proof (*rule nn_integral_diff*)
show (*λx. f x ∗ indicator ?N x*) *∈ borel_measurable M* (*λx. g x ∗ indicator ?N x*) *∈ borel_measurable M*
using *borel N* **by** *auto*
show *AE x in M. g x ∗ indicator ?N x ≤ f x ∗ indicator ?N x*
using *pos* **by** (*auto split*: *split_indicator*)
qed *fact*
also have *... = 0*
unfolding *eq*[*THEN bspec*, *OF N*] **using** *Pg_fin* **by** *auto*
finally have *AE x in M. f x ≤ g x*
using *pos borel nn_integral_PInf_AE*[*OF borel*(*2*) *g_fin*]
by (*subst* (*asm*) *nn_integral_0_iff_AE*)
(*auto split*: *split_indicator simp*: *not_less ennreal_minus_eq_0*) **}**
from *this*[*OF borel pos g_fin eq*] *this*[*OF borel*(*2*,*1*) *pos*(*2*,*1*) *fin*] *eq*
show *AE x in M. f x = g x* **by** *auto*
qed

lemma (**in** *finite_measure*) *density_unique_finite_measure*:
assumes *borel*: *f ∈ borel_measurable M f′ ∈ borel_measurable M*
assumes *pos*: *AE x in M. 0 ≤ f x AE x in M. 0 ≤ f′ x*
assumes *f*: $\bigwedge A.\ A \in sets\ M \Longrightarrow (\int^+ x.\ f\ x * indicator\ A\ x\ \partial M) = (\int^+ x.\ f'\ x * indicator\ A\ x\ \partial M)$
(**is** $\bigwedge A.\ A \in sets\ M \Longrightarrow ?P\ f\ A = ?P\ f'\ A$)
shows *AE x in M. f x = f′ x*
proof −
let *?D* = *λf. density M f*
let *?N* = *λA. ?P f A* **and** *?N′* = *λA. ?P f′ A*
let *?f* = *λA x. f x ∗ indicator A x* **and** *?f′* = *λA x. f′ x ∗ indicator A x*

have *ac*: *absolutely_continuous M* (*density M f*) *sets* (*density M f*) = *sets M*
using *borel* **by** (*auto intro*!: *absolutely_continuousI_density*)
from *split_space_into_finite_sets_and_rest*[*OF this*]

obtain $Q :: nat \Rightarrow 'a\ set$
where Q: *disjoint_family* Q *range* $Q \subseteq sets\ M$
and *in_Q0*: $\bigwedge A.\ A \in sets\ M \Longrightarrow A \cap (\bigcup i.\ Q\ i) = \{\} \Longrightarrow emeasure\ M\ A = 0 \wedge ?D\ f\ A = 0 \vee 0 < emeasure\ M\ A \wedge ?D\ f\ A = \infty$
and *Q_fin*: $\bigwedge i.\ ?D\ f\ (Q\ i) \neq \infty$ **by** *force*
with *borel pos* **have** *in_Q0*: $\bigwedge A.\ A \in sets\ M \Longrightarrow A \cap (\bigcup i.\ Q\ i) = \{\} \Longrightarrow emeasure\ M\ A = 0 \wedge ?N\ A = 0 \vee 0 < emeasure\ M\ A \wedge ?N\ A = \infty$
and *Q_fin*: $\bigwedge i.\ ?N\ (Q\ i) \neq \infty$ **by** (*auto simp*: *emeasure_density subset_eq*)

from Q **have** *Q_sets*[*measurable*]: $\bigwedge i.\ Q\ i \in sets\ M$ **by** *auto*
let $?D = \{x \in space\ M.\ f\ x \neq f'\ x\}$
have $?D \in sets\ M$ **using** *borel* **by** *auto*
have $*$: $\bigwedge i\ x\ A.\ \bigwedge y{::}ennreal.\ y * indicator\ (Q\ i)\ x * indicator\ A\ x = y * indicator\ (Q\ i \cap A)\ x$
unfolding *indicator_def* **by** *auto*
have $\forall i.\ AE\ x\ in\ M.\ ?f\ (Q\ i)\ x = ?f'\ (Q\ i)\ x$ **using** *borel Q_fin Q pos*
by (*intro finite_density_unique*[*THEN iffD1*] *allI*)
(*auto intro*!: *f measure_eqI simp*: *emeasure_density* $*$ *subset_eq*)
moreover have $AE\ x\ in\ M.\ ?f\ (space\ M - (\bigcup i.\ Q\ i))\ x = ?f'\ (space\ M - (\bigcup i.\ Q\ i))\ x$
proof (*rule AE_I'*)
{ **fix** $f :: 'a \Rightarrow ennreal$ **assume** *borel*: $f \in borel\_measurable\ M$
and *eq*: $\bigwedge A.\ A \in sets\ M \Longrightarrow ?N\ A = (\int^+ x.\ f\ x * indicator\ A\ x\ \partial M)$
let $?A = \lambda i.\ (space\ M - (\bigcup i.\ Q\ i)) \cap \{x \in space\ M.\ f\ x < (i{::}nat)\}$
have $(\bigcup i.\ ?A\ i) \in null\_sets\ M$
proof (*rule null_sets_UN*)
fix $i :: nat$ **have** $?A\ i \in sets\ M$
using *borel* **by** *auto*
have $?N\ (?A\ i) \leq (\int^+ x.\ (i{::}ennreal) * indicator\ (?A\ i)\ x\ \partial M)$
unfolding *eq*[*OF* ‹$?A\ i \in sets\ M$›]
by (*auto intro*!: *nn_integral_mono simp*: *indicator_def*)
also have $\ldots = i * emeasure\ M\ (?A\ i)$
using ‹$?A\ i \in sets\ M$› **by** (*auto intro*!: *nn_integral_cmult_indicator*)
also have $\ldots < \infty$ **using** *emeasure_real*[*of* $?A\ i$] **by** (*auto simp*: *ennreal_mult_less_top of_nat_less_top*)
finally have $?N\ (?A\ i) \neq \infty$ **by** *simp*
then show $?A\ i \in null\_sets\ M$ **using** *in_Q0*[*OF* ‹$?A\ i \in sets\ M$›] ‹$?A\ i \in sets\ M$› **by** *auto*
qed
also have $(\bigcup i.\ ?A\ i) = (space\ M - (\bigcup i.\ Q\ i)) \cap \{x \in space\ M.\ f\ x \neq \infty\}$
by (*auto simp*: *ennreal_Ex_less_of_nat less_top*[*symmetric*])
finally have $(space\ M - (\bigcup i.\ Q\ i)) \cap \{x \in space\ M.\ f\ x \neq \infty\} \in null\_sets\ M$ **by** *simp* }
from *this*[*OF borel*(*1*) *refl*] *this*[*OF borel*(*2*) f]
have $(space\ M - (\bigcup i.\ Q\ i)) \cap \{x \in space\ M.\ f\ x \neq \infty\} \in null\_sets\ M\ (space\ M - (\bigcup i.\ Q\ i)) \cap \{x \in space\ M.\ f'\ x \neq \infty\} \in null\_sets\ M$ **by** *simp_all*
then show $((space\ M - (\bigcup i.\ Q\ i)) \cap \{x \in space\ M.\ f\ x \neq \infty\}) \cup ((space\ M - (\bigcup i.\ Q\ i)) \cap \{x \in space\ M.\ f'\ x \neq \infty\}) \in null\_sets\ M$ **by** (*rule null_sets.Un*)
show $\{x \in space\ M.\ ?f\ (space\ M - (\bigcup i.\ Q\ i))\ x \neq ?f'\ (space\ M - (\bigcup i.\ Q$

```
i)) x} ⊆
      ((space M − (⋃i. Q i)) ∩ {x∈space M. f x ≠ ∞}) ∪ ((space M − (⋃i. Q
i)) ∩ {x∈space M. f' x ≠ ∞}) by (auto simp: indicator_def)
  qed
  moreover have AE x in M. (?f (space M − (⋃i. Q i)) x = ?f' (space M −
(⋃i. Q i)) x) ⟶ (∀i. ?f (Q i) x = ?f' (Q i) x) ⟶
    ?f (space M) x = ?f' (space M) x
    by (auto simp: indicator_def)
  ultimately have AE x in M. ?f (space M) x = ?f' (space M) x
    unfolding AE_all_countable[symmetric]
    by eventually_elim (auto split: if_split_asm simp: indicator_def)
  then show AE x in M. f x = f' x by auto
qed

proposition (in sigma_finite_measure) density_unique:
  assumes f: f ∈ borel_measurable M
  assumes f': f' ∈ borel_measurable M
  assumes density_eq: density M f = density M f'
  shows AE x in M. f x = f' x
proof −
  obtain h where h_borel: h ∈ borel_measurable M
    and fin: integral^N M h ≠ ∞ and pos: ⋀x. x ∈ space M ⟹ 0 < h x ∧ h x <
∞ ⋀x. 0 ≤ h x
    using Ex_finite_integrable_function by auto
  then have h_nn: AE x in M. 0 ≤ h x by auto
  let ?H = density M h
  interpret h: finite_measure ?H
    using fin h_borel pos
    by (intro finite_measureI) (simp cong: nn_integral_cong emeasure_density
add: fin)
  let ?fM = density M f
  let ?f'M = density M f'
  { fix A assume A ∈ sets M
    then have {x ∈ space M. h x * indicator A x ≠ 0} = A
      using pos(1) sets.sets_into_space by (force simp: indicator_def)
    then have (∫+x. h x * indicator A x ∂M) = 0 ⟷ A ∈ null_sets M
      using h_borel ‹A ∈ sets M› h_nn by (subst nn_integral_0_iff) auto }
  note h_null_sets = this
  { fix A assume A ∈ sets M
    have (∫+x. f x * (h x * indicator A x) ∂M) = (∫+x. h x * indicator A x
∂?fM)
      using ‹A ∈ sets M› h_borel h_nn f f'
      by (intro nn_integral_density[symmetric]) auto
    also have ... = (∫+x. h x * indicator A x ∂?f'M)
      by (simp_all add: density_eq)
    also have ... = (∫+x. f' x * (h x * indicator A x) ∂M)
      using ‹A ∈ sets M› h_borel h_nn f f'
      by (intro nn_integral_density) auto
    finally have (∫+x. h x * (f x * indicator A x) ∂M) = (∫+x. h x * (f' x *
```

indicator A x) *∂M*)
by (*simp add*: *ac_simps*)
then have (∫⁺*x*. (*f x* ∗ *indicator A x*) *∂?H*) = (∫⁺*x*. (*f' x* ∗ *indicator A x*) *∂?H*)
using ‹*A* ∈ *sets M*› *h_borel h_nn f f'*
by (*subst* (*asm*) (*1 2*) *nn_integral_density*[*symmetric*]) *auto* **}**
then have *AE x in ?H*. *f x* = *f' x* **using** *h_borel h_nn f f'*
by (*intro h.density_unique_finite_measure absolutely_continuous_AE*[*of M*]) *auto*
with *AE_space*[*of M*] *pos* **show** *AE x in M*. *f x* = *f' x*
unfolding *AE_density*[*OF h_borel*] **by** *auto*
qed

lemma (**in** *sigma_finite_measure*) *density_unique_iff*:
assumes *f*: *f* ∈ *borel_measurable M* **and** *f'*: *f'* ∈ *borel_measurable M*
shows *density M f* = *density M f'* ⟷ (*AE x in M*. *f x* = *f' x*)
using *density_unique*[*OF assms*] *density_cong*[*OF f f'*] **by** *auto*

lemma *sigma_finite_density_unique*:
assumes *borel*: *f* ∈ *borel_measurable M g* ∈ *borel_measurable M*
and *fin*: *sigma_finite_measure* (*density M f*)
shows *density M f* = *density M g* ⟷ (*AE x in M*. *f x* = *g x*)
proof
assume *AE x in M*. *f x* = *g x* **with** *borel* **show** *density M f* = *density M g*
by (*auto intro*: *density_cong*)
next
assume *eq*: *density M f* = *density M g*
interpret *f*: *sigma_finite_measure density M f* **by** *fact*
from *f.sigma_finite_incseq* **obtain** *A* **where** *cover*: *range A* ⊆ *sets* (*density M f*)
⋃ (*range A*) = *space* (*density M f*)
⋀*i*. *emeasure* (*density M f*) (*A i*) ≠ ∞
incseq A
by *auto*
have *AE x in M*. ∀ *i*. *x* ∈ *A i* ⟶ *f x* = *g x*
unfolding *AE_all_countable*
proof
fix *i*
have *density* (*density M f*) (*indicator* (*A i*)) = *density* (*density M g*) (*indicator* (*A i*))
unfolding *eq* **..**
moreover have (∫⁺*x*. *f x* ∗ *indicator* (*A i*) *x* *∂M*) ≠ ∞
using *cover*(*1*) *cover*(*3*)[*of i*] *borel* **by** (*auto simp*: *emeasure_density subset_eq*)
ultimately have *AE x in M*. *f x* ∗ *indicator* (*A i*) *x* = *g x* ∗ *indicator* (*A i*) *x*
using *borel cover*(*1*)
by (*intro finite_density_unique*[*THEN iffD1*]) (*auto simp*: *density_density_eq subset_eq*)
then show *AE x in M*. *x* ∈ *A i* ⟶ *f x* = *g x*

```
    by auto
  qed
  with AE_space show AE x in M. f x = g x
    apply eventually_elim
    using cover(2)[symmetric]
    apply auto
    done
qed

lemma (in sigma_finite_measure) sigma_finite_iff_density_finite':
  assumes f: f ∈ borel_measurable M
  shows sigma_finite_measure (density M f) ⟷ (AE x in M. f x ≠ ∞)
    (is sigma_finite_measure ?N ⟷ _)
proof
  assume sigma_finite_measure ?N
  then interpret N: sigma_finite_measure ?N .
  from N.Ex_finite_integrable_function obtain h where
    h: h ∈ borel_measurable M integral^N ?N h ≠ ∞ and
    fin: ∀ x∈space M. 0 < h x ∧ h x < ∞
    by auto
  have AE x in M. f x * h x ≠ ∞
  proof (rule AE_I')
    have integral^N ?N h = (∫⁺x. f x * h x ∂M)
      using f h by (auto intro!: nn_integral_density)
    then have (∫⁺x. f x * h x ∂M) ≠ ∞
      using h(2) by simp
    then show (λx. f x * h x) -` {∞} ∩ space M ∈ null_sets M
     using f h(1) by (auto intro!: nn_integral_PInf[unfolded infinity_ennreal_def]
borel_measurable_vimage)
  qed auto
  then show AE x in M. f x ≠ ∞
   using fin by (auto elim!: AE_Ball_mp simp: less_top ennreal_mult_less_top)
next
  assume AE: AE x in M. f x ≠ ∞
  from sigma_finite obtain Q :: nat ⇒ 'a set
    where Q: range Q ⊆ sets M ⋃ (range Q) = space M ⋀i. emeasure M (Q i)
≠ ∞
    by auto
  define A where A i =
    f -` (case i of 0 ⇒ {∞} | Suc n ⇒ {.. ennreal(of_nat (Suc n))}) ∩ space M
for i
  { fix i j have A i ∩ Q j ∈ sets M
    unfolding A_def using f Q
    apply (rule_tac sets.Int)
    by (cases i) (auto intro: measurable_sets[OF f(1)]) }
  note A_in_sets = this

  show sigma_finite_measure ?N
  proof (standard, intro exI conjI ballI)
```

```
    show countable (range (λ(i, j). A i ∩ Q j))
      by auto
    show range (λ(i, j). A i ∩ Q j) ⊆ sets (density M f)
      using A_in_sets by auto
  next
    have ⋃(range (λ(i, j). A i ∩ Q j)) = (⋃i j. A i ∩ Q j)
      by auto
    also have ... = (⋃i. A i) ∩ space M using Q by auto
    also have (⋃i. A i) = space M
    proof safe
      fix x assume x: x ∈ space M
      show x ∈ (⋃i. A i)
      proof (cases f x rule: ennreal_cases)
        case top with x show ?thesis unfolding A_def by (auto intro: exI[of _
0])
      next
        case (real r)
        with ennreal_Ex_less_of_nat[of f x] obtain n :: nat where f x < n
          by auto
        also have n < (Suc n :: ennreal)
          by simp
        finally show ?thesis
          using x real by (auto simp: A_def ennreal_of_nat_eq_real_of_nat intro!:
exI[of _ Suc n])
      qed
    qed (auto simp: A_def)
    finally show ⋃(range (λ(i, j). A i ∩ Q j)) = space ?N by simp
  next
    fix X assume X ∈ range (λ(i, j). A i ∩ Q j)
    then obtain i j where [simp]:X = A i ∩ Q j by auto
    have (∫⁺x. f x * indicator (A i ∩ Q j) x ∂M) ≠ ∞
    proof (cases i)
      case 0
      have AE x in M. f x * indicator (A i ∩ Q j) x = 0
        using AE by (auto simp: A_def ‹i = 0›)
      from nn_integral_cong_AE[OF this] show ?thesis by simp
    next
      case (Suc n)
      then have (∫⁺x. f x * indicator (A i ∩ Q j) x ∂M) ≤
        (∫⁺x. (Suc n :: ennreal) * indicator (Q j) x ∂M)
       by (auto intro!: nn_integral_mono simp: indicator_def A_def ennreal_of_nat_eq_real_of_nat)
      also have ... = Suc n * emeasure M (Q j)
        using Q by (auto intro!: nn_integral_cmult_indicator)
      also have ... < ∞
        using Q by (auto simp: ennreal_mult_less_top less_top of_nat_less_top)
      finally show ?thesis by simp
    qed
    then show emeasure ?N X ≠ ∞
      using A_in_sets Q f by (auto simp: emeasure_density)
```

qed
qed

lemma (**in** *sigma_finite_measure*) *sigma_finite_iff_density_finite*:
f ∈ *borel_measurable M* ⟹ *sigma_finite_measure* (*density M f*) ⟷ (*AE x in M*. *f x* ≠ ∞)
by (*subst sigma_finite_iff_density_finite'*)
(*auto simp*: *max_def intro*!: *measurable_If*)

6.15.4 Radon-Nikodym derivative

definition *RN_deriv* :: *'a measure* ⇒ *'a measure* ⇒ *'a* ⇒ *ennreal* **where**
RN_deriv M N =
(*if* ∃*f*. *f* ∈ *borel_measurable M* ∧ *density M f* = *N*
then SOME f. *f* ∈ *borel_measurable M* ∧ *density M f* = *N*
else (λ_. *0*))

lemma *RN_derivI*:
assumes *f* ∈ *borel_measurable M density M f* = *N*
shows *density M* (*RN_deriv M N*) = *N*
proof −
have ∗: ∃*f*. *f* ∈ *borel_measurable M* ∧ *density M f* = *N*
using *assms* **by** *auto*
then have *density M* (*SOME f*. *f* ∈ *borel_measurable M* ∧ *density M f* = *N*) = *N*
by (*rule someI2_ex*) *auto*
with ∗ **show** *?thesis*
by (*auto simp*: *RN_deriv_def*)
qed

lemma *borel_measurable_RN_deriv*[*measurable*]: *RN_deriv M N* ∈ *borel_measurable M*
proof −
{ **assume** *ex*: ∃*f*. *f* ∈ *borel_measurable M* ∧ *density M f* = *N*
have *1*: (*SOME f*. *f* ∈ *borel_measurable M* ∧ *density M f* = *N*) ∈ *borel_measurable M*
using *ex* **by** (*rule someI2_ex*) *auto* }
from *this* **show** *?thesis*
by (*auto simp*: *RN_deriv_def*)
qed

lemma *density_RN_deriv_density*:
assumes *f*: *f* ∈ *borel_measurable M*
shows *density M* (*RN_deriv M* (*density M f*)) = *density M f*
by (*rule RN_derivI*[*OF f*]) *simp*

lemma (**in** *sigma_finite_measure*) *density_RN_deriv*:
absolutely_continuous M N ⟹ *sets N* = *sets M* ⟹ *density M* (*RN_deriv M N*) = *N*

by (*metis RN_derivI Radon_Nikodym*)

lemma (**in** *sigma_finite_measure*) *RN_deriv_nn_integral*:
assumes *N*: *absolutely_continuous M N sets N = sets M*
and *f*: *f ∈ borel_measurable M*
shows *integral*^{N} *N f = (∫*$^+$*x. RN_deriv M N x * f x ∂M)*
proof −
have *integral*^{N} *N f = integral*^{N} *(density M (RN_deriv M N)) f*
using *N* **by** (*simp add*: *density_RN_deriv*)
also have *... = (∫*$^+$*x. RN_deriv M N x * f x ∂M)*
using *f* **by** (*simp add*: *nn_integral_density*)
finally show *?thesis* **by** *simp*
qed

lemma (**in** *sigma_finite_measure*) *RN_deriv_unique*:
assumes *f*: *f ∈ borel_measurable M*
and *eq*: *density M f = N*
shows *AE x in M. f x = RN_deriv M N x*
unfolding *eq*[*symmetric*]
by (*intro density_unique_iff*[*THEN iffD1*] *f borel_measurable_RN_deriv*
density_RN_deriv_density[*symmetric*])

lemma *RN_deriv_unique_sigma_finite*:
assumes *f*: *f ∈ borel_measurable M*
and *eq*: *density M f = N* **and** *fin*: *sigma_finite_measure N*
shows *AE x in M. f x = RN_deriv M N x*
using *fin* **unfolding** *eq*[*symmetric*]
by (*intro sigma_finite_density_unique*[*THEN iffD1*] *f borel_measurable_RN_deriv*
density_RN_deriv_density[*symmetric*])

lemma (**in** *sigma_finite_measure*) *RN_deriv_distr*:
fixes *T* :: *'a ⇒ 'b*
assumes *T*: *T ∈ measurable M M'* **and** *T'*: *T' ∈ measurable M' M*
and *inv*: *∀x∈space M. T' (T x) = x*
and *ac*[*simp*]: *absolutely_continuous (distr M M' T) (distr N M' T)*
and *N*: *sets N = sets M*
shows *AE x in M. RN_deriv (distr M M' T) (distr N M' T) (T x) = RN_deriv M N x*
proof (*rule RN_deriv_unique*)
have [*simp*]: *sets N = sets M* **by** *fact*
note *sets_eq_imp_space_eq*[*OF N*, *simp*]
have *measurable_N*[*simp*]: *⋀M'. measurable N M' = measurable M M'* **by** (*auto simp*: *measurable_def*)
{ fix *A* **assume** *A ∈ sets M*
with *inv T T' sets.sets_into_space*[*OF this*]
have *T −‘ T' −‘ A ∩ T −‘ space M' ∩ space M = A*
by (*auto simp*: *measurable_def*) **}**
note *eq = this*[*simp*]
{ fix *A* **assume** *A ∈ sets M*

```
    with inv T T' sets.sets_into_space[OF this]
    have (T' ∘ T) -` A ∩ space M = A
      by (auto simp: measurable_def) }
  note eq2 = this[simp]
  let ?M' = distr M M' T and ?N' = distr N M' T
  interpret M': sigma_finite_measure ?M'
  proof
    from sigma_finite_countable obtain F
      where F: countable F ∧ F ⊆ sets M ∧ ⋃ F = space M ∧ (∀ a∈F. emeasure
M a ≠ ∞) ..
    show ∃ A. countable A ∧ A ⊆ sets (distr M M' T) ∧ ⋃ A = space (distr M M'
T) ∧ (∀ a∈A. emeasure (distr M M' T) a ≠ ∞)
    proof (intro exI conjI ballI)
      show *: (λA. T' -` A ∩ space ?M') ` F ⊆ sets ?M'
        using F T' by (auto simp: measurable_def)
      show ⋃ ((λA. T' -` A ∩ space ?M')`F) = space ?M'
        using F T'[THEN measurable_space] by (auto simp: set_eq_iff)
    next
      fix X assume X ∈ (λA. T' -` A ∩ space ?M')`F
       then obtain A where [simp]: X = T' -` A ∩ space ?M' and A ∈ F by
auto
      have X ∈ sets M' using F T' ‹A∈F› by auto
      moreover
      have Fi: A ∈ sets M using F ‹A∈F› by auto
      ultimately show emeasure ?M' X ≠ ∞
        using F T T' ‹A∈F› by (simp add: emeasure_distr)
    qed (use F in auto)
  qed
  have (RN_deriv ?M' ?N') ∘ T ∈ borel_measurable M
    using T ac by measurable
  then show (λx. RN_deriv ?M' ?N' (T x)) ∈ borel_measurable M
    by (simp add: comp_def)

  have N = distr N M (T' ∘ T)
    by (subst measure_of_of_measure[of N, symmetric])
     (auto simp add: distr_def sets.sigma_sets_eq intro!: measure_of_eq sets.space_closed)
  also have ... = distr (distr N M' T) M T'
    using T T' by (simp add: distr_distr)
  also have ... = distr (density (distr M M' T) (RN_deriv (distr M M' T) (distr
N M' T))) M T'
    using ac by (simp add: M'.density_RN_deriv)
  also have ... = density M (RN_deriv (distr M M' T) (distr N M' T) ∘ T)
    by (simp add: distr_density_distr[OF T T', OF inv])
  finally show density M (λx. RN_deriv (distr M M' T) (distr N M' T) (T x))
= N
    by (simp add: comp_def)
qed

lemma (in sigma_finite_measure) RN_deriv_finite:
```

assumes N: *sigma_finite_measure N* **and** *ac*: *absolutely_continuous M N sets N = sets M*
shows *AE x in M. RN_deriv M N x ≠ ∞*
proof −
interpret N: *sigma_finite_measure N* **by** *fact*
from N **show** *?thesis*
using *sigma_finite_iff_density_finite*[*OF borel_measurable_RN_deriv*, *of N*] *density_RN_deriv*[*OF ac*]
by *simp*
qed

lemma (**in** *sigma_finite_measure*)
assumes N: *sigma_finite_measure N* **and** *ac*: *absolutely_continuous M N sets N = sets M*
and f: $f \in$ *borel_measurable M*
shows *RN_deriv_integrable*: *integrable N f* ⟷
integrable M (λ*x*. *enn2real* (*RN_deriv M N x*) ∗ *f x*) (**is** *?integrable*)
and *RN_deriv_integral*: $\textit{integral}^L$ *N f* = (∫ *x*. *enn2real* (*RN_deriv M N x*) ∗ *f x* ∂*M*) (**is** *?integral*)
proof −
note *ac*(*2*)[*simp*] **and** *sets_eq_imp_space_eq*[*OF ac*(*2*), *simp*]
interpret N: *sigma_finite_measure N* **by** *fact*

have *eq*: *density M* (*RN_deriv M N*) = *density M* (λ*x*. *enn2real* (*RN_deriv M N x*))
proof (*rule density_cong*)
from *RN_deriv_finite*[*OF assms*(*1*,*2*,*3*)]
show *AE x in M. RN_deriv M N x = ennreal* (*enn2real* (*RN_deriv M N x*))
by *eventually_elim* (*auto simp*: *less_top*)
qed (*insert ac*, *auto*)

show *?integrable*
apply (*subst density_RN_deriv*[*OF ac*, *symmetric*])
unfolding *eq*
apply (*intro integrable_real_density f AE_I2 enn2real_nonneg*)
apply (*insert ac*, *auto*)
done

show *?integral*
apply (*subst density_RN_deriv*[*OF ac*, *symmetric*])
unfolding *eq*
apply (*intro integral_real_density f AE_I2 enn2real_nonneg*)
apply (*insert ac*, *auto*)
done
qed

proposition (**in** *sigma_finite_measure*) *real_RN_deriv*:
assumes *finite_measure N*
assumes *ac*: *absolutely_continuous M N sets N = sets M*

```
  obtains D where D ∈ borel_measurable M
    and AE x in M. RN_deriv M N x = ennreal (D x)
    and AE x in N. 0 < D x
    and ⋀x. 0 ≤ D x
proof
  interpret N: finite_measure N by fact

  note RN = borel_measurable_RN_deriv density_RN_deriv[OF ac]

  let ?RN = λt. {x ∈ space M. RN_deriv M N x = t}

  show (λx. enn2real (RN_deriv M N x)) ∈ borel_measurable M
    using RN by auto

  have N (?RN ∞) = (∫⁺ x. RN_deriv M N x * indicator (?RN ∞) x ∂M)
    using RN(1) by (subst RN(2)[symmetric]) (auto simp: emeasure_density)
  also have ... = (∫⁺ x. ∞ * indicator (?RN ∞) x ∂M)
    by (intro nn_integral_cong) (auto simp: indicator_def)
  also have ... = ∞ * emeasure M (?RN ∞)
    using RN by (intro nn_integral_cmult_indicator) auto
  finally have eq: N (?RN ∞) = ∞ * emeasure M (?RN ∞) .
  moreover
  have emeasure M (?RN ∞) = 0
  proof (rule ccontr)
    assume emeasure M {x ∈ space M. RN_deriv M N x = ∞} ≠ 0
    then have 0 < emeasure M {x ∈ space M. RN_deriv M N x = ∞}
      by (auto simp: zero_less_iff_neq_zero)
    with eq have N (?RN ∞) = ∞ by (simp add: ennreal_mult_eq_top_iff)
    with N.emeasure_finite[of ?RN ∞] RN show False by auto
  qed
  ultimately have AE x in M. RN_deriv M N x < ∞
   using RN by (intro AE_iff_measurable[THEN iffD2]) (auto simp: less_top[symmetric])
  then show AE x in M. RN_deriv M N x = ennreal (enn2real (RN_deriv M N
x))
    by auto
  then have eq: AE x in N. RN_deriv M N x = ennreal (enn2real (RN_deriv M
N x))
    using ac absolutely_continuous_AE by auto


  have N (?RN 0) = (∫⁺ x. RN_deriv M N x * indicator (?RN 0) x ∂M)
    by (subst RN(2)[symmetric]) (auto simp: emeasure_density)
  also have ... = (∫⁺ x. 0 ∂M)
    by (intro nn_integral_cong) (auto simp: indicator_def)
  finally have AE x in N. RN_deriv M N x ≠ 0
     using RN by (subst AE_iff_measurable[OF _ refl]) (auto simp: ac cong:
sets_eq_imp_space_eq)
  with eq show AE x in N. 0 < enn2real (RN_deriv M N x)
   by (auto simp: enn2real_positive_iff less_top[symmetric] zero_less_iff_neq_zero)
```

qed (*rule enn2real_nonneg*)

lemma (**in** *sigma_finite_measure*) *RN_deriv_singleton*:
assumes *ac*: *absolutely_continuous M N sets N = sets M*
and *x*: $\{x\} \in$ *sets M*
shows *N* $\{x\}$ = *RN_deriv M N x* $*$ *emeasure M* $\{x\}$
proof −
from ‹$\{x\} \in$ *sets M*›
have *density M* (*RN_deriv M N*) $\{x\}$ = $(\int^+ w.$ *RN_deriv M N x* $*$ *indicator* $\{x\}$ *w* $\partial M)$
by (*auto simp*: *indicator_def emeasure_density intro*!: *nn_integral_cong*)
with *x density_RN_deriv*[*OF ac*] **show** *?thesis*
by (*auto simp*: *max_def*)
qed

end

theory *Set_Integral*
imports *Radon_Nikodym*
begin

definition *set_borel_measurable M A f* ≡ $(\lambda x.$ *indicator A x* $*_R$ *f x*) ∈ *borel_measurable M*

definition *set_integrable M A f* ≡ *integrable M* $(\lambda x.$ *indicator A x* $*_R$ *f x*)

definition *set_lebesgue_integral M A f* ≡ *lebesgue_integral M* $(\lambda x.$ *indicator A x* $*_R$ *f x*)

syntax
_ascii_set_lebesgue_integral :: *pttrn* ⇒ *'a set* ⇒ *'a measure* ⇒ *real* ⇒ *real*
((*4LINT* (_):(_)/|(_)./ _) [*0*,*60*,*110*,*61*] *60*)

translations
LINT x:*A*|*M*. *f* == *CONST set_lebesgue_integral M A* $(\lambda x.$ *f*)

syntax
_lebesgue_borel_integral :: *pttrn* ⇒ *real* ⇒ *real*
((*2LBINT* _./ _) [*0*,*60*] *60*)

syntax
_set_lebesgue_borel_integral :: *pttrn* ⇒ *real set* ⇒ *real* ⇒ *real*
((*3LBINT* _:_./ _) [*0*,*60*,*61*] *60*)

lemma *set_integrable_cong*:
assumes $M = M'$ $A = A'$ $\bigwedge x.\ x \in A \Longrightarrow f\ x = f'\ x$
shows *set_integrable M A f = set_integrable M' A' f'*
proof −
have $(\lambda x.\ indicator\ A\ x *_R f\ x) = (\lambda x.\ indicator\ A'\ x *_R f'\ x)$
using *assms* **by** (*auto simp*: *indicator_def of_bool_def*)
thus *?thesis* **by** (*simp add*: *set_integrable_def assms*)
qed

lemma *set_borel_measurable_sets*:
fixes $f :: \_ \Rightarrow \_ ::$*real_normed_vector*
assumes *set_borel_measurable M X f B ∈ sets borel X ∈ sets M*
shows $f -` B \cap X \in sets\ M$
proof −
have $f \in borel\_measurable\ (restrict\_space\ M\ X)$
using *assms* **unfolding** *set_borel_measurable_def* **by** (*subst borel_measurable_restrict_space_iff*) *auto*
then have $f -` B \cap space\ (restrict\_space\ M\ X) \in sets\ (restrict\_space\ M\ X)$
by (*rule measurable_sets*) *fact*
with ‹$X \in sets\ M$› **show** *?thesis*
by (*subst* (*asm*) *sets_restrict_space_iff*) (*auto simp*: *space_restrict_space*)
qed

lemma *set_integrable_bound*:
fixes $f :: 'a \Rightarrow 'b ::$ {*banach*, *second_countable_topology*}
and $g :: 'a \Rightarrow 'c ::$ {*banach*, *second_countable_topology*}
assumes *set_integrable M A f set_borel_measurable M A g*
assumes *AE x in M. x ∈ A ⟶ norm (g x) ≤ norm (f x)*
shows *set_integrable M A g*
unfolding *set_integrable_def*
proof (*rule Bochner_Integration.integrable_bound*)
from *assms*(*1*) **show** $integrable\ M\ (\lambda x.\ indicator\ A\ x *_R f\ x)$
by (*simp add*: *set_integrable_def*)
from *assms*(*2*) **show** $(\lambda x.\ indicat\_real\ A\ x *_R g\ x) \in borel\_measurable\ M$
by (*simp add*: *set_borel_measurable_def*)
from *assms*(*3*) **show** *AE x in M. norm (indicat_real A x ∗R g x) ≤ norm (indicat_real A x ∗R f x)*
by *eventually_elim* (*simp add*: *indicator_def*)
qed

lemma *set_lebesgue_integral_zero* [*simp*]: *set_lebesgue_integral M A* $(\lambda x.\ 0) = 0$
by (*auto simp*: *set_lebesgue_integral_def*)

lemma *set_lebesgue_integral_cong*:

assumes $A \in sets\ M$ **and** $\forall x.\ x \in A \longrightarrow f\ x = g\ x$
shows $(LINT\ x{:}A|M.\ f\ x) = (LINT\ x{:}A|M.\ g\ x)$
unfolding *set_lebesgue_integral_def*
using *assms*
by (*metis indicator_simps*(*2*) *real_vector.scale_zero_left*)

lemma *set_lebesgue_integral_cong_AE*:
assumes [*measurable*]: $A \in sets\ M\ f \in borel\_measurable\ M\ g \in borel\_measurable\ M$
assumes $AE\ x \in A\ in\ M.\ f\ x = g\ x$
shows $LINT\ x{:}A|M.\ f\ x = LINT\ x{:}A|M.\ g\ x$
proof−
have $AE\ x\ in\ M.\ indicator\ A\ x *_R f\ x = indicator\ A\ x *_R g\ x$
using *assms* **by** *auto*
thus *?thesis*
unfolding *set_lebesgue_integral_def* **by** (*intro integral_cong_AE*) *auto*
qed

lemma *set_integrable_cong_AE*:
$f \in borel\_measurable\ M \Longrightarrow g \in borel\_measurable\ M \Longrightarrow$
$AE\ x \in A\ in\ M.\ f\ x = g\ x \Longrightarrow A \in sets\ M \Longrightarrow$
$set\_integrable\ M\ A\ f = set\_integrable\ M\ A\ g$
unfolding *set_integrable_def*
by (*rule integrable_cong_AE*) *auto*

lemma *set_integrable_subset*:
fixes $M\ A\ B$ **and** $f :: \_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$
assumes $set\_integrable\ M\ A\ f\ B \in sets\ M\ B \subseteq A$
shows $set\_integrable\ M\ B\ f$
proof −
have $set\_integrable\ M\ B\ (\lambda x.\ indicator\ A\ x *_R f\ x)$
using *assms integrable_mult_indicator set_integrable_def* **by** *blast*
with ‹$B \subseteq A$› **show** *?thesis*
unfolding *set_integrable_def*
by (*simp add*: *indicator_inter_arith*[*symmetric*] *Int_absorb2*)
qed

lemma *set_integrable_restrict_space*:
fixes $f :: 'a \Rightarrow 'b::\{banach, second\_countable\_topology\}$
assumes f: $set\_integrable\ M\ S\ f$ **and** T: $T \in sets\ (restrict\_space\ M\ S)$
shows $set\_integrable\ M\ T\ f$
proof −
obtain T' **where** T_eq: $T = S \cap T'$ **and** $T' \in sets\ M$
using T **by** (*auto simp*: *sets_restrict_space*)
have ‹$integrable\ M\ (\lambda x.\ indicator\ T'\ x *_R (indicator\ S\ x *_R f\ x))$›
using ‹$T' \in sets\ M$› f *integrable_mult_indicator set_integrable_def* **by** *blast*
then show *?thesis*
unfolding *set_integrable_def*
unfolding *T_eq indicator_inter_arith* **by** (*simp add*: *ac_simps*)

qed

lemma *set_integral_scaleR_right* [*simp*]: *LINT t:A|M. a *$_R$* f t = a *$_R$* (LINT t:A|M. f t)*
unfolding *set_lebesgue_integral_def*
by (*subst integral_scaleR_right*[*symmetric*]) (*auto intro*!: *Bochner_Integration.integral_cong*)

lemma *set_integral_mult_right* [*simp*]:
fixes *a* :: *'a*::{*real_normed_field*, *second_countable_topology*}
shows *LINT t:A|M. a * f t = a * (LINT t:A|M. f t)*
unfolding *set_lebesgue_integral_def*
by (*subst integral_mult_right_zero*[*symmetric*]) *auto*

lemma *set_integral_mult_left* [*simp*]:
fixes *a* :: *'a*::{*real_normed_field*, *second_countable_topology*}
shows *LINT t:A|M. f t * a = (LINT t:A|M. f t) * a*
unfolding *set_lebesgue_integral_def*
by (*subst integral_mult_left_zero*[*symmetric*]) *auto*

lemma *set_integral_divide_zero* [*simp*]:
fixes *a* :: *'a*::{*real_normed_field*, *field*, *second_countable_topology*}
shows *LINT t:A|M. f t / a = (LINT t:A|M. f t) / a*
unfolding *set_lebesgue_integral_def*
by (*subst integral_divide_zero*[*symmetric*], *intro Bochner_Integration.integral_cong*)
(*auto split*: *split_indicator*)

lemma *set_integrable_scaleR_right* [*simp*, *intro*]:
shows *(a ≠ 0 ⟹ set_integrable M A f) ⟹ set_integrable M A (λt. a *$_R$* f t)*
unfolding *set_integrable_def*
unfolding *scaleR_left_commute* **by** (*rule integrable_scaleR_right*)

lemma *set_integrable_scaleR_left* [*simp*, *intro*]:
fixes *a* :: *_* :: {*banach*, *second_countable_topology*}
shows *(a ≠ 0 ⟹ set_integrable M A f) ⟹ set_integrable M A (λt. f t *$_R$* a)*
unfolding *set_integrable_def*
using *integrable_scaleR_left*[*of a M λx. indicator A x *$_R$* f x*] **by** *simp*

lemma *set_integrable_mult_right* [*simp*, *intro*]:
fixes *a* :: *'a*::{*real_normed_field*, *second_countable_topology*}
shows *(a ≠ 0 ⟹ set_integrable M A f) ⟹ set_integrable M A (λt. a * f t)*
unfolding *set_integrable_def*
using *integrable_mult_right*[*of a M λx. indicator A x *$_R$* f x*] **by** *simp*

lemma *set_integrable_mult_right_iff* [*simp*]:
fixes *a* :: *'a*::{*real_normed_field*, *second_countable_topology*}
assumes *a ≠ 0*

shows *set_integrable M A* (λ*t*. *a* ∗ *f t*) ⟷ *set_integrable M A f*
proof
assume *set_integrable M A* (λ*t*. *a* ∗ *f t*)
then have *set_integrable M A* (λ*t*. *1*/*a* ∗ (*a* ∗ *f t*))
using *set_integrable_mult_right* **by** *blast*
then show *set_integrable M A f*
using *assms* **by** *auto*
qed *auto*

lemma *set_integrable_mult_left* [*simp*, *intro*]:
fixes *a* :: *'a*::{*real_normed_field*, *second_countable_topology*}
shows (*a* ≠ *0* ⟹ *set_integrable M A f*) ⟹ *set_integrable M A* (λ*t*. *f t* ∗ *a*)
unfolding *set_integrable_def*
using *integrable_mult_left*[*of a M* λ*x*. *indicator A x* ∗*R f x*] **by** *simp*

lemma *set_integrable_mult_left_iff* [*simp*]:
fixes *a* :: *'a*::{*real_normed_field*, *second_countable_topology*}
assumes *a* ≠ *0*
shows *set_integrable M A* (λ*t*. *f t* ∗ *a*) ⟷ *set_integrable M A f*
using *assms* **by** (*subst set_integrable_mult_right_iff* [*symmetric*]) (*auto simp*: *mult.commute*)

lemma *set_integrable_divide* [*simp*, *intro*]:
fixes *a* :: *'a*::{*real_normed_field*, *field*, *second_countable_topology*}
assumes *a* ≠ *0* ⟹ *set_integrable M A f*
shows *set_integrable M A* (λ*t*. *f t* / *a*)
proof −
have *integrable M* (λ*x*. *indicator A x* ∗*R f x* / *a*)
using *assms* **unfolding** *set_integrable_def* **by** (*rule integrable_divide_zero*)
also have (λ*x*. *indicator A x* ∗*R f x* / *a*) = (λ*x*. *indicator A x* ∗*R* (*f x* / *a*))
by (*auto split*: *split_indicator*)
finally show *?thesis*
unfolding *set_integrable_def* **.**
qed

lemma *set_integrable_mult_divide_iff* [*simp*]:
fixes *a* :: *'a*::{*real_normed_field*, *second_countable_topology*}
assumes *a* ≠ *0*
shows *set_integrable M A* (λ*t*. *f t* / *a*) ⟷ *set_integrable M A f*
by (*simp add*: *divide_inverse assms*)

lemma *set_integral_add* [*simp*, *intro*]:
fixes *f g* :: *_* ⇒ *_* :: {*banach*, *second_countable_topology*}
assumes *set_integrable M A f set_integrable M A g*
shows *set_integrable M A* (λ*x*. *f x* + *g x*)
and *LINT x*:*A*|*M*. *f x* + *g x* = (*LINT x*:*A*|*M*. *f x*) + (*LINT x*:*A*|*M*. *g x*)
using *assms* **unfolding** *set_integrable_def set_lebesgue_integral_def* **by** (*simp_all add*: *scaleR_add_right*)

lemma *set_integral_diff* [*simp*, *intro*]:
assumes *set_integrable M A f set_integrable M A g*
shows *set_integrable M A* (λ*x*. *f x* − *g x*) **and** *LINT x*:*A*|*M*. *f x* − *g x* =
(*LINT x*:*A*|*M*. *f x*) − (*LINT x*:*A*|*M*. *g x*)
using *assms* **unfolding** *set_integrable_def set_lebesgue_integral_def* **by** (*simp_all add*: *scaleR_diff_right*)

lemma *set_integral_uminus*: *set_integrable M A f* ⟹ *LINT x*:*A*|*M*. − *f x* = − (*LINT x*:*A*|*M*. *f x*)
unfolding *set_integrable_def set_lebesgue_integral_def*
by (*subst integral_minus*[*symmetric*]) *simp_all*

lemma *set_integral_complex_of_real*:
LINT x:*A*|*M*. *complex_of_real* (*f x*) = *of_real* (*LINT x*:*A*|*M*. *f x*)
unfolding *set_lebesgue_integral_def*
by (*subst integral_complex_of_real*[*symmetric*])
(*auto intro*!: *Bochner_Integration.integral_cong split*: *split_indicator*)

lemma *set_integral_mono*:
fixes *f g* :: _ ⇒ *real*
assumes *set_integrable M A f set_integrable M A g*
⋀*x*. *x* ∈ *A* ⟹ *f x* ≤ *g x*
shows (*LINT x*:*A*|*M*. *f x*) ≤ (*LINT x*:*A*|*M*. *g x*)
using *assms* **unfolding** *set_integrable_def set_lebesgue_integral_def*
by (*auto intro*: *integral_mono split*: *split_indicator*)

lemma *set_integral_mono_AE*:
fixes *f g* :: _ ⇒ *real*
assumes *set_integrable M A f set_integrable M A g*
AE x ∈ *A in M*. *f x* ≤ *g x*
shows (*LINT x*:*A*|*M*. *f x*) ≤ (*LINT x*:*A*|*M*. *g x*)
using *assms* **unfolding** *set_integrable_def set_lebesgue_integral_def*
by (*auto intro*: *integral_mono_AE split*: *split_indicator*)

lemma *set_integrable_abs*: *set_integrable M A f* ⟹ *set_integrable M A* (λ*x*. |*f x*| :: *real*)
using *integrable_abs*[*of M* λ*x*. *f x* ∗ *indicator A x*]**unfolding** *set_integrable_def*
by (*simp add*: *abs_mult ac_simps*)

lemma *set_integrable_abs_iff*:
fixes *f* :: _ ⇒ *real*
shows *set_borel_measurable M A f* ⟹ *set_integrable M A* (λ*x*. |*f x*|) = *set_integrable M A f*
unfolding *set_integrable_def set_borel_measurable_def*
by (*subst* (*2*) *integrable_abs_iff*[*symmetric*]) (*simp_all add*: *abs_mult ac_simps*)

lemma *set_integrable_abs_iff'*:
fixes *f* :: _ ⇒ *real*
shows *f* ∈ *borel_measurable M* ⟹ *A* ∈ *sets M* ⟹

set_integrable M A (λ*x*. |*f x*|) = *set_integrable M A f*
by (*simp add*: *set_borel_measurable_def set_integrable_abs_iff*)

lemma *set_integrable_discrete_difference*:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second_countable_topology*}
assumes *countable X*
assumes *diff*: (*A* − *B*) ∪ (*B* − *A*) ⊆ *X*
assumes ⋀*x*. *x* ∈ *X* ⟹ *emeasure M* {*x*} = *0* ⋀*x*. *x* ∈ *X* ⟹ {*x*} ∈ *sets M*
shows *set_integrable M A f* ⟷ *set_integrable M B f*
unfolding *set_integrable_def*
proof (*rule integrable_discrete_difference*[**where** *X*=*X*])
show ⋀*x*. *x* ∈ *space M* ⟹ *x* ∉ *X* ⟹ *indicator A x* ∗*R f x* = *indicator B x* ∗*R* *f x*
using *diff* **by** (*auto split*: *split_indicator*)
qed *fact*+

lemma *set_integral_discrete_difference*:
fixes *f* :: *'a* ⇒ *'b*::{*banach*, *second_countable_topology*}
assumes *countable X*
assumes *diff*: (*A* − *B*) ∪ (*B* − *A*) ⊆ *X*
assumes ⋀*x*. *x* ∈ *X* ⟹ *emeasure M* {*x*} = *0* ⋀*x*. *x* ∈ *X* ⟹ {*x*} ∈ *sets M*
shows *set_lebesgue_integral M A f* = *set_lebesgue_integral M B f*
unfolding *set_lebesgue_integral_def*
proof (*rule integral_discrete_difference*[**where** *X*=*X*])
show ⋀*x*. *x* ∈ *space M* ⟹ *x* ∉ *X* ⟹ *indicator A x* ∗*R f x* = *indicator B x* ∗*R* *f x*
using *diff* **by** (*auto split*: *split_indicator*)
qed *fact*+

lemma *set_integrable_Un*:
fixes *f g* :: _ ⇒ _ :: {*banach*, *second_countable_topology*}
assumes *f_A*: *set_integrable M A f* **and** *f_B*: *set_integrable M B f*
and [*measurable*]: *A* ∈ *sets M B* ∈ *sets M*
shows *set_integrable M* (*A* ∪ *B*) *f*
proof −
have *set_integrable M* (*A* − *B*) *f*
using *f_A* **by** (*rule set_integrable_subset*) *auto*
with *f_B* **have** *integrable M* (λ*x*. *indicator* (*A* − *B*) *x* ∗*R f x* + *indicator B x* ∗*R* *f x*)
unfolding *set_integrable_def* **using** *integrable_add* **by** *blast*
then show *?thesis*
unfolding *set_integrable_def*
by (*rule integrable_cong*[*THEN iffD1*, *rotated 2*]) (*auto split*: *split_indicator*)
qed

lemma *set_integrable_empty* [*simp*]: *set_integrable M* {} *f*
by (*auto simp*: *set_integrable_def*)

lemma *set_integrable_UN*:

fixes $f :: \_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$
assumes *finite* I $\bigwedge i.\ i \in I \Longrightarrow set\_integrable\ M\ (A\ i)\ f$
$\bigwedge i.\ i \in I \Longrightarrow A\ i \in sets\ M$
shows $set\_integrable\ M\ (\bigcup i \in I.\ A\ i)\ f$
using *assms*
by (*induct* I) (*auto simp*: *set_integrable_Un sets.finite_UN*)

lemma *set_integral_Un*:
fixes $f :: \_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$
assumes $A \cap B = \{\}$
and $set\_integrable\ M\ A\ f$
and $set\_integrable\ M\ B\ f$
shows $LINT\ x{:}A \cup B|M.\ f\ x = (LINT\ x{:}A|M.\ f\ x) + (LINT\ x{:}B|M.\ f\ x)$
using *assms*
unfolding *set_integrable_def set_lebesgue_integral_def*
by (*auto simp add*: *indicator_union_arith indicator_inter_arith*[*symmetric*] *scaleR_add_left*)

lemma *set_integral_cong_set*:
fixes $f :: \_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$
assumes *set_borel_measurable M A f set_borel_measurable M B f*
and *ae*: *AE x in M*. $x \in A \longleftrightarrow x \in B$
shows $LINT\ x{:}B|M.\ f\ x = LINT\ x{:}A|M.\ f\ x$
unfolding *set_lebesgue_integral_def*
proof (*rule integral_cong_AE*)
show *AE x in M*. $indicator\ B\ x *_R f\ x = indicator\ A\ x *_R f\ x$
using *ae* **by** (*auto simp*: *subset_eq split*: *split_indicator*)
qed (*use assms* **in** ‹*auto simp*: *set_borel_measurable_def*›)

proposition *set_borel_measurable_subset*:
fixes $f :: \_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$
assumes [*measurable*]: *set_borel_measurable M A f B* $\in$ *sets M* **and** $B \subseteq A$
shows *set_borel_measurable M B f*
proof−
have *set_borel_measurable M B* ($\lambda x.$ *indicator A x* $*_R$ *f x*)
using *assms* **unfolding** *set_borel_measurable_def*
using *borel_measurable_indicator borel_measurable_scaleR* **by** *blast*
moreover have ($\lambda x.$ *indicator B x* $*_R$ *indicator A x* $*_R$ *f x*) = ($\lambda x.$ *indicator B x* $*_R$ *f x*)
using ‹$B \subseteq A$› **by** (*auto simp*: *fun_eq_iff split*: *split_indicator*)
ultimately show *?thesis*
unfolding *set_borel_measurable_def* **by** *simp*
qed

lemma *set_integral_Un_AE*:
fixes $f :: \_ \Rightarrow \_ :: \{banach, second\_countable\_topology\}$
assumes *ae*: *AE x in M*. $\neg\ (x \in A \wedge x \in B)$ **and** [*measurable*]: $A \in sets\ M\ B \in sets\ M$
and *set_integrable M A f*
and *set_integrable M B f*

shows *LINT x:A∪B|M. f x = (LINT x:A|M. f x) + (LINT x:B|M. f x)*
proof −
have *f*: *set_integrable M (A ∪ B) f*
by (*intro set_integrable_Un assms*)
then have *f′*: *set_borel_measurable M (A ∪ B) f*
using *integrable_iff_bounded set_borel_measurable_def set_integrable_def* **by** *blast*
have *LINT x:A∪B|M. f x = LINT x:(A − A ∩ B) ∪ (B − A ∩ B)|M. f x*
proof (*rule set_integral_cong_set*)
show *AE x in M. (x ∈ A − A ∩ B ∪ (B − A ∩ B)) = (x ∈ A ∪ B)*
using *ae* **by** *auto*
show *set_borel_measurable M (A − A ∩ B ∪ (B − A ∩ B)) f*
using *f′* **by** (*rule set_borel_measurable_subset*) *auto*
qed *fact*
also have *... = (LINT x:(A − A ∩ B)|M. f x) + (LINT x:(B − A ∩ B)|M. f x)*
by (*auto intro*!: *set_integral_Un set_integrable_subset*[*OF f*])
also have *... = (LINT x:A|M. f x) + (LINT x:B|M. f x)*
using *ae*
by (*intro arg_cong2*[**where** *f*=(+)] *set_integral_cong_set*)
(*auto intro*!: *set_borel_measurable_subset*[*OF f′*])
finally show *?thesis* **.**
qed

lemma *set_integral_finite_Union*:
fixes *f* :: *_* ⇒ *_* :: {*banach*, *second_countable_topology*}
assumes *finite I disjoint_family_on A I*
and ⋀*i*. *i* ∈ *I* ⟹ *set_integrable M (A i) f* ⋀*i*. *i* ∈ *I* ⟹ *A i* ∈ *sets M*
shows (*LINT x*:(⋃ *i*∈*I*. *A i*)|*M*. *f x*) = (∑ *i*∈*I*. *LINT x*:*A i*|*M*. *f x*)
using *assms*
proof *induction*
case (*insert x F*)
then have *A x* ∩ ⋃(*A* ‘ *F*) = {}
by (*meson disjoint_family_on_insert*)
with *insert* **show** *?case*
by (*simp add*: *set_integral_Un set_integrable_Un set_integrable_UN disjoint_family_on_insert*)
qed (*simp add*: *set_lebesgue_integral_def*)

lemma *pos_integrable_to_top*:
fixes *l*::*real*
assumes ⋀*i*. *A i* ∈ *sets M mono A*
assumes *nneg*: ⋀*x i*. *x* ∈ *A i* ⟹ 0 ≤ *f x*
and *intgbl*: ⋀*i*::*nat*. *set_integrable M (A i) f*
and *lim*: (λ*i*::*nat*. *LINT x*:*A i*|*M*. *f x*) ⟶ *l*
shows *set_integrable M* (⋃ *i*. *A i*) *f*
unfolding *set_integrable_def*
apply (*rule integrable_monotone_convergence*[**where** *f* = λ*i*::*nat*. λ*x*. *indicator*

```
(A i) x *R f x and x = l])
  apply (rule intgbl [unfolded set_integrable_def])
  prefer 3 apply (rule lim [unfolded set_lebesgue_integral_def])
  apply (rule AE_I2)
  using ‹mono A› apply (auto simp: mono_def nneg split: split_indicator) []
proof (rule AE_I2)
  { fix x assume x ∈ space M
    show (λi. indicator (A i) x *R f x) ⟶ indicator (⋃ i. A i) x *R f x
    proof cases
      assume ∃ i. x ∈ A i
      then obtain i where x ∈ A i ..
      then have *: eventually (λi. x ∈ A i) sequentially
       using ‹x ∈ A i› ‹mono A› by (auto simp: eventually_sequentially mono_def)
      show ?thesis
        apply (intro tendsto_eventually)
        using *
        apply eventually_elim
        apply (auto split: split_indicator)
        done
    qed auto }
  then show (λx. indicator (⋃ i. A i) x *R f x) ∈ borel_measurable M
    apply (rule borel_measurable_LIMSEQ_real)
    apply assumption
    using intgbl set_integrable_def by blast
qed


lemma lebesgue_integral_countable_add:
  fixes f :: _ ⇒ 'a :: {banach, second_countable_topology}
  assumes meas[intro]: ⋀i::nat. A i ∈ sets M
    and disj: ⋀i j. i ≠ j ⟹ A i ∩ A j = {}
    and intgbl: set_integrable M (⋃ i. A i) f
  shows LINT x:(⋃ i. A i)|M. f x = (∑ i. (LINT x:(A i)|M. f x))
    unfolding set_lebesgue_integral_def
proof (subst integral_suminf[symmetric])
  show int_A: integrable M (λx. indicat_real (A i) x *R f x) for i
    using intgbl unfolding set_integrable_def [symmetric]
    by (rule set_integrable_subset) auto
  { fix x assume x ∈ space M
    have (λi. indicator (A i) x *R f x) sums (indicator (⋃ i. A i) x *R f x)
      by (intro sums_scaleR_left indicator_sums) fact }
  note sums = this

  have norm_f: ⋀i. set_integrable M (A i) (λx. norm (f x))
    using int_A[THEN integrable_norm] unfolding set_integrable_def by auto

  show AE x in M. summable (λi. norm (indicator (A i) x *R f x))
   using disj by (intro AE_I2) (auto intro!: summable_mult2 sums_summable[OF
indicator_sums])
```

show *summable* (λ*i*. *LINT x*|*M*. *norm* (*indicator* (*A i*) *x* ∗$_R$ *f x*))
proof (*rule summableI_nonneg_bounded*)
fix *n*
show *0* ≤ *LINT x*|*M*. *norm* (*indicator* (*A n*) *x* ∗$_R$ *f x*)
using *norm_f* **by** (*auto intro*!: *integral_nonneg_AE*)

have (∑ *i*<*n*. *LINT x*|*M*. *norm* (*indicator* (*A i*) *x* ∗$_R$ *f x*)) = (∑ *i*<*n*. *LINT x*:*A i*|*M*. *norm* (*f x*))
by (*simp add*: *abs_mult set_lebesgue_integral_def*)
also have ... = *set_lebesgue_integral M* (⋃ *i*<*n*. *A i*) (λ*x*. *norm* (*f x*))
using *norm_f*
by (*subst set_integral_finite_Union*) (*auto simp*: *disjoint_family_on_def disj*)
also have ... ≤ *set_lebesgue_integral M* (⋃ *i*. *A i*) (λ*x*. *norm* (*f x*))
using *intgbl*[*unfolded set_integrable_def*, *THEN integrable_norm*] *norm_f*
unfolding *set_lebesgue_integral_def set_integrable_def*
apply (*intro integral_mono set_integrable_UN*[*of* {..<*n*}, *unfolded set_integrable_def*])
apply (*auto split*: *split_indicator*)
done
finally show (∑ *i*<*n*. *LINT x*|*M*. *norm* (*indicator* (*A i*) *x* ∗$_R$ *f x*)) ≤ *set_lebesgue_integral M* (⋃ *i*. *A i*) (λ*x*. *norm* (*f x*))
by *simp*
qed
show *LINT x*|*M*. *indicator* (⋃ (*A* ‘ *UNIV*)) *x* ∗$_R$ *f x* = *LINT x*|*M*. (∑ *i*. *indicator* (*A i*) *x* ∗$_R$ *f x*)
apply (*rule Bochner_Integration.integral_cong*[*OF refl*])
apply (*subst suminf_scaleR_left*[*OF sums_summable*[*OF indicator_sums*, *OF disj*], *symmetric*])
using *sums_unique*[*OF indicator_sums*[*OF disj*]]
apply *auto*
done
qed

lemma *set_integral_cont_up*:
fixes *f* :: _ ⇒ ′*a* :: {*banach*, *second_countable_topology*}
assumes [*measurable*]: ⋀*i*. *A i* ∈ *sets M* **and** *A*: *incseq A*
and *intgbl*: *set_integrable M* (⋃ *i*. *A i*) *f*
shows (λ*i*. *LINT x*:(*A i*)|*M*. *f x*) ⟶ *LINT x*:(⋃ *i*. *A i*)|*M*. *f x*
unfolding *set_lebesgue_integral_def*
proof (*intro integral_dominated_convergence*[**where** *w*=λ*x*. *indicator* (⋃ *i*. *A i*) *x* ∗$_R$ *norm* (*f x*)])
have *int_A*: ⋀*i*. *set_integrable M* (*A i*) *f*
using *intgbl* **by** (*rule set_integrable_subset*) *auto*
show ⋀*i*. (λ*x*. *indicator* (*A i*) *x* ∗$_R$ *f x*) ∈ *borel_measurable M*
using *int_A integrable_iff_bounded set_integrable_def* **by** *blast*
show (λ*x*. *indicator* (⋃ (*A* ‘ *UNIV*)) *x* ∗$_R$ *f x*) ∈ *borel_measurable M*
using *integrable_iff_bounded intgbl set_integrable_def* **by** *blast*
show *integrable M* (λ*x*. *indicator* (⋃ *i*. *A i*) *x* ∗$_R$ *norm* (*f x*))

using *int_A intgbl integrable_norm* **unfolding** *set_integrable_def*
by *fastforce*
{ fix *x i* **assume** *x* ∈ *A i*
with *A* **have** (λ*xa*. *indicator* (*A xa*) *x*::*real*) ⟶ *1* ⟷ (λ*xa*. *1*::*real*) ⟶ *1*
by (*intro filterlim_cong refl*)
(*fastforce simp*: *eventually_sequentially incseq_def subset_eq intro*!: *exI*[*of* _ *i*]) **}**
then show *AE x in M*. (λ*i*. *indicator* (*A i*) *x* ∗$_R$ *f x*) ⟶ *indicator* (⋃ *i*. *A i*) *x* ∗$_R$ *f x*
by (*intro AE_I2 tendsto_intros*) (*auto split*: *split_indicator*)
qed (*auto split*: *split_indicator*)

lemma *set_integral_cont_down*:
fixes *f* :: _ ⇒ *'a* :: {*banach*, *second_countable_topology*}
assumes [*measurable*]: ⋀*i*. *A i* ∈ *sets M* **and** *A*: *decseq A*
and *int0*: *set_integrable M* (*A 0*) *f*
shows (λ*i*::*nat*. *LINT x*:(*A i*)|*M*. *f x*) ⟶ *LINT x*:(⋂ *i*. *A i*)|*M*. *f x*
unfolding *set_lebesgue_integral_def*
proof (*rule integral_dominated_convergence*)
have *int_A*: ⋀*i*. *set_integrable M* (*A i*) *f*
using *int0* **by** (*rule set_integrable_subset*) (*insert A*, *auto simp*: *decseq_def*)
have *integrable M* (λ*c*. *norm* (*indicat_real* (*A 0*) *c* ∗$_R$ *f c*))
by (*metis* (*no_types*) *int0 integrable_norm set_integrable_def*)
then show *integrable M* (λ*x*. *indicator* (*A 0*) *x* ∗$_R$ *norm* (*f x*))
by *force*
have *set_integrable M* (⋂ *i*. *A i*) *f*
using *int0* **by** (*rule set_integrable_subset*) (*insert A*, *auto simp*: *decseq_def*)
with *int_A* **show** (λ*x*. *indicat_real* (⋂ (*A* ‘ *UNIV*)) *x* ∗$_R$ *f x*) ∈ *borel_measurable M*
⋀*i*. (λ*x*. *indicat_real* (*A i*) *x* ∗$_R$ *f x*) ∈ *borel_measurable M*
by (*auto simp*: *set_integrable_def*)
show ⋀*i*. *AE x in M*. *norm* (*indicator* (*A i*) *x* ∗$_R$ *f x*) ≤ *indicator* (*A 0*) *x* ∗$_R$ *norm* (*f x*)
using *A* **by** (*auto split*: *split_indicator simp*: *decseq_def*)
{ fix *x i* **assume** *x* ∈ *space M x* ∉ *A i*
with *A* **have** (λ*i*. *indicator* (*A i*) *x*::*real*) ⟶ *0* ⟷ (λ*i*. *0*::*real*) ⟶ *0*
by (*intro filterlim_cong refl*)
(*auto split*: *split_indicator simp*: *eventually_sequentially decseq_def intro*!: *exI*[*of* _ *i*]) **}**
then show *AE x in M*. (λ*i*. *indicator* (*A i*) *x* ∗$_R$ *f x*) ⟶ *indicator* (⋂ *i*. *A i*) *x* ∗$_R$ *f x*
by (*intro AE_I2 tendsto_intros*) (*auto split*: *split_indicator*)
qed

lemma *set_integral_at_point*:
fixes *a* :: *real*
assumes *set_integrable M* {*a*} *f*

and [*simp*]: $\{a\} \in sets\ M$ **and** $(emeasure\ M)\ \{a\} \neq \infty$
shows $(LINT\ x{:}\{a\} \mid M.\ f\ x) = f\ a * measure\ M\ \{a\}$
proof−
have *set_lebesgue_integral M* $\{a\}$ *f* = *set_lebesgue_integral M* $\{a\}$ (%*x*. *f a*)
by (*intro set_lebesgue_integral_cong*) *simp_all*
then show *?thesis* **using** *assms*
unfolding *set_lebesgue_integral_def* **by** *simp*
qed

abbreviation *complex_integrable* :: *'a measure* ⇒ (*'a* ⇒ *complex*) ⇒ *bool* **where**
complex_integrable M f ≡ *integrable M f*

abbreviation *complex_lebesgue_integral* :: *'a measure* ⇒ (*'a* ⇒ *complex*) ⇒ *complex* (*integral*^{C}) **where**
integral^C *M f* == *integral*^{L} *M f*

syntax
_complex_lebesgue_integral :: *pttrn* ⇒ *complex* ⇒ *'a measure* ⇒ *complex*
(∫^{C} *_*. *_* ∂*_* [*60*,*61*] *110*)

translations
∫^{C}*x*. *f* ∂*M* == *CONST complex_lebesgue_integral M* (λ*x*. *f*)

syntax
_ascii_complex_lebesgue_integral :: *pttrn* ⇒ *'a measure* ⇒ *real* ⇒ *real*
((*3CLINT _*|*_*. *_*) [*0*,*110*,*60*] *60*)

translations
CLINT x|*M*. *f* == *CONST complex_lebesgue_integral M* (λ*x*. *f*)

lemma *complex_integrable_cnj* [*simp*]:
complex_integrable M (λ*x*. *cnj* (*f x*)) ⟷ *complex_integrable M f*
proof
assume *complex_integrable M* (λ*x*. *cnj* (*f x*))
then have *complex_integrable M* (λ*x*. *cnj* (*cnj* (*f x*)))
by (*rule integrable_cnj*)
then show *complex_integrable M f*
by *simp*
qed *simp*

lemma *complex_of_real_integrable_eq*:
complex_integrable M (λ*x*. *complex_of_real* (*f x*)) ⟷ *integrable M f*
proof
assume *complex_integrable M* (λ*x*. *complex_of_real* (*f x*))
then have *integrable M* (λ*x*. *Re* (*complex_of_real* (*f x*)))
by (*rule integrable_Re*)
then show *integrable M f*
by *simp*

qed *simp*

abbreviation *complex_set_integrable* :: *'a measure* ⇒ *'a set* ⇒ (*'a* ⇒ *complex*) ⇒ *bool* **where**
complex_set_integrable M A f ≡ *set_integrable M A f*

abbreviation *complex_set_lebesgue_integral* :: *'a measure* ⇒ *'a set* ⇒ (*'a* ⇒ *complex*) ⇒ *complex* **where**
complex_set_lebesgue_integral M A f ≡ *set_lebesgue_integral M A f*

syntax
_ascii_complex_set_lebesgue_integral :: *pttrn* ⇒ *'a set* ⇒ *'a measure* ⇒ *real* ⇒ *real*
((*4CLINT _:_|_. _*) [*0*,*60*,*110*,*61*] *60*)

translations
CLINT x:A|M. f == *CONST complex_set_lebesgue_integral M A* (λ*x. f*)

lemma *set_measurable_continuous_on_ivl*:
assumes *continuous_on* {*a..b*} (*f* :: *real* ⇒ *real*)
shows *set_borel_measurable borel* {*a..b*} *f*
unfolding *set_borel_measurable_def*
by (*rule borel_measurable_continuous_on_indicator*[*OF _ assms*]) *simp*

This notation is from Sébastien Gouëzel: His use is not directly in line with the notations in this file, they are more in line with sum, and more readable he thinks.

abbreviation *set_nn_integral M A f* ≡ *nn_integral M* (λ*x. f x* ∗ *indicator A x*)

syntax
_set_nn_integral :: *pttrn* => *'a set* => *'a measure* => *ereal* => *ereal*
((∫⁺((_)∈(_)./ _)/∂_) [*0*,*60*,*110*,*61*] *60*)

_set_lebesgue_integral :: *pttrn* => *'a set* => *'a measure* => *ereal* => *ereal*
((∫((_)∈(_)./ _)/∂_) [*0*,*60*,*110*,*61*] *60*)

translations
∫⁺*x* ∈ *A. f* ∂*M* == *CONST set_nn_integral M A* (λ*x. f*)
∫ *x* ∈ *A. f* ∂*M* == *CONST set_lebesgue_integral M A* (λ*x. f*)

lemma *set_nn_integral_cong*:
assumes *M* = *M'* *A* = *B* ⋀*x. x* ∈ *space M* ∩ *A* ⟹ *f x* = *g x*
shows *set_nn_integral M A f* = *set_nn_integral M' B g*
by (*metis* (*mono_tags*, *lifting*) *IntI assms indicator_simps*(*2*) *mult_eq_0_iff nn_integral_cong*)

lemma *nn_integral_disjoint_pair*:
assumes [*measurable*]: *f* ∈ *borel_measurable M*

$B \in sets\ M\ C \in sets\ M$
$B \cap C = \{\}$
shows $(\int^+ x \in B \cup C.\ f\ x\ \partial M) = (\int^+ x \in B.\ f\ x\ \partial M) + (\int^+ x \in C.\ f\ x\ \partial M)$
proof −
have *mes*: $\bigwedge D.\ D \in sets\ M \Longrightarrow (\lambda x.\ f\ x * indicator\ D\ x) \in borel\_measurable\ M$
by *simp*
have *pos*: $\bigwedge D.\ AE\ x\ in\ M.\ f\ x * indicator\ D\ x \geq 0$ **using** *assms*(2) **by** *auto*
have $\bigwedge x.\ f\ x * indicator\ (B \cup C)\ x = f\ x * indicator\ B\ x + f\ x * indicator\ C\ x$
using *assms*(4)
by (*auto split*: *split_indicator*)
then have $(\int^+ x.\ f\ x * indicator\ (B \cup C)\ x\ \partial M) = (\int^+ x.\ f\ x * indicator\ B\ x + f\ x * indicator\ C\ x\ \partial M)$
by *simp*
also have $... = (\int^+ x.\ f\ x * indicator\ B\ x\ \partial M) + (\int^+ x.\ f\ x * indicator\ C\ x\ \partial M)$
by (*rule nn_integral_add*) (*auto simp add*: *assms mes pos*)
finally show *?thesis* **by** *simp*
qed

lemma *nn_integral_disjoint_pair_countspace*:
assumes $B \cap C = \{\}$
shows $(\int^+ x \in B \cup C.\ f\ x\ \partial count\_space\ UNIV) = (\int^+ x \in B.\ f\ x\ \partial count\_space\ UNIV) + (\int^+ x \in C.\ f\ x\ \partial count\_space\ UNIV)$
by (*rule nn_integral_disjoint_pair*) (*simp_all add*: *assms*)

lemma *nn_integral_null_delta*:
assumes $A \in sets\ M\ B \in sets\ M$
$(A - B) \cup (B - A) \in null\_sets\ M$
shows $(\int^+ x \in A.\ f\ x\ \partial M) = (\int^+ x \in B.\ f\ x\ \partial M)$
proof (*rule nn_integral_cong_AE*)
have ∗: $AE\ a\ in\ M.\ a \notin (A - B) \cup (B - A)$
using *assms*(3) *AE_not_in* **by** *blast*
then show ‹$AE\ x\ in\ M.\ f\ x * indicator\ A\ x = f\ x * indicator\ B\ x$›
by *auto*
qed

proposition *nn_integral_disjoint_family*:
assumes [*measurable*]: $f \in borel\_measurable\ M$ $\bigwedge$(*n*::*nat*). $B\ n \in sets\ M$
and *disjoint_family B*
shows $(\int^+ x \in (\bigcup n.\ B\ n).\ f\ x\ \partial M) = (\sum n.\ (\int^+ x \in B\ n.\ f\ x\ \partial M))$
proof −
have $(\int^+ x.\ (\sum n.\ f\ x * indicator\ (B\ n)\ x)\ \partial M) = (\sum n.\ (\int^+ x.\ f\ x * indicator\ (B\ n)\ x\ \partial M))$
by (*rule nn_integral_suminf*) *simp*
moreover have $(\sum n.\ f\ x * indicator\ (B\ n)\ x) = f\ x * indicator\ (\bigcup n.\ B\ n)\ x$
for *x*
proof (*cases*)
assume $x \in (\bigcup n.\ B\ n)$
then obtain *n* **where** $x \in B\ n$ **by** *blast*
have *a*: *finite* {*n*} **by** *simp*

```
    have ⋀i. i ≠ n ⟹ x ∉ B i using ‹x ∈ B n› assms(3) disjoint_family_on_def
      by (metis IntI UNIV_I empty_iff)
   then have ⋀i. i ∉ {n} ⟹ indicator (B i) x = (0::ennreal) using indicator_def
by simp
    then have b: ⋀i. i ∉ {n} ⟹ f x * indicator (B i) x = (0::ennreal) by simp

    define h where h = (λi. f x * indicator (B i) x)
    then have ⋀i. i ∉ {n} ⟹ h i = 0 using b by simp
    then have (∑ i. h i) = (∑ i∈{n}. h i)
      by (metis sums_unique[OF sums_finite[OF a]])
    then have (∑ i. h i) = h n by simp
     then have (∑ n. f x * indicator (B n) x) = f x * indicator (B n) x using
h_def by simp
    then have (∑ n. f x * indicator (B n) x) = f x using ‹x ∈ B n› indicator_def
by simp
    then show ?thesis using ‹x ∈ (⋃ n. B n)› by auto
  next
    assume x ∉ (⋃ n. B n)
    then have ⋀n. f x * indicator (B n) x = 0 by simp
    have (∑ n. f x * indicator (B n) x) = 0
      by (simp add: ‹⋀n. f x * indicator (B n) x = 0›)
    then show ?thesis using ‹x ∉ (⋃ n. B n)› by auto
  qed
  ultimately show ?thesis by simp
qed

lemma nn_set_integral_add:
  assumes [measurable]: f ∈ borel_measurable M g ∈ borel_measurable M
          A ∈ sets M
  shows (∫⁺x ∈ A. (f x + g x) ∂M) = (∫⁺x ∈ A. f x ∂M) + (∫⁺x ∈ A. g x ∂M)
proof −
  have (∫⁺x ∈ A. (f x + g x) ∂M) = (∫⁺x. (f x * indicator A x + g x * indicator
A x) ∂M)
    by (auto simp add: indicator_def intro!: nn_integral_cong)
  also have ... = (∫⁺x. f x * indicator A x ∂M) + (∫⁺x. g x * indicator A x ∂M)
    apply (rule nn_integral_add) using assms(1) assms(2) by auto
  finally show ?thesis by simp
qed

lemma nn_set_integral_cong:
  assumes AE x in M. f x = g x
  shows (∫⁺x ∈ A. f x ∂M) = (∫⁺x ∈ A. g x ∂M)
apply (rule nn_integral_cong_AE) using assms(1) by auto

lemma nn_set_integral_set_mono:
  A ⊆ B ⟹ (∫⁺ x ∈ A. f x ∂M) ≤ (∫⁺ x ∈ B. f x ∂M)
by (auto intro!: nn_integral_mono split: split_indicator)

lemma nn_set_integral_mono:
```

assumes [*measurable*]: $f \in$ *borel_measurable* M $g \in$ *borel_measurable* M
$A \in$ *sets* M
and *AE* $x{\in}A$ *in* M. $f\ x \leq g\ x$
shows $(\int^+ x \in A.\ f\ x\ \partial M) \leq (\int^+ x \in A.\ g\ x\ \partial M)$
by (*auto intro*!: *nn_integral_mono_AE split*: *split_indicator simp*: *assms*)

lemma *nn_set_integral_space* [*simp*]:
shows $(\int^+ x \in$ *space* $M.\ f\ x\ \partial M) = (\int^+ x.\ f\ x\ \partial M)$
by (*metis* (*mono_tags*, *lifting*) *indicator_simps*(1) *mult.right_neutral nn_integral_cong*)

lemma *nn_integral_count_compose_inj*:
assumes *inj_on g A*
shows $(\int^+ x \in A.\ f\ (g\ x)\ \partial$*count_space UNIV*$) = (\int^+ y \in g`A.\ f\ y\ \partial$*count_space UNIV*$)$
proof −
have $(\int^+ x \in A.\ f\ (g\ x)\ \partial$*count_space UNIV*$) = (\int^+ x.\ f\ (g\ x)\ \partial$*count_space* $A)$
by (*auto simp add*: *nn_integral_count_space_indicator*[*symmetric*])
also have ... $= (\int^+ y.\ f\ y\ \partial$*count_space* $(g`A))$
by (*simp add*: *assms nn_integral_bij_count_space inj_on_imp_bij_betw*)
also have ... $= (\int^+ y \in g`A.\ f\ y\ \partial$*count_space UNIV*$)$
by (*auto simp add*: *nn_integral_count_space_indicator*[*symmetric*])
finally show *?thesis* **by** *simp*
qed

lemma *nn_integral_count_compose_bij*:
assumes *bij_betw g A B*
shows $(\int^+ x \in A.\ f\ (g\ x)\ \partial$*count_space UNIV*$) = (\int^+ y \in B.\ f\ y\ \partial$*count_space UNIV*$)$
proof −
have *inj_on g A* **using** *assms bij_betw_def* **by** *auto*
then have $(\int^+ x \in A.\ f\ (g\ x)\ \partial$*count_space UNIV*$) = (\int^+ y \in g`A.\ f\ y\ \partial$*count_space UNIV*$)$
by (*rule nn_integral_count_compose_inj*)
then show *?thesis* **using** *assms* **by** (*simp add*: *bij_betw_def*)
qed

lemma *set_integral_null_delta*:
fixes f::_ ⇒ _ :: {*banach*, *second_countable_topology*}
assumes [*measurable*]: *integrable* $M\ f\ A \in$ *sets* $M\ B \in$ *sets* M
and *null*: $(A - B) \cup (B - A) \in$ *null_sets* M
shows $(\int x \in A.\ f\ x\ \partial M) = (\int x \in B.\ f\ x\ \partial M)$
proof (*rule set_integral_cong_set*)
have ∗: *AE a in M*. $a \notin (A - B) \cup (B - A)$
using *null AE_not_in* **by** *blast*
then show *AE x in M*. $(x \in B) = (x \in A)$
by *auto*
qed (*simp_all add*: *set_borel_measurable_def*)

lemma *set_integral_space*:

assumes *integrable M f*
shows $(\int x \in space\ M.\ f\ x\ \partial M) = (\int x.\ f\ x\ \partial M)$
by (*metis* (*no_types*, *lifting*) *indicator_simps*(*1*) *integral_cong scaleR_one set_lebesgue_integral_def*)

lemma *null_if_pos_func_has_zero_nn_int*:
fixes $f::'a \Rightarrow ennreal$
assumes [*measurable*]: $f \in borel\_measurable\ M\ A \in sets\ M$
and $AE\ x{\in}A\ in\ M.\ f\ x > 0\ (\int^+ x{\in}A.\ f\ x\ \partial M) = 0$
shows $A \in null\_sets\ M$
proof −
have $AE\ x\ in\ M.\ f\ x * indicator\ A\ x = 0$
by (*subst nn_integral_0_iff_AE*[*symmetric*], *auto simp add*: *assms*(*4*))
then have $AE\ x{\in}A\ in\ M.\ False$ **using** *assms*(*3*) **by** *auto*
then show $A \in null\_sets\ M$ **using** *assms*(*2*) **by** (*simp add*: *AE_iff_null_sets*)
qed

lemma *null_if_pos_func_has_zero_int*:
assumes [*measurable*]: $integrable\ M\ f\ A \in sets\ M$
and $AE\ x{\in}A\ in\ M.\ f\ x > 0\ (\int x{\in}A.\ f\ x\ \partial M) = (0::real)$
shows $A \in null\_sets\ M$
proof −
have $AE\ x\ in\ M.\ indicator\ A\ x * f\ x = 0$
apply (*subst integral_nonneg_eq_0_iff_AE*[*symmetric*])
using *assms integrable_mult_indicator*[*OF* ‹$A \in sets\ M$› *assms*(*1*)]
by (*auto simp*: *set_lebesgue_integral_def*)
then have $AE\ x{\in}A\ in\ M.\ f\ x = 0$ **by** *auto*
then have $AE\ x{\in}A\ in\ M.\ False$ **using** *assms*(*3*) **by** *auto*
then show $A \in null\_sets\ M$ **using** *assms*(*2*) **by** (*simp add*: *AE_iff_null_sets*)
qed

The next lemma is a variant of *density_unique*. Note that it uses the notation for nonnegative set integrals introduced earlier.

lemma (**in** *sigma_finite_measure*) *density_unique2*:
assumes [*measurable*]: $f \in borel\_measurable\ M\ f' \in borel\_measurable\ M$
assumes *density_eq*: $\bigwedge A.\ A \in sets\ M \Longrightarrow (\int^+ x \in A.\ f\ x\ \partial M) = (\int^+ x \in A.\ f'\ x\ \partial M)$
shows $AE\ x\ in\ M.\ f\ x = f'\ x$
proof (*rule density_unique*)
show $density\ M\ f = density\ M\ f'$
by (*intro measure_eqI*) (*auto simp*: *emeasure_density intro*!: *density_eq*)
qed (*auto simp add*: *assms*)

The next lemma implies the same statement for Banach-space valued functions using Hahn-Banach theorem and linear forms. Since they are not yet easily available, I only formulate it for real-valued functions.

lemma *density_unique_real*:
fixes $f\ f'::\_ \Rightarrow real$
assumes M[*measurable*]: $integrable\ M\ f\ integrable\ M\ f'$

assumes *density_eq*: $\bigwedge A.\ A \in sets\ M \Longrightarrow (\int x \in A.\ f\ x\ \partial M) = (\int x \in A.\ f'\ x\ \partial M)$
shows *AE x in M. f x = f' x*
proof −
define *A* **where** $A = \{x \in space\ M.\ f\ x < f'\ x\}$
then have [*measurable*]: $A \in sets\ M$ **by** *simp*
have $(\int x \in A.\ (f'\ x - f\ x)\ \partial M) = (\int x \in A.\ f'\ x\ \partial M) - (\int x \in A.\ f\ x\ \partial M)$
using ‹$A \in sets\ M$› *M integrable_mult_indicator set_integrable_def* **by** *blast*
then have $(\int x \in A.\ (f'\ x - f\ x)\ \partial M) = 0$ **using** *assms*(*3*) **by** *simp*
then have *A ∈ null_sets M*
using *A_def null_if_pos_func_has_zero_int*[**where** *?f* = $\lambda x.\ f'\ x - f\ x$ **and** *?A = A*] *assms* **by** *auto*
then have *AE x in M. x ∉ A* **by** (*simp add*: *AE_not_in*)
then have ∗: *AE x in M. f' x ≤ f x* **unfolding** *A_def* **by** *auto*

define *B* **where** $B = \{x \in space\ M.\ f'\ x < f\ x\}$
then have [*measurable*]: $B \in sets\ M$ **by** *simp*
have $(\int x \in B.\ (f\ x - f'\ x)\ \partial M) = (\int x \in B.\ f\ x\ \partial M) - (\int x \in B.\ f'\ x\ \partial M)$
using ‹$B \in sets\ M$› *M integrable_mult_indicator set_integrable_def* **by** *blast*
then have $(\int x \in B.\ (f\ x - f'\ x)\ \partial M) = 0$ **using** *assms*(*3*) **by** *simp*
then have *B ∈ null_sets M*
using *B_def null_if_pos_func_has_zero_int*[**where** *?f* = $\lambda x.\ f\ x - f'\ x$ **and** *?A = B*] *assms* **by** *auto*
then have *AE x in M. x ∉ B* **by** (*simp add*: *AE_not_in*)
then have *AE x in M. f' x ≥ f x* **unfolding** *B_def* **by** *auto*
then show *?thesis* **using** ∗ **by** *auto*
qed

The next lemma shows that L^1 convergence of a sequence of functions follows from almost everywhere convergence and the weaker condition of the convergence of the integrated norms (or even just the nontrivial inequality about them). Useful in a lot of contexts! This statement (or its variations) are known as Scheffe lemma.

The formalization is more painful as one should jump back and forth between reals and ereals and justify all the time positivity or integrability (thankfully, measurability is handled more or less automatically).

proposition *Scheffe_lemma1*:
assumes $\bigwedge n.$ *integrable M (F n) integrable M f*
AE x in M. $(\lambda n.\ F\ n\ x) \longrightarrow f\ x$
limsup $(\lambda n.\ \int^+ x.\ norm(F\ n\ x)\ \partial M) \le (\int^+ x.\ norm(f\ x)\ \partial M)$
shows $(\lambda n.\ \int^+ x.\ norm(F\ n\ x - f\ x)\ \partial M) \longrightarrow 0$
proof −
have [*measurable*]: $\bigwedge n.$ *F n ∈ borel_measurable M f ∈ borel_measurable M*
using *assms*(*1*) *assms*(*2*) **by** *simp_all*
define *G* **where** $G = (\lambda n\ x.\ norm(f\ x) + norm(F\ n\ x) - norm(F\ n\ x - f\ x))$
have [*measurable*]: $\bigwedge n.$ *G n ∈ borel_measurable M* **unfolding** *G_def* **by** *simp*
have *G_pos*[*simp*]: $\bigwedge n\ x.\ G\ n\ x \ge 0$
unfolding *G_def* **by** (*metis ge_iff_diff_ge_0 norm_minus_commute norm_triangle_ineq4*)

have *finint*: $(\int^+ x.\ norm(f\ x)\ \partial M) \neq \infty$
using *has_bochner_integral_implies_finite_norm*[*OF has_bochner_integral_integrable*[*OF* ‹*integrable M f*›]]
by *simp*
then have *fin2*: $2 * (\int^+ x.\ norm(f\ x)\ \partial M) \neq \infty$
by (*auto simp*: *ennreal_mult_eq_top_iff*)

{
fix x **assume** $*$: $(\lambda n.\ F\ n\ x) \longrightarrow f\ x$
then have $(\lambda n.\ norm(F\ n\ x)) \longrightarrow norm(f\ x)$ **using** *tendsto_norm* **by** *blast*
moreover have $(\lambda n.\ norm(F\ n\ x - f\ x)) \longrightarrow 0$ **using** $*$ *Lim_null tendsto_norm_zero_iff* **by** *fastforce*
ultimately have *a*: $(\lambda n.\ norm(F\ n\ x) - norm(F\ n\ x - f\ x)) \longrightarrow norm(f\ x)$ **using** *tendsto_diff* **by** *fastforce*
have $(\lambda n.\ norm(f\ x) + (norm(F\ n\ x) - norm(F\ n\ x - f\ x))) \longrightarrow norm(f\ x) + norm(f\ x)$
by (*rule tendsto_add*) (*auto simp add*: *a*)
moreover have $\bigwedge n.\ G\ n\ x = norm(f\ x) + (norm(F\ n\ x) - norm(F\ n\ x - f\ x))$ **unfolding** *G_def* **by** *simp*
ultimately have $(\lambda n.\ G\ n\ x) \longrightarrow 2 * norm(f\ x)$ **by** *simp*
then have $(\lambda n.\ ennreal(G\ n\ x)) \longrightarrow ennreal(2 * norm(f\ x))$ **by** *simp*
then have *liminf* $(\lambda n.\ ennreal(G\ n\ x)) = ennreal(2 * norm(f\ x))$
using *sequentially_bot tendsto_iff_Liminf_eq_Limsup* **by** *blast*
}
then have *AE x in M*. *liminf* $(\lambda n.\ ennreal(G\ n\ x)) = ennreal(2 * norm(f\ x))$ **using** *assms*(*3*) **by** *auto*
then have $(\int^+ x.\ liminf\ (\lambda n.\ ennreal\ (G\ n\ x))\ \partial M) = (\int^+ x.\ 2 * ennreal(norm(f\ x))\ \partial M)$
by (*simp add*: *nn_integral_cong_AE ennreal_mult*)
also have ... $= 2 * (\int^+ x.\ norm(f\ x)\ \partial M)$ **by** (*rule nn_integral_cmult*) *auto*
finally have *int_liminf*: $(\int^+ x.\ liminf\ (\lambda n.\ ennreal\ (G\ n\ x))\ \partial M) = 2 * (\int^+ x.\ norm(f\ x)\ \partial M)$
by *simp*

have $(\int^+ x.\ ennreal(norm(f\ x)) + ennreal(norm(F\ n\ x))\ \partial M) = (\int^+ x.\ norm(f\ x)\ \partial M) + (\int^+ x.\ norm(F\ n\ x)\ \partial M)$ **for** *n*
by (*rule nn_integral_add*) (*auto simp add*: *assms*)
then have *limsup* $(\lambda n.\ (\int^+ x.\ ennreal(norm(f\ x)) + ennreal(norm(F\ n\ x))\ \partial M)) =$
limsup $(\lambda n.\ (\int^+ x.\ norm(f\ x)\ \partial M) + (\int^+ x.\ norm(F\ n\ x)\ \partial M))$
by *simp*
also have ... $= (\int^+ x.\ norm(f\ x)\ \partial M) + limsup\ (\lambda n.\ (\int^+ x.\ norm(F\ n\ x)\ \partial M))$
by (*rule Limsup_const_add*, *auto simp add*: *finint*)
also have ... $\leq (\int^+ x.\ norm(f\ x)\ \partial M) + (\int^+ x.\ norm(f\ x)\ \partial M)$
using *assms*(*4*) **by** (*simp add*: *add_left_mono*)
also have ... $= 2 * (\int^+ x.\ norm(f\ x)\ \partial M)$
unfolding *one_add_one*[*symmetric*] *distrib_right* **by** *simp*
ultimately have *a*: *limsup* $(\lambda n.\ (\int^+ x.\ ennreal(norm(f\ x)) + ennreal(norm(F\ n\ x))\ \partial M)) \leq$

$2 * (\int^+ x.\ norm(f\ x)\ \partial M)$ **by** *simp*

have *le*: *ennreal* (*norm* (*F n x* − *f x*)) ≤ *ennreal* (*norm* (*f x*)) + *ennreal* (*norm* (*F n x*)) **for** *n x*
by (*simp add*: *norm_minus_commute norm_triangle_ineq4 ennreal_minus flip*: *ennreal_plus*)
then have *le2*: $(\int^+ x.\ ennreal\ (norm\ (F\ n\ x - f\ x))\ \partial M) \le (\int^+ x.\ ennreal\ (norm\ (f\ x)) + ennreal\ (norm\ (F\ n\ x))\ \partial M)$ **for** *n*
by (*rule nn_integral_mono*)

have $2 * (\int^+ x.\ norm(f\ x)\ \partial M) = (\int^+ x.\ liminf\ (\lambda n.\ ennreal\ (G\ n\ x))\ \partial M)$
by (*simp add*: *int_liminf*)
also have ... ≤ $liminf\ (\lambda n.\ (\int^+ x.\ G\ n\ x\ \partial M))$
by (*rule nn_integral_liminf*) *auto*
also have $liminf\ (\lambda n.\ (\int^+ x.\ G\ n\ x\ \partial M)) =$
$liminf\ (\lambda n.\ (\int^+ x.\ ennreal(norm(f\ x)) + ennreal(norm(F\ n\ x))\ \partial M) - (\int^+ x.\ norm(F\ n\ x - f\ x)\ \partial M))$
proof (*intro arg_cong*[**where** *f*=*liminf*] *ext*)
fix *n*
have $\bigwedge x.\ ennreal(G\ n\ x) = ennreal(norm(f\ x)) + ennreal(norm(F\ n\ x)) - ennreal(norm(F\ n\ x - f\ x))$
unfolding *G_def* **by** (*simp add*: *ennreal_minus flip*: *ennreal_plus*)
moreover have $(\int^+ x.\ ennreal(norm(f\ x)) + ennreal(norm(F\ n\ x)) - ennreal(norm(F\ n\ x - f\ x))\ \partial M)$
$= (\int^+ x.\ ennreal(norm(f\ x)) + ennreal(norm(F\ n\ x))\ \partial M) - (\int^+ x.\ norm(F\ n\ x - f\ x)\ \partial M)$
proof (*rule nn_integral_diff*)
from *le* **show** *AE x in M*. *ennreal* (*norm* (*F n x* − *f x*)) ≤ *ennreal* (*norm* (*f x*)) + *ennreal* (*norm* (*F n x*))
by *simp*
from *le2* **have** $(\int^+ x.\ ennreal\ (norm\ (F\ n\ x - f\ x))\ \partial M) < \infty$ **using** *assms*(*1*) *assms*(*2*)
by (*metis has_bochner_integral_implies_finite_norm integrable.simps Bochner_Integration.integrable_diff*)
then show $(\int^+ x.\ ennreal\ (norm\ (F\ n\ x - f\ x))\ \partial M) \neq \infty$ **by** *simp*
qed (*auto simp add*: *assms*)
ultimately show $(\int^+ x.\ G\ n\ x\ \partial M) = (\int^+ x.\ ennreal(norm(f\ x)) + ennreal(norm(F\ n\ x))\ \partial M) - (\int^+ x.\ norm(F\ n\ x - f\ x)\ \partial M)$
by *simp*
qed
finally have $2 * (\int^+ x.\ norm(f\ x)\ \partial M) + limsup\ (\lambda n.\ (\int^+ x.\ norm(F\ n\ x - f\ x)\ \partial M)) \le$
$liminf\ (\lambda n.\ (\int^+ x.\ ennreal(norm(f\ x)) + ennreal(norm(F\ n\ x))\ \partial M) - (\int^+ x.\ norm(F\ n\ x - f\ x)\ \partial M)) +$
$limsup\ (\lambda n.\ (\int^+ x.\ norm(F\ n\ x - f\ x)\ \partial M))$
by (*intro add_mono*) *auto*
also have ... ≤ $(limsup\ (\lambda n.\ \int^+ x.\ ennreal(norm(f\ x)) + ennreal(norm(F\ n\ x))\ \partial M) - limsup\ (\lambda n.\ \int^+ x.\ norm\ (F\ n\ x - f\ x)\ \partial M)) +$
$limsup\ (\lambda n.\ (\int^+ x.\ norm(F\ n\ x - f\ x)\ \partial M))$

by (*intro add_mono liminf_minus_ennreal le2*) *auto*
also have ... = *limsup* (λ*n*. (∫⁺*x*. *ennreal*(*norm*(*f x*)) + *ennreal*(*norm*(*F n x*)) ∂*M*))
by (*intro diff_add_cancel_ennreal Limsup_mono always_eventually allI le2*)
also have ... ≤ *2* ∗ (∫⁺*x*. *norm*(*f x*) ∂*M*)
by *fact*
finally have *limsup* (λ*n*. (∫⁺*x*. *norm*(*F n x* − *f x*) ∂*M*)) = *0*
using *fin2* **by** *simp*
then show *?thesis*
by (*rule tendsto_0_if_Limsup_eq_0_ennreal*)
qed

proposition *Scheffe_lemma2*:
fixes *F*::*nat* ⇒ 'a ⇒ 'b::{*banach*, *second_countable_topology*}
assumes ⋀ *n*::*nat*. *F n* ∈ *borel_measurable M integrable M f*
AE x in M. (λ*n*. *F n x*) ⟶ *f x*
⋀*n*. (∫⁺ *x*. *norm*(*F n x*) ∂*M*) ≤ (∫⁺ *x*. *norm*(*f x*) ∂*M*)
shows (λ*n*. ∫⁺ *x*. *norm*(*F n x* − *f x*) ∂*M*) ⟶ *0*
proof (*rule Scheffe_lemma1*)
fix *n*::*nat*
have (∫⁺ *x*. *norm*(*f x*) ∂*M*) < ∞ **using** *assms*(*2*) **by** (*metis has_bochner_integral_implies_finite_norm integrable.cases*)
then have (∫⁺ *x*. *norm*(*F n x*) ∂*M*) < ∞ **using** *assms*(*4*)[*of n*] **by** *auto*
then show *integrable M* (*F n*) **by** (*subst integrable_iff_bounded*, *simp add*: *assms*(*1*)[*of n*])
qed (*auto simp add*: *assms Limsup_bounded*)

lemma *tendsto_set_lebesgue_integral_at_right*:
fixes *a b* :: *real* **and** *f* :: *real* ⇒ 'a :: {*banach*,*second_countable_topology*}
assumes *a* < *b* **and** *sets*: ⋀*a'*. *a'* ∈ {*a*<..*b*} ⟹ {*a'*..*b*} ∈ *sets M*
and *set_integrable M* {*a*<..*b*} *f*
shows ((λ*a'*. *set_lebesgue_integral M* {*a'*..*b*} *f*) ⟶
set_lebesgue_integral M {*a*<..*b*} *f*) (*at_right a*)
proof (*rule tendsto_at_right_sequentially*[*OF assms*(*1*)], *goal_cases*)
case (*1 S*)
have *eq*: (⋃ *n*. {*S n*..*b*}) = {*a*<..*b*}
proof *safe*
fix *x n* **assume** *x* ∈ {*S n*..*b*}
with *1*(*1*,*2*)[*of n*] **show** *x* ∈ {*a*<..*b*} **by** *auto*
next
fix *x* **assume** *x* ∈ {*a*<..*b*}
with *order_tendstoD*[*OF* ‹*S* ⟶ *a*›, *of x*] **show** *x* ∈ (⋃ *n*. {*S n*..*b*})
by (*force simp*: *eventually_at_top_linorder dest*: *less_imp_le*)
qed
have (λ*n*. *set_lebesgue_integral M* {*S n*..*b*} *f*) ⟶ *set_lebesgue_integral M* (⋃ *n*. {*S n*..*b*}) *f*
by (*rule set_integral_cont_up*) (*insert assms 1*, *auto simp*: *eq incseq_def decseq_def less_imp_le*)
with *eq* **show** *?case* **by** *simp*

qed

The next lemmas relate convergence of integrals over an interval to improper integrals.

lemma *tendsto_set_lebesgue_integral_at_left*:
fixes *a b* :: *real* **and** *f* :: *real* ⇒ *'a* :: {*banach*,*second_countable_topology*}
assumes *a* < *b* **and** *sets*: ⋀*b'*. *b'* ∈ {*a*..<*b*} ⟹ {*a*..*b'*} ∈ *sets M*
and *set_integrable M* {*a*..<*b*} *f*
shows ((λ*b'*. *set_lebesgue_integral M* {*a*..*b'*} *f*) ⟶
set_lebesgue_integral M {*a*..<*b*} *f*) (*at_left b*)
proof (*rule tendsto_at_left_sequentially*[*OF assms*(*1*)], *goal_cases*)
case (*1 S*)
have *eq*: (⋃ *n*. {*a*..*S n*}) = {*a*..<*b*}
proof *safe*
fix *x n* **assume** *x* ∈ {*a*..*S n*}
with *1*(*1*,*2*)[*of n*] **show** *x* ∈ {*a*..<*b*} **by** *auto*
next
fix *x* **assume** *x* ∈ {*a*..<*b*}
with *order_tendstoD*[*OF* ‹*S* ⟶ *b*›, *of x*] **show** *x* ∈ (⋃ *n*. {*a*..*S n*})
by (*force simp*: *eventually_at_top_linorder dest*: *less_imp_le*)
qed
have (λ*n*. *set_lebesgue_integral M* {*a*..*S n*} *f*) ⟶ *set_lebesgue_integral M* (⋃ *n*. {*a*..*S n*}) *f*
by (*rule set_integral_cont_up*) (*insert assms 1*, *auto simp*: *eq incseq_def decseq_def less_imp_le*)
with *eq* **show** *?case* **by** *simp*
qed

proposition *tendsto_set_lebesgue_integral_at_top*:
fixes *f* :: *real* ⇒ *'a*::{*banach*, *second_countable_topology*}
assumes *sets*: ⋀*b*. *b* ≥ *a* ⟹ {*a*..*b*} ∈ *sets M*
and *int*: *set_integrable M* {*a*..} *f*
shows ((λ*b*. *set_lebesgue_integral M* {*a*..*b*} *f*) ⟶ *set_lebesgue_integral M* {*a*..} *f*) *at_top*
proof (*rule tendsto_at_topI_sequentially*)
fix *X* :: *nat* ⇒ *real* **assume** *filterlim X at_top sequentially*
show (λ*n*. *set_lebesgue_integral M* {*a*..*X n*} *f*) ⟶ *set_lebesgue_integral M* {*a*..} *f*
unfolding *set_lebesgue_integral_def*
proof (*rule integral_dominated_convergence*)
show *integrable M* (λ*x*. *indicat_real* {*a*..} *x* ∗*R* *norm* (*f x*))
using *integrable_norm*[*OF int*[*unfolded set_integrable_def*]] **by** *simp*
show *AE x in M*. (λ*n*. *indicator* {*a*..*X n*} *x* ∗*R* *f x*) ⟶ *indicat_real* {*a*..} *x* ∗*R* *f x*
proof
fix *x*
from ‹*filterlim X at_top sequentially*›
have *eventually* (λ*n*. *x* ≤ *X n*) *sequentially*
unfolding *filterlim_at_top_ge*[**where** *c*=*x*] **by** *auto*

```
    then show (λn. indicator {a..X n} x *R f x) ⟶ indicat_real {a..} x *R
f x
        by (intro tendsto_eventually) (auto split: split_indicator elim!: eventu-
ally_mono)
    qed
    fix n show AE x in M. norm (indicator {a..X n} x *R f x) ≤
                          indicator {a..} x *R norm (f x)
      by (auto split: split_indicator)
  next
    from int show (λx. indicat_real {a..} x *R f x) ∈ borel_measurable M
      by (simp add: set_integrable_def)
  next
    fix n :: nat
    from sets have {a..X n} ∈ sets M by (cases X n ≥ a) auto
    with int have set_integrable M {a..X n} f
      by (rule set_integrable_subset) auto
    thus (λx. indicat_real {a..X n} x *R f x) ∈ borel_measurable M
      by (simp add: set_integrable_def)
  qed
qed

proposition tendsto_set_lebesgue_integral_at_bot:
  fixes f :: real ⇒ 'a::{banach, second_countable_topology}
  assumes sets: ⋀a. a ≤ b ⟹ {a..b} ∈ sets M
      and int: set_integrable M {..b} f
    shows ((λa. set_lebesgue_integral M {a..b} f) ⟶ set_lebesgue_integral M
{..b} f) at_bot
proof (rule tendsto_at_botI_sequentially)
  fix X :: nat ⇒ real assume filterlim X at_bot sequentially
  show (λn. set_lebesgue_integral M {X n..b} f) ⟶ set_lebesgue_integral M
{..b} f
    unfolding set_lebesgue_integral_def
  proof (rule integral_dominated_convergence)
    show integrable M (λx. indicat_real {..b} x *R norm (f x))
      using integrable_norm[OF int[unfolded set_integrable_def]] by simp
    show AE x in M. (λn. indicator {X n..b} x *R f x) ⟶ indicat_real {..b}
x *R f x
    proof
      fix x
      from ‹filterlim X at_bot sequentially›
      have eventually (λn. x ≥ X n) sequentially
        unfolding filterlim_at_bot_le[where c=x] by auto
      then show (λn. indicator {X n..b} x *R f x) ⟶ indicat_real {..b} x *R
f x
        by (intro tendsto_eventually) (auto split: split_indicator elim!: eventu-
ally_mono)
    qed
    fix n show AE x in M. norm (indicator {X n..b} x *R f x) ≤
                          indicator {..b} x *R norm (f x)
```

```
    by (auto split: split_indicator)
  next
    from int show (λx. indicat_real {..b} x *R f x) ∈ borel_measurable M
      by (simp add: set_integrable_def)
  next
    fix n :: nat
    from sets have {X n..b} ∈ sets M by (cases X n ≤ b) auto
    with int have set_integrable M {X n..b} f
      by (rule set_integrable_subset) auto
    thus (λx. indicat_real {X n..b} x *R f x) ∈ borel_measurable M
      by (simp add: set_integrable_def)
  qed
qed



theorem integral_Markov_inequality':
  fixes u :: 'a ⇒ real
  assumes [measurable]: set_integrable M A u and A ∈ sets M
  assumes AE x in M. x ∈ A ⟶ u x ≥ 0 and 0 < (c::real)
  shows emeasure M {x∈A. u x ≥ c} ≤ (1/c::real) * (∫ x∈A. u x ∂M)
proof −
  have (λx. u x * indicator A x) ∈ borel_measurable M
    using assms by (auto simp: set_integrable_def mult_ac)
  hence (λx. ennreal (u x * indicator A x)) ∈ borel_measurable M
    by measurable
  also have (λx. ennreal (u x * indicator A x)) = (λx. ennreal (u x) * indicator
A x)
    by (intro ext) (auto simp: indicator_def)
  finally have meas: ... ∈ borel_measurable M .
  from assms(3) have AE: AE x in M. 0 ≤ u x * indicator A x
    by eventually_elim (auto simp: indicator_def)
  have nonneg: set_lebesgue_integral M A u ≥ 0
    unfolding set_lebesgue_integral_def
   by (intro Bochner_Integration.integral_nonneg_AE eventually_mono[OF AE])
(auto simp: mult_ac)

  have A: A ⊆ space M
    using ‹A ∈ sets M› by (simp add: sets.sets_into_space)
  have {x ∈ A. u x ≥ c} = {x ∈ A. ennreal(1/c) * u x ≥ 1}
    using ‹c>0› A by (auto simp: ennreal_mult'[symmetric])
  then have emeasure M {x ∈ A. u x ≥ c} = emeasure M ({x ∈ A. ennreal(1/c)
* u x ≥ 1})
    by simp
  also have ... ≤ ennreal(1/c) * (∫⁺ x. ennreal(u x) * indicator A x ∂M)
    by (intro nn_integral_Markov_inequality meas assms)
 also have (∫⁺ x. ennreal(u x) * indicator A x ∂M) = ennreal (set_lebesgue_integral
M A u)
     unfolding set_lebesgue_integral_def nn_integral_set_ennreal using assms
```

AE
by (*subst nn_integral_eq_integral*) (*simp_all add*: *mult_ac set_integrable_def*)
finally show *?thesis*
using ‹*c* > *0*› *nonneg* **by** (*subst ennreal_mult*) *auto*
qed

theorem *integral_Markov_inequality'_measure*:
assumes [*measurable*]: *set_integrable M A u* **and** *A* ∈ *sets M*
and *AE x in M*. *x* ∈ *A* ⟶ *0* ≤ *u x 0* < (*c*::*real*)
shows *measure M* {*x*∈*A*. *u x* ≥ *c*} ≤ (∫ *x*∈*A*. *u x* ∂*M*) / *c*
proof −
have *nonneg*: *set_lebesgue_integral M A u* ≥ *0*
unfolding *set_lebesgue_integral_def*
by (*intro Bochner_Integration.integral_nonneg_AE eventually_mono*[*OF assms*(*3*)])
(*auto simp*: *mult_ac*)
have *le*: *emeasure M* {*x*∈*A*. *u x* ≥ *c*} ≤ *ennreal* ((*1* / *c*) ∗ (∫ *x*∈*A*. *u x* ∂*M*))
by (*rule integral_Markov_inequality'*) (*use assms* **in** *auto*)
also have ... < *top*
by *auto*
finally have *ennreal* (*measure M* {*x*∈*A*. *u x* ≥ *c*}) = *emeasure M* {*x*∈*A*. *u x* ≥ *c*}
by (*intro emeasure_eq_ennreal_measure* [*symmetric*]) *auto*
also note *le*
finally show *?thesis* **using** *nonneg*
by (*subst* (*asm*) *ennreal_le_iff*)
(*auto intro*!: *divide_nonneg_pos Bochner_Integration.integral_nonneg_AE assms*)
qed

theorem (**in** *finite_measure*) *Chernoff_ineq_ge*:
assumes *s*: *s* > *0*
assumes *integrable*: *set_integrable M A* (λ*x*. *exp* (*s* ∗ *f x*)) **and** *A* ∈ *sets M*
shows *measure M* {*x*∈*A*. *f x* ≥ *a*} ≤ *exp* (−*s* ∗ *a*) ∗ (∫ *x*∈*A*. *exp* (*s* ∗ *f x*) ∂*M*)
proof −
have {*x*∈*A*. *f x* ≥ *a*} = {*x*∈*A*. *exp* (*s* ∗ *f x*) ≥ *exp* (*s* ∗ *a*)}
using *s* **by** *auto*
also have *measure M* ... ≤ *set_lebesgue_integral M A* (λ*x*. *exp* (*s* ∗ *f x*)) / *exp* (*s* ∗ *a*)
by (*intro integral_Markov_inequality'_measure assms*) *auto*
finally show *?thesis*
by (*simp add*: *exp_minus field_simps*)
qed

theorem (**in** *finite_measure*) *Chernoff_ineq_le*:
assumes *s*: *s* > *0*
assumes *integrable*: *set_integrable M A* (λ*x*. *exp* (−*s* ∗ *f x*)) **and** *A* ∈ *sets M*
shows *measure M* {*x*∈*A*. *f x* ≤ *a*} ≤ *exp* (*s* ∗ *a*) ∗ (∫ *x*∈*A*. *exp* (−*s* ∗ *f x*) ∂*M*)
proof −
have {*x*∈*A*. *f x* ≤ *a*} = {*x*∈*A*. *exp* (−*s* ∗ *f x*) ≥ *exp* (−*s* ∗ *a*)}

using *s* **by** *auto*
also have *measure M ... ≤ set_lebesgue_integral M A* (λ*x*. *exp* (−*s* ∗ *f x*)) / *exp* (−*s* ∗ *a*)
by (*intro integral_Markov_inequality'_measure assms*) *auto*
finally show *?thesis*
by (*simp add*: *exp_minus field_simps*)
qed

end

6.16 Homeomorphism Theorems

theory *Homeomorphism*
imports *Homotopy*
begin

lemma *homeomorphic_spheres'*:
fixes *a* ::'*a*::*euclidean_space* **and** *b* ::'*b*::*euclidean_space*
assumes *0* < *δ* **and** *dimeq*: *DIM*('*a*) = *DIM*('*b*)
shows (*sphere a δ*) *homeomorphic* (*sphere b δ*)
proof −
obtain *f* :: '*a*⇒'*b* **and** *g* **where** *linear f linear g*
and *fg*: ⋀*x*. *norm*(*f x*) = *norm x* ⋀*y*. *norm*(*g y*) = *norm y* ⋀*x*. *g*(*f x*) = *x* ⋀*y*. *f*(*g y*) = *y*
by (*blast intro*: *isomorphisms_UNIV_UNIV* [*OF dimeq*])
then have *continuous_on UNIV f continuous_on UNIV g*
using *linear_continuous_on linear_linear* **by** *blast*+
then show *?thesis*
unfolding *homeomorphic_minimal*
apply(*rule_tac x*=λ*x*. *b* + *f*(*x* − *a*) **in** *exI*)
apply(*rule_tac x*=λ*x*. *a* + *g*(*x* − *b*) **in** *exI*)
using *assms*
apply (*force intro*: *continuous_intros*
continuous_on_compose2 [*of* _ *f*] *continuous_on_compose2* [*of* _ *g*] *simp*: *dist_commute dist_norm fg*)
done
qed

lemma *homeomorphic_spheres_gen*:
fixes *a* :: '*a*::*euclidean_space* **and** *b* :: '*b*::*euclidean_space*
assumes *0* < *r 0* < *s DIM*('*a*::*euclidean_space*) = *DIM*('*b*::*euclidean_space*)
shows (*sphere a r homeomorphic sphere b s*)
using *assms homeomorphic_trans* [*OF homeomorphic_spheres homeomorphic_spheres'*]
by *auto*

6.16.1 Homeomorphism of all convex compact sets with nonempty interior

proposition

```
  fixes S :: 'a::euclidean_space set
  assumes compact S and 0: 0 ∈ rel_interior S
      and star: ⋀x. x ∈ S ⟹ open_segment 0 x ⊆ rel_interior S
    shows starlike_compact_projective1_0:
            S − rel_interior S homeomorphic sphere 0 1 ∩ affine hull S
            (is ?SMINUS homeomorphic ?SPHER)
      and starlike_compact_projective2_0:
            S homeomorphic cball 0 1 ∩ affine hull S
            (is S homeomorphic ?CBALL)
proof −
  have starI: (u *R x) ∈ rel_interior S if x ∈ S 0 ≤ u u < 1 for x u
  proof (cases x=0 ∨ u=0)
    case True with 0 show ?thesis by force
  next
    case False with that show ?thesis
      by (auto simp: in_segment intro: star [THEN subsetD])
  qed
  have 0 ∈ S  using assms rel_interior_subset by auto
  define proj where proj ≡ λx::'a. x /R norm x
  have eqI: x = y if proj x = proj y norm x = norm y for x y
    using that  by (force simp: proj_def)
  then have iff_eq: ⋀x y. (proj x = proj y ∧ norm x = norm y) ⟷ x = y
    by blast
  have projI: x ∈ affine hull S ⟹ proj x ∈ affine hull S for x
    by (metis ‹0 ∈ S› affine_hull_span_0 hull_inc span_mul proj_def)
  have nproj1 [simp]: x ≠ 0 ⟹ norm(proj x) = 1 for x
    by (simp add: proj_def)
  have proj0_iff [simp]: proj x = 0 ⟷ x = 0 for x
    by (simp add: proj_def)
  have cont_proj: continuous_on (UNIV − {0}) proj
    unfolding proj_def by (rule continuous_intros | force)+
  have proj_spherI: ⋀x. ⟦x ∈ affine hull S; x ≠ 0⟧ ⟹ proj x ∈ ?SPHER
    by (simp add: projI)
  have bounded S closed S
    using ‹compact S› compact_eq_bounded_closed by blast+
  have inj_on_proj: inj_on proj (S − rel_interior S)
  proof
    fix x y
    assume x: x ∈ S − rel_interior S
      and y: y ∈ S − rel_interior S and eq: proj x = proj y
    then have xynot: x ≠ 0 y ≠ 0 x ∈ S y ∈ S x ∉ rel_interior S y ∉ rel_interior
S
      using 0 by auto
    consider norm x = norm y | norm x < norm y | norm x > norm y by linarith
    then show x = y
    proof cases
      assume norm x = norm y
        with iff_eq eq show x = y by blast
    next
```

```
      assume ∗: norm x < norm y
      have x /R norm x = norm x ∗R (x /R norm x) /R norm (norm x ∗R (x /R
norm x))
        by force
      then have proj ((norm x / norm y) ∗R y) = proj x
        by (metis (no_types) divide_inverse local.proj_def eq scaleR_scaleR)
      then have [simp]: (norm x / norm y) ∗R y = x
        by (rule eqI) (simp add: ‹y ≠ 0›)
      have no: 0 ≤ norm x / norm y norm x / norm y < 1
        using ∗ by (auto simp: field_split_simps)
      then show x = y
        using starI [OF ‹y ∈ S› no] xynot by auto
    next
      assume ∗: norm x > norm y
      have y /R norm y = norm y ∗R (y /R norm y) /R norm (norm y ∗R (y /R
norm y))
        by force
      then have proj ((norm y / norm x) ∗R x) = proj y
        by (metis (no_types) divide_inverse local.proj_def eq scaleR_scaleR)
      then have [simp]: (norm y / norm x) ∗R x = y
        by (rule eqI) (simp add: ‹x ≠ 0›)
      have no: 0 ≤ norm y / norm x norm y / norm x < 1
        using ∗ by (auto simp: field_split_simps)
      then show x = y
        using starI [OF ‹x ∈ S› no] xynot by auto
    qed
  qed
  have ∃surf. homeomorphism (S − rel_interior S) ?SPHER proj surf
  proof (rule homeomorphism_compact)
    show compact (S − rel_interior S)
      using ‹compact S› compact_rel_boundary by blast
    show continuous_on (S − rel_interior S) proj
      using 0 by (blast intro: continuous_on_subset [OF cont_proj])
    show proj ‘ (S − rel_interior S) = ?SPHER
    proof
      show proj ‘ (S − rel_interior S) ⊆ ?SPHER
        using 0 by (force simp: hull_inc projI intro: nproj1)
      show ?SPHER ⊆ proj ‘ (S − rel_interior S)
      proof (clarsimp simp: proj_def)
        fix x
        assume x ∈ affine hull S and nox: norm x = 1
        then have x ≠ 0 by auto
        obtain d where 0 < d and dx: (d ∗R x) ∈ rel_frontier S
          and ri: ⋀e. [[0 ≤ e; e < d]] ⟹ (e ∗R x) ∈ rel_interior S
          using ray_to_rel_frontier [OF ‹bounded S› 0] ‹x ∈ affine hull S› ‹x ≠
0› by auto
        show x ∈ (λx. x /R norm x) ‘ (S − rel_interior S)
        proof
          show x = d ∗R x /R norm (d ∗R x)
```

```
          using ‹0 < d› by (auto simp: nox)
        show d *R x ∈ S − rel_interior S
          using dx ‹closed S› by (auto simp: rel_frontier_def)
      qed
    qed
  qed
  qed (rule inj_on_proj)
  then obtain surf where surf: homeomorphism (S − rel_interior S) ?SPHER
proj surf
    by blast
  then have cont_surf: continuous_on (proj ‘ (S − rel_interior S)) surf
    by (auto simp: homeomorphism_def)
  have surf_nz: ⋀x. x ∈ ?SPHER ⟹ surf x ≠ 0
    by (metis 0 DiffE homeomorphism_def imageI surf)
  have cont_nosp: continuous_on (?SPHER) (λx. norm x *R ((surf o proj) x))
  proof (intro continuous_intros)
    show continuous_on (sphere 0 1 ∩ affine hull S) proj
      by (rule continuous_on_subset [OF cont_proj], force)
    show continuous_on (proj ‘ (sphere 0 1 ∩ affine hull S)) surf
      by (intro continuous_on_subset [OF cont_surf]) (force simp: homeomor-
phism_image1 [OF surf] dest: proj_spherI)
  qed
  have surfpS: ⋀x. ⟦norm x = 1; x ∈ affine hull S⟧ ⟹ surf (proj x) ∈ S
   by (metis (full_types) DiffE ‹0 ∈ S› homeomorphism_def image_eqI norm_zero
proj_spherI real_vector.scale_zero_left scaleR_one surf)
  have *: ∃y. norm y = 1 ∧ y ∈ affine hull S ∧ x = surf (proj y)
        if x ∈ S x ∉ rel_interior S for x
  proof −
    have proj x ∈ ?SPHER
      by (metis (full_types) 0 hull_inc proj_spherI that)
    moreover have surf (proj x) = x
      by (metis Diff_iff homeomorphism_def surf that)
    ultimately show ?thesis
     by (metis ‹⋀x. x ∈ ?SPHER ⟹ surf x ≠ 0› hull_inc inverse_1 local.proj_def
norm_sgn projI scaleR_one sgn_div_norm that(1))
  qed
   have surfp_notin: ⋀x. ⟦norm x = 1; x ∈ affine hull S⟧ ⟹ surf (proj x) ∉
rel_interior S
     by (metis (full_types) DiffE one_neq_zero homeomorphism_def image_eqI
norm_zero proj_spherI surf)
  have no_sp_im: (λx. norm x *R surf (proj x)) ‘ (?SPHER) = S − rel_interior
S
    by (auto simp: surfpS image_def Bex_def surfp_notin *)
  have inj_spher: inj_on (λx. norm x *R surf (proj x)) ?SPHER
  proof
    fix x y
    assume xy: x ∈ ?SPHER y ∈ ?SPHER
      and eq:  norm x *R surf (proj x) = norm y *R surf (proj y)
    then have norm x = 1 norm y = 1 x ∈ affine hull S y ∈ affine hull S
```

```
    using 0 by auto
   with eq show x = y
    by (simp add: proj_def) (metis surf xy homeomorphism_def)
  qed
  have co01: compact ?SPHER
   by (simp add: compact_Int_closed)
  show ?SMINUS homeomorphic ?SPHER
   using homeomorphic_def surf by blast
  have proj_scaleR: ⋀a x. 0 < a ⟹ proj (a *R x) = proj x
   by (simp add: proj_def)
  have cont_sp0: continuous_on (affine hull S − {0}) (surf o proj)
  proof (rule continuous_on_compose [OF continuous_on_subset [OF cont_proj]])
   show continuous_on (proj ‘ (affine hull S − {0})) surf
   using homeomorphism_image1 proj_spherI surf by (intro continuous_on_subset
[OF cont_surf]) fastforce
  qed auto
  obtain B where B>0 and B: ⋀x. x ∈ S ⟹ norm x ≤ B
   by (metis compact_imp_bounded ‹compact S› bounded_pos_less less_eq_real_def)
  have cont_nosp: continuous (at x within ?CBALL) (λx. norm x *R surf (proj
x))
       if norm x ≤ 1 x ∈ affine hull S for x
  proof (cases x=0)
   case True
   have (norm ⟶ 0) (at 0 within cball 0 1 ∩ affine hull S)
    by (simp add: tendsto_norm_zero eventually_at)
   with True show ?thesis
    apply (simp add: continuous_within)
    apply (rule lim_null_scaleR_bounded [where B=B])
    using B ‹0 < B› local.proj_def projI surfpS by (auto simp: eventually_at)
  next
   case False
   then have ∀F x in at x. (x ∈ affine hull S − {0}) = (x ∈ affine hull S)
    by (force simp: False eventually_at)
   moreover
   have continuous (at x within affine hull S − {0}) (λx. surf (proj x))
   using cont_sp0 False that by (auto simp add: continuous_on_eq_continuous_within)
   ultimately have *: continuous (at x within affine hull S) (λx. surf (proj x))
   by (simp add: continuous_within Lim_transform_within_set continuous_on_eq_continuous_within)
   show ?thesis
   by (intro continuous_within_subset [where S = affine hull S, OF _ Int_lower2]
continuous_intros *)
  qed
  have cont_nosp2: continuous_on ?CBALL (λx. norm x *R ((surf o proj) x))
   by (simp add: continuous_on_eq_continuous_within cont_nosp)
  have norm y *R surf (proj y) ∈ S if y ∈ cball 0 1 and yaff: y ∈ affine hull S
for y
  proof (cases y=0)
   case True then show ?thesis
    by (simp add: ‹0 ∈ S›)
```

```
  next
    case False
    then have norm y *R surf (proj y) = norm y *R surf (proj (y /R norm y))
      by (simp add: proj_def)
    have norm y ≤ 1 using that by simp
    have surf (proj (y /R norm y)) ∈ S
      using False local.proj_def nproj1 projI surfpS yaff by blast
    then have surf (proj y) ∈ S
      by (simp add: False proj_def)
    then show norm y *R surf (proj y) ∈ S
    by (metis dual_order.antisym le_less_linear norm_ge_zero rel_interior_subset
scaleR_one
              starI subset_eq ‹norm y ≤ 1›)
  qed
  moreover have x ∈ (λx. norm x *R surf (proj x)) ` (?CBALL) if x ∈ S for x
  proof (cases x=0)
    case True with that hull_inc  show ?thesis by fastforce
  next
    case False
    then have psp: proj (surf (proj x)) = proj x
      by (metis homeomorphism_def hull_inc proj_spherI surf that)
    have nxx: norm x *R proj x = x
      by (simp add: False local.proj_def)
    have affineI: (1 / norm (surf (proj x))) *R x ∈ affine hull S
      by (metis ‹0 ∈ S› affine_hull_span_0 hull_inc span_clauses(4) that)
    have sproj_nz: surf (proj x) ≠ 0
      by (metis False proj0_iff psp)
    then have proj x = proj (proj x)
      by (metis False nxx proj_scaleR zero_less_norm_iff)
    moreover have scaleproj: ⋀a r. r *R proj a = (r / norm a) *R a
      by (simp add: divide_inverse local.proj_def)
    ultimately have (norm (surf (proj x)) / norm x) *R x ∉ rel_interior S
     by (metis (no_types) sproj_nz divide_self_if hull_inc norm_eq_zero nproj1
projI psp scaleR_one surfp_notin that)
    then have (norm (surf (proj x)) / norm x) ≥ 1
     using starI [OF that] by (meson starI [OF that] le_less_linear norm_ge_zero
zero_le_divide_iff)
    then have nole: norm x ≤ norm (surf (proj x))
      by (simp add: le_divide_eq_1)
    let ?inx = x /R norm (surf (proj x))
    show ?thesis
    proof
      show x = norm ?inx *R surf (proj ?inx)
      by (simp add: field_simps) (metis inverse_eq_divide nxx positive_imp_inverse_positive
proj_scaleR psp scaleproj sproj_nz zero_less_norm_iff)
      qed (auto simp: field_split_simps nole affineI)
  qed
  ultimately have im_cball: (λx. norm x *R surf (proj x)) ` ?CBALL = S
    by blast
```

```
  have inj_cball: inj_on (λx. norm x *R surf (proj x)) ?CBALL
  proof
    fix x y
    assume x ∈ ?CBALL y ∈ ?CBALL
      and eq: norm x *R surf (proj x) = norm y *R surf (proj y)
    then have x: x ∈ affine hull S and y: y ∈ affine hull S
      using 0 by auto
    show x = y
    proof (cases x=0 ∨ y=0)
      case True then show x = y using eq proj_spherI surf_nz x y by force
    next
      case False
      with x y have speq: surf (proj x) = surf (proj y)
      by (metis eq homeomorphism_apply2 proj_scaleR proj_spherI surf zero_less_norm_iff)
      then have norm x = norm y
      by (metis ‹x ∈ affine hull S› ‹y ∈ affine hull S› eq proj_spherI real_vector.scale_cancel_right
surf_nz)
      moreover have proj x = proj y
        by (metis (no_types) False speq homeomorphism_apply2 proj_spherI surf x
y)
      ultimately show x = y
        using eq eqI by blast
    qed
  qed
  have co01: compact ?CBALL
    by (simp add: compact_Int_closed)
  show S homeomorphic ?CBALL
    using homeomorphic_compact [OF co01 cont_nosp2 [unfolded o_def] im_cball
inj_cball] homeomorphic_sym by blast
qed

corollary
  fixes S :: 'a::euclidean_space set
  assumes compact S and a: a ∈ rel_interior S
      and star: ⋀x. x ∈ S ⟹ open_segment a x ⊆ rel_interior S
    shows starlike_compact_projective1:
            S − rel_interior S homeomorphic sphere a 1 ∩ affine hull S
      and starlike_compact_projective2:
            S homeomorphic cball a 1 ∩ affine hull S
proof −
  have 1: compact ((+) (−a) ` S) by (meson assms compact_translation)
  have 2: 0 ∈ rel_interior ((+) (−a) ` S)
    using a rel_interior_translation [of − a S] by (simp cong: image_cong_simp)
  have 3: open_segment 0 x ⊆ rel_interior ((+) (−a) ` S) if x ∈ ((+) (−a) ` S)
for x
  proof −
    have x+a ∈ S using that by auto
    then have open_segment a (x+a) ⊆ rel_interior S by (metis star)
    then show ?thesis using open_segment_translation [of a 0 x]
```

using *rel_interior_translation* [*of* − *a S*] **by** (*fastforce simp add*: *ac_simps image_iff cong*: *image_cong_simp*)
qed
have *S* − *rel_interior S homeomorphic* ((+) (−*a*) ‘ *S*) − *rel_interior* ((+) (−*a*) ‘ *S*)
by (*metis rel_interior_translation translation_diff homeomorphic_translation*)
also have ... *homeomorphic sphere 0 1* ∩ *affine hull* ((+) (−*a*) ‘ *S*)
by (*rule starlike_compact_projective1_0* [*OF 1 2 3*])
also have ... = (+) (−*a*) ‘ (*sphere a 1* ∩ *affine hull S*)
by (*metis affine_hull_translation left_minus sphere_translation translation_Int*)
also have ... *homeomorphic sphere a 1* ∩ *affine hull S*
using *homeomorphic_translation homeomorphic_sym* **by** *blast*
finally show *S* − *rel_interior S homeomorphic sphere a 1* ∩ *affine hull S* **.**

have *S homeomorphic* ((+) (−*a*) ‘ *S*)
by (*metis homeomorphic_translation*)
also have ... *homeomorphic cball 0 1* ∩ *affine hull* ((+) (−*a*) ‘ *S*)
by (*rule starlike_compact_projective2_0* [*OF 1 2 3*])
also have ... = (+) (−*a*) ‘ (*cball a 1* ∩ *affine hull S*)
by (*metis affine_hull_translation left_minus cball_translation translation_Int*)
also have ... *homeomorphic cball a 1* ∩ *affine hull S*
using *homeomorphic_translation homeomorphic_sym* **by** *blast*
finally show *S homeomorphic cball a 1* ∩ *affine hull S* **.**
qed

corollary *starlike_compact_projective_special*:
assumes *compact S*
and *cb01*: *cball* (*0*::*'a*::*euclidean_space*) *1* ⊆ *S*
and *scale*: ⋀*x u*. ⟦*x* ∈ *S*; *0* ≤ *u*; *u* < *1*⟧ ⟹ *u* ∗*R x* ∈ *S* − *frontier S*
shows *S homeomorphic* (*cball* (*0*::*'a*::*euclidean_space*) *1*)
proof −
have *ball 0 1* ⊆ *interior S*
using *cb01 interior_cball interior_mono* **by** *blast*
then have *0*: *0* ∈ *rel_interior S*
by (*meson centre_in_ball subsetD interior_subset_rel_interior le_numeral_extra*(*2*) *not_le*)
have [*simp*]: *affine hull S* = *UNIV*
using ‹*ball 0 1* ⊆ *interior S*› **by** (*auto intro*!: *affine_hull_nonempty_interior*)
have *star*: *open_segment 0 x* ⊆ *rel_interior S* **if** *x* ∈ *S* **for** *x*
proof
fix *p* **assume** *p* ∈ *open_segment 0 x*
then obtain *u* **where** *x* ≠ *0* **and** *u*: *0* ≤ *u u* < *1* **and** *p*: *u* ∗*R x* = *p*
by (*auto simp*: *in_segment*)
then show *p* ∈ *rel_interior S*
using *scale* [*OF that u*] *closure_subset frontier_def interior_subset_rel_interior*
by *fastforce*
qed
show *?thesis*
using *starlike_compact_projective2_0* [*OF* ‹*compact S*› *0 star*] **by** *simp*

qed

lemma *homeomorphic_convex_lemma*:
fixes $S :: 'a::euclidean\_space\ set$ **and** $T :: 'b::euclidean\_space\ set$
assumes *convex S compact S convex T compact T*
and *affeq*: *aff_dim S = aff_dim T*
shows *(S − rel_interior S) homeomorphic (T − rel_interior T) ∧ S homeomorphic T*
proof *(cases rel_interior S = {} ∨ rel_interior T = {})*
case *True*
then show *?thesis*
by *(metis Diff_empty affeq ‹convex S› ‹convex T› aff_dim_empty homeomorphic_empty rel_interior_eq_empty aff_dim_empty)*
next
case *False*
then obtain *a b* **where** *a*: *a ∈ rel_interior S* **and** *b*: *b ∈ rel_interior T* **by** *auto*
have *starS*: *⋀x. x ∈ S ⟹ open_segment a x ⊆ rel_interior S*
using *rel_interior_closure_convex_segment a ‹convex S› closure_subset subsetCE* **by** *blast*
have *starT*: *⋀x. x ∈ T ⟹ open_segment b x ⊆ rel_interior T*
using *rel_interior_closure_convex_segment b ‹convex T› closure_subset subsetCE* **by** *blast*
let *?aS = (+) (−a) ‘ S* **and** *?bT = (+) (−b) ‘ T*
have *0*: *0 ∈ affine hull ?aS 0 ∈ affine hull ?bT*
by *(metis a b subsetD hull_inc image_eqI left_minus rel_interior_subset)+*
have *subs*: *subspace (span ?aS) subspace (span ?bT)*
by *(rule subspace_span)+*
moreover
have *dim (span ((+) (− a) ‘ S)) = dim (span ((+) (− b) ‘ T))*
by *(metis 0 aff_dim_translation_eq aff_dim_zero affeq dim_span nat_int)*
ultimately obtain *f g* **where** *linear f linear g*
and *fim*: *f ‘ span ?aS = span ?bT*
and *gim*: *g ‘ span ?bT = span ?aS*
and *fno*: *⋀x. x ∈ span ?aS ⟹ norm(f x) = norm x*
and *gno*: *⋀x. x ∈ span ?bT ⟹ norm(g x) = norm x*
and *gf*: *⋀x. x ∈ span ?aS ⟹ g(f x) = x*
and *fg*: *⋀x. x ∈ span ?bT ⟹ f(g x) = x*
by *(rule isometries_subspaces) blast*
have *[simp]*: *continuous_on A f* **for** *A*
using *‹linear f› linear_conv_bounded_linear linear_continuous_on* **by** *blast*
have *[simp]*: *continuous_on B g* **for** *B*
using *‹linear g› linear_conv_bounded_linear linear_continuous_on* **by** *blast*
have *eqspanS*: *affine hull ?aS = span ?aS*
by *(metis a affine_hull_span_0 subsetD hull_inc image_eqI left_minus rel_interior_subset)*
have *eqspanT*: *affine hull ?bT = span ?bT*
by *(metis b affine_hull_span_0 subsetD hull_inc image_eqI left_minus rel_interior_subset)*
have *S homeomorphic cball a 1 ∩ affine hull S*
by *(rule starlike_compact_projective2 [OF ‹compact S› a starS])*
also have *... homeomorphic (+) (−a) ‘ (cball a 1 ∩ affine hull S)*

```
    by (metis homeomorphic_translation)
  also have ... = cball 0 1 ∩ (+) (−a) ‘ (affine hull S)
    by (auto simp: dist_norm)
  also have ... = cball 0 1 ∩ span ?aS
    using eqspanS affine_hull_translation by blast
  also have ... homeomorphic cball 0 1 ∩ span ?bT
  proof (rule homeomorphicI)
    show fim1: f ‘ (cball 0 1 ∩ span ?aS) = cball 0 1 ∩ span ?bT
    proof
      show f ‘ (cball 0 1 ∩ span ?aS) ⊆ cball 0 1 ∩ span ?bT
        using fim fno by auto
      show cball 0 1 ∩ span ?bT ⊆ f ‘ (cball 0 1 ∩ span ?aS)
        by clarify (metis IntI fg gim gno image_eqI mem_cball_0)
    qed
    show g ‘ (cball 0 1 ∩ span ?bT) = cball 0 1 ∩ span ?aS
    proof
      show g ‘ (cball 0 1 ∩ span ?bT) ⊆ cball 0 1 ∩ span ?aS
        using gim gno by auto
      show cball 0 1 ∩ span ?aS ⊆ g ‘ (cball 0 1 ∩ span ?bT)
        by clarify (metis IntI fim1 gf image_eqI)
    qed
  qed (auto simp: fg gf)
  also have ... = cball 0 1 ∩ (+) (−b) ‘ (affine hull T)
    using eqspanT affine_hull_translation by blast
  also have ... = (+) (−b) ‘ (cball b 1 ∩ affine hull T)
    by (auto simp: dist_norm)
  also have ... homeomorphic (cball b 1 ∩ affine hull T)
    by (metis homeomorphic_translation homeomorphic_sym)
  also have ... homeomorphic T
    by (metis starlike_compact_projective2 [OF ‹compact T› b starT] homeomor-
phic_sym)
  finally have 1: S homeomorphic T .

  have S − rel_interior S homeomorphic sphere a 1 ∩ affine hull S
    by (rule starlike_compact_projective1 [OF ‹compact S› a starS])
  also have ... homeomorphic (+) (−a) ‘ (sphere a 1 ∩ affine hull S)
    by (metis homeomorphic_translation)
  also have ... = sphere 0 1 ∩ (+) (−a) ‘ (affine hull S)
    by (auto simp: dist_norm)
  also have ... = sphere 0 1 ∩ span ?aS
    using eqspanS affine_hull_translation by blast
  also have ... homeomorphic sphere 0 1 ∩ span ?bT
  proof (rule homeomorphicI)
    show fim1: f ‘ (sphere 0 1 ∩ span ?aS) = sphere 0 1 ∩ span ?bT
    proof
      show f ‘ (sphere 0 1 ∩ span ?aS) ⊆ sphere 0 1 ∩ span ?bT
        using fim fno by auto
      show sphere 0 1 ∩ span ?bT ⊆ f ‘ (sphere 0 1 ∩ span ?aS)
        by clarify (metis IntI fg gim gno image_eqI mem_sphere_0)
```

```
    qed
    show g ‘ (sphere 0 1 ∩ span ?bT) = sphere 0 1 ∩ span ?aS
    proof
      show g ‘ (sphere 0 1 ∩ span ?bT) ⊆ sphere 0 1 ∩ span ?aS
        using gim gno by auto
      show sphere 0 1 ∩ span ?aS ⊆ g ‘ (sphere 0 1 ∩ span ?bT)
        by clarify (metis IntI fim1 gf image_eqI)
    qed
  qed (auto simp: fg gf)
  also have ... = sphere 0 1 ∩ (+) (−b) ‘ (affine hull T)
    using eqspanT affine_hull_translation by blast
  also have ... = (+) (−b) ‘ (sphere b 1 ∩ affine hull T)
    by (auto simp: dist_norm)
  also have ... homeomorphic (sphere b 1 ∩ affine hull T)
    by (metis homeomorphic_translation homeomorphic_sym)
  also have ... homeomorphic T − rel_interior T
    by (metis starlike_compact_projective1 [OF ‹compact T› b starT] homeomor-
phic_sym)
  finally have 2: S − rel_interior S homeomorphic T − rel_interior T .
  show ?thesis
    using 1 2 by blast
qed

lemma homeomorphic_convex_compact_sets:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes convex S compact S convex T compact T
      and affeq: aff_dim S = aff_dim T
    shows S homeomorphic T
using homeomorphic_convex_lemma [OF assms] assms
by (auto simp: rel_frontier_def)

lemma homeomorphic_rel_frontiers_convex_bounded_sets:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes convex S bounded S convex T bounded T
      and affeq: aff_dim S = aff_dim T
    shows  rel_frontier S homeomorphic rel_frontier T
using assms homeomorphic_convex_lemma [of closure S closure T]
by (simp add: rel_frontier_def convex_rel_interior_closure)
```

6.16.2 Homeomorphisms between punctured spheres and affine sets

Including the famous stereoscopic projection of the 3-D sphere to the complex plane

The special case with centre 0 and radius 1

```
lemma homeomorphic_punctured_affine_sphere_affine_01:
  assumes b ∈ sphere 0 1 affine T 0 ∈ T b ∈ T affine p
      and affT: aff_dim T = aff_dim p + 1
```

```
    shows (sphere 0 1 ∩ T) − {b} homeomorphic p
proof −
  have [simp]: norm b = 1 b•b = 1
    using assms by (auto simp: norm_eq_1)
  have [simp]: T ∩ {v. b•v = 0} ≠ {}
    using ‹0 ∈ T› by auto
  have [simp]: ¬ T ⊆ {v. b•v = 0}
    using ‹norm b = 1› ‹b ∈ T› by auto
  define f where f ≡ λx. 2 *R b + (2 / (1 − b•x)) *R (x − b)
  define g where g ≡ λy. b + (4 / (norm y ^ 2 + 4)) *R (y − 2 *R b)
  have fg[simp]: ⋀x. ⟦x ∈ T; b•x = 0⟧ ⟹ f (g x) = x
   unfolding f_def g_def by (simp add: algebra_simps field_split_simps add_nonneg_eq_0_iff)
  have no: (norm (f x))^2 = 4 * (1 + b • x) / (1 − b • x)
    if norm x = 1 and b • x ≠ 1 for x
    using that sum_sqs_eq [of 1 b • x]
    apply (simp flip: dot_square_norm add: norm_eq_1 nonzero_eq_divide_eq)
    apply (simp add: f_def vector_add_divide_simps inner_simps)
    apply (auto simp add: field_split_simps inner_commute)
    done
  have [simp]: ⋀u::real. 8 + u * (u * 8) = u * 16 ⟷ u=1
    by algebra
  have gf[simp]: ⋀x. ⟦norm x = 1; b • x ≠ 1⟧ ⟹ g (f x) = x
    unfolding g_def no by (auto simp: f_def field_split_simps)
  have g1: norm (g x) = 1 if x ∈ T and b • x = 0 for x
    using that
    apply (simp only: g_def)
    apply (rule power2_eq_imp_eq)
   apply (simp_all add: dot_square_norm [symmetric] divide_simps vector_add_divide_simps)
    apply (simp add: algebra_simps inner_commute)
    done
  have ne1: b • g x ≠ 1 if x ∈ T and b • x = 0 for x
    using that unfolding g_def
   apply (simp_all add: dot_square_norm [symmetric] divide_simps vector_add_divide_simps
add_nonneg_eq_0_iff)
    apply (auto simp: algebra_simps)
    done
  have subspace T
    by (simp add: assms subspace_affine)
  have gT: ⋀x. ⟦x ∈ T; b • x = 0⟧ ⟹ g x ∈ T
    unfolding g_def
   by (blast intro: ‹subspace T› ‹b ∈ T› subspace_add subspace_mul subspace_diff)
  have f ‘ {x. norm x = 1 ∧ b•x ≠ 1} ⊆ {x. b•x = 0}
    unfolding f_def using ‹norm b = 1› norm_eq_1
    by (force simp: field_simps inner_add_right inner_diff_right)
  moreover have f ‘ T ⊆ T
    unfolding f_def using assms ‹subspace T›
    by (auto simp add: inner_add_right inner_diff_right mem_affine_3_minus
subspace_mul)
  moreover have {x. b•x = 0} ∩ T ⊆ f ‘ ({x. norm x = 1 ∧ b•x ≠ 1} ∩ T)
```

```
    by clarify (metis (mono_tags) IntI ne1 fg gT g1 imageI mem_Collect_eq)
  ultimately have imf: f ‘ ({x. norm x = 1 ∧ b•x ≠ 1} ∩ T) = {x. b•x = 0} ∩
T
    by blast
  have no4: ⋀y. b•y = 0 ⟹ norm ((y•y + 4) *R b + 4 *R (y − 2 *R b)) = y•y
+ 4
    apply (rule power2_eq_imp_eq)
    apply (simp_all flip: dot_square_norm)
    apply (auto simp: power2_eq_square algebra_simps inner_commute)
    done
  have [simp]: ⋀x. ⟦norm x = 1; b • x ≠ 1⟧ ⟹ b • f x = 0
    by (simp add: f_def algebra_simps field_split_simps)
  have [simp]: ⋀x. ⟦x ∈ T; norm x = 1; b • x ≠ 1⟧ ⟹ f x ∈ T
    unfolding f_def
   by (blast intro: ‹subspace T› ‹b ∈ T› subspace_add subspace_mul subspace_diff)
  have g ‘ {x. b•x = 0} ⊆ {x. norm x = 1 ∧ b•x ≠ 1}
    unfolding g_def
   apply (clarsimp simp: no4 vector_add_divide_simps divide_simps add_nonneg_eq_0_iff
dot_square_norm [symmetric])
    apply (auto simp: algebra_simps)
    done
  moreover have g ‘ T ⊆ T
    unfolding g_def
   by (blast intro: ‹subspace T› ‹b ∈ T› subspace_add subspace_mul subspace_diff)
  moreover have {x. norm x = 1 ∧ b•x ≠ 1} ∩ T ⊆ g ‘ ({x. b•x = 0} ∩ T)
    by clarify (metis (mono_tags, lifting) IntI gf image_iff imf mem_Collect_eq)
  ultimately have img: g ‘ ({x. b•x = 0} ∩ T) = {x. norm x = 1 ∧ b•x ≠ 1} ∩
T
    by blast
  have aff: affine ({x. b•x = 0} ∩ T)
    by (blast intro: affine_hyperplane assms)
  have contf: continuous_on ({x. norm x = 1 ∧ b•x ≠ 1} ∩ T) f
    unfolding f_def by (rule continuous_intros | force)+
  have contg: continuous_on ({x. b•x = 0} ∩ T) g
   unfolding g_def by (rule continuous_intros | force simp: add_nonneg_eq_0_iff)+
  have (sphere 0 1 ∩ T) − {b} = {x. norm x = 1 ∧ (b•x ≠ 1)} ∩ T
    using ‹norm b = 1› by (auto simp: norm_eq_1) (metis vector_eq ‹b•b = 1›)
  also have ... homeomorphic {x. b•x = 0} ∩ T
    by (rule homeomorphicI [OF imf img contf contg]) auto
  also have ... homeomorphic p
  proof (rule homeomorphic_affine_sets [OF aff ‹affine p›])
    show aff_dim ({x. b • x = 0} ∩ T) = aff_dim p
      by (simp add: Int_commute aff_dim_affine_Int_hyperplane [OF ‹affine T›]
affT)
  qed
  finally show ?thesis .
qed

theorem homeomorphic_punctured_affine_sphere_affine:
```

fixes $a :: 'a :: euclidean\_space$
assumes $0 < r$ $b \in sphere\ a\ r$ $affine\ T$ $a \in T$ $b \in T$ $affine\ p$
and *aff*: $aff\_dim\ T = aff\_dim\ p + 1$
shows $(sphere\ a\ r \cap T) - \{b\}$ *homeomorphic* p
proof −
have $a \neq b$ **using** *assms* **by** *auto*
then have *inj*: $inj\ (\lambda x::'a.\ x\ /_R\ norm\ (a - b))$
by (*simp add*: *inj_on_def*)
have $((sphere\ a\ r \cap T) - \{b\})$ *homeomorphic*
$(+)\ (-a)\ `\ ((sphere\ a\ r \cap T) - \{b\})$
by (*rule homeomorphic_translation*)
also have ... *homeomorphic* $(*_R)\ (inverse\ r)\ `\ (+)\ (-\ a)\ `\ (sphere\ a\ r \cap T - \{b\})$
by (*metis* ‹$0 < r$› *homeomorphic_scaling inverse_inverse_eq inverse_zero less_irrefl*)
also have ... = $sphere\ 0\ 1 \cap ((*_R)\ (inverse\ r)\ `\ (+)\ (-\ a)\ `\ T) - \{(b - a)\ /_R\ r\}$
using *assms* **by** (*auto simp*: *dist_norm norm_minus_commute divide_simps*)
also have ... *homeomorphic* p
using *assms affine_translation* [*symmetric*, *of* $-\ a$] *aff_dim_translation_eq* [*of* $-\ a$]
by (*intro homeomorphic_punctured_affine_sphere_affine_01*) (*auto simp*: *dist_norm norm_minus_commute affine_scaling inj*)
finally show *?thesis* **.**
qed

corollary *homeomorphic_punctured_sphere_affine*:
fixes $a :: 'a :: euclidean\_space$
assumes $0 < r$ **and** *b*: $b \in sphere\ a\ r$
and $affine\ T$ **and** *affS*: $aff\_dim\ T + 1 = DIM('a)$
shows $(sphere\ a\ r - \{b\})$ *homeomorphic* T
using *homeomorphic_punctured_affine_sphere_affine* [*of r b a UNIV T*] *assms*
by *auto*

corollary *homeomorphic_punctured_sphere_hyperplane*:
fixes $a :: 'a :: euclidean\_space$
assumes $0 < r$ **and** *b*: $b \in sphere\ a\ r$
and $c \neq 0$
shows $(sphere\ a\ r - \{b\})$ *homeomorphic* $\{x::'a.\ c \cdot x = d\}$
using *assms*
by (*intro homeomorphic_punctured_sphere_affine*) (*auto simp*: *affine_hyperplane of_nat_diff*)

proposition *homeomorphic_punctured_sphere_affine_gen*:
fixes $a :: 'a :: euclidean\_space$
assumes *convex S bounded S* **and** *a*: $a \in rel\_frontier\ S$
and $affine\ T$ **and** *affS*: $aff\_dim\ S = aff\_dim\ T + 1$
shows $rel\_frontier\ S - \{a\}$ *homeomorphic* T
proof −

obtain $U :: 'a\ set$ **where** *affine U convex U* **and** *affdS*: *aff_dim U = aff_dim S*
using *choose_affine_subset* [*OF affine_UNIV aff_dim_geq*]
by (*meson aff_dim_affine_hull affine_affine_hull affine_imp_convex*)
have $S \neq \{\}$ **using** *assms* **by** *auto*
then obtain *z* **where** $z \in U$
by (*metis aff_dim_negative_iff equals0I affdS*)
then have *bne*: *ball z 1 ∩ U ≠ {}* **by** *force*
then have [*simp*]: *aff_dim*(*ball z 1 ∩ U*) = *aff_dim U*
using *aff_dim_convex_Int_open* [*OF* ‹*convex U*› *open_ball*]
by (*fastforce simp add*: *Int_commute*)
have *rel_frontier S homeomorphic rel_frontier* (*ball z 1 ∩ U*)
by (*rule homeomorphic_rel_frontiers_convex_bounded_sets*)
(*auto simp*: ‹*affine U*› *affine_imp_convex convex_Int affdS assms*)
also have ... = *sphere z 1 ∩ U*
using *convex_affine_rel_frontier_Int* [*of ball z 1 U*]
by (*simp add*: ‹*affine U*› *bne*)
finally have *rel_frontier S homeomorphic sphere z 1 ∩ U* **.**
then obtain *h k* **where** *him*: *h ‘ rel_frontier S = sphere z 1 ∩ U*
and *kim*: *k ‘ (sphere z 1 ∩ U) = rel_frontier S*
and *hcon*: *continuous_on* (*rel_frontier S*) *h*
and *kcon*: *continuous_on* (*sphere z 1 ∩ U*) *k*
and *kh*: $\bigwedge x.\ x \in$ *rel_frontier S* $\Longrightarrow k(h(x)) = x$
and *hk*: $\bigwedge y.\ y \in$ *sphere z 1 ∩ U* $\Longrightarrow h(k(y)) = y$
unfolding *homeomorphic_def homeomorphism_def* **by** *auto*
have *rel_frontier S − {a} homeomorphic* (*sphere z 1 ∩ U*) *− {h a}*
proof (*rule homeomorphicI*)
show *h*: *h ‘ (rel_frontier S − {a}) = sphere z 1 ∩ U − {h a}*
using *him a kh* **by** *auto metis*
show *k ‘ (sphere z 1 ∩ U − {h a}) = rel_frontier S − {a}*
by (*force simp*: *h* [*symmetric*] *image_comp o_def kh*)
qed (*auto intro*: *continuous_on_subset hcon kcon simp*: *kh hk*)
also have ... *homeomorphic T*
by (*rule homeomorphic_punctured_affine_sphere_affine*)
(*use a him* **in** ‹*auto simp*: *affS affdS* ‹*affine T*› ‹*affine U*› ‹*z ∈ U*››)
finally show *?thesis* **.**
qed

When dealing with AR, ANR and ANR later, it's useful to know that every set is homeomorphic to a closed subset of a convex set, and if the set is locally compact we can take the convex set to be the universe.

proposition *homeomorphic_closedin_convex*:
fixes $S :: 'm{::}euclidean\_space\ set$
assumes *aff_dim S* < *DIM*(*'n*)
obtains *U* **and** $T :: 'n{::}euclidean\_space\ set$
where *convex U U ≠ {} closedin* (*top_of_set U*) *T*
S homeomorphic T
proof (*cases S = {}*)
case *True* **then show** *?thesis*
by (*rule_tac U=UNIV* **and** *T={}* **in** *that*) *auto*

next
case *False*
then obtain *a* **where** $a \in S$ **by** *auto*
obtain *i*::*'n* **where** *i*: $i \in Basis\ i \neq 0$
using *SOME_Basis Basis_zero* **by** *force*
have $0 \in affine\ hull\ ((+)\ (-\ a)\ `\ S)$
by (*simp add*: ‹$a \in S$› *hull_inc*)
then have $dim\ ((+)\ (-\ a)\ `\ S) = aff\_dim\ ((+)\ (-\ a)\ `\ S)$
by (*simp add*: *aff_dim_zero*)
also have ... $< DIM('n)$
by (*simp add*: *aff_dim_translation_eq_subtract assms cong*: *image_cong_simp*)
finally have *dd*: $dim\ ((+)\ (-\ a)\ `\ S) < DIM('n)$
by *linarith*
have *span*: $span\ \{x.\ i \cdot x = 0\} = \{x.\ i \cdot x = 0\}$
using *span_eq_iff* [*symmetric*, *of* $\{x.\ i \cdot x = 0\}$] *subspace_hyperplane* [*of i*]
by *simp*
have $dim\ ((+)\ (-\ a)\ `\ S) \leq dim\ \{x.\ i \cdot x = 0\}$
using *dd* **by** (*simp add*: *dim_hyperplane* [*OF* ‹$i \neq 0$›])
then obtain *T* **where** *subspace T* **and** *Tsub*: $T \subseteq \{x.\ i \cdot x = 0\}$
and *dimT*: $dim\ T = dim\ ((+)\ (-\ a)\ `\ S)$
by (*rule choose_subspace_of_subspace*) (*simp add*: *span*)
have *subspace* (*span* ((+) (− *a*) ‘ *S*))
using *subspace_span* **by** *blast*
then obtain *h k* **where** *linear h linear k*
and *heq*: $h\ `\ span\ ((+)\ (-\ a)\ `\ S) = T$
and *keq*:$k\ `\ T = span\ ((+)\ (-\ a)\ `\ S)$
and *hinv* [*simp*]: $\bigwedge x.\ x \in span\ ((+)\ (-\ a)\ `\ S) \Longrightarrow k(h\ x) = x$
and *kinv* [*simp*]: $\bigwedge x.\ x \in T \Longrightarrow h(k\ x) = x$
by (*auto simp*: *dimT intro*: *isometries_subspaces* [*OF _* ‹*subspace T*›] *dimT*)
have *hcont*: *continuous_on A h* **and** *kcont*: *continuous_on B k* **for** *A B*
using ‹*linear h*› ‹*linear k*› *linear_continuous_on linear_conv_bounded_linear*
by *blast*+
have *ihhhh*[*simp*]: $\bigwedge x.\ x \in S \Longrightarrow i \cdot h\ (x - a) = 0$
using *Tsub* [*THEN subsetD*] *heq span_superset* **by** *fastforce*
have *sphere 0 1* − {*i*} *homeomorphic* $\{x.\ i \cdot x = 0\}$
proof (*rule homeomorphic_punctured_sphere_affine*)
show *affine* $\{x.\ i \cdot x = 0\}$
by (*auto simp*: *affine_hyperplane*)
show $aff\_dim\ \{x.\ i \cdot x = 0\} + 1 = int\ DIM('n)$
using *i* **by** *clarsimp* (*metis DIM_positive Suc_pred add.commute of_nat_Suc*)
qed (*use i* **in** *auto*)
then obtain *f g* **where** *fg*: *homeomorphism* (*sphere 0 1* − {*i*}) $\{x.\ i \cdot x = 0\}$ *f g*
by (*force simp*: *homeomorphic_def*)
show *?thesis*
proof
have $h\ `\ (+)\ (-\ a)\ `\ S \subseteq T$
using *heq span_superset span_linear_image* **by** *blast*
then have $g\ `\ h\ `\ (+)\ (-\ a)\ `\ S \subseteq g\ `\ \{x.\ i \cdot x = 0\}$

using *Tsub* **by** (*simp add*: *image_mono*)
also have ... ⊆ *sphere 0 1* − {*i*}
by (*simp add*: *fg* [*unfolded homeomorphism_def*])
finally have *gh_sub_sph*: (*g* ∘ *h*) ‘ (+) (− *a*) ‘ *S* ⊆ *sphere 0 1* − {*i*}
by (*metis image_comp*)
then have *gh_sub_cb*: (*g* ∘ *h*) ‘ (+) (− *a*) ‘ *S* ⊆ *cball 0 1*
by (*metis Diff_subset order_trans sphere_cball*)
have [*simp*]: ⋀*u*. *u* ∈ *S* ⟹ *norm* (*g* (*h* (*u* − *a*))) = *1*
using *gh_sub_sph* [*THEN subsetD*] **by** (*auto simp*: *o_def*)
show *convex* (*ball 0 1* ∪ (*g* ∘ *h*) ‘ (+) (− *a*) ‘ *S*)
by (*meson ball_subset_cball convex_intermediate_ball gh_sub_cb sup.bounded_iff sup.cobounded1*)
show *closedin* (*top_of_set* (*ball 0 1* ∪ (*g* ∘ *h*) ‘ (+) (− *a*) ‘ *S*)) ((*g* ∘ *h*) ‘ (+) (− *a*) ‘ *S*)
unfolding *closedin_closed*
by (*rule_tac x=sphere 0 1* **in** *exI*) *auto*
have *ghcont*: *continuous_on* ((λ*x*. *x* − *a*) ‘ *S*) (λ*x*. *g* (*h x*))
by (*rule continuous_on_compose2* [*OF homeomorphism_cont2* [*OF fg*] *hcont*], *force*)
have *kfcont*: *continuous_on* ((λ*x*. *g* (*h* (*x* − *a*))) ‘ *S*) (λ*x*. *k* (*f x*))
proof (*rule continuous_on_compose2* [*OF kcont*])
show *continuous_on* ((λ*x*. *g* (*h* (*x* − *a*))) ‘ *S*) *f*
using *homeomorphism_cont1* [*OF fg*] *gh_sub_sph* **by** (*fastforce intro*: *continuous_on_subset*)
qed *auto*
have *S homeomorphic* (+) (− *a*) ‘ *S*
by (*fact homeomorphic_translation*)
also have ... *homeomorphic* (*g* ∘ *h*) ‘ (+) (− *a*) ‘ *S*
apply (*simp add*: *homeomorphic_def homeomorphism_def cong*: *image_cong_simp*)
apply (*rule_tac x=g* ∘ *h* **in** *exI*)
apply (*rule_tac x=k* ∘ *f* **in** *exI*)
apply (*auto simp*: *ghcont kfcont span_base homeomorphism_apply2* [*OF fg*] *image_comp cong*: *image_cong_simp*)
done
finally show *S homeomorphic* (*g* ∘ *h*) ‘ (+) (− *a*) ‘ *S* **.**
qed *auto*
qed

6.16.3 Locally compact sets in an open set

Locally compact sets are closed in an open set and are homeomorphic to an absolutely closed set if we have one more dimension to play with.

lemma *locally_compact_open_Int_closure*:
fixes *S* :: *'a* :: *metric_space set*
assumes *locally compact S*
obtains *T* **where** *open T S* = *T* ∩ *closure S*
proof −
have ∀ *x*∈*S*. ∃ *T v u*. *u* = *S* ∩ *T* ∧ *x* ∈ *u* ∧ *u* ⊆ *v* ∧ *v* ⊆ *S* ∧ *open T* ∧ *compact v*

```
    by (metis assms locally_compact openin_open)
  then obtain t v where
       tv: ⋀x. x ∈ S
           ⟹ v x ⊆ S ∧ open (t x) ∧ compact (v x) ∧ (∃u. x ∈ u ∧ u ⊆ v x ∧ u
= S ∩ t x)
    by metis
  then have o: open (⋃(t ` S))
    by blast
  have S = ⋃ (v ` S)
    using tv by blast
  also have ... = ⋃(t ` S) ∩ closure S
  proof
    show ⋃(v ` S) ⊆ ⋃(t ` S) ∩ closure S
      by clarify (meson IntD2 IntI UN_I closure_subset subsetD tv)
    have t x ∩ closure S ⊆ v x if x ∈ S for x
    proof −
      have t x ∩ closure S ⊆ closure (t x ∩ S)
        by (simp add: open_Int_closure_subset that tv)
      also have ... ⊆ v x
        by (metis Int_commute closure_minimal compact_imp_closed that tv)
      finally show ?thesis .
    qed
    then show ⋃(t ` S) ∩ closure S ⊆ ⋃(v ` S)
      by blast
  qed
  finally have e: S = ⋃(t ` S) ∩ closure S .
  show ?thesis
    by (rule that [OF o e])
qed


lemma locally_compact_closedin_open:
    fixes S :: 'a :: metric_space set
    assumes locally compact S
    obtains T where open T closedin (top_of_set T) S
  by (metis locally_compact_open_Int_closure [OF assms] closed_closure closedin_closed_Int)


lemma locally_compact_homeomorphism_projection_closed:
  assumes locally compact S
  obtains T and f :: 'a ⇒ 'a :: euclidean_space × 'b :: euclidean_space
  where closed T homeomorphism S T f fst
proof (cases closed S)
  case True
  show ?thesis
  proof
    show homeomorphism S (S × {0}) (λx. (x, 0)) fst
      by (auto simp: homeomorphism_def continuous_intros)
  qed (use True closed_Times in auto)
```

next
case *False*
obtain *U* **where** *open U* **and** *US*: $U \cap$ *closure S* $= S$
by (*metis locally_compact_open_Int_closure* [*OF assms*])
with *False* **have** *Ucomp*: $-U \neq \{\}$
using *closure_eq* **by** *auto*
have [*simp*]: *closure* $(-U) = -U$
by (*simp add*: ‹*open U*› *closed_Compl*)
define *f* :: $'a \Rightarrow 'a \times 'b$ **where** $f \equiv \lambda x.\ (x,\ One\ /_R\ setdist\ \{x\}\ (-U))$
have *continuous_on U* $(\lambda x.\ (x,\ One\ /_R\ setdist\ \{x\}\ (-U)))$
proof (*intro continuous_intros continuous_on_setdist*)
show $\forall x{\in}U.\ setdist\ \{x\}\ (-U) \neq 0$
by (*simp add*: *Ucomp setdist_eq_0_sing_1*)
qed
then have *homU*: *homeomorphism U* (*f*‘*U*) *f fst*
by (*auto simp*: *f_def homeomorphism_def image_iff continuous_intros*)
have *cloS*: *closedin* (*top_of_set U*) *S*
by (*metis US closed_closure closedin_closed_Int*)
have *cont*: *isCont* $((\lambda x.\ setdist\ \{x\}\ (-U))\ o\ fst)\ z$ **for** $z :: 'a \times 'b$
by (*rule continuous_at_compose continuous_intros continuous_at_setdist*)+
have *setdist1D*: *setdist* $\{a\}\ (-U) *_R b = One \Longrightarrow setdist\ \{a\}\ (-U) \neq 0$ **for** *a*::$'a$ **and** *b*::$'b$
by *force*
have $*$: $r *_R b = One \Longrightarrow b = (1\ /\ r) *_R One$ **for** *r* **and** *b*::$'b$
by (*metis One_non_0 nonzero_divide_eq_eq real_vector.scale_eq_0_iff real_vector.scale_scale scaleR_one*)
have $\bigwedge a\ b{::}'b.\ setdist\ \{a\}\ (-U) *_R b = One \Longrightarrow (a,b) \in (\lambda x.\ (x,\ (1\ /\ setdist\ \{x\}\ (-U)) *_R One))\ `\ U$
by (*metis* (*mono_tags*, *lifting*) $*$ *ComplI image_eqI setdist1D setdist_sing_in_set*)
then have $f\ `\ U = (\lambda z.\ (setdist\ \{fst\ z\}\ (-U) *_R snd\ z)) -` \{One\}$
by (*auto simp*: *f_def setdist_eq_0_sing_1 field_simps Ucomp*)
then have *clfU*: *closed* $(f\ `\ U)$
by (*force intro*: *continuous_intros cont* [*unfolded o_def*] *continuous_closed_vimage*)
have *closed* $(f\ `\ S)$
by (*metis closedin_closed_trans* [*OF _ clfU*] *homeomorphism_imp_closed_map* [*OF homU cloS*])
then show *?thesis*
by (*metis US homU homeomorphism_of_subsets inf_sup_ord*(1) *that*)
qed

lemma *locally_compact_closed_Int_open*:
fixes *S* :: $'a$ :: *euclidean_space set*
shows *locally compact S* $\longleftrightarrow$ $(\exists U\ V.\ closed\ U \wedge open\ V \wedge S = U \cap V)$ (**is** *?lhs* = *?rhs*)
proof
show *?lhs* $\Longrightarrow$ *?rhs*
by (*metis closed_closure inf_commute locally_compact_open_Int_closure*)
show *?rhs* $\Longrightarrow$ *?lhs*
by (*meson closed_imp_locally_compact locally_compact_Int open_imp_locally_compact*)

qed

lemma *lowerdim_embeddings*:
assumes $DIM('a) < DIM('b)$
obtains $f :: 'a::euclidean\_space*real \Rightarrow 'b::euclidean\_space$
and $g :: 'b \Rightarrow 'a*real$
and $j :: 'b$
where *linear f linear g* $\bigwedge z.\ g\ (f\ z) = z\ j \in Basis\ \bigwedge x.\ f(x,0) \cdot j = 0$
proof −
let *?B* = *Basis* :: $('a*real)\ set$
have *b01*: $(0,1) \in ?B$
by (*simp add*: *Basis_prod_def*)
have $DIM('a * real) \leq DIM('b)$
by (*simp add*: *Suc_leI assms*)
then obtain *basf* :: $'a*real \Rightarrow 'b$ **where** *sbf*: *basf ‘ ?B* ⊆ *Basis* **and** *injbf*: *inj_on basf Basis*
by (*metis finite_Basis card_le_inj*)
define *basg*:: $'b \Rightarrow 'a * real$ **where**
basg ≡ λ*i*. *if i* ∈ *basf ‘ Basis then inv_into Basis basf i else* (0,1)
have *bgf*[*simp*]: *basg* (*basf i*) = *i* **if** *i* ∈ *Basis* **for** *i*
using *inv_into_f_f injbf that* **by** (*force simp*: *basg_def*)
have *sbg*: *basg ‘ Basis* ⊆ *?B*
by (*force simp*: *basg_def injbf b01*)
define *f* :: $'a*real \Rightarrow 'b$ **where** $f \equiv \lambda u.\ \sum j{\in}Basis.\ (u \cdot basg\ j) *_R j$
define *g* :: $'b \Rightarrow 'a*real$ **where** $g \equiv \lambda z.\ (\sum i{\in}Basis.\ (z \cdot basf\ i) *_R i)$
show *?thesis*
proof
show *linear f*
unfolding *f_def*
by (*intro linear_compose_sum linearI ballI*) (*auto simp*: *algebra_simps*)
show *linear g*
unfolding *g_def*
by (*intro linear_compose_sum linearI ballI*) (*auto simp*: *algebra_simps*)
have $*$: $(\sum a \in Basis.\ a \cdot basf\ b * (x \cdot basg\ a)) = x \cdot b$ **if** $b \in Basis$ **for** $x\ b$
using *sbf that* **by** *auto*
show *gf*: *g* (*f x*) = *x* **for** *x*
proof (*rule euclidean_eqI*)
show $\bigwedge b.\ b \in Basis \Longrightarrow g\ (f\ x) \cdot b = x \cdot b$
using *f_def g_def sbf* **by** *auto*
qed
show *basf*(0,1) ∈ *Basis*
using *b01 sbf* **by** *auto*
then show $f(x,0) \cdot basf(0,1) = 0$ **for** *x*
unfolding *f_def inner_sum_left*
using *b01 inner_not_same_Basis*
by (*fastforce intro*: *comm_monoid_add_class.sum.neutral*)
qed
qed

proposition *locally_compact_homeomorphic_closed*:
fixes S :: *'a::euclidean_space set*
assumes *locally compact S* **and** *dimlt*: $DIM('a) < DIM('b)$
obtains T :: *'b::euclidean_space set* **where** *closed T S homeomorphic T*
proof −
obtain U:: *('a∗real)set* **and** h
where *closed U* **and** *homU*: *homeomorphism S U h fst*
using *locally_compact_homeomorphism_projection_closed assms* **by** *metis*
obtain f :: *'a∗real ⇒ 'b* **and** g :: *'b ⇒ 'a∗real*
where *linear f linear g* **and** *gf* [*simp*]: $\bigwedge z.\ g\ (f\ z) = z$
using *lowerdim_embeddings* [*OF dimlt*] **by** *metis*
then have *inj f*
by (*metis injI*)
have *gfU*: $g\ `\ f\ `\ U = U$
by (*simp add*: *image_comp o_def*)
have *S homeomorphic U*
using *homU homeomorphic_def* **by** *blast*
also have ... *homeomorphic f ` U*
proof (*rule homeomorphicI* [*OF refl gfU*])
show *continuous_on U f*
by (*meson ‹inj f› ‹linear f› homeomorphism_cont2 linear_homeomorphism_image*)
show *continuous_on* $(f\ `\ U)\ g$
using *‹linear g› linear_continuous_on linear_conv_bounded_linear* **by** *blast*
qed (*auto simp*: *o_def*)
finally show *?thesis*
using *‹closed U› ‹inj f› ‹linear f› closed_injective_linear_image that* **by** *blast*
qed

lemma *homeomorphic_convex_compact_lemma*:
fixes S :: *'a::euclidean_space set*
assumes *convex S*
and *compact S*
and *cball 0 1* $\subseteq S$
shows *S homeomorphic* (*cball* $(0::'a)$ *1*)
proof (*rule starlike_compact_projective_special*[*OF assms*(*2*−*3*)])
fix $x\ u$
assume $x \in S$ **and** $0 \le u$ **and** $u < (1::real)$
have *open* (*ball* $(u *_R x)\ (1 - u)$)
by (*rule open_ball*)
moreover have $u *_R x \in$ *ball* $(u *_R x)\ (1 - u)$
unfolding *centre_in_ball* **using** *‹u < 1›* **by** *simp*
moreover have *ball* $(u *_R x)\ (1 - u) \subseteq S$
proof
fix y
assume $y \in$ *ball* $(u *_R x)\ (1 - u)$
then have *dist* $(u *_R x)\ y < 1 - u$
unfolding *mem_ball* .
with *‹u < 1›* **have** *inverse* $(1 - u) *_R (y - u *_R x) \in$ *cball 0 1*

```
    by (simp add: dist_norm inverse_eq_divide norm_minus_commute)
   with assms(3) have inverse (1 - u) *R (y - u *R x) ∈ S ..
   with assms(1) have (1 - u) *R ((y - u *R x) /R (1 - u)) + u *R x ∈ S
    using ‹x ∈ S› ‹0 ≤ u› ‹u < 1› [THEN less_imp_le] by (rule convexD_alt)
   then show y ∈ S using ‹u < 1›
    by simp
  qed
  ultimately have u *R x ∈ interior S ..
  then show u *R x ∈ S - frontier S
   using frontier_def and interior_subset by auto
qed

proposition homeomorphic_convex_compact_cball:
  fixes e :: real
   and S :: 'a::euclidean_space set
  assumes S: convex S compact S interior S ≠ {} and e > 0
  shows S homeomorphic (cball (b::'a) e)
proof (rule homeomorphic_trans[OF _ homeomorphic_balls(2)])
  obtain a where a ∈ interior S
   using assms by auto
  then show S homeomorphic cball (0::'a) 1
   by (metis (no_types) aff_dim_cball S compact_cball convex_cball
       homeomorphic_convex_lemma interior_rel_interior_gen zero_less_one)
qed (use ‹e>0› in auto)

corollary homeomorphic_convex_compact:
  fixes S :: 'a::euclidean_space set
   and T :: 'a set
  assumes convex S compact S interior S ≠ {}
   and convex T compact T interior T ≠ {}
  shows S homeomorphic T
  using assms
 by (meson zero_less_one homeomorphic_trans homeomorphic_convex_compact_cball
homeomorphic_sym)

lemma homeomorphic_closed_intervals:
  fixes a :: 'a::euclidean_space and b and c :: 'a::euclidean_space and d
  assumes box a b ≠ {} and box c d ≠ {}
   shows (cbox a b) homeomorphic (cbox c d)
  by (simp add: assms homeomorphic_convex_compact)

lemma homeomorphic_closed_intervals_real:
  fixes a::real and b and c::real and d
  assumes a<b and c<d
  shows {a..b} homeomorphic {c..d}
  using assms by (auto intro: homeomorphic_convex_compact)
```

6.16.4 Covering spaces and lifting results for them

definition *covering_space*
:: *'a::topological_space set* ⇒ (*'a* ⇒ *'b*) ⇒ *'b::topological_space set* ⇒ *bool*
where
covering_space c p S ≡
continuous_on c p ∧ *p ' c* = *S* ∧
(∀ *x* ∈ *S*. ∃ *T*. *x* ∈ *T* ∧ *openin* (*top_of_set S*) *T* ∧
(∃ *v*. ⋃ *v* = *c* ∩ *p* −' *T* ∧
(∀ *u* ∈ *v*. *openin* (*top_of_set c*) *u*) ∧
pairwise disjnt v ∧
(∀ *u* ∈ *v*. ∃ *q*. *homeomorphism u T p q*)))

lemma *covering_spaceI* [*intro?*]:
assumes *continuous_on c p p ' c* = *S*
⋀*x*. *x* ∈ *S* ⟹ ∃ *T*. *x* ∈ *T* ∧ *openin* (*top_of_set S*) *T* ∧
(∃ *v*. ⋃ *v* = *c* ∩ *p* −' *T* ∧ (∀ *u* ∈ *v*. *openin* (*top_of_set c*) *u*) ∧
pairwise disjnt v ∧ (∀ *u* ∈ *v*. ∃ *q*. *homeomorphism u T p q*))
shows *covering_space c p S*
using *assms* **unfolding** *covering_space_def* **by** *auto*

lemma *covering_space_imp_continuous*: *covering_space c p S* ⟹ *continuous_on c p*
by (*simp add*: *covering_space_def*)

lemma *covering_space_imp_surjective*: *covering_space c p S* ⟹ *p ' c* = *S*
by (*simp add*: *covering_space_def*)

lemma *homeomorphism_imp_covering_space*: *homeomorphism S T f g* ⟹ *covering_space S f T*
apply (*clarsimp simp add*: *homeomorphism_def covering_space_def*)
apply (*rule_tac x*=*T* **in** *exI*, *simp*)
apply (*rule_tac x*=*{S}* **in** *exI*, *auto*)
done

lemma *covering_space_local_homeomorphism*:
assumes *covering_space c p S x* ∈ *c*
obtains *T u q* **where** *x* ∈ *T openin* (*top_of_set c*) *T*
p x ∈ *u openin* (*top_of_set S*) *u*
homeomorphism T u p q
using *assms*
by (*clarsimp simp add*: *covering_space_def*) (*metis IntI UnionE vimage_eq*)

lemma *covering_space_local_homeomorphism_alt*:
assumes *p*: *covering_space c p S* **and** *y* ∈ *S*
obtains *x T U q* **where** *p x* = *y*
x ∈ *T openin* (*top_of_set c*) *T*
y ∈ *U openin* (*top_of_set S*) *U*

homeomorphism T U p q
proof −
obtain *x* **where** *p x = y x ∈ c*
using *assms covering_space_imp_surjective* **by** *blast*
show *?thesis*
using *that* ‹*p x = y*› **by** (*auto intro*: *covering_space_local_homeomorphism* [*OF p* ‹*x ∈ c*›])
qed

proposition *covering_space_open_map*:
fixes *S* :: *'a* :: *metric_space set* **and** *T* :: *'b* :: *metric_space set*
assumes *p*: *covering_space c p S* **and** *T*: *openin* (*top_of_set c*) *T*
shows *openin* (*top_of_set S*) (*p ‘ T*)
proof −
have *pce*: *p ‘ c = S*
and *covs*:
⋀*x*. *x ∈ S ⟹*
∃ *X VS*. *x ∈ X ∧ openin* (*top_of_set S*) *X ∧*
⋃ *VS = c ∩ p −‘ X ∧*
(∀ *u ∈ VS*. *openin* (*top_of_set c*) *u*) *∧*
pairwise disjnt VS ∧
(∀ *u ∈ VS*. ∃ *q*. *homeomorphism u X p q*)
using *p* **by** (*auto simp*: *covering_space_def*)
have *T ⊆ c* **by** (*metis openin_euclidean_subtopology_iff T*)
have ∃ *X*. *openin* (*top_of_set S*) *X ∧ y ∈ X ∧ X ⊆ p ‘ T*
if *y ∈ p ‘ T* **for** *y*
proof −
have *y ∈ S* **using** ‹*T ⊆ c*› *pce that* **by** *blast*
obtain *U VS* **where** *y ∈ U* **and** *U*: *openin* (*top_of_set S*) *U*
and *VS*: ⋃ *VS = c ∩ p −‘ U*
and *openVS*: ∀ *V ∈ VS*. *openin* (*top_of_set c*) *V*
and *homVS*: ⋀*V*. *V ∈ VS ⟹* ∃ *q*. *homeomorphism V U p q*
using *covs* [*OF* ‹*y ∈ S*›] **by** *auto*
obtain *x* **where** *x ∈ c p x ∈ U x ∈ T p x = y*
using *T* [*unfolded openin_euclidean_subtopology_iff*] ‹*y ∈ U*› ‹*y ∈ p ‘ T*›
by *blast*
with *VS* **obtain** *V* **where** *x ∈ V V ∈ VS* **by** *auto*
then obtain *q* **where** *q*: *homeomorphism V U p q* **using** *homVS* **by** *blast*
then have *ptV*: *p ‘* (*T ∩ V*) = *U ∩ q −‘* (*T ∩ V*)
using *VS* ‹*V ∈ VS*› **by** (*auto simp*: *homeomorphism_def*)
have *ocv*: *openin* (*top_of_set c*) *V*
by (*simp add*: ‹*V ∈ VS*› *openVS*)
have *openin* (*top_of_set* (*q ‘ U*)) (*T ∩ V*)
using *q* **unfolding** *homeomorphism_def*
by (*metis T inf.absorb_iff2 ocv openin_imp_subset openin_subtopology_Int subtopology_subtopology*)
then have *openin* (*top_of_set U*) (*U ∩ q −‘* (*T ∩ V*))
using *continuous_on_open homeomorphism_def q* **by** *blast*
then have *os*: *openin* (*top_of_set S*) (*U ∩ q −‘* (*T ∩ V*))

```
    using openin_trans [of U] by (simp add: Collect_conj_eq U)
   show ?thesis
   proof (intro exI conjI)
    show openin (top_of_set S) (p ‘ (T ∩ V))
     by (simp only: ptV os)
   qed (use ‹p x = y› ‹x ∈ V› ‹x ∈ T› in auto)
  qed
  with openin_subopen show ?thesis by blast
qed

lemma covering_space_lift_unique_gen:
  fixes f :: 'a::topological_space ⇒ 'b::topological_space
  fixes g1 :: 'a ⇒ 'c::real_normed_vector
  assumes cov: covering_space c p S
     and eq: g1 a = g2 a
     and f: continuous_on T f  f ∈ T → S
     and g1: continuous_on T g1  g1 ∈ T → c
     and fg1: ⋀x. x ∈ T ⟹ f x = p(g1 x)
     and g2: continuous_on T g2  g2 ∈ T → c
     and fg2: ⋀x. x ∈ T ⟹ f x = p(g2 x)
     and u_compt: U ∈ components T and a ∈ U x ∈ U
   shows g1 x = g2 x
proof −
  have U ⊆ T by (rule in_components_subset [OF u_compt])
  define G12 where G12 ≡ {x ∈ U. g1 x − g2 x = 0}
  have connected U by (rule in_components_connected [OF u_compt])
  have contu: continuous_on U g1 continuous_on U g2
      using ‹U ⊆ T› continuous_on_subset g1 g2 by blast+
  have o12: openin (top_of_set U) G12
  unfolding G12_def
  proof (subst openin_subopen, clarify)
   fix z
   assume z: z ∈ U g1 z − g2 z = 0
   obtain v w q where g1 z ∈ v and ocv: openin (top_of_set c) v
    and p (g1 z) ∈ w and osw: openin (top_of_set S) w
    and hom: homeomorphism v w p q
   proof (rule covering_space_local_homeomorphism [OF cov])
    show g1 z ∈ c
     using ‹U ⊆ T› ‹z ∈ U› g1(2) by blast
   qed auto
   have g2 z ∈ v using ‹g1 z ∈ v› z by auto
   have gg: U ∩ g −‘ v = U ∩ g −‘ (v ∩ g ‘ U) for g
    by auto
   have openin (top_of_set (g1 ‘ U)) (v ∩ g1 ‘ U)
    using ocv ‹U ⊆ T› g1 by (fastforce simp add: openin_open)
   then have 1: openin (top_of_set U) (U ∩ g1 −‘ v)
     unfolding gg by (blast intro: contu continuous_on_open [THEN iffD1,
rule_format])
   have openin (top_of_set (g2 ‘ U)) (v ∩ g2 ‘ U)
```

```
    using ocv ‹U ⊆ T› g2 by (fastforce simp add: openin_open)
  then have 2: openin (top_of_set U) (U ∩ g2 -` v)
    unfolding gg by (blast intro: contu continuous_on_open [THEN iffD1,
rule_format])
  let ?T = (U ∩ g1 -` v) ∩ (U ∩ g2 -` v)
  show ∃T. openin (top_of_set U) T ∧ z ∈ T ∧ T ⊆ {z ∈ U. g1 z - g2 z =
0}
  proof (intro exI conjI)
    show openin (top_of_set U) ?T
      using 1 2 by blast
    show z ∈ ?T
      using z by (simp add: ‹g1 z ∈ v› ‹g2 z ∈ v›)
    show ?T ⊆ {z ∈ U. g1 z - g2 z = 0}
      using hom
      by (clarsimp simp: homeomorphism_def) (metis ‹U ⊆ T› fg1 fg2 subsetD)
  qed
 qed
 have c12: closedin (top_of_set U) G12
  unfolding G12_def
  by (intro continuous_intros continuous_closedin_preimage_constant contu)
 have G12 = {} ∨ G12 = U
   by (intro connected_clopen [THEN iffD1, rule_format] ‹connected U› conjI
o12 c12)
 with eq ‹a ∈ U› have ⋀x. x ∈ U ⟹ g1 x - g2 x = 0 by (auto simp: G12_def)
 then show ?thesis
  using ‹x ∈ U› by force
qed

proposition covering_space_lift_unique:
 fixes f :: 'a::topological_space ⇒ 'b::topological_space
 fixes g1 :: 'a ⇒ 'c::real_normed_vector
 assumes covering_space c p S
         g1 a = g2 a
         continuous_on T f  f ∈ T → S
         continuous_on T g1  g1 ∈ T → c  ⋀x. x ∈ T ⟹ f x = p(g1 x)
         continuous_on T g2  g2 ∈ T → c  ⋀x. x ∈ T ⟹ f x = p(g2 x)
         connected T  a ∈ T   x ∈ T
  shows g1 x = g2 x
  using covering_space_lift_unique_gen [of c p S] in_components_self assms
ex_in_conv
 by blast

lemma covering_space_locally:
 fixes p :: 'a::real_normed_vector ⇒ 'b::real_normed_vector
 assumes loc: locally φ C and cov: covering_space C p S
    and pim: ⋀T. ⟦T ⊆ C; φ T⟧ ⟹ ψ(p ` T)
  shows locally ψ S
proof -
 have locally ψ (p ` C)
```

proof (*rule locally_open_map_image* [*OF loc*])
show *continuous_on C p*
using *cov covering_space_imp_continuous* **by** *blast*
show $\bigwedge T$. *openin* (*top_of_set C*) *T* $\Longrightarrow$ *openin* (*top_of_set* (*p ' C*)) (*p ' T*)
using *cov covering_space_imp_surjective covering_space_open_map* **by** *blast*
qed (*simp add*: *pim*)
then show *?thesis*
using *covering_space_imp_surjective* [*OF cov*] **by** *metis*
qed

proposition *covering_space_locally_eq*:
fixes *p* :: *'a*::*real_normed_vector* $\Rightarrow$ *'b*::*real_normed_vector*
assumes *cov*: *covering_space C p S*
and *pim*: $\bigwedge T$. $[\![T \subseteq C; \varphi\ T]\!] \Longrightarrow \psi(p\ `\ T)$
and *qim*: $\bigwedge q\ U$. $[\![U \subseteq S$; *continuous_on U q*; $\psi\ U]\!] \Longrightarrow \varphi(q\ `\ U)$
shows *locally* ψ *S* $\longleftrightarrow$ *locally* φ *C*
(**is** *?lhs* = *?rhs*)
proof
assume *L*: *?lhs*
show *?rhs*
proof (*rule locallyI*)
fix *V x*
assume *V*: *openin* (*top_of_set C*) *V* **and** $x \in V$
have *p x* $\in$ *p ' C*
by (*metis IntE V* ‹$x \in V$› *imageI openin_open*)
then obtain *T* $\mathcal{V}$ **where** *p x* $\in$ *T*
and *opeT*: *openin* (*top_of_set S*) *T*
and *veq*: $\bigcup\mathcal{V} = C \cap p -` T$
and *ope*: $\forall\, U{\in}\mathcal{V}$. *openin* (*top_of_set C*) *U*
and *hom*: $\forall\, U{\in}\mathcal{V}$. $\exists\, q$. *homeomorphism U T p q*
using *cov* **unfolding** *covering_space_def* **by** (*blast intro*: *that*)
have $x \in \bigcup\mathcal{V}$
using *V veq* ‹*p x* $\in$ *T*› ‹$x \in V$› *openin_imp_subset* **by** *fastforce*
then obtain *U* **where** $x \in U$ $U \in \mathcal{V}$
by *blast*
then obtain *q* **where** *opeU*: *openin* (*top_of_set C*) *U* **and** *q*: *homeomorphism U T p q*
using *ope hom* **by** *blast*
with *V* **have** *openin* (*top_of_set C*) (*U* $\cap$ *V*)
by *blast*
then have *UV*: *openin* (*top_of_set S*) (*p ' (U* $\cap$ *V)*)
using *cov covering_space_open_map* **by** *blast*
obtain *W W'* **where** *opeW*: *openin* (*top_of_set S*) *W* **and** ψ *W' p x* $\in$ *W*
W $\subseteq$ *W'* **and** *W'sub*: *W'* $\subseteq$ *p ' (U* $\cap$ *V)*
using *locallyE* [*OF L UV*] ‹$x \in U$› ‹$x \in V$› **by** *blast*
then have *W* $\subseteq$ *T*
by (*metis Int_lower1 q homeomorphism_image1 image_Int_subset order_trans*)
show $\exists\, U\ Z$. *openin* (*top_of_set C*) *U* $\wedge$

```
                φ Z ∧ x ∈ U ∧ U ⊆ Z ∧ Z ⊆ V
    proof (intro exI conjI)
      have openin (top_of_set T) W
       by (meson opeW opeT openin_imp_subset openin_subset_trans ‹W ⊆ T›)
      then have openin (top_of_set U) (q ‘ W)
       by (meson homeomorphism_imp_open_map homeomorphism_symD q)
      then show openin (top_of_set C) (q ‘ W)
       using opeU openin_trans by blast
      show φ (q ‘ W′)
       by (metis (mono_tags, lifting) Int_subset_iff UV W′sub ‹ψ W′› continuous_on_subset dual_order.trans homeomorphism_def image_Int_subset openin_imp_subset q qim)
      show x ∈ q ‘ W
       by (metis ‹p x ∈ W› ‹x ∈ U› homeomorphism_def imageI q)
      show q ‘ W ⊆ q ‘ W′
       using ‹W ⊆ W′› by blast
      have W′ ⊆ p ‘ V
       using W′sub by blast
      then show q ‘ W′ ⊆ V
       using W′sub homeomorphism_apply1 [OF q] by auto
      qed
  qed
next
  assume ?rhs
  then show ?lhs
    using cov covering_space_locally pim by blast
qed

lemma covering_space_locally_compact_eq:
  fixes p :: 'a::real_normed_vector ⇒ 'b::real_normed_vector
  assumes covering_space C p S
  shows locally compact S ⟷ locally compact C
proof (rule covering_space_locally_eq [OF assms])
  show ⋀T. ⟦T ⊆ C; compact T⟧ ⟹ compact (p ‘ T)
     by (meson assms compact_continuous_image continuous_on_subset covering_space_imp_continuous)
qed (use compact_continuous_image in blast)

lemma covering_space_locally_connected_eq:
  fixes p :: 'a::real_normed_vector ⇒ 'b::real_normed_vector
  assumes covering_space C p S
    shows locally connected S ⟷ locally connected C
proof (rule covering_space_locally_eq [OF assms])
  show ⋀T. ⟦T ⊆ C; connected T⟧ ⟹ connected (p ‘ T)
     by (meson connected_continuous_image assms continuous_on_subset covering_space_imp_continuous)
qed (use connected_continuous_image in blast)

lemma covering_space_locally_path_connected_eq:
```

fixes $p :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$
assumes *covering_space C p S*
shows *locally path_connected S* $\longleftrightarrow$ *locally path_connected C*
proof (*rule covering_space_locally_eq* [*OF assms*])
show $\bigwedge T.\ [\![T \subseteq C;\ path\_connected\ T]\!] \Longrightarrow path\_connected\ (p\ `\ T)$
by (*meson path_connected_continuous_image assms continuous_on_subset covering_space_imp_continuous*)
qed (*use path_connected_continuous_image* **in** *blast*)

lemma *covering_space_locally_compact*:
fixes $p :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$
assumes *locally compact C covering_space C p S*
shows *locally compact S*
using *assms covering_space_locally_compact_eq* **by** *blast*

lemma *covering_space_locally_connected*:
fixes $p :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$
assumes *locally connected C covering_space C p S*
shows *locally connected S*
using *assms covering_space_locally_connected_eq* **by** *blast*

lemma *covering_space_locally_path_connected*:
fixes $p :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$
assumes *locally path_connected C covering_space C p S*
shows *locally path_connected S*
using *assms covering_space_locally_path_connected_eq* **by** *blast*

proposition *covering_space_lift_homotopy*:
fixes $p :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$
and $h :: real \times 'c::real\_normed\_vector \Rightarrow 'b$
assumes *cov*: *covering_space C p S*
and *conth*: *continuous_on* $(\{0..1\} \times U)\ h$
and *him*: $h \in (\{0..1\} \times U) \to S$
and *heq*: $\bigwedge y.\ y \in U \Longrightarrow h\ (0,y) = p(f\ y)$
and *contf*: *continuous_on U f* **and** *fim*: $f \in U \to C$
obtains k **where** *continuous_on* $(\{0..1\} \times U)\ k$
$k \in (\{0..1\} \times U) \to C$
$\bigwedge y.\ y \in U \Longrightarrow k(0,\ y) = f\ y$
$\bigwedge z.\ z \in \{0..1\} \times U \Longrightarrow h\ z = p(k\ z)$
proof −
have $\exists V\ k.\ openin\ (top\_of\_set\ U)\ V \wedge y \in V \wedge$
$continuous\_on\ (\{0..1\} \times V)\ k \wedge k\ `\ (\{0..1\} \times V) \subseteq C \wedge$
$(\forall z \in V.\ k(0,\ z) = f\ z) \wedge (\forall z \in \{0..1\} \times V.\ h\ z = p(k\ z))$
if $y \in U$ **for** y
proof −
obtain *UU* **where** *UU*: $\bigwedge s.\ s \in S \Longrightarrow s \in (UU\ s) \wedge openin\ (top\_of\_set\ S)\ (UU\ s) \wedge$

```
                                    (∃ 𝒱. ⋃𝒱 = C ∩ p -‘ UU s ∧
                                       (∀ U ∈ 𝒱. openin (top_of_set C) U) ∧
                                       pairwise disjnt 𝒱 ∧
                                      (∀ U ∈ 𝒱. ∃ q. homeomorphism U (UU s) p q))
     using cov unfolding covering_space_def by (metis (mono_tags))
   then have ope: ⋀s. s ∈ S ⟹ s ∈ (UU s) ∧ openin (top_of_set S) (UU s)
     by blast
   have ∃ k n i. open k ∧ open n ∧
                 t ∈ k ∧ y ∈ n ∧ i ∈ S ∧ h ‘ (({0..1} ∩ k) × (U ∩ n)) ⊆ UU i if t
∈ {0..1} for t
   proof −
     have hinS: h (t, y) ∈ S
       using ‹y ∈ U› him that by blast
     then have (t,y) ∈ ({0..1} × U) ∩ h −‘ UU(h(t, y))
       using ‹y ∈ U› ‹t ∈ {0..1}› by (auto simp: ope)
    moreover have ope_01U: openin (top_of_set ({0..1} × U)) (({0..1} × U)
∩ h −‘ UU(h(t, y)))
       using hinS ope continuous_on_open_gen [OF him] conth by blast
      ultimately obtain V W where opeV: open V and t ∈ {0..1} ∩ V t ∈
{0..1} ∩ V
                           and opeW: open W and y ∈ U y ∈ W
                           and VW: ({0..1} ∩ V) × (U ∩ W) ⊆ (({0..1} × U) ∩
h −‘ UU(h(t, y)))
       by (rule Times_in_interior_subtopology) (auto simp: openin_open)
     then show ?thesis
       using hinS by blast
   qed
   then obtain K NN X where
         K: ⋀t. t ∈ {0..1} ⟹ open (K t)
       and NN: ⋀t. t ∈ {0..1} ⟹ open (NN t)
       and inUS: ⋀t. t ∈ {0..1} ⟹ t ∈ K t ∧ y ∈ NN t ∧ X t ∈ S
       and him: ⋀t. t ∈ {0..1} ⟹ h ‘ (({0..1} ∩ K t) × (U ∩ NN t)) ⊆ UU
(X t)
   by (metis (mono_tags))
   obtain 𝒯 where 𝒯 ⊆ ((λi. K i × NN i)) ‘ {0..1} finite 𝒯 {0::real..1} × {y}
⊆ ⋃𝒯
   proof (rule compactE)
     show compact ({0::real..1} × {y})
       by (simp add: compact_Times)
     show {0..1} × {y} ⊆ (⋃i∈{0..1}. K i × NN i)
       using K inUS by auto
     show ⋀B. B ∈ (λi. K i × NN i) ‘ {0..1} ⟹ open B
       using K NN by (auto simp: open_Times)
   qed blast
   then obtain tk where tk ⊆ {0..1} finite tk
                    and tk: {0::real..1} × {y} ⊆ (⋃ i ∈ tk. K i × NN i)
     by (metis (no_types, lifting) finite_subset_image)
   then have tk ≠ {}
     by auto
```

```
  define n where n = ⋂(NN ‘ tk)
  have y ∈ n open n
    using inUS NN ‹tk ⊆ {0..1}› ‹finite tk›
    by (auto simp: n_def open_INT subset_iff)
  obtain δ where 0 < δ and δ: ⋀T. ⟦T ⊆ {0..1}; diameter T < δ⟧ ⟹ ∃B∈K
‘ tk. T ⊆ B
  proof (rule Lebesgue_number_lemma [of {0..1} K ‘ tk])
    show K ‘ tk ≠ {}
      using ‹tk ≠ {}› by auto
    show {0..1} ⊆ ⋃(K ‘ tk)
      using tk by auto
    show ⋀B. B ∈ K ‘ tk ⟹ open B
      using ‹tk ⊆ {0..1}› K by auto
  qed auto
  obtain N::nat where N: N > 1 / δ
    using reals_Archimedean2 by blast
  then have N > 0
    using ‹0 < δ› order.asym by force
  have ∗: ∃ V k. openin (top_of_set U) V ∧ y ∈ V ∧
                 continuous_on ({0..of_nat n / N} × V) k ∧
                 k ‘ ({0..of_nat n / N} × V) ⊆ C ∧
                 (∀ z∈V. k (0, z) = f z) ∧
                 (∀ z∈{0..of_nat n / N} × V. h z = p (k z)) if n ≤ N for n
    using that
  proof (induction n)
    case 0
    show ?case
      apply (rule_tac x=U in exI)
      apply (rule_tac x=f ∘ snd in exI)
      apply (intro conjI ‹y ∈ U› continuous_intros continuous_on_subset [OF
contf])
      using fim  apply (auto simp: heq)
      done
  next
    case (Suc n)
    then obtain V k where opeUV: openin (top_of_set U) V
                      and y ∈ V
                      and contk: continuous_on ({0..n/N} × V) k
                      and kim: k ‘ ({0..n/N} × V) ⊆ C
                      and keq: ⋀z. z ∈ V ⟹ k (0, z) = f z
                      and heq: ⋀z. z ∈ {0..n/N} × V ⟹ h z = p (k z)
      using Suc_leD by auto
    have n ≤ N
      using Suc.prems by auto
    obtain t where t ∈ tk and t: {n/N .. (1 + real n) / N} ⊆ K t
    proof (rule bexE [OF δ])
      show {n/N .. (1 + real n) / N} ⊆ {0..1}
        using Suc.prems by (auto simp: field_split_simps)
      show diameter_less: diameter {n/N .. (1 + real n) / N} < δ
```

```
    using ‹0 < δ› N by (auto simp: field_split_simps)
qed blast
have t01: t ∈ {0..1}
  using ‹t ∈ tk› ‹tk ⊆ {0..1}› by blast
obtain 𝒱 where 𝒱: ⋃𝒱 = C ∩ p -‘ UU (X t)
          and openC: ⋀U. U ∈ 𝒱 ⟹ openin (top_of_set C) U
          and pairwise disjnt 𝒱
         and homuu: ⋀U. U ∈ 𝒱 ⟹ ∃q. homeomorphism U (UU (X t)) p q
  using inUS [OF t01] UU by meson
have n_div_N_in: n/N ∈ {n/N .. (1 + real n) / N}
  using N by (auto simp: field_split_simps)
with t have nN_in_kkt: n/N ∈ K t
  by blast
have k (n/N, y) ∈ C ∩ p -‘ UU (X t)
proof (simp, rule conjI)
  show k (n/N, y) ∈ C
    using ‹y ∈ V› kim keq by force
  have p (k (n/N, y)) = h (n/N, y)
    by (simp add: ‹y ∈ V› heq)
  also have ... ∈ h ‘ (({0..1} ∩ K t) × (U ∩ NN t))
    using ‹y ∈ V› t01 ‹n ≤ N›
    by (simp add: nN_in_kkt ‹y ∈ U› inUS field_split_simps)
  also have ... ⊆ UU (X t)
    using him t01 by blast
  finally show p (k (n/N, y)) ∈ UU (X t) .
qed
with 𝒱 have k (n/N, y) ∈ ⋃𝒱
  by blast
then obtain W where W: k (n/N, y) ∈ W and W ∈ 𝒱
  by blast
then obtain p' where openC': openin (top_of_set C) W
                 and hom': homeomorphism W (UU (X t)) p p'
  using homuu openC by blast
then have W ⊆ C
  using openin_imp_subset by blast
define W' where W' = UU(X t)
have opeVW: openin (top_of_set V) (V ∩ (k ∘ Pair (n / N)) -‘ W)
proof (rule continuous_openin_preimage [OF _ _ openC'])
  show continuous_on V (k ∘ Pair (n/N))
    by (intro continuous_intros continuous_on_subset [OF contk], auto)
  show (k ∘ Pair (n/N)) ∈ V → C
    using kim by (auto simp: ‹y ∈ V› W)
qed
obtain N' where opeUN': openin (top_of_set U) N'
        and y ∈ N' and kimw: k ‘ ({(n/N)} × N') ⊆ W
proof
  show openin (top_of_set U) (V ∩ (k ∘ Pair (n/N)) -‘ W)
    using opeUV opeVW openin_trans by blast
qed (use ‹y ∈ V› W in ‹force+›)
```

obtain Q Q' **where** *opeUQ*: *openin* (*top_of_set* U) Q
and *cloUQ'*: *closedin* (*top_of_set* U) Q'
and $y \in Q$ $Q \subseteq Q'$
and Q': $Q' \subseteq (U \cap NN(t)) \cap N' \cap V$
proof −
obtain *VO VX* **where** *open VO open VX* **and** *VO*: $V = U \cap VO$ **and** *VX*: $N' = U \cap VX$
using *opeUV opeUN'* **by** (*auto simp*: *openin_open*)
then have *open* ($NN(t) \cap VO \cap VX$)
using *NN t01* **by** *blast*
then obtain e **where** $e > 0$ **and** e: *cball* y $e \subseteq NN(t) \cap VO \cap VX$
by (*metis Int_iff* ‹$N' = U \cap VX$› ‹$V = U \cap VO$› ‹$y \in N'$› ‹$y \in V$› *inUS open_contains_cball t01*)
show *?thesis*
proof
show *openin* (*top_of_set* U) ($U \cap$ *ball* y e)
by *blast*
show *closedin* (*top_of_set* U) ($U \cap$ *cball* y e)
using e **by** (*auto simp*: *closedin_closed*)
qed (*use* ‹$y \in U$› ‹$e > 0$› *VO VX e* **in** *auto*)
qed
then have $y \in Q'$ $Q \subseteq (U \cap NN(t)) \cap N' \cap V$
by *blast+*
have *neq*: $\{0..n/N\} \cup \{n/N..(1 + \text{real } n) / N\} = \{0..(1 + \text{real } n) / N\}$
apply (*auto simp*: *field_split_simps*)
by (*metis not_less of_nat_0_le_iff of_nat_0_less_iff order_trans zero_le_mult_iff*)
then have *neqQ'*: $\{0..n/N\} \times Q' \cup \{n/N..(1 + \text{real } n) / N\} \times Q' = \{0..(1 + \text{real } n) / N\} \times Q'$
by *blast*
have *cont*: *continuous_on* ($\{0..(1 + \text{real } n) / N\} \times Q'$) ($\lambda x.$ *if* $x \in \{0..n/N\} \times Q'$ *then* k x *else* ($p' \circ h$) x)
unfolding *neqQ'* [*symmetric*]
proof (*rule continuous_on_cases_local*, *simp_all add*: *neqQ' del*: *comp_apply*)
have $\exists T.$ *closed* $T \wedge \{0..n/N\} \times Q' = \{0..(1+n)/N\} \times Q' \cap T$
using *n_div_N_in*
by (*rule_tac* x=$\{0 .. n/N\} \times$ *UNIV* **in** *exI*) (*auto simp*: *closed_Times*)
then show *closedin* (*top_of_set* ($\{0..(1 + \text{real } n) / N\} \times Q'$)) ($\{0..n/N\} \times Q'$)
by (*simp add*: *closedin_closed*)
have $\exists T.$ *closed* $T \wedge \{n/N..(1+n)/N\} \times Q' = \{0..(1+n)/N\} \times Q' \cap T$
by (*rule_tac* x=$\{n/N..(1+n)/N\} \times$ *UNIV* **in** *exI*) (*auto simp*: *closed_Times order_trans* [*rotated*])
then show *closedin* (*top_of_set* ($\{0..(1 + \text{real } n) / N\} \times Q'$)) ($\{n/N..(1 + \text{real } n) / N\} \times Q'$)
by (*simp add*: *closedin_closed*)
show *continuous_on* ($\{0..n/N\} \times Q'$) k
using Q' **by** (*auto intro*: *continuous_on_subset* [*OF contk*])
have *continuous_on* ($\{n/N..(1 + \text{real } n) / N\} \times Q'$) h
proof (*rule continuous_on_subset* [*OF conth*])

```
      show {n/N..(1 + real n) / N} × Q' ⊆ {0..1} × U
      proof (clarsimp, intro conjI)
        fix a b
        assume b ∈ Q' and a: n/N ≤ a a ≤ (1 + real n) / N
        have 0 ≤ n/N (1 + real n) / N ≤ 1
          using a Suc.prems by (auto simp: divide_simps)
        with a show 0 ≤ a  a ≤ 1
          by linarith+
        show b ∈ U
          using ‹b ∈ Q'› cloUQ' closedin_imp_subset by blast
      qed
    qed
    moreover have continuous_on (h ‘ ({n/N..(1 + real n) / N} × Q')) p'
    proof (rule continuous_on_subset [OF homeomorphism_cont2 [OF hom']])
      have h ‘ ({n/N..(1 + real n) / N} × Q') ⊆ h ‘ (({0..1} ∩ K t) × (U ∩
NN t))
      proof (rule image_mono)
        show {n/N..(1 + real n) / N} × Q' ⊆ ({0..1} ∩ K t) × (U ∩ NN t)
        proof (clarsimp, intro conjI)
          fix a::real and b
          assume b ∈ Q' n/N ≤ a a ≤ (1 + real n) / N
          show 0 ≤ a
              by (meson ‹n/N ≤ a› divide_nonneg_nonneg of_nat_0_le_iff
order_trans)
          show a ≤ 1
            using Suc.prems ‹a ≤ (1 + real n) / N› order_trans by force
          show a ∈ K t
            using ‹a ≤ (1 + real n) / N› ‹n/N ≤ a› t by auto
          show b ∈ U
            using ‹b ∈ Q'› cloUQ' closedin_imp_subset by blast
          show b ∈ NN t
            using Q' ‹b ∈ Q'› by auto
        qed
      qed
      with him show h ‘ ({n/N..(1 + real n) / N} × Q') ⊆ UU (X t)
        using t01 by blast
    qed
    ultimately show continuous_on ({n/N..(1 + real n) / N} × Q') (p' ∘ h)
      by (rule continuous_on_compose)
    have k (n/N, b) = p' (h (n/N, b)) if b ∈ Q' for b
    proof −
      have k (n/N, b) ∈ W
        using that Q' kimw  by force
      then have k (n/N, b) = p' (p (k (n/N, b)))
        by (simp add:  homeomorphism_apply1 [OF hom'])
      then show ?thesis
        using Q' that by (force simp: heq)
    qed
    then show ⋀x. x ∈ {n/N..(1 + real n) / N} × Q' ∧
```

```
            x ∈ {0..n/N} × Q' ⟹ k x = (p' ∘ h) x
        by auto
      qed
      have h_in_UU: h (x, y) ∈ UU (X t) if y ∈ Q ¬ x ≤ n/N 0 ≤ x x ≤ (1 +
real n) / N for x y
      proof −
        have x ≤ 1
          using Suc.prems that order_trans by force
        moreover have x ∈ K t
          by (meson atLeastAtMost_iff le_less not_le subset_eq t that)
        moreover have y ∈ U
          using ‹y ∈ Q› opeUQ openin_imp_subset by blast
        moreover have y ∈ NN t
          using Q' ‹Q ⊆ Q'› ‹y ∈ Q› by auto
        ultimately have (x, y) ∈ (({0..1} ∩ K t) × (U ∩ NN t))
          using that by auto
        then have h (x, y) ∈ h ‘ (({0..1} ∩ K t) × (U ∩ NN t))
          by blast
        also have ... ⊆ UU (X t)
          by (metis him t01)
        finally show ?thesis .
      qed
      let ?k = (λx. if x ∈ {0..n/N} × Q' then k x else (p' ∘ h) x)
      show ?case
      proof (intro exI conjI)
        show continuous_on ({0..real (Suc n) / N} × Q) ?k
          using ‹Q ⊆ Q'› by (auto intro: continuous_on_subset [OF cont])
        have ⋀x y. ⟦x ≤ n/N; y ∈ Q'; 0 ≤ x⟧ ⟹ k (x, y) ∈ C
          using kim Q' by force
        moreover have p' (h (x, y)) ∈ C if y ∈ Q ¬ x ≤ n/N 0 ≤ x x ≤ (1 +
real n) / N for x y
        proof (rule ‹W ⊆ C› [THEN subsetD])
          show p' (h (x, y)) ∈ W
            using homeomorphism_image2 [OF hom', symmetric] h_in_UU Q'
‹Q ⊆ Q'› ‹W ⊆ C› that by auto
        qed
        ultimately show ?k ‘ ({0..real (Suc n) / N} × Q) ⊆ C
          using Q' ‹Q ⊆ Q'› by force
        show ∀z∈Q. ?k (0, z) = f z
          using Q' keq ‹Q ⊆ Q'› by auto
        show ∀z ∈ {0..real (Suc n) / N} × Q. h z = p(?k z)
          using ‹Q ⊆ U ∩ NN t ∩ N' ∩ V› heq Q' ‹Q ⊆ Q'›
            by (auto simp: homeomorphism_apply2 [OF hom'] dest: h_in_UU)
        qed (auto simp: ‹y ∈ Q› opeUQ)
      qed
      show ?thesis
        using ∗[OF order_refl] N ‹0 < δ› by (simp add: split: if_split_asm)
    qed
    then obtain V fs where opeV: ⋀y. y ∈ U ⟹ openin (top_of_set U) (V y)
```

```
        and V: ⋀y. y ∈ U ⟹ y ∈ V y
        and contfs: ⋀y. y ∈ U ⟹ continuous_on ({0..1} × V y) (fs y)
        and ∗: ⋀y. y ∈ U ⟹ (fs y) ‘ ({0..1} × V y) ⊆ C ∧
                            (∀z ∈ V y. fs y (0, z) = f z) ∧
                            (∀z ∈ {0..1} × V y. h z = p(fs y z))
    by (metis (mono_tags))
  then have VU: ⋀y. y ∈ U ⟹ V y ⊆ U
    by (meson openin_imp_subset)
  obtain k where contk: continuous_on ({0..1} × U) k
          and k: ⋀x i. ⟦i ∈ U; x ∈ {0..1} × U ∩ {0..1} × V i⟧ ⟹ k x = fs i x
  proof (rule pasting_lemma_exists)
    let ?X = top_of_set ({0..1::real} × U)
    show topspace ?X ⊆ (⋃i∈U. {0..1} × V i)
      using V by force
    show ⋀i. i ∈ U ⟹ openin (top_of_set ({0..1} × U)) ({0..1} × V i)
      by (simp add: Abstract_Topology.openin_Times opeV)
    show ⋀i. i ∈ U ⟹ continuous_map
           (subtopology (top_of_set ({0..1} × U)) ({0..1} × V i)) euclidean (fs
i)
      by (metis contfs subtopology_subtopology continuous_map_iff_continuous
Times_Int_Times VU inf.absorb_iff2 inf.idem)
    show fs i x = fs j x  if i ∈ U j ∈ U and x: x ∈ topspace ?X ∩ {0..1} × V i
∩ {0..1} × V j
      for i j x
    proof −
      obtain u y where x = (u, y) y ∈ V i y ∈ V j 0 ≤ u u ≤ 1
        using x by auto
      show ?thesis
      proof (rule covering_space_lift_unique [OF cov, of _ (0,y) _ {0..1} × {y}
h])
        show fs i (0, y) = fs j (0, y)
          using∗V by (simp add: ‹y ∈ V i› ‹y ∈ V j› that)
        show conth_y: continuous_on ({0..1} × {y}) h
          using VU ‹y ∈ V j› that by (auto intro: continuous_on_subset [OF
conth])
        show h ∈ ({0..1} × {y}) → S
          using ‹y ∈ V i› assms(3) VU that by fastforce
        show continuous_on ({0..1} × {y}) (fs i)
          using continuous_on_subset [OF contfs] ‹i ∈ U›
          by (simp add: ‹y ∈ V i› subset_iff)
        show fs i ∈ ({0..1} × {y}) → C
          using ∗ ‹y ∈ V i› ‹i ∈ U› by fastforce
        show ⋀x. x ∈ {0..1} × {y} ⟹ h x = p (fs i x)
          using ∗ ‹y ∈ V i› ‹i ∈ U› by blast
        show continuous_on ({0..1} × {y}) (fs j)
          using continuous_on_subset [OF contfs] ‹j ∈ U›
          by (simp add: ‹y ∈ V j› subset_iff)
        show fs j ∈ ({0..1} × {y}) → C
          using ∗ ‹y ∈ V j› ‹j ∈ U› by fastforce
```

```
      show ⋀x. x ∈ {0..1} × {y} ⟹ h x = p (fs j x)
        using ∗ ‹y ∈ V j› ‹j ∈ U› by blast
      show connected ({0..1::real} × {y})
        using connected_Icc connected_Times connected_sing by blast
      show (0, y) ∈ {0..1::real} × {y}
        by force
      show x ∈ {0..1} × {y}
        using ‹x = (u, y)› x by blast
    qed
  qed
 qed force
 show ?thesis
 proof
  show k ∈ ({0..1} × U) → C
    using V∗k VU by fastforce
  show ⋀y. y ∈ U ⟹ k (0, y) = f y
    by (simp add: V∗k)
  show ⋀z. z ∈ {0..1} × U ⟹ h z = p (k z)
    using V∗k by auto
 qed (auto simp: contk)
qed

corollary covering_space_lift_homotopy_alt:
 fixes p :: 'a::real_normed_vector ⇒ 'b::real_normed_vector
   and h :: 'c::real_normed_vector × real ⇒ 'b
 assumes cov: covering_space C p S
     and conth: continuous_on (U × {0..1}) h
     and him: h ∈ (U × {0..1}) → S
     and heq: ⋀y. y ∈ U ⟹ h (y,0) = p(f y)
     and contf: continuous_on U f and fim: f ∈ U → C
 obtains k where continuous_on (U × {0..1}) k
                 k ∈ (U × {0..1}) → C
                 ⋀y. y ∈ U ⟹ k(y, 0) = f y
                 ⋀z. z ∈ U × {0..1} ⟹ h z = p(k z)
proof −
 have continuous_on ({0..1} × U) (h ∘ (λz. (snd z, fst z)))
   by (intro continuous_intros continuous_on_subset [OF conth]) auto
 then obtain k where contk: continuous_on ({0..1} × U) k
                 and kim:  k ‘ ({0..1} × U) ⊆ C
                 and k0: ⋀y. y ∈ U ⟹ k(0, y) = f y
                 and heqp: ⋀z. z ∈ {0..1} × U ⟹ (h ∘ (λz. Pair (snd z) (fst z)))
z = p(k z)
   apply (rule covering_space_lift_homotopy [OF cov _ _ _ contf fim])
   using him  by (auto simp: contf heq)
 show ?thesis
 proof
  show continuous_on (U × {0..1}) (k ∘ (λz. (snd z, fst z)))
    by (intro continuous_intros continuous_on_subset [OF contk]) auto
 qed (use kim heqp in ‹auto simp: k0›)
```

qed

corollary *covering_space_lift_homotopic_function*:
fixes p :: *'a*::*real_normed_vector* ⇒ *'b*::*real_normed_vector* **and** g:: *'c*::*real_normed_vector* ⇒ *'a*
assumes *cov*: *covering_space C p S*
and *contg*: *continuous_on U g*
and *gim*: $g \in U \to C$
and *pgeq*: $\bigwedge y.\ y \in U \Longrightarrow p(g\ y) = f\ y$
and *hom*: *homotopic_with_canon* (λ*x*. *True*) *U S f f'*
obtains *g'* **where** *continuous_on U g' image g' U* ⊆ *C* $\bigwedge y.\ y \in U \Longrightarrow p(g'\ y) = f'\ y$
proof −
obtain *h* **where** *conth*: *continuous_on* ({*0..1*::*real*} × *U*) *h*
and *him*: $h \in (\{0..1\} \times U) \to S$
and *h0*: $\bigwedge x.\ h(0, x) = f\ x$
and *h1*: $\bigwedge x.\ h(1, x) = f'\ x$
using *hom* **by** (*auto simp*: *homotopic_with_def*)
have $\bigwedge y.\ y \in U \Longrightarrow h\ (0, y) = p\ (g\ y)$
by (*simp add*: *h0 pgeq*)
then obtain *k* **where** *contk*: *continuous_on* ({*0..1*} × *U*) *k*
and *kim*: *k* ‘ ({*0..1*} × *U*) ⊆ *C*
and *k0*: $\bigwedge y.\ y \in U \Longrightarrow k(0, y) = g\ y$
and *heq*: $\bigwedge z.\ z \in \{0..1\} \times U \Longrightarrow h\ z = p(k\ z)$
using *covering_space_lift_homotopy* [*OF cov conth him _ contg gim*] **by** (*metis image_subset_iff_funcset*)
show *?thesis*
proof
show *continuous_on U* (*k* ∘ *Pair 1*)
by (*meson contk atLeastAtMost_iff continuous_on_o_Pair order_refl zero_le_one*)
show (*k* ∘ *Pair 1*) ‘ *U* ⊆ *C*
using *kim* **by** *auto*
show $\bigwedge y.\ y \in U \Longrightarrow p\ ((k \circ Pair\ 1)\ y) = f'\ y$
by (*auto simp*: *h1 heq* [*symmetric*])
qed
qed

corollary *covering_space_lift_inessential_function*:
fixes p :: *'a*::*real_normed_vector* ⇒ *'b*::*real_normed_vector* **and** *U* :: *'c*::*real_normed_vector set*
assumes *cov*: *covering_space C p S*
and *hom*: *homotopic_with_canon* (λ*x*. *True*) *U S f* (λ*x*. *a*)
obtains *g* **where** *continuous_on U g g* ‘ *U* ⊆ *C* $\bigwedge y.\ y \in U \Longrightarrow p(g\ y) = f\ y$
proof (*cases U* = {})
case *True*
then show *?thesis*
using *that continuous_on_empty* **by** *blast*
next
case *False*

then obtain b **where** b: $b \in C\ p\ b = a$
using *covering_space_imp_surjective* [*OF cov*] *homotopic_with_imp_subset2* [*OF hom*]
by *auto*
then have *gim*: $(\lambda y.\ b) \in U \to C$
by *blast*
show *?thesis*
proof (*rule covering_space_lift_homotopic_function* [*OF cov continuous_on_const gim*])
show $\bigwedge y.\ y \in U \Longrightarrow p\ b = a$
using b **by** *auto*
qed (*use that homotopic_with_symD* [*OF hom*] **in** *auto*)
qed

6.16.5 Lifting of general functions to covering space

proposition *covering_space_lift_path_strong*:
fixes p :: $'a$::*real_normed_vector* ⇒ $'b$::*real_normed_vector*
and f :: $'c$::*real_normed_vector* ⇒ $'b$
assumes *cov*: *covering_space* $C\ p\ S$ **and** $a \in C$
and *path* g **and** *pag*: *path_image* $g \subseteq S$ **and** *pas*: *pathstart* $g = p\ a$
obtains h **where** *path* h *path_image* $h \subseteq C$ *pathstart* $h = a$
and $\bigwedge t.\ t \in \{0..1\} \Longrightarrow p(h\ t) = g\ t$
proof −
obtain k:: *real* × $'c$ ⇒ $'a$
where *contk*: *continuous_on* $(\{0..1\} \times \{undefined\})\ k$
and *kim*: k ‘ $(\{0..1\} \times \{undefined\}) \subseteq C$
and *k0*: $k\ (0,\ undefined) = a$
and *pk*: $\bigwedge z.\ z \in \{0..1\} \times \{undefined\} \Longrightarrow p(k\ z) = (g \circ fst)\ z$
proof (*rule covering_space_lift_homotopy* [*OF cov*, *of* $\{undefined\}$ $g \circ fst$])
show *continuous_on* $(\{0..1$::*real*$\} \times \{undefined$::$'c\})\ (g \circ fst)$
using ‹*path g*› **by** (*intro continuous_intros*) (*simp add*: *path_def*)
show $(g \circ fst) \in (\{0..1\} \times \{undefined\}) \to S$
using *pag* **by** (*auto simp*: *path_image_def*)
show $(g \circ fst)\ (0,\ y) = p\ a$ **if** $y \in \{undefined\}$ **for** y::$'c$
by (*metis comp_def fst_conv pas pathstart_def*)
qed (*use assms* **in** *auto*)
show *?thesis*
proof
show *path* $(k \circ (\lambda t.\ Pair\ t\ undefined))$
unfolding *path_def*
by (*intro continuous_on_compose continuous_intros continuous_on_subset* [*OF contk*]) *auto*
show *path_image* $(k \circ (\lambda t.\ (t,\ undefined))) \subseteq C$
using *kim* **by** (*auto simp*: *path_image_def*)
show *pathstart* $(k \circ (\lambda t.\ (t,\ undefined))) = a$
by (*auto simp*: *pathstart_def k0*)
show $\bigwedge t.\ t \in \{0..1\} \Longrightarrow p\ ((k \circ (\lambda t.\ (t,\ undefined)))\ t) = g\ t$
by (*auto simp*: *pk*)

qed
qed

corollary *covering_space_lift_path*:
fixes $p :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$
assumes *cov*: *covering_space C p S* **and** *path g* **and** *pig*: *path_image g* $\subseteq S$
obtains *h* **where** *path h path_image h* $\subseteq C$ $\bigwedge t.\ t \in \{0..1\} \Longrightarrow p(h\ t) = g\ t$
proof −
obtain *a* **where** $a \in C$ *pathstart g* $= p\ a$
by (*metis pig cov covering_space_imp_surjective imageE pathstart_in_path_image subsetCE*)
show *?thesis*
using *covering_space_lift_path_strong* [*OF cov* ‹$a \in C$› ‹*path g*› *pig*]
by (*metis* ‹*pathstart g* $= p\ a$› *that*)
qed

proposition *covering_space_lift_homotopic_paths*:
fixes $p :: 'a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector$
assumes *cov*: *covering_space C p S*
and *path g1* **and** *pig1*: *path_image g1* $\subseteq S$
and *path g2* **and** *pig2*: *path_image g2* $\subseteq S$
and *hom*: *homotopic_paths S g1 g2*
and *path h1* **and** *pih1*: *path_image h1* $\subseteq C$ **and** *ph1*: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow p(h1\ t) = g1\ t$
and *path h2* **and** *pih2*: *path_image h2* $\subseteq C$ **and** *ph2*: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow p(h2\ t) = g2\ t$
and *h1h2*: *pathstart h1* $=$ *pathstart h2*
shows *homotopic_paths C h1 h2*
proof −
obtain $h :: real \times real \Rightarrow 'b$
where *conth*: *continuous_on* $(\{0..1\} \times \{0..1\})\ h$
and *him*: $h \in (\{0..1\} \times \{0..1\}) \to S$
and *h0*: $\bigwedge x.\ h\ (0, x) = g1\ x$ **and** *h1*: $\bigwedge x.\ h\ (1, x) = g2\ x$
and *heq0*: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow h\ (t, 0) = g1\ 0$
and *heq1*: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow h\ (t, 1) = g1\ 1$
using *hom* **by** (*auto simp*: *homotopic_paths_def homotopic_with_def pathstart_def pathfinish_def image_subset_iff_funcset*)
obtain *k* **where** *contk*: *continuous_on* $(\{0..1\} \times \{0..1\})\ k$
and *kim*: $k \in (\{0..1\} \times \{0..1\}) \to C$
and *kh2*: $\bigwedge y.\ y \in \{0..1\} \Longrightarrow k\ (y, 0) = h2\ 0$
and *hpk*: $\bigwedge z.\ z \in \{0..1\} \times \{0..1\} \Longrightarrow h\ z = p\ (k\ z)$
proof (*rule covering_space_lift_homotopy_alt* [*OF cov conth him*])
show $\bigwedge y.\ y \in \{0..1\} \Longrightarrow h\ (y, 0) = p\ (h2\ 0)$
by (*metis atLeastAtMost_iff h1h2 heq0 order_refl pathstart_def ph1 zero_le_one*)
qed (*use path_image_def pih2* **in** ‹*fastforce+*›)
have *contg1*: *continuous_on* $\{0..1\}$ *g1* **and** *contg2*: *continuous_on* $\{0..1\}$ *g2*
using ‹*path g1*› ‹*path g2*› *path_def* **by** *blast+*
have *g1im*: $g1 \in \{0..1\} \to S$ **and** *g2im*: $g2 \in \{0..1\} \to S$

```
    using path_image_def pig1 pig2 by auto
  have conth1: continuous_on {0..1} h1 and conth2: continuous_on {0..1} h2
    using ‹path h1› ‹path h2› path_def by blast+
  have h1im: h1 ∈ {0..1} → C and h2im: h2 ∈ {0..1} → C
    using path_image_def pih1 pih2 by auto
  show ?thesis
    unfolding homotopic_paths pathstart_def pathfinish_def
  proof (intro exI conjI ballI)
    show keqh1: k(0, x) = h1 x if x ∈ {0..1} for x
    proof (rule covering_space_lift_unique [OF cov _ contg1 g1im])
      show k (0,0) = h1 0
       by (metis atLeastAtMost_iff h1h2 kh2 order_refl pathstart_def zero_le_one)
      show continuous_on {0..1} (λa. k (0, a))
        by (intro continuous_intros continuous_on_compose2 [OF contk]) auto
      show ⋀x. x ∈ {0..1} ⟹ g1 x = p (k (0, x))
       by (metis atLeastAtMost_iff h0 hpk zero_le_one mem_Sigma_iff order_refl)
    qed (use conth1 h1im kim that in ‹auto simp: ph1›)
    show k(1, x) = h2 x if x ∈ {0..1} for x
    proof (rule covering_space_lift_unique [OF cov _ contg2 g2im])
      show k (1,0) = h2 0
        by (metis atLeastAtMost_iff kh2 order_refl zero_le_one)
      show continuous_on {0..1} (λa. k (1, a))
        by (intro continuous_intros continuous_on_compose2 [OF contk]) auto
      show ⋀x. x ∈ {0..1} ⟹ g2 x = p (k (1, x))
       by (metis atLeastAtMost_iff h1 hpk mem_Sigma_iff order_refl zero_le_one)
    qed (use conth2 h2im kim that in ‹auto simp: ph2›)
    show ⋀t. t ∈ {0..1} ⟹ (k ∘ Pair t) 0 = h1 0
      by (metis comp_apply h1h2 kh2 pathstart_def)
    show (k ∘ Pair t) 1 = h1 1 if t ∈ {0..1} for t
    proof (rule covering_space_lift_unique
          [OF cov, of λa. (k ∘ Pair a) 1 0 λa. h1 1 {0..1}  λx. g1 1])
      show (k ∘ Pair 0) 1 = h1 1
        using keqh1 by auto
      show continuous_on {0..1} (λa. (k ∘ Pair a) 1)
        by (auto intro!: continuous_intros continuous_on_compose2 [OF contk])
      show ⋀x. x ∈ {0..1} ⟹ g1 1 = p ((k ∘ Pair x) 1)
        using heq1 hpk by auto
    qed (use contk kim g1im h1im that in ‹auto simp: ph1›)
  qed (use contk kim in auto)
qed


corollary covering_space_monodromy:
  fixes p :: 'a::real_normed_vector ⇒ 'b::real_normed_vector
  assumes cov: covering_space C p S
      and path g1 and pig1: path_image g1 ⊆ S
      and path g2 and pig2: path_image g2 ⊆ S
      and hom: homotopic_paths S g1 g2
       and path h1 and pih1: path_image h1 ⊆ C and ph1: ⋀t. t ∈ {0..1} ⟹
```

$p(h1\ t) = g1\ t$
and *path h2* **and** *pih2*: *path_image h2* $\subseteq$ *C* **and** *ph2*: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow$ $p(h2\ t) = g2\ t$
and *h1h2*: *pathstart h1* = *pathstart h2*
shows *pathfinish h1* = *pathfinish h2*
using *covering_space_lift_homotopic_paths* [*OF assms*] *homotopic_paths_imp_pathfinish*
by *blast*

corollary *covering_space_lift_homotopic_path*:
fixes *p* :: *'a*::*real_normed_vector* $\Rightarrow$ *'b*::*real_normed_vector*
assumes *cov*: *covering_space C p S*
and *hom*: *homotopic_paths S f f'*
and *path g* **and** *pig*: *path_image g* $\subseteq$ *C*
and *a*: *pathstart g* = *a* **and** *b*: *pathfinish g* = *b*
and *pgeq*: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow p(g\ t) = f\ t$
obtains *g'* **where** *path g' path_image g'* $\subseteq$ *C*
pathstart g' = *a pathfinish g'* = *b* $\bigwedge t.\ t \in \{0..1\} \Longrightarrow p(g'\ t) = f'\ t$
proof (*rule covering_space_lift_path_strong* [*OF cov, of a f'*])
show *a* $\in$ *C*
using *a pig* **by** *auto*
show *path f' path_image f'* $\subseteq$ *S*
using *hom homotopic_paths_imp_path homotopic_paths_imp_subset* **by** *blast+*
show *pathstart f'* = *p a*
by (*metis a atLeastAtMost_iff hom homotopic_paths_imp_pathstart order_refl pathstart_def pgeq zero_le_one*)
qed (*metis* (*mono_tags*, *lifting*) *assms cov covering_space_monodromy hom homotopic_paths_imp_path homotopic_paths_imp_subset pgeq pig*)

proposition *covering_space_lift_general*:
fixes *p* :: *'a*::*real_normed_vector* $\Rightarrow$ *'b*::*real_normed_vector*
and *f* :: *'c*::*real_normed_vector* $\Rightarrow$ *'b*
assumes *cov*: *covering_space C p S* **and** *a* $\in$ *C z* $\in$ *U*
and *U*: *path_connected U locally path_connected U*
and *contf*: *continuous_on U f* **and** *fim*: *f* $\in$ *U* $\to$ *S*
and *feq*: *f z* = *p a*
and *hom*: $\bigwedge r.\ [\![$*path r*; *path_image r* $\subseteq$ *U*; *pathstart r* = *z*; *pathfinish r* = *z*$]\!]$
$\Longrightarrow \exists q.$ *path q* $\wedge$ *path_image q* $\subseteq$ *C* $\wedge$
pathstart q = *a* $\wedge$ *pathfinish q* = *a* $\wedge$
homotopic_paths S (*f* $\circ$ *r*) (*p* $\circ$ *q*)
obtains *g* **where** *continuous_on U g g* $\in$ *U* $\to$ *C g z* = *a* $\bigwedge y.\ y \in U \Longrightarrow p(g\ y) = f\ y$
proof −
have $*$: $\exists g\ h.$ *path g* $\wedge$ *path_image g* $\subseteq$ *U* $\wedge$
pathstart g = *z* $\wedge$ *pathfinish g* = *y* $\wedge$
path h $\wedge$ *path_image h* $\subseteq$ *C* $\wedge$ *pathstart h* = *a* $\wedge$
$(\forall t \in \{0..1\}.\ p(h\ t) = f(g\ t))$
if *y* $\in$ *U* **for** *y*

proof −
obtain *g* **where** *path g path_image g* ⊆ *U* **and** *pastg*: *pathstart g* = *z*
and *pafig*: *pathfinish g* = *y*
using *U* ‹*z* ∈ *U*› ‹*y* ∈ *U*› **by** (*force simp*: *path_connected_def*)
obtain *h* **where** *path h path_image h* ⊆ *C pathstart h* = *a*
and ⋀*t*. *t* ∈ {*0..1*} ⟹ *p*(*h t*) = (*f* ∘ *g*) *t*
proof (*rule covering_space_lift_path_strong* [*OF cov* ‹*a* ∈ *C*›])
show *path* (*f* ∘ *g*)
using ‹*path g*› ‹*path_image g* ⊆ *U*› *contf continuous_on_subset path_continuous_image*
by *blast*
show *path_image* (*f* ∘ *g*) ⊆ *S*
by (*metis* ‹*path_image g* ⊆ *U*› *fim image_mono path_image_compose*
subset_trans image_subset_iff_funcset)
show *pathstart* (*f* ∘ *g*) = *p a*
by (*simp add*: *feq pastg pathstart_compose*)
qed *auto*
then show *?thesis*
by (*metis* ‹*path g*› ‹*path_image g* ⊆ *U*› *comp_apply pafig pastg*)
qed
have ∃*l*. ∀ *g h*. *path g* ∧ *path_image g* ⊆ *U* ∧ *pathstart g* = *z* ∧ *pathfinish g* =
y ∧
path h ∧ *path_image h* ⊆ *C* ∧ *pathstart h* = *a* ∧
(∀ *t* ∈ {*0..1*}. *p*(*h t*) = *f*(*g t*)) ⟶ *pathfinish h* = *l* **for** *y*
proof −
have *pathfinish h* = *pathfinish h′*
if *g*: *path g path_image g* ⊆ *U pathstart g* = *z pathfinish g* = *y*
and *h*: *path h path_image h* ⊆ *C pathstart h* = *a*
and *phg*: ⋀*t*. *t* ∈ {*0..1*} ⟹ *p*(*h t*) = *f*(*g t*)
and *g′*: *path g′ path_image g′* ⊆ *U pathstart g′* = *z pathfinish g′* = *y*
and *h′*: *path h′ path_image h′* ⊆ *C pathstart h′* = *a*
and *phg′*: ⋀*t*. *t* ∈ {*0..1*} ⟹ *p*(*h′ t*) = *f*(*g′ t*)
for *g h g′ h′*
proof −
obtain *q* **where** *path q* **and** *piq*: *path_image q* ⊆ *C* **and** *pastq*: *pathstart q*
= *a* **and** *pafiq*: *pathfinish q* = *a*
and *homS*: *homotopic_paths S* (*f* ∘ *g* +++ *reversepath g′*) (*p* ∘ *q*)
using *g g′ hom* [*of g* +++ *reversepath g′*] **by** (*auto simp*: *subset_path_image_join*)
have *papq*: *path* (*p* ∘ *q*)
using *homS homotopic_paths_imp_path* **by** *blast*
have *pipq*: *path_image* (*p* ∘ *q*) ⊆ *S*
using *homS homotopic_paths_imp_subset* **by** *blast*
obtain *q′* **where** *path q′ path_image q′* ⊆ *C*
and *pathstart q′* = *pathstart q pathfinish q′* = *pathfinish q*
and *pq′_eq*: ⋀*t*. *t* ∈ {*0..1*} ⟹ *p* (*q′ t*) = (*f* ∘ *g* +++ *reversepath*
g′) *t*
using *covering_space_lift_homotopic_path* [*OF cov homotopic_paths_sym*
[*OF homS*] ‹*path q*› *piq refl refl*]
by *auto*
have *q′ t* = (*h* ∘ (∗$_R$) *2*) *t* **if** *0* ≤ *t t* ≤ *1/2* **for** *t*

proof (*rule covering_space_lift_unique* [*OF cov*, *of* q' *0* $h \circ (*_R)$ *2* $\{0..1/2\}$ $f \circ g \circ (*_R)$ *2 t*])
show q' *0* $= (h \circ (*_R)$ *2*$)$ *0*
by (*metis* ‹*pathstart* $q' =$ *pathstart q*› *comp_def h*(*3*) *pastq pathstart_def pth_4*(*2*))
show *continuous_on* $\{0..1/2\}$ $(f \circ g \circ (*_R)$ *2*$)$
proof (*intro continuous_intros continuous_on_path* [*OF* ‹*path g*›] *continuous_on_subset* [*OF contf*])
show *g* ‘ $(*_R)$ *2* ‘ $\{0..1/2\} \subseteq U$
using *g path_image_def* **by** *fastforce*
qed *auto*
show $(f \circ g \circ (*_R)$ *2*$) \in \{0..1/2\} \to S$
using *g*(*2*) *fim* **by** (*fastforce simp*: *path_image_def image_subset_iff_funcset*)
show $(h \circ (*_R)$ *2*$) \in \{0..1/2\} \to C$
using *h path_image_def* **by** *fastforce*
show $q' \in \{0..1/2\} \to C$
using ‹*path_image* $q' \subseteq C$› *path_image_def* **by** *fastforce*
show $\bigwedge x.\ x \in \{0..1/2\} \Longrightarrow (f \circ g \circ (*_R)$ *2*$)\ x = p\ (q'\ x)$
by (*auto simp*: *joinpaths_def pq'_eq*)
show $\bigwedge x.\ x \in \{0..1/2\} \Longrightarrow (f \circ g \circ (*_R)$ *2*$)\ x = p\ ((h \circ (*_R)$ *2*$)\ x)$
by (*simp add*: *phg*)
show *continuous_on* $\{0..1/2\}$ q'
by (*simp add*: *continuous_on_path* ‹*path* q'›)
show *continuous_on* $\{0..1/2\}$ $(h \circ (*_R)$ *2*$)$
by (*intro continuous_intros continuous_on_path* [*OF* ‹*path h*›]) *auto*
qed (*use that* **in** *auto*)
moreover have $q'\ t = (\mathit{reversepath}\ h' \circ (\lambda t.\ 2 *_R t - 1))\ t$ **if** $1/2 < t\ t \leq 1$ **for** *t*
proof (*rule covering_space_lift_unique* [*OF cov*, *of* q' *1 reversepath* $h' \circ (\lambda t.\ 2 *_R t - 1)$ $\{1/2<..1\}$ *f* $\circ$ *reversepath* $g' \circ (\lambda t.\ 2 *_R t - 1)$ *t*])
show q' *1* $=$ (*reversepath* $h' \circ (\lambda t.\ 2 *_R t - 1)$) *1*
using h' ‹*pathfinish* $q' =$ *pathfinish q*› *pafiq*
by (*simp add*: *pathstart_def pathfinish_def reversepath_def*)
show *continuous_on* $\{1/2<..1\}$ $(f \circ$ *reversepath* $g' \circ (\lambda t.\ 2 *_R t - 1))$
proof (*intro continuous_intros continuous_on_path* ‹*path* g'› *continuous_on_subset* [*OF contf*])
show *reversepath* g' ‘ $(\lambda t.\ 2 *_R t - 1)$ ‘ $\{1/2<..1\} \subseteq U$
using g' **by** (*auto simp*: *path_image_def reversepath_def*)
qed (*use* g' **in** *auto*)
show $(f \circ$ *reversepath* $g' \circ (\lambda t.\ 2 *_R t - 1)) \in \{1/2<..1\} \to S$
using g'(*2*) *path_image_def fim* **by** (*auto simp*: *image_subset_iff path_image_def reversepath_def*)
show $q' \in \{1/2<..1\} \to C$
using ‹*path_image* $q' \subseteq C$› *path_image_def* **by** *fastforce*
show (*reversepath* $h' \circ (\lambda t.\ 2 *_R t - 1)) \in \{1/2<..1\} \to C$
using h' **by** (*simp add*: *path_image_def reversepath_def subset_eq*)
show $\bigwedge x.\ x \in \{1/2<..1\} \Longrightarrow (f \circ$ *reversepath* $g' \circ (\lambda t.\ 2 *_R t - 1))\ x = p\ (q'\ x)$
by (*auto simp*: *joinpaths_def pq'_eq*)

```
      show ⋀x. x ∈ {1/2<..1} ⟹
             (f ∘ reversepath g' ∘ (λt. 2 *R t − 1)) x = p ((reversepath h' ∘ (λt.
2 *R t − 1)) x)
        by (simp add: phg' reversepath_def)
      show continuous_on {1/2<..1} q'
        by (auto intro: continuous_on_path [OF ‹path q'›])
      show continuous_on {1/2<..1} (reversepath h' ∘ (λt. 2 *R t − 1))
         by (intro continuous_intros continuous_on_path ‹path h'›) (use h' in
auto)
    qed (use that in auto)
    ultimately have q' t = (h +++ reversepath h') t if 0 ≤ t t ≤ 1 for t
      using that by (simp add: joinpaths_def)
    then have path(h +++ reversepath h')
      by (auto intro: path_eq [OF ‹path q'›])
    then show ?thesis
      by (auto simp: ‹path h› ‹path h'›)
  qed
  then show ?thesis by metis
 qed
 then obtain l :: 'c ⇒ 'a
       where l: ⋀y g h. ⟦path g; path_image g ⊆ U; pathstart g = z; pathfinish
g = y;
                        path h; path_image h ⊆ C; pathstart h = a;
                        ⋀t. t ∈ {0..1} ⟹ p(h t) = f(g t)⟧ ⟹ pathfinish h = l y
  by metis
 show ?thesis
 proof
  show pleq: p (l y) = f y if y ∈ U for y
   using∗[OF ‹y ∈ U›] by (metis l atLeastAtMost_iff order_refl pathfinish_def
zero_le_one)
  show l z = a
   using l [of linepath z z z linepath a a] by (auto simp: assms)
  show LC: l ∈ U → C
   by (clarify dest!: ∗) (metis (full_types) l pathfinish_in_path_image subsetCE)
  have ∃ T. openin (top_of_set U) T ∧ y ∈ T ∧ T ⊆ U ∩ l −‘ X
      if X: openin (top_of_set C) X and y ∈ U l y ∈ X for X y
  proof −
   have X ⊆ C
    using X openin_euclidean_subtopology_iff by blast
   have f y ∈ S
    using fim ‹y ∈ U› by blast
   then obtain W V
         where WV: f y ∈ W ∧ openin (top_of_set S) W ∧
                   (⋃V = C ∩ p −‘ W ∧
                    (∀ U ∈ V. openin (top_of_set C) U) ∧
                    pairwise disjnt V ∧
                    (∀ U ∈ V. ∃ q. homeomorphism U W p q))
    using cov by (force simp: covering_space_def)
   then have l y ∈ ⋃V
```

using ‹$X \subseteq C$› *pleq that* **by** *auto*
then obtain W' **where** $l\ y \in W'$ **and** $W' \in \mathcal{V}$
by *blast*
with WV **obtain** p' **where** *opeCW'*: *openin* (*top_of_set* C) W'
and *homUW'*: *homeomorphism* $W'\ W\ p\ p'$
by *blast*
then have *contp'*: *continuous_on* $W\ p'$ **and** *p'im*: $p'\ `\ W \subseteq W'$
using *homUW' homeomorphism_image2 homeomorphism_cont2* **by** *fastforce+*
obtain V **where** $y \in V\ y \in U$ **and** *fimW*: $f\ `\ V \subseteq W\ V \subseteq U$
and *path_connected* V **and** *opeUV*: *openin* (*top_of_set* U) V
proof −
have *openin* (*top_of_set* U) $(U \cap f -` W)$
using WV *contf continuous_on_open_gen fim* **by** *auto*
then obtain UO **where** *openin* (*top_of_set* U) $UO \wedge$ *path_connected* $UO \wedge\ y \in UO \wedge\ UO \subseteq U \cap f -` W$
using $U\ WV$ ‹$y \in U$› **unfolding** *locally_path_connected* **by** (*meson IntI vimage_eq*)
then show *?thesis*
by (*meson* ‹$y \in U$› *image_subset_iff_subset_vimage le_inf_iff that*)
qed
have $W' \subseteq C\ W \subseteq S$
using *opeCW' WV openin_imp_subset* **by** *auto*
have *p'im*: $p'\ `\ W \subseteq W'$
using *homUW' homeomorphism_image2* **by** *fastforce*
show *?thesis*
proof (*intro exI conjI*)
have *openin* (*top_of_set* S) $(W \cap p' -` (W' \cap X))$
proof (*rule openin_trans*)
show *openin* (*top_of_set* W) $(W \cap p' -` (W' \cap X))$
using X ‹$W' \subseteq C$›
by (*metis continuous_on_open contp' homUW' homeomorphism_image2 inf.assoc inf.orderE openin_open*)
show *openin* (*top_of_set* S) W
using WV **by** *blast*
qed
then show *openin* (*top_of_set* U) $(V \cap (U \cap (f -` (W \cap (p' -` (W' \cap X))))))$
by (*blast intro*: *opeUV openin_subtopology_self continuous_openin_preimage* [*OF contf fim*])
have $p'\ (f\ y) \in X$
using ‹$l\ y \in W'$› *homeomorphism_apply1* [*OF homUW'*] *pleq* ‹$y \in U$› ‹$l\ y \in X$› **by** *fastforce*
then show $y \in V \cap (U \cap f -` (W \cap p' -` (W' \cap X)))$
using ‹$y \in U$› ‹$y \in V$› WV *p'im* **by** *auto*
show $V \cap (U \cap f -` (W \cap p' -` (W' \cap X))) \subseteq U \cap l -` X$
proof (*intro subsetI IntI*; *clarify*)
fix y'
assume y': $y' \in V\ y' \in U\ f\ y' \in W\ p'\ (f\ y') \in W'\ p'\ (f\ y') \in X$

```
      then obtain γ where path γ path_image γ ⊆ V pathstart γ = y pathfinish
γ = y'
        by (meson ‹path_connected V› ‹y ∈ V› path_connected_def)
       obtain pp qq where pp: path pp path_image pp ⊆ U pathstart pp = z
pathfinish pp = y
                  and qq: path qq path_image qq ⊆ C pathstart qq = a
                  and pqqeq: ⋀t. t ∈ {0..1} ⟹ p(qq t) = f(pp t)
        using*[OF ‹y ∈ U›] by blast
      have finW: ⋀x. ⟦0 ≤ x; x ≤ 1⟧ ⟹ f (γ x) ∈ W
      using ‹path_image γ ⊆ V› by (auto simp: image_subset_iff path_image_def
finW [THEN subsetD])
        have pathfinish (qq +++ (p' ∘ f ∘ γ)) = l y'
        proof (rule l [of pp +++ γ y' qq +++ (p' ∘ f ∘ γ)])
          show path (pp +++ γ)
           by (simp add: ‹path γ› ‹path pp› ‹pathfinish pp = y› ‹pathstart γ = y›)
          show path_image (pp +++ γ) ⊆ U
                using ‹V ⊆ U› ‹path_image γ ⊆ V› ‹path_image pp ⊆ U›
not_in_path_image_join by blast
          show pathstart (pp +++ γ) = z
            by (simp add: ‹pathstart pp = z›)
          show pathfinish (pp +++ γ) = y'
            by (simp add: ‹pathfinish γ = y'›)
          have pathfinish qq = l y
            using ‹path pp› ‹path qq› ‹path_image pp ⊆ U› ‹path_image qq ⊆ C›
‹pathfinish pp = y› ‹pathstart pp = z› ‹pathstart qq = a› l pqqeq by blast
          also have ... = p' (f y)
             using ‹l y ∈ W'› homUW' homeomorphism_apply1 pleq that(2) by
fastforce
          finally have pathfinish qq = p' (f y) .
          then have paqq: pathfinish qq = pathstart (p' ∘ f ∘ γ)
            by (simp add: ‹pathstart γ = y› pathstart_compose)
          have continuous_on (path_image γ) (p' ∘ f)
          proof (rule continuous_on_compose)
            show continuous_on (path_image γ) f
               using ‹path_image γ ⊆ V› ‹V ⊆ U› contf continuous_on_subset
by blast
            show continuous_on (f ‘ path_image γ) p'
            proof (rule continuous_on_subset [OF contp'])
              show f ‘ path_image γ ⊆ W
               by (auto simp: path_image_def pathfinish_def pathstart_def finW)
            qed
          qed
          then show path (qq +++ (p' ∘ f ∘ γ))
            using ‹path γ› ‹path qq› paqq path_continuous_image path_join_imp
by blast
          show path_image (qq +++ (p' ∘ f ∘ γ)) ⊆ C
          proof (rule subset_path_image_join)
            show path_image qq ⊆ C
              by (simp add: ‹path_image qq ⊆ C›)
```

```
            show path_image (p' ∘ f ∘ γ) ⊆ C
             by (metis ‹W' ⊆ C› ‹path_image γ ⊆ V› dual_order.trans fimW(1)
image_comp image_mono p'im path_image_compose)
          qed
          show pathstart (qq +++ (p' ∘ f ∘ γ)) = a
            by (simp add: ‹pathstart qq = a›)
          show p ((qq +++ (p' ∘ f ∘ γ)) ξ) = f ((pp +++ γ) ξ) if ξ: ξ ∈ {0..1}
for ξ
          proof (simp add: joinpaths_def, safe)
            show p (qq (2*ξ)) = f (pp (2*ξ)) if ξ*2 ≤ 1
              using ‹ξ ∈ {0..1}› pqqeq that by auto
            show p (p' (f (γ (2*ξ − 1)))) = f (γ (2*ξ − 1)) if ¬ ξ*2 ≤ 1
              using that ξ by (auto intro: homeomorphism_apply2 [OF homUW'
finW])
          qed
        qed
        with ‹pathfinish γ = y'›  ‹p' (f y') ∈ X› show y' ∈ l −‘ X
          unfolding pathfinish_join by (simp add: pathfinish_def)
      qed
    qed
  qed
  then show continuous_on U l
    using vimage_eq openin_subopen continuous_on_open_gen [OF LC]
   by (metis IntD1 IntD2 vimage_eq openin_subopen continuous_on_open_gen
[OF LC])
 qed
qed

corollary covering_space_lift_stronger:
 fixes p :: 'a::real_normed_vector ⇒ 'b::real_normed_vector
   and f :: 'c::real_normed_vector ⇒ 'b
 assumes cov: covering_space C p S a ∈ C z ∈ U
     and U: path_connected U locally path_connected U
     and contf: continuous_on U f and fim: f ∈ U → S
     and feq: f z = p a
     and hom: ⋀r. ⟦path r; path_image r ⊆ U; pathstart r = z; pathfinish r = z⟧
                  ⟹ ∃ b. homotopic_paths S (f ∘ r) (linepath b b)
 obtains g where continuous_on U g g ∈ U → C g z = a ⋀y. y ∈ U ⟹ p(g
y) = f y
proof (rule covering_space_lift_general [OF cov U contf fim feq])
 fix r
 assume path r path_image r ⊆ U pathstart r = z pathfinish r = z
 then obtain b where b: homotopic_paths S (f ∘ r) (linepath b b)
   using hom by blast
 then have f (pathstart r) = b
  by (metis homotopic_paths_imp_pathstart pathstart_compose pathstart_linepath)
 then have homotopic_paths S (f ∘ r) (linepath (f z) (f z))
   by (simp add: b ‹pathstart r = z›)
 then have homotopic_paths S (f ∘ r) (p ∘ linepath a a)
```

```
    by (simp add: o_def feq linepath_def)
  then show ∃q. path q ∧
                path_image q ⊆ C ∧
                pathstart q = a ∧ pathfinish q = a ∧ homotopic_paths S (f ∘ r) (p
∘ q)
    by (force simp: ‹a ∈ C›)
qed auto

corollary covering_space_lift_strong:
  fixes p :: 'a::real_normed_vector ⇒ 'b::real_normed_vector
    and f :: 'c::real_normed_vector ⇒ 'b
  assumes cov: covering_space C p S a ∈ C z ∈ U
      and scU: simply_connected U and lpcU: locally path_connected U
      and contf: continuous_on U f and fim: f ∈ U → S
      and feq: f z = p a
  obtains g where continuous_on U g g ∈ U → C g z = a ⋀y. y ∈ U ⟹ p(g
y) = f y
proof (rule covering_space_lift_stronger [OF cov _ lpcU contf fim feq])
  show path_connected U
    using scU simply_connected_eq_contractible_loop_some by blast
  fix r
  assume r: path r path_image r ⊆ U pathstart r = z pathfinish r = z
  have linepath (f z) (f z) = f ∘ linepath z z
    by (simp add: o_def linepath_def)
  then have homotopic_paths S (f ∘ r) (linepath (f z) (f z))
   by (metis r contf fim homotopic_paths_continuous_image scU simply_connected_eq_contractible_path)
  then show ∃b. homotopic_paths S (f ∘ r) (linepath b b)
    by blast
qed blast

corollary covering_space_lift:
  fixes p :: 'a::real_normed_vector ⇒ 'b::real_normed_vector
    and f :: 'c::real_normed_vector ⇒ 'b
  assumes cov: covering_space C p S
      and U: simply_connected U  locally path_connected U
      and contf: continuous_on U f and fim: f ∈ U → S
   obtains g where continuous_on U g g ∈ U → C ⋀y. y ∈ U ⟹ p(g y) = f y
proof (cases U = {})
  case True
  with that show ?thesis by auto
next
  case False
  then obtain z where z ∈ U by blast
  then obtain a where a ∈ C f z = p a
    by (metis cov covering_space_imp_surjective fim image_iff Pi_iff)
  then show ?thesis
    by (metis that covering_space_lift_strong [OF cov _ ‹z ∈ U› U contf fim])
qed
```

6.16.6 Homeomorphisms of arc images

lemma *homeomorphism_arc*:
fixes *g* :: *real* ⇒ *'a*::*t2_space*
assumes *arc g*
obtains *h* **where** *homeomorphism* {*0..1*} (*path_image g*) *g h*
using *assms* **by** (*force simp*: *arc_def homeomorphism_compact path_def path_image_def*)

lemma *homeomorphic_arc_image_interval*:
fixes *g* :: *real* ⇒ *'a*::*t2_space* **and** *a*::*real*
assumes *arc g a* < *b*
shows (*path_image g*) *homeomorphic* {*a..b*}
proof −
have (*path_image g*) *homeomorphic* {*0..1*::*real*}
by (*meson assms*(*1*) *homeomorphic_def homeomorphic_sym homeomorphism_arc*)
also have ... *homeomorphic* {*a..b*}
using *assms* **by** (*force intro*: *homeomorphic_closed_intervals_real*)
finally show *?thesis* **.**
qed

lemma *homeomorphic_arc_images*:
fixes *g* :: *real* ⇒ *'a*::*t2_space* **and** *h* :: *real* ⇒ *'b*::*t2_space*
assumes *arc g arc h*
shows (*path_image g*) *homeomorphic* (*path_image h*)
proof −
have (*path_image g*) *homeomorphic* {*0..1*::*real*}
by (*meson assms homeomorphic_def homeomorphic_sym homeomorphism_arc*)
also have ... *homeomorphic* (*path_image h*)
by (*meson assms homeomorphic_def homeomorphism_arc*)
finally show *?thesis* **.**
qed

end

theory *Equivalence_Lebesgue_Henstock_Integration*
imports
Lebesgue_Measure
Henstock_Kurzweil_Integration
Complete_Measure
Set_Integral
Homeomorphism
Cartesian_Euclidean_Space
begin

lemma *LIMSEQ_if_less*: (λ*k*. *if i* < *k then a else b*) ⟶ *a*
by (*rule_tac k*=*Suc i* **in** *LIMSEQ_offset*) *auto*

Note that the rhs is an implication. This lemma plays a specific role in one proof.

lemma *le_left_mono*: $x \leq y \Longrightarrow y \leq a \longrightarrow x \leq (a::'a::preorder)$
by (*auto intro*: *order_trans*)

lemma *ball_trans*:
assumes $y \in ball\ z\ q\ r + q \leq s$ **shows** $ball\ y\ r \subseteq ball\ z\ s$
using *assms* **by** *metric*

lemma *has_integral_implies_lebesgue_measurable_cbox*:
fixes $f :: 'a :: euclidean\_space \Rightarrow real$
assumes f: $(f\ has\_integral\ I)\ (cbox\ x\ y)$
shows $f \in lebesgue\_on\ (cbox\ x\ y) \rightarrow_M borel$
proof (*rule cld_measure.borel_measurable_cld*)
let $?L = lebesgue\_on\ (cbox\ x\ y)$
let $?\mu = emeasure\ ?L$
let $?\mu' = outer\_measure\_of\ ?L$
interpret L: *finite_measure* $?L$
proof
show $?\mu\ (space\ ?L) \neq \infty$
by (*simp add*: *emeasure_restrict_space space_restrict_space emeasure_lborel_cbox_eq*)
qed

show *cld_measure* $?L$
proof
fix $B\ A$ **assume** $B \subseteq A\ A \in null\_sets\ ?L$
then show $B \in sets\ ?L$
using *null_sets_completion_subset*[*OF* ‹$B \subseteq A$›, *of lborel*]
by (*auto simp add*: *null_sets_restrict_space sets_restrict_space_iff intro*:)
next
fix A **assume** $A \subseteq space\ ?L\ \bigwedge B.\ B \in sets\ ?L \Longrightarrow ?\mu\ B < \infty \Longrightarrow A \cap B \in sets\ ?L$
from *this*(*1*) *this*(*2*)[*of space* $?L$] **show** $A \in sets\ ?L$
by (*auto simp*: *Int_absorb2 less_top*[*symmetric*])
qed *auto*
then interpret *cld_measure* $?L$
.

have *content_eq_L*: $A \in sets\ borel \Longrightarrow A \subseteq cbox\ x\ y \Longrightarrow content\ A = measure\ ?L\ A$ **for** A
by (*subst measure_restrict_space*) (*auto simp*: *measure_def*)

fix E **and** $a\ b :: real$ **assume** $E \in sets\ ?L\ a < b\ 0 < ?\mu\ E\ ?\mu\ E < \infty$
then obtain $M :: real$ **where** $?\mu\ E = M\ 0 < M$
by (*cases* $?\mu\ E$) *auto*
define e **where** $e = M\ /\ (4 + 2\ /\ (b - a))$
from ‹$a < b$› ‹$0 < M$› **have** $0 < e$
by (*auto intro*!: *divide_pos_pos simp*: *field_simps e_def*)

have $e < M\ /\ (3 + 2\ /\ (b - a))$
using ‹$a < b$› ‹$0 < M$›

```
    unfolding e_def by (intro divide_strict_left_mono add_strict_right_mono
mult_pos_pos) (auto simp: field_simps)
  then have 2 * e < (b - a) * (M - e * 3)
    using ‹0<M› ‹0 < e› ‹a < b› by (simp add: field_simps)

  have e_less_M: e < M / 1
    unfolding e_def using ‹a < b› ‹0<M› by (intro divide_strict_left_mono)
(auto simp: field_simps)

  obtain d
    where gauge d
      and integral_f: ∀ p. p tagged_division_of cbox x y ∧ d fine p ⟶
        norm ((∑ (x,k) ∈ p. content k *R f x) - I) < e
    using ‹0<e› f unfolding has_integral by auto

  define C where C X m = X ∩ {x. ball x (1/Suc m) ⊆ d x} for X m
  have incseq (C X) for X
    unfolding C_def [abs_def]
    by (intro monoI Collect_mono conj_mono imp_refl le_left_mono subset_ball
divide_left_mono Int_mono) auto

  { fix X assume X ⊆ space ?L and eq: ?μ' X = ?μ E
    have (SUP m. outer_measure_of ?L (C X m)) = outer_measure_of ?L (⋃ m.
C X m)
      using ‹X ⊆ space ?L› by (intro SUP_outer_measure_of_incseq ‹incseq (C
X)›) (auto simp: C_def)
    also have (⋃ m. C X m) = X
    proof -
      { fix x
        obtain e where 0 < e ball x e ⊆ d x
          using gaugeD[OF ‹gauge d›, of x] unfolding open_contains_ball by auto
        moreover
        obtain n where 1 / (1 + real n) < e
          using reals_Archimedean[OF ‹0<e›] by (auto simp: inverse_eq_divide)
        then have ball x (1 / (1 + real n)) ⊆ ball x e
          by (intro subset_ball) auto
        ultimately have ∃ n. ball x (1 / (1 + real n)) ⊆ d x
          by blast }
      then show ?thesis
        by (auto simp: C_def)
    qed
    finally have (SUP m. outer_measure_of ?L (C X m)) = ?μ E
      using eq by auto
    also have ... > M - e
      using ‹0 < M› ‹?μ E = M› ‹0<e› by (auto intro!: ennreal_lessI)
    finally have ∃ m. M - e < outer_measure_of ?L (C X m)
      unfolding less_SUP_iff by auto }
  note C = this
```

```
  let ?E = {x∈E. f x ≤ a} and ?F = {x∈E. b ≤ f x}

  have ¬ (?μ' ?E = ?μ E ∧ ?μ' ?F = ?μ E)
  proof
    assume eq: ?μ' ?E = ?μ E ∧ ?μ' ?F = ?μ E
   with C[of ?E] C[of ?F] ‹E ∈ sets ?L›[THEN sets.sets_into_space] obtain ma
mb
     where M − e < outer_measure_of ?L (C ?E ma) M − e < outer_measure_of
?L (C ?F mb)
      by auto
    moreover define m where m = max ma mb
    ultimately have M_minus_e: M − e < outer_measure_of ?L (C ?E m) M
− e < outer_measure_of ?L (C ?F m)
      using
        incseqD[OF ‹incseq (C ?E)›, of ma m, THEN outer_measure_of_mono]
        incseqD[OF ‹incseq (C ?F)›, of mb m, THEN outer_measure_of_mono]
      by (auto intro: less_le_trans)
    define d' where d' x = d x ∩ ball x (1 / (3 * Suc m)) for x
    have gauge d'
      unfolding d'_def by (intro gauge_Int ‹gauge d› gauge_ball) auto
    then obtain p where p: p tagged_division_of cbox x y d' fine p
      by (rule fine_division_exists)
    then have d fine p
      unfolding d'_def[abs_def] fine_def by auto

    define s where s = {(x::'a, k). k ∩ (C ?E m) ≠ {} ∧ k ∩ (C ?F m) ≠ {}}
    define T where T E k = (SOME x. x ∈ k ∩ C E m) for E k
    let ?A = (λ(x, k). (T ?E k, k)) ‘ (p ∩ s) ∪ (p − s)
    let ?B = (λ(x, k). (T ?F k, k)) ‘ (p ∩ s) ∪ (p − s)

    { fix X assume X_eq: X = ?E ∨ X = ?F
      let ?T = (λ(x, k). (T X k, k))
      let ?p = ?T ‘ (p ∩ s) ∪ (p − s)

      have in_s: (x, k) ∈ s ⟹ T X k ∈ k ∩ C X m for x k
        using someI_ex[of λx. x ∈ k ∩ C X m] X_eq unfolding ex_in_conv by
(auto simp: T_def s_def)

      { fix x k assume (x, k) ∈ p (x, k) ∈ s
        have k: k ⊆ ball x (1 / (3 * Suc m))
          using ‹d' fine p›[THEN fineD, OF ‹(x, k) ∈ p›] by (auto simp: d'_def)
        then have x ∈ ball (T X k) (1 / (3 * Suc m))
       using in_s[OF ‹(x, k) ∈ s›] by (auto simp: C_def subset_eq dist_commute)
        then have ball x (1 / (3 * Suc m)) ⊆ ball (T X k) (1 / Suc m)
          by (rule ball_trans) (auto simp: field_split_simps)
        with k in_s[OF ‹(x, k) ∈ s›] have k ⊆ d (T X k)
          by (auto simp: C_def) }
      then have d fine ?p
        using ‹d fine p› by (auto intro!: fineI)
```

```
    moreover
    have ?p tagged_division_of cbox x y
    proof (rule tagged_division_ofI)
      show finite ?p
        using p(1) by auto
    next
      fix z k assume *: (z, k) ∈ ?p
      then consider (z, k) ∈ p (z, k) ∉ s
        | x' where (x', k) ∈ p (x', k) ∈ s z = T X k
        by (auto simp: T_def)
      then have z ∈ k ∧ k ⊆ cbox x y ∧ (∃ a b. k = cbox a b)
        using p(1) by cases (auto dest: in_s)
      then show z ∈ k k ⊆ cbox x y ∃ a b. k = cbox a b
        by auto
    next
      fix z k z' k' assume (z, k) ∈ ?p (z', k') ∈ ?p (z, k) ≠ (z', k')
      with tagged_division_ofD(5)[OF p(1), of _ k _ k']
      show interior k ∩ interior k' = {}
        by (auto simp: T_def dest: in_s)
    next
      have {k. ∃ x. (x, k) ∈ ?p} = {k. ∃ x. (x, k) ∈ p}
        by (auto simp: T_def image_iff Bex_def)
      then show ⋃{k. ∃ x. (x, k) ∈ ?p} = cbox x y
        using p(1) by auto
    qed
    ultimately have I: norm ((∑(x,k) ∈ ?p. content k *R f x) − I) < e
      using integral_f by auto

    have (∑(x,k) ∈ ?p. content k *R f x) =
      (∑(x,k) ∈ ?T ‘ (p ∩ s). content k *R f x) + (∑(x,k) ∈ p − s. content k
*R f x)
      using p(1)[THEN tagged_division_ofD(1)]
      by (safe intro!: sum.union_inter_neutral) (auto simp: s_def T_def)
     also have (∑(x,k) ∈ ?T ‘ (p ∩ s). content k *R f x) = (∑(x,k) ∈ p ∩ s.
content k *R f (T X k))
    proof (subst sum.reindex_nontrivial, safe)
      fix x1 x2 k assume 1: (x1, k) ∈ p (x1, k) ∈ s and 2: (x2, k) ∈ p (x2, k)
∈ s
        and eq: content k *R f (T X k) ≠ 0
      with tagged_division_ofD(5)[OF p(1), of x1 k x2 k] tagged_division_ofD(4)[OF
p(1), of x1 k]
      show x1 = x2
        by (auto simp: content_eq_0_interior)
    qed (use p in ‹auto intro!: sum.cong›)
    finally have eq: (∑(x,k) ∈ ?p. content k *R f x) =
      (∑(x,k) ∈ p ∩ s. content k *R f (T X k)) + (∑(x,k) ∈ p − s. content k
*R f x) .

    have in_T: (x, k) ∈ s ⟹ T X k ∈ X for x k
```

```
      using in_s[of x k] by (auto simp: C_def)

    note I_eq_in_T }
  note parts = this

  have p_in_L: (x, k) ∈ p ⟹ k ∈ sets ?L for x k
  using tagged_division_ofD(3, 4)[OF p(1), of x k] by (auto simp: sets_restrict_space)

  have [simp]: finite p
    using tagged_division_ofD(1)[OF p(1)] .

  have (M − 3∗e) ∗ (b − a) ≤ (∑(x,k) ∈ p ∩ s. content k) ∗ (b − a)
  proof (intro mult_right_mono)
    have fin: ?μ (E ∩ ⋃{k∈snd‘p. k ∩ C X m = {}}) < ∞ for X
      using ‹?μ E < ∞› by (rule le_less_trans[rotated]) (auto intro!: emeasure_mono ‹E ∈ sets ?L›)
    have sets: (E ∩ ⋃{k∈snd‘p. k ∩ C X m = {}}) ∈ sets ?L for X
      using tagged_division_ofD(1)[OF p(1)] by (intro sets.Diff ‹E ∈ sets ?L› sets.finite_Union sets.Int) (auto intro: p_in_L)
    { fix X assume X ⊆ E M − e < ?μ' (C X m)
      have M − e ≤ ?μ' (C X m)
        by (rule less_imp_le) fact
      also have ... ≤ ?μ' (E − (E ∩ ⋃{k∈snd‘p. k ∩ C X m = {}}))
      proof (intro outer_measure_of_mono subsetI)
        fix v assume v ∈ C X m
        then have v ∈ cbox x y v ∈ E
          using ‹E ⊆ space ?L› ‹X ⊆ E› by (auto simp: space_restrict_space C_def)
        then obtain z k where (z, k) ∈ p v ∈ k
          using tagged_division_ofD(6)[OF p(1), symmetric] by auto
        then show v ∈ E − E ∩ (⋃{k∈snd‘p. k ∩ C X m = {}})
          using ‹v ∈ C X m› ‹v ∈ E› by auto
      qed
      also have ... = ?μ E − ?μ (E ∩ ⋃{k∈snd‘p. k ∩ C X m = {}})
        using ‹E ∈ sets ?L› fin[of X] sets[of X] by (auto intro!: emeasure_Diff)
      finally have ?μ (E ∩ ⋃{k∈snd‘p. k ∩ C X m = {}}) ≤ e
        using ‹0 < e› e_less_M
        by (cases ?μ (E ∩ ⋃{k∈snd‘p. k ∩ C X m = {}})) (auto simp add: ‹?μ E = M› ennreal_minus ennreal_le_iff2)
      note this }
    note upper_bound = this

    have ?μ (E ∩ ⋃(snd‘(p − s))) =
      ?μ ((E ∩ ⋃{k∈snd‘p. k ∩ C ?E m = {}}) ∪ (E ∩ ⋃{k∈snd‘p. k ∩ C ?F m = {}}))
      by (intro arg_cong[where f=?μ]) (auto simp: s_def image_def Bex_def)
      also have ... ≤ ?μ (E ∩ ⋃{k∈snd‘p. k ∩ C ?E m = {}}) + ?μ (E ∩ ⋃{k∈snd‘p. k ∩ C ?F m = {}})
      using sets[of ?E] sets[of ?F] M_minus_e by (intro emeasure_subadditive)
```

auto
also have ... ≤ *e* + *ennreal e*
using *upper_bound*[*of ?E*] *upper_bound*[*of ?F*] *M_minus_e* **by** (*intro add_mono*) *auto*
finally have *?μ E* − *2*∗*e* ≤ *?μ* (*E* − (*E* ∩ ⋃(*snd*‘(*p* − *s*))))
using ‹*0* < *e*› ‹*E* ∈ *sets ?L*› *tagged_division_ofD*(*1*)[*OF p*(*1*)]
by (*subst emeasure_Diff*)
(*auto simp*: *top_unique simp flip*: *ennreal_plus*
intro!: *sets.Int sets.finite_UN ennreal_mono_minus intro*: *p_in_L*)
also have ... ≤ *?μ* (⋃*x*∈*p* ∩ *s*. *snd x*)
proof (*safe intro*!: *emeasure_mono subsetI*)
fix *v* **assume** *v* ∈ *E* **and** *not*: *v* ∉ (⋃*x*∈*p* ∩ *s*. *snd x*)
then have *v* ∈ *cbox x y*
using ‹*E* ⊆ *space ?L*› **by** (*auto simp*: *space_restrict_space*)
then obtain *z k* **where** (*z*, *k*) ∈ *p v* ∈ *k*
using *tagged_division_ofD*(*6*)[*OF p*(*1*), *symmetric*] **by** *auto*
with *not* **show** *v* ∈ ⋃(*snd* ‘ (*p* − *s*))
by (*auto intro*!: *bexI*[*of* _ (*z*, *k*)] *elim*: *ballE*[*of* _ _ (*z*, *k*)])
qed (*auto intro*!: *sets.Int sets.finite_UN ennreal_mono_minus intro*: *p_in_L*)
also have ... = *measure ?L* (⋃*x*∈*p* ∩ *s*. *snd x*)
by (*auto intro*!: *emeasure_eq_ennreal_measure*)
finally have *M* − *2* ∗ *e* ≤ *measure ?L* (⋃*x*∈*p* ∩ *s*. *snd x*)
unfolding ‹*?μ E* = *M*› **using** ‹*0* < *e*› **by** (*simp add*: *ennreal_minus*)
also have *measure ?L* (⋃*x*∈*p* ∩ *s*. *snd x*) = *content* (⋃*x*∈*p* ∩ *s*. *snd x*)
using *tagged_division_ofD*(*1*,*3*,*4*) [*OF p*(*1*)]
by (*intro content_eq_L*[*symmetric*])
(*fastforce intro*!: *sets.finite_UN UN_least del*: *subsetI*)+
also have *content* (⋃*x*∈*p* ∩ *s*. *snd x*) ≤ (∑ *k*∈*p* ∩ *s*. *content* (*snd k*))
using *p*(*1*) **by** (*auto simp*: *emeasure_lborel_cbox_eq intro*!: *measure_subadditive_finite*
dest!: *p*(*1*)[*THEN tagged_division_ofD*(*4*)])
finally show *M* − *3* ∗ *e* ≤ (∑ (*x*, *y*)∈*p* ∩ *s*. *content y*)
using ‹*0* < *e*› **by** (*simp add*: *split_beta*)
qed (*use* ‹*a* < *b*› **in** *auto*)
also have ... = (∑ (*x*,*k*) ∈ *p* ∩ *s*. *content k* ∗ (*b* − *a*))
by (*simp add*: *sum_distrib_right split_beta*′)
also have ... ≤ (∑ (*x*,*k*) ∈ *p* ∩ *s*. *content k* ∗ (*f* (*T ?F k*) − *f* (*T ?E k*)))
using *parts*(*3*) **by** (*auto intro*!: *sum_mono mult_left_mono diff_mono*)
also have ... = (∑ (*x*,*k*) ∈ *p* ∩ *s*. *content k* ∗ *f* (*T ?F k*)) − (∑ (*x*,*k*) ∈ *p* ∩ *s*. *content k* ∗ *f* (*T ?E k*))
by (*auto intro*!: *sum.cong simp*: *field_simps sum_subtractf*[*symmetric*])
also have ... = (∑ (*x*,*k*) ∈ *?B*. *content k* ∗$_R$ *f x*) − (∑ (*x*,*k*) ∈ *?A*. *content k* ∗$_R$ *f x*)
by (*subst* (*1 2*) *parts*) *auto*
also have ... ≤ *norm* ((∑ (*x*,*k*) ∈ *?B*. *content k* ∗$_R$ *f x*) − (∑ (*x*,*k*) ∈ *?A*. *content k* ∗$_R$ *f x*))
by *auto*
also have ... ≤ *e* + *e*
using *parts*(*1*)[*of ?E*] *parts*(*1*)[*of ?F*] **by** (*intro norm_diff_triangle_le*[*of* _ *I*]) *auto*

```
    finally show False
      using ‹2 * e < (b - a) * (M - e * 3)› by (auto simp: field_simps)
  qed
  moreover have ?μ' ?E ≤ ?μ E ?μ' ?F ≤ ?μ E
   unfolding outer_measure_of_eq[OF ‹E ∈ sets ?L›, symmetric] by (auto intro!:
outer_measure_of_mono)
  ultimately show min (?μ' ?E) (?μ' ?F) < ?μ E
    unfolding min_less_iff_disj by (auto simp: less_le)
qed

lemma has_integral_implies_lebesgue_measurable_real:
  fixes f :: 'a :: euclidean_space ⇒ real
  assumes f: (f has_integral I) Ω
  shows (λx. f x * indicator Ω x) ∈ lebesgue →M borel
proof −
  define B :: nat ⇒ 'a set where B n = cbox (− real n *R One) (real n *R One)
for n
  show (λx. f x * indicator Ω x) ∈ lebesgue →M borel
  proof (rule measurable_piecewise_restrict)
    have (⋃n. box (− real n *R One) (real n *R One)) ⊆ ⋃(B ‘ UNIV)
      unfolding B_def by (intro UN_mono box_subset_cbox order_refl)
    then show countable (range B) space lebesgue ⊆ ⋃(B ‘ UNIV)
      by (auto simp: B_def UN_box_eq_UNIV)
  next
    fix Ω' assume Ω' ∈ range B
    then obtain n where Ω': Ω' = B n by auto
    then show Ω' ∩ space lebesgue ∈ sets lebesgue
      by (auto simp: B_def)

    have f integrable_on Ω
      using f by auto
    then have (λx. f x * indicator Ω x) integrable_on Ω
      by (auto simp: integrable_on_def cong: has_integral_cong)
    then have (λx. f x * indicator Ω x) integrable_on (Ω ∪ B n)
      by (rule integrable_on_superset) auto
    then have (λx. f x * indicator Ω x) integrable_on B n
      unfolding B_def by (rule integrable_on_subcbox) auto
    then show (λx. f x * indicator Ω x) ∈ lebesgue_on Ω' →M borel
    unfolding B_def Ω' by (auto intro: has_integral_implies_lebesgue_measurable_cbox
simp: integrable_on_def)
  qed
qed

lemma has_integral_implies_lebesgue_measurable:
  fixes f :: 'a :: euclidean_space ⇒ 'b :: euclidean_space
  assumes f: (f has_integral I) Ω
  shows (λx. indicator Ω x *R f x) ∈ lebesgue →M borel
proof (intro borel_measurable_euclidean_space[where 'c='b, THEN iffD2] ballI)
  fix i :: 'b assume i ∈ Basis
```

```
  have (λx. (f x • i) * indicator Ω x) ∈ borel_measurable (completion lborel)
    using has_integral_linear[OF f bounded_linear_inner_left, of i]
   by (intro has_integral_implies_lebesgue_measurable_real) (auto simp: comp_def)
  then show (λx. indicator Ω x *R f x • i) ∈ borel_measurable (completion lborel)
    by (simp add: ac_simps)
qed
```

6.16.7 Equivalence Lebesgue integral on *lborel* and HK-integral

```
lemma has_integral_measure_lborel:
  fixes A :: 'a::euclidean_space set
  assumes A[measurable]: A ∈ sets borel and finite: emeasure lborel A < ∞
  shows ((λx. 1) has_integral measure lborel A) A
proof −
  { fix l u :: 'a
    have ((λx. 1) has_integral measure lborel (box l u)) (box l u)
    proof cases
      assume ∀ b∈Basis. l • b ≤ u • b
      then show ?thesis
        using has_integral_const[of 1::real l u]
      by (simp flip: has_integral_restrict[OF box_subset_cbox] add: has_integral_spike_interior)
    next
      assume ¬ (∀ b∈Basis. l • b ≤ u • b)
      then have box l u = {}
        unfolding box_eq_empty by (auto simp: not_le intro: less_imp_le)
      then show ?thesis
        by simp
    qed }
  note has_integral_box = this

  { fix a b :: 'a let ?M = λA. measure lborel (A ∩ box a b)
    have Int_stable  (range (λ(a, b). box a b))
      by (auto simp: Int_stable_def box_Int_box)
    moreover have (range (λ(a, b). box a b)) ⊆ Pow UNIV
      by auto
    moreover have A ∈ sigma_sets UNIV (range (λ(a, b). box a b))
      using A unfolding borel_eq_box by simp
    ultimately have ((λx. 1) has_integral ?M A) (A ∩ box a b)
    proof (induction rule: sigma_sets_induct_disjoint)
      case (basic A) then show ?case
        by (auto simp: box_Int_box has_integral_box)
    next
      case empty then show ?case
        by simp
    next
      case (compl A)
      then have [measurable]: A ∈ sets borel
        by (simp add: borel_eq_box)
```

```
    have ((λx. 1) has_integral ?M (box a b)) (box a b)
      by (simp add: has_integral_box)
    moreover have ((λx. if x ∈ A ∩ box a b then 1 else 0) has_integral ?M A)
(box a b)
      by (subst has_integral_restrict) (auto intro: compl)
    ultimately have ((λx. 1 − (if x ∈ A ∩ box a b then 1 else 0)) has_integral
?M (box a b) − ?M A) (box a b)
      by (rule has_integral_diff)
    then have ((λx. (if x ∈ (UNIV − A) ∩ box a b then 1 else 0)) has_integral
?M (box a b) − ?M A) (box a b)
      by (rule has_integral_cong[THEN iffD1, rotated 1]) auto
    then have ((λx. 1) has_integral ?M (box a b) − ?M A) ((UNIV − A) ∩ box
a b)
      by (subst (asm) has_integral_restrict) auto
    also have ?M (box a b) − ?M A = ?M (UNIV − A)
      by (subst measure_Diff[symmetric]) (auto simp: emeasure_lborel_box_eq
Diff_Int_distrib2)
    finally show ?case .
  next
    case (union F)
    then have [measurable]: ⋀i. F i ∈ sets borel
      by (simp add: borel_eq_box subset_eq)
    have ((λx. if x ∈ ⋃(F ‘ UNIV) ∩ box a b then 1 else 0) has_integral ?M
(⋃ i. F i)) (box a b)
    proof (rule has_integral_monotone_convergence_increasing)
      let ?f = λk x. ∑ i<k. if x ∈ F i ∩ box a b then 1 else 0 :: real
      show ⋀k. (?f k has_integral (∑ i<k. ?M (F i))) (box a b)
        using union.IH by (auto intro!: has_integral_sum simp del: Int_iff)
      show ⋀k x. ?f k x ≤ ?f (Suc k) x
        by (intro sum_mono2) auto
      from union(1) have ∗: ⋀x i j. x ∈ F i ⟹ x ∈ F j ⟷ j = i
        by (auto simp add: disjoint_family_on_def)
      show (λk. ?f k x) ⟶ (if x ∈ ⋃(F ‘ UNIV) ∩ box a b then 1 else 0) for
x
        by (auto simp: ∗ sum.If_cases Iio_Int_singleton if_distrib LIMSEQ_if_less
cong: if_cong)
      have ∗: emeasure lborel ((⋃ x. F x) ∩ box a b) ≤ emeasure lborel (box a b)
        by (intro emeasure_mono) auto

      with union(1) show (λk. ∑ i<k. ?M (F i)) ⟶ ?M (⋃ i. F i)
        unfolding sums_def[symmetric] UN_extend_simps
        by (intro measure_UNION) (auto simp: disjoint_family_on_def emea-
sure_lborel_box_eq top_unique)
    qed
    then show ?case
      by (subst (asm) has_integral_restrict) auto
  qed }
 note ∗ = this
```

```
  show ?thesis
  proof (rule has_integral_monotone_convergence_increasing)
    let ?B = λn::nat. box (− real n *R One) (real n *R One) :: 'a set
    let ?f = λn::nat. λx. if x ∈ A ∩ ?B n then 1 else 0 :: real
    let ?M = λn. measure lborel (A ∩ ?B n)

    show ⋀n::nat. (?f n has_integral ?M n) A
      using ∗ by (subst has_integral_restrict) simp_all
    show ⋀k x. ?f k x ≤ ?f (Suc k) x
      by (auto simp: box_def)
    { fix x assume x ∈ A
    moreover have (λk. indicator (A ∩ ?B k) x :: real) ⟶ indicator (⋃k::nat.
A ∩ ?B k) x
        by (intro LIMSEQ_indicator_incseq) (auto simp: incseq_def box_def)
      ultimately show (λk. if x ∈ A ∩ ?B k then 1 else 0::real) ⟶ 1
        by (simp add: indicator_def of_bool_def UN_box_eq_UNIV) }

    have (λn. emeasure lborel (A ∩ ?B n)) ⟶ emeasure lborel (⋃n::nat. A ∩
?B n)
      by (intro Lim_emeasure_incseq) (auto simp: incseq_def box_def)
   also have (λn. emeasure lborel (A ∩ ?B n)) = (λn. measure lborel (A ∩ ?B n))
    proof (intro ext emeasure_eq_ennreal_measure)
      fix n have emeasure lborel (A ∩ ?B n) ≤ emeasure lborel (?B n)
        by (intro emeasure_mono) auto
      then show emeasure lborel (A ∩ ?B n) ≠ top
        by (auto simp: top_unique)
    qed
    finally show (λn. measure lborel (A ∩ ?B n)) ⟶ measure lborel A
      using emeasure_eq_ennreal_measure[of lborel A] finite
      by (simp add: UN_box_eq_UNIV less_top)
  qed
qed

lemma nn_integral_has_integral:
  fixes f::'a::euclidean_space ⇒ real
  assumes f: f ∈ borel_measurable borel ⋀x. 0 ≤ f x (∫+x. f x ∂lborel) = ennreal
r 0 ≤ r
  shows (f has_integral r) UNIV
using f proof (induct f arbitrary: r rule: borel_measurable_induct_real)
  case (set A)
  then have ((λx. 1) has_integral measure lborel A) A
    by (intro has_integral_measure_lborel) (auto simp: ennreal_indicator)
  with set show ?case
   by (simp add: ennreal_indicator measure_def) (simp add: indicator_def of_bool_def)
next
  case (mult g c)
  then have ennreal c ∗ (∫+ x. g x ∂lborel) = ennreal r
    by (subst nn_integral_cmult[symmetric]) (auto simp: ennreal_mult)
  with ‹0 ≤ r› ‹0 ≤ c›
```

obtain r' **where** $(c = 0 \wedge r = 0) \vee (0 \leq r' \wedge (\int^+ x.\ ennreal\ (g\ x)\ \partial lborel) = ennreal\ r' \wedge r = c * r')$
by $(cases \int^+ x.\ ennreal\ (g\ x)\ \partial lborel\ rule{:}\ ennreal\_cases)$
$(auto\ split{:}\ if\_split\_asm\ simp{:}\ ennreal\_mult\_top\ ennreal\_mult[symmetric])$
with *mult* **show** *?case*
by $(auto\ intro!{:}\ has\_integral\_cmult\_real)$
next
case $(add\ g\ h)$
then have $(\int^+ x.\ h\ x + g\ x\ \partial lborel) = (\int^+ x.\ h\ x\ \partial lborel) + (\int^+ x.\ g\ x\ \partial lborel)$
by $(simp\ add{:}\ nn\_integral\_add)$
with *add* **obtain** $a\ b$ **where** $0 \leq a\ 0 \leq b\ (\int^+ x.\ h\ x\ \partial lborel) = ennreal\ a\ (\int^+ x.\ g\ x\ \partial lborel) = ennreal\ b\ r = a + b$
by $(cases \int^+ x.\ h\ x\ \partial lborel \int^+ x.\ g\ x\ \partial lborel\ rule{:}\ ennreal2\_cases)$
$(auto\ simp{:}\ add\_top\ nn\_integral\_add\ top\_add\ simp\ flip{:}\ ennreal\_plus)$
with *add* **show** *?case*
by $(auto\ intro!{:}\ has\_integral\_add)$
next
case $(seq\ U)$
note $seq(1)[measurable]$ **and** $f[measurable]$

have $U\_le\_f{:}\ U\ i\ x \leq f\ x$ **for** $i\ x$
by $(metis\ (no\_types)\ LIMSEQ\_le\_const\ UNIV\_I\ incseq\_def\ le\_fun\_def\ seq.hyps(4)\ seq.hyps(5)\ space\_borel)$

{ fix i
have $(\int^+x.\ U\ i\ x\ \partial lborel) \leq (\int^+x.\ f\ x\ \partial lborel)$
using $seq(2)\ f(2)\ U\_le\_f$ **by** $(intro\ nn\_integral\_mono)\ simp$
then obtain p **where** $(\int^+x.\ U\ i\ x\ \partial lborel) = ennreal\ p\ p \leq r\ 0 \leq p$
using $seq(6)\ \langle 0 \leq r \rangle$ **by** $(cases \int^+x.\ U\ i\ x\ \partial lborel\ rule{:}\ ennreal\_cases)\ (auto\ simp{:}\ top\_unique)$
moreover note *seq*
ultimately have $\exists p.\ (\int^+x.\ U\ i\ x\ \partial lborel) = ennreal\ p \wedge 0 \leq p \wedge p \leq r \wedge (U\ i\ has\_integral\ p)\ UNIV$
by *auto* **}**
then obtain p **where** $p{:}\ \bigwedge i.\ (\int^+x.\ ennreal\ (U\ i\ x)\ \partial lborel) = ennreal\ (p\ i)$
and $bnd{:}\ \bigwedge i.\ p\ i \leq r\ \bigwedge i.\ 0 \leq p\ i$
and $U\_int{:}\ \bigwedge i.(U\ i\ has\_integral\ (p\ i))\ UNIV$ **by** *metis*

have $int\_eq{:}\ \bigwedge i.\ integral\ UNIV\ (U\ i) = p\ i$ **using** U_int **by** $(rule\ integral\_unique)$

have $*{:}\ f\ integrable\_on\ UNIV \wedge (\lambda k.\ integral\ UNIV\ (U\ k)) \longrightarrow integral\ UNIV\ f$
proof $(rule\ monotone\_convergence\_increasing)$
show $\bigwedge k.\ U\ k\ integrable\_on\ UNIV$ **using** U_int **by** *auto*
show $\bigwedge k\ x.\ x \in UNIV \Longrightarrow U\ k\ x \leq U\ (Suc\ k)\ x$ **using** $\langle incseq\ U \rangle$ **by** $(auto\ simp{:}\ incseq\_def\ le\_fun\_def)$
then show $bounded\ (range\ (\lambda k.\ integral\ UNIV\ (U\ k)))$
using $bnd\ int\_eq$ **by** $(auto\ simp{:}\ bounded\_real\ intro!{:}\ exI[of\ \_\ r])$
show $\bigwedge x.\ x \in UNIV \Longrightarrow (\lambda k.\ U\ k\ x) \longrightarrow f\ x$

```
    using seq by auto
  qed
  moreover have (λi. (∫⁺x. U i x ∂lborel)) ⟶ (∫⁺x. f x ∂lborel)
    using seq f(2) U_le_f by (intro nn_integral_dominated_convergence[where
w=f]) auto
  ultimately have integral UNIV f = r
    by (auto simp add: bnd int_eq p seq intro: LIMSEQ_unique)
  with ∗ show ?case
    by (simp add: has_integral_integral)
qed

lemma nn_integral_lborel_eq_integral:
  fixes f::'a::euclidean_space ⇒ real
  assumes f: f ∈ borel_measurable borel ⋀x. 0 ≤ f x (∫⁺x. f x ∂lborel) < ∞
  shows (∫⁺x. f x ∂lborel) = integral UNIV f
proof −
  from f(3) obtain r where r: (∫⁺x. f x ∂lborel) = ennreal r 0 ≤ r
    by (cases ∫⁺x. f x ∂lborel rule: ennreal_cases) auto
  then show ?thesis
    using nn_integral_has_integral[OF f(1,2) r] by (simp add: integral_unique)
qed

lemma nn_integral_integrable_on:
  fixes f::'a::euclidean_space ⇒ real
  assumes f: f ∈ borel_measurable borel ⋀x. 0 ≤ f x (∫⁺x. f x ∂lborel) < ∞
  shows f integrable_on UNIV
proof −
  from f(3) obtain r where r: (∫⁺x. f x ∂lborel) = ennreal r 0 ≤ r
    by (cases ∫⁺x. f x ∂lborel rule: ennreal_cases) auto
  then show ?thesis
   by (intro has_integral_integrable[where i=r] nn_integral_has_integral[where
r=r] f)
qed

lemma nn_integral_has_integral_lborel:
  fixes f :: 'a::euclidean_space ⇒ real
  assumes f_borel: f ∈ borel_measurable borel and nonneg: ⋀x. 0 ≤ f x
  assumes I: (f has_integral I) UNIV
  shows integral^N lborel f = I
proof −
  from f_borel have (λx. ennreal (f x)) ∈ borel_measurable lborel by auto
  from borel_measurable_implies_simple_function_sequence'[OF this]
  obtain F where F: ⋀i. simple_function lborel (F i) incseq F
                ⋀i x. F i x < top ⋀x. (SUP i. F i x) = ennreal (f x)
    by blast
  then have [measurable]: ⋀i. F i ∈ borel_measurable lborel
    by (metis borel_measurable_simple_function)
  let ?B = λi::nat. box (− (real i ∗R One)) (real i ∗R One) :: 'a set
```

```
  have 0 ≤ I
    using I by (rule has_integral_nonneg) (simp add: nonneg)

  have F_le_f: ennreal2 (F i x) ≤ f x for i x
    using F(3,4)[where x=x] nonneg SUP_upper[of i UNIV λi. F i x]
    by (cases F i x rule: ennreal_cases) auto
  let ?F = λi x. F i x * indicator (?B i) x
  have (∫⁺ x. ennreal (f x) ∂lborel) = (SUP i. integral^N lborel (λx. ?F i x))
  proof (subst nn_integral_monotone_convergence_SUP[symmetric])
    { fix x
      obtain j where j: x ∈ ?B j
        using UN_box_eq_UNIV by auto

      have ennreal (f x) = (SUP i. F i x)
        using F(4)[of x] nonneg[of x] by (simp add: max_def)
      also have ... = (SUP i. ?F i x)
      proof (rule SUP_eq)
        fix i show ∃j∈UNIV. F i x ≤ ?F j x
          using j F(2)
          by (intro bexI[of _ max i j])
             (auto split: split_max split_indicator simp: incseq_def le_fun_def
box_def)
      qed (auto intro!: F split: split_indicator)
      finally have ennreal (f x) = (SUP i. ?F i x) . }
    then show (∫⁺ x. ennreal (f x) ∂lborel) = (∫⁺ x. (SUP i. ?F i x) ∂lborel)
      by simp
  qed (insert F, auto simp: incseq_def le_fun_def box_def split: split_indicator)
  also have ... ≤ ennreal I
  proof (rule SUP_least)
    fix i :: nat
    have finite_F: (∫⁺ x. ennreal (enn2real (F i x) * indicator (?B i) x) ∂lborel)
< ∞
    proof (rule nn_integral_bound_simple_function)
      have emeasure lborel {x ∈ space lborel. ennreal (enn2real (F i x) * indicator
(?B i) x) ≠ 0} ≤
        emeasure lborel (?B i)
        by (intro emeasure_mono) (auto split: split_indicator)
      then show emeasure lborel {x ∈ space lborel. ennreal (enn2real (F i x) *
indicator (?B i) x) ≠ 0} < ∞
        by (auto simp: less_top[symmetric] top_unique)
    qed (auto split: split_indicator
         intro!: F simple_function_compose1[where g=enn2real] simple_function_ennreal)

    have int_F: (λx. enn2real (F i x) * indicator (?B i) x) integrable_on UNIV
      using F(4) finite_F
    by (intro nn_integral_integrable_on) (auto split: split_indicator simp: enn2real_nonneg)

    have (∫⁺ x. F i x * indicator (?B i) x ∂lborel) =
      (∫⁺ x. ennreal (enn2real (F i x) * indicator (?B i) x) ∂lborel)
```

```
    using F(3,4)
   by (intro nn_integral_cong) (auto simp: image_iff eq_commute split: split_indicator)
   also have ... = ennreal (integral UNIV (λx. enn2real (F i x) * indicator (?B
i) x))
    using F
    by (intro nn_integral_lborel_eq_integral[OF _ _ finite_F])
       (auto split: split_indicator intro: enn2real_nonneg)
   also have ... ≤ ennreal I
    by (auto intro!: has_integral_le[OF integrable_integral[OF int_F] I] nonneg
F_le_f
            simp: ‹0 ≤ I› split: split_indicator )
   finally show (∫⁺ x. F i x * indicator (?B i) x ∂lborel) ≤ ennreal I .
  qed
  finally have (∫⁺ x. ennreal (f x) ∂lborel) < ∞
   by (auto simp: less_top[symmetric] top_unique)
  from nn_integral_lborel_eq_integral[OF assms(1,2) this] I show ?thesis
   by (simp add: integral_unique)
qed

lemma has_integral_iff_emeasure_lborel:
  fixes A :: 'a::euclidean_space set
  assumes A[measurable]: A ∈ sets borel and [simp]: 0 ≤ r
  shows ((λx. 1) has_integral r) A ⟷ emeasure lborel A = ennreal r
proof (cases emeasure lborel A = ∞)
  case emeasure_A: True
  have ¬ (λx. 1::real) integrable_on A
  proof
    assume int: (λx. 1::real) integrable_on A
    then have (indicator A::'a ⇒ real) integrable_on UNIV
      unfolding indicator_def of_bool_def integrable_restrict_UNIV .
    then obtain r where ((indicator A::'a⇒real) has_integral r) UNIV
      by auto
    from nn_integral_has_integral_lborel[OF _ _ this] emeasure_A show False
      by (simp add: ennreal_indicator)
  qed
  with emeasure_A show ?thesis
    by auto
next
  case False
  then have ((λx. 1) has_integral measure lborel A) A
    by (simp add: has_integral_measure_lborel less_top)
  with False show ?thesis
    by (auto simp: emeasure_eq_ennreal_measure has_integral_unique)
qed

lemma ennreal_max_0: ennreal (max 0 x) = ennreal x
  by (auto simp: max_def ennreal_neg)

lemma has_integral_integral_real:
```

```
  fixes f::'a::euclidean_space ⇒ real
  assumes f: integrable lborel f
  shows (f has_integral (integral^L lborel f)) UNIV
proof −
  from integrableE[OF f] obtain r q
    where 0 ≤ r 0 ≤ q
      and r: (∫⁺ x. ennreal (max 0 (f x)) ∂lborel) = ennreal r
      and q: (∫⁺ x. ennreal (max 0 (− f x)) ∂lborel) = ennreal q
      and f: f ∈ borel_measurable lborel and eq: integral^L lborel f = r − q
    unfolding ennreal_max_0 by auto
  then have ((λx. max 0 (f x)) has_integral r) UNIV ((λx. max 0 (− f x))
has_integral q) UNIV
    using nn_integral_has_integral[OF _ _ r] nn_integral_has_integral[OF _ _
q] by auto
  note has_integral_diff[OF this]
  moreover have (λx. max 0 (f x) − max 0 (− f x)) = f
    by auto
  ultimately show ?thesis
    by (simp add: eq)
qed

lemma has_integral_AE:
  assumes ae: AE x in lborel. x ∈ Ω ⟶ f x = g x
  shows (f has_integral x) Ω = (g has_integral x) Ω
proof −
  from ae obtain N
    where N: N ∈ sets borel emeasure lborel N = 0 {x. ¬ (x ∈ Ω ⟶ f x = g x)}
⊆ N
    by (auto elim!: AE_E)
  then have not_N: AE x in lborel. x ∉ N
    by (simp add: AE_iff_measurable)
  show ?thesis
  proof (rule has_integral_spike_eq[symmetric])
    show ⋀x. x∈Ω − N ⟹ f x = g x using N(3) by auto
    show negligible N
      unfolding negligible_def
    proof (intro allI)
      fix a b :: 'a
      let ?F = λx::'a. if x ∈ cbox a b then indicator N x else 0 :: real
      have integrable lborel ?F = integrable lborel (λx::'a. 0::real)
        using not_N N(1) by (intro integrable_cong_AE) auto
      moreover have (LINT x|lborel. ?F x) = (LINT x::'a|lborel. 0::real)
        using not_N N(1) by (intro integral_cong_AE) auto
      ultimately have (?F has_integral 0) UNIV
        using has_integral_integral_real[of ?F] by simp
      then show (indicator N has_integral (0::real)) (cbox a b)
        unfolding has_integral_restrict_UNIV .
    qed
  qed
```

qed

lemma *nn_integral_has_integral_lebesgue*:
fixes $f :: 'a::euclidean\_space \Rightarrow real$
assumes *nonneg*: $\bigwedge x.\ x \in \Omega \Longrightarrow 0 \le f\ x$ **and** I: $(f\ has\_integral\ I)\ \Omega$
shows $integral^N\ lborel\ (\lambda x.\ indicator\ \Omega\ x * f\ x) = I$
proof −
from I **have** $(\lambda x.\ indicator\ \Omega\ x *_R f\ x) \in lebesgue \to_M borel$
by (*rule has_integral_implies_lebesgue_measurable*)
then obtain $f' :: 'a \Rightarrow real$
where [*measurable*]: $f' \in borel \to_M borel$ **and** *eq*: $AE\ x\ in\ lborel.\ indicator\ \Omega\ x * f\ x = f'\ x$
by (*auto dest*: *completion_ex_borel_measurable_real*)

from I **have** $((\lambda x.\ abs\ (indicator\ \Omega\ x * f\ x))\ has\_integral\ I)\ UNIV$
using *nonneg* **by** (*simp add*: *indicator_def of_bool_def if_distrib*[*of* $\lambda x.\ x * f\ y$ **for** y] *cong*: *if_cong*)
also have $((\lambda x.\ abs\ (indicator\ \Omega\ x * f\ x))\ has\_integral\ I)\ UNIV \longleftrightarrow ((\lambda x.\ abs\ (f'\ x))\ has\_integral\ I)\ UNIV$
using *eq* **by** (*intro has_integral_AE*) *auto*
finally have $integral^N\ lborel\ (\lambda x.\ abs\ (f'\ x)) = I$
by (*rule nn_integral_has_integral_lborel*[*rotated 2*]) *auto*
also have $integral^N\ lborel\ (\lambda x.\ abs\ (f'\ x)) = integral^N\ lborel\ (\lambda x.\ abs\ (indicator\ \Omega\ x * f\ x))$
using *eq* **by** (*intro nn_integral_cong_AE*) *auto*
also have $(\lambda x.\ abs\ (indicator\ \Omega\ x * f\ x)) = (\lambda x.\ indicator\ \Omega\ x * f\ x)$
using *nonneg* **by** (*auto simp*: *indicator_def fun_eq_iff*)
finally show *?thesis* **.**
qed

lemma *has_integral_iff_nn_integral_lebesgue*:
assumes f: $\bigwedge x.\ 0 \le f\ x$
shows $(f\ has\_integral\ r)\ UNIV \longleftrightarrow (f \in lebesgue \to_M borel \wedge integral^N\ lebesgue\ f = r \wedge 0 \le r)$ (**is** *?I* = *?N*)
proof
assume *?I*
have $0 \le r$
using *has_integral_nonneg*[*OF* ‹*?I*›] f **by** *auto*
then show *?N*
using *nn_integral_has_integral_lebesgue*[*OF* f ‹*?I*›]
has_integral_implies_lebesgue_measurable[*OF* ‹*?I*›]
by (*auto simp*: *nn_integral_completion*)
next
assume *?N*
then obtain f' **where** f': $f' \in borel \to_M borel\ AE\ x\ in\ lborel.\ f\ x = f'\ x$
by (*auto dest*: *completion_ex_borel_measurable_real*)
moreover have $(\int^+ x.\ ennreal\ |f'\ x|\ \partial lborel) = (\int^+ x.\ ennreal\ |f\ x|\ \partial lborel)$
using f' **by** (*intro nn_integral_cong_AE*) *auto*
moreover have $((\lambda x.\ |f'\ x|)\ has\_integral\ r)\ UNIV \longleftrightarrow ((\lambda x.\ |f\ x|)\ has\_integral$

```
r) UNIV
    using f' by (intro has_integral_AE) auto
  moreover note nn_integral_has_integral[of λx. |f' x| r] ‹?N›
  ultimately show ?I
    using f by (auto simp: nn_integral_completion)
qed

lemma set_nn_integral_lborel_eq_integral:
  fixes f::'a::euclidean_space ⇒ real
  assumes set_borel_measurable borel A f
  assumes ⋀x. x ∈ A ⟹ 0 ≤ f x (∫⁺x∈A. f x ∂lborel) < ∞
  shows (∫⁺x∈A. f x ∂lborel) = integral A f
proof −
  have eq: (∫⁺x∈A. f x ∂lborel) = (∫⁺x. ennreal (indicator A x * f x) ∂lborel)
    by (intro nn_integral_cong) (auto simp: indicator_def)
  also have ... = integral UNIV (λx. indicator A x * f x)
    using assms eq by (intro nn_integral_lborel_eq_integral)
                      (auto simp: indicator_def set_borel_measurable_def)
  also have integral UNIV (λx. indicator A x * f x) = integral A (λx. indicator A
x * f x)
    by (rule integral_spike_set) (auto intro: empty_imp_negligible)

  also have ... = integral A f
    by (rule integral_cong) (auto simp: indicator_def)
  finally show ?thesis .
qed

lemma nn_integral_has_integral_lebesgue':
  fixes f :: 'a::euclidean_space ⇒ real
  assumes nonneg: ⋀x. x ∈ Ω ⟹ 0 ≤ f x and I: (f has_integral I) Ω
  shows integral^N lborel (λx. ennreal (f x) * indicator Ω x) = ennreal I
proof −
  have integral^N lborel (λx. ennreal (f x) * indicator Ω x) =
        integral^N lborel (λx. ennreal (indicator Ω x * f x))
    by (intro nn_integral_cong) (auto simp: indicator_def)
  also have ... = ennreal I
    using assms by (intro nn_integral_has_integral_lebesgue)
  finally show ?thesis .
qed

context
  fixes f::'a::euclidean_space ⇒ 'b::euclidean_space
begin

lemma has_integral_integral_lborel:
  assumes f: integrable lborel f
  shows (f has_integral (integral^L lborel f)) UNIV
proof −
  have ((λx. ∑ b∈Basis. (f x • b) *_R b) has_integral (∑ b∈Basis. integral^L lborel
```

$(\lambda x.\ f\ x \cdot b) *_R b))$ *UNIV*
using *f* **by** (*intro has_integral_sum finite_Basis ballI has_integral_scaleR_left has_integral_integral_real*) *auto*
also have *eq_f*: $(\lambda x.\ \sum b{\in}Basis.\ (f\ x \cdot b) *_R b) = f$
by (*simp add*: *fun_eq_iff euclidean_representation*)
also have $(\sum b{\in}Basis.\ integral^L\ lborel\ (\lambda x.\ f\ x \cdot b) *_R b) = integral^L\ lborel\ f$
using *f* **by** (*subst* (*2*) *eq_f*[*symmetric*]) *simp*
finally show *?thesis* **.**
qed

lemma *integrable_on_lborel*: *integrable lborel f* $\Longrightarrow$ *f integrable_on UNIV*
using *has_integral_integral_lborel* **by** *auto*

lemma *integral_lborel*: *integrable lborel f* $\Longrightarrow$ *integral UNIV f* $= (\int x.\ f\ x\ \partial lborel)$
using *has_integral_integral_lborel* **by** *auto*

end

context
begin

private lemma *has_integral_integral_lebesgue_real*:
fixes *f* :: *'a*::*euclidean_space* $\Rightarrow$ *real*
assumes *f*: *integrable lebesgue f*
shows (*f has_integral* (*integral*^{L} *lebesgue f*)) *UNIV*
proof −
obtain *f'* **where** *f'*: $f' \in borel \to_M borel$ *AE x in lborel. f x* $=$ *f' x*
using *completion_ex_borel_measurable_real*[*OF borel_measurable_integrable*[*OF f*]] **by** *auto*
moreover have $(\int^+ x.\ ennreal\ (norm\ (f\ x))\ \partial lborel) = (\int^+ x.\ ennreal\ (norm\ (f'\ x))\ \partial lborel)$
using *f'* **by** (*intro nn_integral_cong_AE*) *auto*
ultimately have *integrable lborel f'*
using *f* **by** (*auto simp*: *integrable_iff_bounded nn_integral_completion cong*: *nn_integral_cong_AE*)
note *has_integral_integral_real*[*OF this*]
moreover have *integral*^{L} *lebesgue f* $=$ *integral*^{L} *lebesgue f'*
using *f' f* **by** (*intro integral_cong_AE*) (*auto intro*: *AE_completion measurable_completion*)
moreover have *integral*^{L} *lebesgue f'* $=$ *integral*^{L} *lborel f'*
using *f'* **by** (*simp add*: *integral_completion*)
moreover have (*f' has_integral integral*^{L} *lborel f'*) *UNIV* $\longleftrightarrow$ (*f has_integral integral*^{L} *lborel f'*) *UNIV*
using *f'* **by** (*intro has_integral_AE*) *auto*
ultimately show *?thesis*
by *auto*
qed

lemma *has_integral_integral_lebesgue*:

fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes f: *integrable lebesgue f*
shows (f *has_integral* (*integral*^{L} *lebesgue f*)) *UNIV*
proof −
have (($\lambda x.\ \sum b{\in}Basis.\ (f\ x \cdot b) *_R b$) *has_integral* ($\sum b{\in}Basis.$ *integral*^{L} *lebesgue* ($\lambda x.\ f\ x \cdot b) *_R b$)) *UNIV*
using *f* **by** (*intro has_integral_sum finite_Basis ballI has_integral_scaleR_left has_integral_integral_lebesgue_real*) *auto*
also have *eq_f*: ($\lambda x.\ \sum b{\in}Basis.\ (f\ x \cdot b) *_R b$) = *f*
by (*simp add*: *fun_eq_iff euclidean_representation*)
also have ($\sum b{\in}Basis.$ *integral*^{L} *lebesgue* ($\lambda x.\ f\ x \cdot b) *_R b$) = *integral*^{L} *lebesgue f*
using *f* **by** (*subst* (*2*) *eq_f*[*symmetric*]) *simp*
finally show *?thesis* **.**
qed

lemma *has_integral_integral_lebesgue_on*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes *integrable* (*lebesgue_on S*) *f S* ∈ *sets lebesgue*
shows (f *has_integral* (*integral*^{L} (*lebesgue_on S*) *f*)) *S*
proof −
let *?f* = $\lambda x.$ *if x* ∈ *S then f x else 0*
have *integrable lebesgue* ($\lambda x.$ *indicat_real S x* $*_R$ *f x*)
using *indicator_scaleR_eq_if* [*of S _ f*] *assms*
by (*metis* (*full_types*) *integrable_restrict_space sets.Int_space_eq2*)
then have *integrable lebesgue ?f*
using *indicator_scaleR_eq_if* [*of S _ f*] *assms* **by** *auto*
then have (*?f has_integral* (*integral*^{L} *lebesgue ?f*)) *UNIV*
by (*rule has_integral_integral_lebesgue*)
then have (f *has_integral* (*integral*^{L} *lebesgue ?f*)) *S*
using *has_integral_restrict_UNIV* **by** *blast*
moreover
have *S* ∩ *space lebesgue* ∈ *sets lebesgue*
by (*simp add*: *assms*)
then have (*integral*^{L} *lebesgue ?f*) = (*integral*^{L} (*lebesgue_on S*) *f*)
by (*simp add*: *integral_restrict_space indicator_scaleR_eq_if*)
ultimately show *?thesis*
by *auto*
qed

lemma *lebesgue_integral_eq_integral*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes *integrable* (*lebesgue_on S*) *f S* ∈ *sets lebesgue*
shows *integral*^{L} (*lebesgue_on S*) *f* = *integral S f*
by (*metis has_integral_integral_lebesgue_on assms integral_unique*)

lemma *integrable_on_lebesgue*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
shows *integrable lebesgue f* ⟹ *f integrable_on UNIV*

using *has_integral_integral_lebesgue* **by** *auto*

lemma *integral_lebesgue*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
shows *integrable lebesgue f* $\Longrightarrow$ *integral UNIV f* $= (\int x.\ f\ x\ \partial lebesgue)$
using *has_integral_integral_lebesgue* **by** *auto*

end

6.16.8 Absolute integrability (this is the same as Lebesgue integrability)

translations
LBINT x. f == CONST lebesgue_integral CONST lborel $(\lambda x.\ f)$

translations
LBINT x:A. f == CONST set_lebesgue_integral CONST lborel A $(\lambda x.\ f)$

lemma *set_integral_reflect*:
fixes S **and** $f :: real \Rightarrow 'a :: \{banach, second\_countable\_topology\}$
shows $(LBINT\ x : S.\ f\ x) = (LBINT\ x : \{x.\ -\ x \in S\}.\ f\ (-\ x))$
unfolding *set_lebesgue_integral_def*
by (*subst lborel_integral_real_affine*[**where** $c=-1$ **and** $t=0$])
(*auto intro*!: *Bochner_Integration.integral_cong split*: *split_indicator*)

lemma *borel_integrable_atLeastAtMost'*:
fixes $f :: real \Rightarrow 'a::\{banach, second\_countable\_topology\}$
assumes f: *continuous_on* $\{a..b\}$ f
shows *set_integrable lborel* $\{a..b\}$ f
unfolding *set_integrable_def*
by (*intro borel_integrable_compact compact_Icc f*)

lemma *integral_FTC_atLeastAtMost*:
fixes $f :: real \Rightarrow 'a :: euclidean\_space$
assumes $a \le b$
and F: $\bigwedge x.\ a \le x \Longrightarrow x \le b \Longrightarrow$ (*F has_vector_derivative f x*) (*at x within* $\{a\ ..\ b\}$)
and f: *continuous_on* $\{a\ ..\ b\}$ f
shows $integral^L$ *lborel* $(\lambda x.\ indicator\ \{a\ ..\ b\}\ x *_R f\ x) = F\ b - F\ a$
proof −
let *?f* $= \lambda x.\ indicator\ \{a\ ..\ b\}\ x *_R f\ x$
have (*?f has_integral* $(\int x.\ ?f\ x\ \partial lborel)$) *UNIV*
using *borel_integrable_atLeastAtMost'*[*OF f*]
unfolding *set_integrable_def* **by** (*rule has_integral_integral_lborel*)
moreover
have (*f has_integral F b − F a*) $\{a\ ..\ b\}$
by (*intro fundamental_theorem_of_calculus ballI assms*) *auto*
then have (*?f has_integral F b − F a*) $\{a\ ..\ b\}$
by (*subst has_integral_cong*[**where** $g=f$]) *auto*

then have (*?f has_integral F b − F a*) *UNIV*
by (*intro has_integral_on_superset*[**where** *T*=*UNIV* **and** *S*={*a..b*}]) *auto*
ultimately show $integral^L$ *lborel ?f* = *F b − F a*
by (*rule has_integral_unique*)
qed

lemma *set_borel_integral_eq_integral*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *set_integrable lborel S f*
shows *f integrable_on S LINT x* : *S* | *lborel*. *f x* = *integral S f*
proof −
let *?f* = *λx*. *indicator S x* ∗$_R$ *f x*
have (*?f has_integral LINT x* : *S* | *lborel*. *f x*) *UNIV*
using *assms has_integral_integral_lborel*
unfolding *set_integrable_def set_lebesgue_integral_def* **by** *blast*
hence *1*: (*f has_integral* (*set_lebesgue_integral lborel S f*)) *S*
by (*simp add*: *indicator_scaleR_eq_if*)
thus *f integrable_on S*
by (*auto simp add*: *integrable_on_def*)
with *1* **have** (*f has_integral* (*integral S f*)) *S*
by (*intro integrable_integral*, *auto simp add*: *integrable_on_def*)
thus *LINT x* : *S* | *lborel*. *f x* = *integral S f*
by (*intro has_integral_unique* [*OF 1*])
qed

lemma *has_integral_set_lebesgue*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *f*: *set_integrable lebesgue S f*
shows (*f has_integral* (*LINT x*:*S*|*lebesgue*. *f x*)) *S*
using *has_integral_integral_lebesgue f*
by (*fastforce simp add*: *set_integrable_def set_lebesgue_integral_def indicator_def*
of_bool_def if_distrib[*of λx*. *x* ∗$_R$ *f* _] *cong*: *if_cong*)

lemma *set_lebesgue_integral_eq_integral*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *f*: *set_integrable lebesgue S f*
shows *f integrable_on S LINT x*:*S* | *lebesgue*. *f x* = *integral S f*
using *has_integral_set_lebesgue*[*OF f*] **by** (*auto simp*: *integral_unique integrable_on_def*)

lemma *lmeasurable_iff_has_integral*:
S ∈ *lmeasurable* ⟷ ((*indicator S*) *has_integral measure lebesgue S*) *UNIV*
by (*subst has_integral_iff_nn_integral_lebesgue*)
(*auto simp*: *ennreal_indicator emeasure_eq_measure2 borel_measurable_indicator_iff intro*!: *fmeasurableI*)

abbreviation
absolutely_integrable_on :: (*'a*::*euclidean_space* ⇒ *'b*::{*banach*, *second_countable_topology*})
⇒ *'a set* ⇒ *bool*

(**infixr** *absolutely'_integrable'_on 46*)
where *f absolutely_integrable_on s ≡ set_integrable lebesgue s f*

lemma *absolutely_integrable_zero* [*simp*]: (*λx. 0*) *absolutely_integrable_on S*
by (*simp add*: *set_integrable_def*)

lemma *absolutely_integrable_on_def*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
shows *f absolutely_integrable_on S* ⟷ *f integrable_on S* ∧ (*λx. norm* (*f x*)) *integrable_on S*
proof *safe*
assume *f*: *f absolutely_integrable_on S*
then have *nf*: *integrable lebesgue* (*λx. norm* (*indicator S x* ∗$_R$ *f x*))
using *integrable_norm set_integrable_def* **by** *blast*
show *f integrable_on S*
by (*rule set_lebesgue_integral_eq_integral*[*OF f*])
have (*λx. norm* (*indicator S x* ∗$_R$ *f x*)) = (*λx. if x* ∈ *S then norm* (*f x*) *else 0*)
by *auto*
with *integrable_on_lebesgue*[*OF nf*] **show** (*λx. norm* (*f x*)) *integrable_on S*
by (*simp add*: *integrable_restrict_UNIV*)
next
assume *f*: *f integrable_on S* **and** *nf*: (*λx. norm* (*f x*)) *integrable_on S*
show *f absolutely_integrable_on S*
unfolding *set_integrable_def*
proof (*rule integrableI_bounded*)
show (*λx. indicator S x* ∗$_R$ *f x*) ∈ *borel_measurable lebesgue*
using *f has_integral_implies_lebesgue_measurable*[*of f _ S*] **by** (*auto simp*: *integrable_on_def*)
show ($\int^+$ *x. ennreal* (*norm* (*indicator S x* ∗$_R$ *f x*)) *∂lebesgue*) < ∞
using *nf nn_integral_has_integral_lebesgue*[*of _ λx. norm* (*f x*)]
by (*auto simp*: *integrable_on_def nn_integral_completion*)
qed
qed

lemma *integrable_on_lebesgue_on*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *f*: *integrable* (*lebesgue_on S*) *f* **and** *S*: *S* ∈ *sets lebesgue*
shows *f integrable_on S*
proof −
have *integrable lebesgue* (*λx. indicator S x* ∗$_R$ *f x*)
using *S f inf_top.comm_neutral integrable_restrict_space* **by** *blast*
then show *?thesis*
using *absolutely_integrable_on_def set_integrable_def* **by** *blast*
qed

lemma *absolutely_integrable_imp_integrable*:
assumes *f absolutely_integrable_on S S* ∈ *sets lebesgue*
shows *integrable* (*lebesgue_on S*) *f*

by (*meson assms integrable_restrict_space set_integrable_def sets.Int sets.top*)

lemma *absolutely_integrable_on_null* [*intro*]:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
shows *content* (*cbox a b*) = *0* ⟹ *f absolutely_integrable_on* (*cbox a b*)
by (*auto simp*: *absolutely_integrable_on_def*)

lemma *absolutely_integrable_on_open_interval*:
fixes *f* :: *'a* :: *euclidean_space* ⇒ *'b* :: *euclidean_space*
shows *f absolutely_integrable_on box a b* ⟷
f absolutely_integrable_on cbox a b
by (*auto simp*: *integrable_on_open_interval absolutely_integrable_on_def*)

lemma *absolutely_integrable_restrict_UNIV*:
(λ*x*. *if x* ∈ *S then f x else 0*) *absolutely_integrable_on UNIV* ⟷ *f absolutely_integrable_on S*
unfolding *set_integrable_def*
by (*intro arg_cong2*[**where** *f*=*integrable*]) *auto*

lemma *absolutely_integrable_onI*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
shows *f integrable_on S* ⟹ (λ*x*. *norm* (*f x*)) *integrable_on S* ⟹ *f absolutely_integrable_on S*
unfolding *absolutely_integrable_on_def* **by** *auto*

lemma *nonnegative_absolutely_integrable_1*:
fixes *f* :: *'a* :: *euclidean_space* ⇒ *real*
assumes *f*: *f integrable_on A* **and** ⋀*x*. *x* ∈ *A* ⟹ *0* ≤ *f x*
shows *f absolutely_integrable_on A*
by (*rule absolutely_integrable_onI* [*OF f*]) (*use assms* **in** ‹*simp add*: *integrable_eq*›)

lemma *absolutely_integrable_on_iff_nonneg*:
fixes *f* :: *'a* :: *euclidean_space* ⇒ *real*
assumes ⋀*x*. *x* ∈ *S* ⟹ *0* ≤ *f x* **shows** *f absolutely_integrable_on S* ⟷ *f integrable_on S*
proof −
{ **assume** *f integrable_on S*
then have (λ*x*. *if x* ∈ *S then f x else 0*) *integrable_on UNIV*
by (*simp add*: *integrable_restrict_UNIV*)
then have (λ*x*. *if x* ∈ *S then f x else 0*) *absolutely_integrable_on UNIV*
using ‹*f integrable_on S*› *absolutely_integrable_restrict_UNIV assms nonnegative_absolutely_integrable_1* **by** *blast*
then have *f absolutely_integrable_on S*
using *absolutely_integrable_restrict_UNIV* **by** *blast*
}
then show *?thesis*
unfolding *absolutely_integrable_on_def* **by** *auto*
qed

lemma *absolutely_integrable_on_scaleR_iff*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
shows
$(\lambda x.\ c *_R f\ x)$ *absolutely_integrable_on* $S \longleftrightarrow$
$c = 0 \vee f$ *absolutely_integrable_on* S
proof (*cases* $c=0$)
case *False*
then show *?thesis*
unfolding *absolutely_integrable_on_def*
by (*simp add*: *norm_mult*)
qed *auto*

lemma *absolutely_integrable_spike*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes f *absolutely_integrable_on* T **and** S: *negligible* S $\bigwedge x.\ x \in T - S \Longrightarrow g\ x = f\ x$
shows g *absolutely_integrable_on* T
using *assms integrable_spike*
unfolding *absolutely_integrable_on_def* **by** *metis*

lemma *absolutely_integrable_negligible*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes *negligible* S
shows f *absolutely_integrable_on* S
using *assms* **by** (*simp add*: *absolutely_integrable_on_def integrable_negligible*)

lemma *absolutely_integrable_spike_eq*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes *negligible* S $\bigwedge x.\ x \in T - S \Longrightarrow g\ x = f\ x$
shows (f *absolutely_integrable_on* $T \longleftrightarrow g$ *absolutely_integrable_on* T)
using *assms* **by** (*blast intro*: *absolutely_integrable_spike sym*)

lemma *absolutely_integrable_spike_set_eq*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes *negligible* $\{x \in S - T.\ f\ x \neq 0\}$ *negligible* $\{x \in T - S.\ f\ x \neq 0\}$
shows (f *absolutely_integrable_on* $S \longleftrightarrow f$ *absolutely_integrable_on* T)
proof −
have $(\lambda x.$ *if* $x \in S$ *then* $f\ x$ *else* $0)$ *absolutely_integrable_on UNIV* $\longleftrightarrow$
$(\lambda x.$ *if* $x \in T$ *then* $f\ x$ *else* $0)$ *absolutely_integrable_on UNIV*
proof (*rule absolutely_integrable_spike_eq*)
show *negligible* $(\{x \in S - T.\ f\ x \neq 0\} \cup \{x \in T - S.\ f\ x \neq 0\})$
by (*rule negligible_Un* [*OF assms*])
qed *auto*
with *absolutely_integrable_restrict_UNIV* **show** *?thesis*
by *blast*
qed

lemma *absolutely_integrable_spike_set*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$

assumes *f*: *f absolutely_integrable_on S* **and** *neg*: *negligible* {*x* ∈ *S* − *T*. *f x* ≠ *0*} *negligible* {*x* ∈ *T* − *S*. *f x* ≠ *0*}
shows *f absolutely_integrable_on T*
using *absolutely_integrable_spike_set_eq f neg* **by** *blast*

lemma *absolutely_integrable_reflect*[*simp*]:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
shows (λ*x*. *f*(−*x*)) *absolutely_integrable_on cbox* (−*b*) (−*a*) ⟷ *f absolutely_integrable_on cbox a b*
unfolding *absolutely_integrable_on_def*
by (*metis* (*mono_tags*, *lifting*) *integrable_eq integrable_reflect*)

lemma *absolutely_integrable_reflect_real*[*simp*]:
fixes *f* :: *real* ⇒ *'b*::*euclidean_space*
shows (λ*x*. *f*(−*x*)) *absolutely_integrable_on* {−*b* .. −*a*} ⟷ *f absolutely_integrable_on* {*a*..*b*::*real*}
unfolding *box_real*[*symmetric*] **by** (*rule absolutely_integrable_reflect*)

lemma *absolutely_integrable_on_subcbox*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
shows ⟦*f absolutely_integrable_on S*; *cbox a b* ⊆ *S*⟧ ⟹ *f absolutely_integrable_on cbox a b*
by (*meson absolutely_integrable_on_def integrable_on_subcbox*)

lemma *absolutely_integrable_on_subinterval*:
fixes *f* :: *real* ⇒ *'b*::*euclidean_space*
shows ⟦*f absolutely_integrable_on S*; {*a*..*b*} ⊆ *S*⟧ ⟹ *f absolutely_integrable_on* {*a*..*b*}
using *absolutely_integrable_on_subcbox* **by** *fastforce*

lemma *integrable_subinterval*:
fixes *f* :: *real* ⇒ *'a*::*euclidean_space*
assumes *integrable* (*lebesgue_on* {*a*..*b*}) *f*
and {*c*..*d*} ⊆ {*a*..*b*}
shows *integrable* (*lebesgue_on* {*c*..*d*}) *f*
proof (*rule absolutely_integrable_imp_integrable*)
show *f absolutely_integrable_on* {*c*..*d*}
proof −
have *f integrable_on* {*c*..*d*}
using *assms integrable_on_lebesgue_on integrable_on_subinterval* **by** *fastforce*
moreover have (λ*x*. *norm* (*f x*)) *integrable_on* {*c*..*d*}
proof (*rule integrable_on_subinterval*)
show (λ*x*. *norm* (*f x*)) *integrable_on* {*a*..*b*}
by (*simp add*: *assms integrable_on_lebesgue_on*)
qed (*use assms* **in** *auto*)
ultimately show *?thesis*
by (*auto simp*: *absolutely_integrable_on_def*)
qed

qed *auto*

lemma *indefinite_integral_continuous_real*:
fixes f :: *real* ⇒ *'a*::*euclidean_space*
assumes *integrable* (*lebesgue_on* {*a*..*b*}) f
shows *continuous_on* {*a*..*b*} (λ*x*. *integral*^{L} (*lebesgue_on* {*a*..*x*}) f)
proof −
have f *integrable_on* {*a*..*b*}
by (*simp add*: *assms integrable_on_lebesgue_on*)
then have *continuous_on* {*a*..*b*} (λ*x*. *integral* {*a*..*x*} f)
using *indefinite_integral_continuous_1* **by** *blast*
moreover have *integral*^{L} (*lebesgue_on* {*a*..*x*}) f = *integral* {*a*..*x*} f **if** $a \leq x$ $x \leq b$ **for** x
proof −
have {*a*..*x*} ⊆ {*a*..*b*}
using *that* **by** *auto*
then have *integrable* (*lebesgue_on* {*a*..*x*}) f
using *integrable_subinterval assms* **by** *blast*
then show *integral*^{L} (*lebesgue_on* {*a*..*x*}) f = *integral* {*a*..*x*} f
by (*simp add*: *lebesgue_integral_eq_integral*)
qed
ultimately show *?thesis*
by (*metis* (*no_types*, *lifting*) *atLeastAtMost_iff continuous_on_cong*)
qed

lemma *lmeasurable_iff_integrable_on*: $S \in$ *lmeasurable* ⟷ (λ*x*. *1*::*real*) *integrable_on* S
by (*subst absolutely_integrable_on_iff_nonneg*[*symmetric*])
(*simp_all add*: *lmeasurable_iff_integrable set_integrable_def*)

lemma *lmeasure_integral_UNIV*: $S \in$ *lmeasurable* ⟹ *measure lebesgue* S = *integral UNIV* (*indicator* S)
by (*simp add*: *lmeasurable_iff_has_integral integral_unique*)

lemma *lmeasure_integral*: $S \in$ *lmeasurable* ⟹ *measure lebesgue* S = *integral* S (λ*x*. *1*::*real*)
by (*fastforce simp add*: *lmeasure_integral_UNIV indicator_def* [*abs_def*] *of_bool_def lmeasurable_iff_integrable_on*)

lemma *integrable_on_const*: $S \in$ *lmeasurable* ⟹ (λ*x*. *c*) *integrable_on* S
unfolding *lmeasurable_iff_integrable*
by (*metis* (*mono_tags*, *lifting*) *integrable_eq integrable_on_scaleR_left lmeasurable_iff_integrable lmeasurable_iff_integrable_on scaleR_one*)

lemma *integral_indicator*:
assumes ($S \cap T$) ∈ *lmeasurable*
shows *integral* T (*indicator* S) = *measure lebesgue* ($S \cap T$)
proof −
have *integral UNIV* (*indicator* ($S \cap T$)) = *integral UNIV* (λ*a*. *if* $a \in S \cap T$

then 1::*real else 0*)
by (*simp add*: *indicator_def* [*abs_def*] *of_bool_def*)
moreover have (*indicator* (*S* ∩ *T*) *has_integral measure lebesgue* (*S* ∩ *T*)) *UNIV*
using *assms* **by** (*simp add*: *lmeasurable_iff_has_integral*)
ultimately have *integral UNIV* (λ*x*. *if x* ∈ *S* ∩ *T then 1 else 0*) = *measure lebesgue* (*S* ∩ *T*)
by (*metis* (*no_types*) *integral_unique*)
moreover have *integral T* (λ*a*. *if a* ∈ *S then 1*::*real else 0*) = *integral* (*S* ∩ *T* ∩ *UNIV*) (λ*a*. *1*)
by (*simp add*: *Henstock_Kurzweil_Integration.integral_restrict_Int*)
moreover have *integral T* (*indicat_real S*) = *integral T* (λ*a*. *if a* ∈ *S then 1 else 0*)
by (*simp add*: *indicator_def* [*abs_def*] *of_bool_def*)
ultimately show *?thesis*
by (*simp add*: *assms lmeasure_integral*)
qed

lemma *measurable_integrable*:
fixes *S* :: *'a*::*euclidean_space set*
shows *S* ∈ *lmeasurable* ⟷ (*indicat_real S*) *integrable_on UNIV*
by (*auto simp*: *lmeasurable_iff_integrable absolutely_integrable_on_iff_nonneg* [*symmetric*] *set_integrable_def*)

lemma *integrable_on_indicator*:
fixes *S* :: *'a*::*euclidean_space set*
shows *indicat_real S integrable_on T* ⟷ (*S* ∩ *T*) ∈ *lmeasurable*
unfolding *measurable_integrable*
unfolding *integrable_restrict_UNIV* [*of T*, *symmetric*]
by (*fastforce simp add*: *indicator_def elim*: *integrable_eq*)

lemma
assumes *D*: *D division_of S*
shows *lmeasurable_division*: *S* ∈ *lmeasurable* (**is** *?l*)
and *content_division*: (∑ *k*∈*D*. *measure lebesgue k*) = *measure lebesgue S* (**is** *?m*)
proof −
{ fix *d1 d2* **assume** ∗: *d1* ∈ *D d2* ∈ *D d1* ≠ *d2*
then obtain *a b c d* **where** *d1* = *cbox a b d2* = *cbox c d*
using *division_ofD*(*4*)[*OF D*] **by** *blast*
with *division_ofD*(*5*)[*OF D* ∗]
have *d1* ∈ *sets lborel d2* ∈ *sets lborel d1* ∩ *d2* ⊆ (*cbox a b* − *box a b*) ∪ (*cbox c d* − *box c d*)
by *auto*
moreover have (*cbox a b* − *box a b*) ∪ (*cbox c d* − *box c d*) ∈ *null_sets lborel*
by (*intro null_sets.Un null_sets_cbox_Diff_box*)
ultimately have *d1* ∩ *d2* ∈ *null_sets lborel*
by (*blast intro*: *null_sets_subset*) **}**
then show *?l ?m*

unfolding *division_ofD*(6)[*OF* $\mathcal{D}$, *symmetric*]
using *division_ofD*(1,4)[*OF* $\mathcal{D}$]
by (*auto intro*!: *measure_Union_AE*[*symmetric*] *simp*: *completion.AE_iff_null_sets Int_def*[*symmetric*] *pairwise_def null_sets_def*)
qed

lemma *has_measure_limit*:
assumes $S \in$ *lmeasurable* $e > 0$
obtains B **where** $B > 0$
$\bigwedge a\ b.$ *ball* 0 $B \subseteq$ *cbox* $a\ b \Longrightarrow |$*measure lebesgue* $(S \cap$ *cbox* $a\ b) -$ *measure lebesgue* $S| < e$
using *assms* **unfolding** *lmeasurable_iff_has_integral has_integral_alt'*
by (*force simp*: *integral_indicator integrable_on_indicator*)

lemma *lmeasurable_iff_indicator_has_integral*:
fixes $S :: 'a{::}euclidean\_space\ set$
shows $S \in$ *lmeasurable* $\wedge\ m =$ *measure lebesgue* $S \longleftrightarrow$ (*indicat_real* S *has_integral* m) *UNIV*
by (*metis has_integral_iff lmeasurable_iff_has_integral measurable_integrable*)

lemma *has_measure_limit_iff*:
fixes $f :: 'n{::}euclidean\_space \Rightarrow 'a{::}banach$
shows $S \in$ *lmeasurable* $\wedge\ m =$ *measure lebesgue* $S \longleftrightarrow$
$(\forall e{>}0.\ \exists B{>}0.\ \forall a\ b.$ *ball* 0 $B \subseteq$ *cbox* $a\ b \longrightarrow$
$(S \cap$ *cbox* $a\ b) \in$ *lmeasurable* $\wedge\ |$*measure lebesgue* $(S \cap$ *cbox* $a\ b) - m| < e)$ (**is** *?lhs* = *?rhs*)
proof
assume *?lhs* **then show** *?rhs*
by (*meson has_measure_limit fmeasurable.Int lmeasurable_cbox*)
next
assume *RHS* [*rule_format*]: *?rhs*
then show *?lhs*
apply (*simp add*: *lmeasurable_iff_indicator_has_integral has_integral'* [**where** $i{=}m$])
by (*metis* (*full_types*) *integral_indicator integrable_integral integrable_on_indicator*)
qed

6.16.9 Applications to Negligibility

lemma *negligible_iff_null_sets*: *negligible* $S \longleftrightarrow S \in$ *null_sets lebesgue*
proof
assume *negligible* S
then have (*indicator* S *has_integral* (0::*real*)) *UNIV*
by (*auto simp*: *negligible*)
then show $S \in$ *null_sets lebesgue*
by (*subst* (*asm*) *has_integral_iff_nn_integral_lebesgue*)
(*auto simp*: *borel_measurable_indicator_iff nn_integral_0_iff_AE AE_iff_null_sets indicator_eq_0_iff*)
next

```
  assume S: S ∈ null_sets lebesgue
  show negligible S
    unfolding negligible_def
  proof (safe intro!: has_integral_iff_nn_integral_lebesgue[THEN iffD2]
                      has_integral_restrict_UNIV[where s=cbox _ _, THEN iffD1])
    fix a b
    show (λx. if x ∈ cbox a b then indicator S x else 0) ∈ lebesgue →M borel
      using S by (auto intro!: measurable_If)
   then show (∫+ x. ennreal (if x ∈ cbox a b then indicator S x else 0) ∂lebesgue)
= ennreal 0
      using S[THEN AE_not_in] by (auto intro!: nn_integral_0_iff_AE[THEN
iffD2])
  qed auto
qed

corollary eventually_ae_filter_negligible:
  eventually P (ae_filter lebesgue) ⟷ (∃N. negligible N ∧ {x. ¬ P x} ⊆ N)
 by (auto simp: completion.AE_iff_null_sets negligible_iff_null_sets null_sets_completion_subset)

lemma starlike_negligible:
  assumes closed S
     and eq1: ⋀c x. (a + c *R x) ∈ S ⟹ 0 ≤ c ⟹ a + x ∈ S ⟹ c = 1
    shows negligible S
proof −
  have negligible ((+) (−a) ` S)
  proof (subst negligible_on_intervals, intro allI)
    fix u v
    show negligible ((+) (− a) ` S ∩ cbox u v)
       using ‹closed S› eq1 by (auto simp add: negligible_iff_null_sets alge-
bra_simps
        intro!: closed_translation_subtract starlike_negligible_compact cong: im-
age_cong_simp)
      (metis add_diff_eq diff_add_cancel scale_right_diff_distrib)
  qed
  then show ?thesis
    by (rule negligible_translation_rev)
qed

lemma starlike_negligible_strong:
  assumes closed S
     and star: ⋀c x. ⟦0 ≤ c; c < 1; a+x ∈ S⟧ ⟹ a + c *R x ∉ S
    shows negligible S
proof −
  show ?thesis
  proof (rule starlike_negligible [OF ‹closed S›, of a])
    fix c x
    assume cx: a + c *R x ∈ S 0 ≤ c a + x ∈ S
    with star have ¬ (c < 1) by auto
    moreover have ¬ (c > 1)
```

```
    using star [of 1/c c *R x] cx by force
   ultimately show c = 1 by arith
  qed
qed

lemma negligible_hyperplane:
  assumes a ≠ 0 ∨ b ≠ 0 shows negligible {x. a · x = b}
proof −
  obtain x where x: a · x ≠ b
    using assms by (metis euclidean_all_zero_iff inner_zero_right)
  moreover have c = 1 if a · (x + c *R w) = b a · (x + w) = b for c w
    using that
    by (metis (no_types, opaque_lifting) add_diff_eq diff_0 diff_minus_eq_add
inner_diff_right inner_scaleR_right mult_cancel_right2 right_minus_eq x)
  ultimately
  show ?thesis
    using starlike_negligible [OF closed_hyperplane, of x] by simp
qed

lemma negligible_lowdim:
  fixes S :: 'N :: euclidean_space set
  assumes dim S < DIM('N)
    shows negligible S
proof −
  obtain a where a ≠ 0 and a: span S ⊆ {x. a · x = 0}
    using lowdim_subset_hyperplane [OF assms] by blast
  have negligible (span S)
    using ‹a ≠ 0› a negligible_hyperplane by (blast intro: negligible_subset)
  then show ?thesis
    using span_base by (blast intro: negligible_subset)
qed

proposition negligible_convex_frontier:
  fixes S :: 'N :: euclidean_space set
  assumes convex S
    shows negligible(frontier S)
proof −
  have nf: negligible(frontier S) if convex S 0 ∈ S for S :: 'N set
  proof −
    obtain B where B ⊆ S and indB: independent B
              and spanB: S ⊆ span B and cardB: card B = dim S
      by (metis basis_exists)
    consider dim S < DIM('N) | dim S = DIM('N)
      using dim_subset_UNIV le_eq_less_or_eq by auto
    then show ?thesis
    proof cases
      case 1
      show ?thesis
        by (rule negligible_subset [of closure S])
```

```
        (simp_all add: frontier_def negligible_lowdim 1)
    next
      case 2
      obtain a where a ∈ interior (convex hull insert 0 B)
      proof (rule interior_simplex_nonempty [OF indB])
        show finite B
          by (simp add: indB independent_finite)
        show card B = DIM('N)
          by (simp add: cardB 2)
      qed
      then have a: a ∈ interior S
        by (metis ‹B ⊆ S› ‹0 ∈ S› ‹convex S› insert_absorb insert_subset interior_mono subset_hull)
      show ?thesis
      proof (rule starlike_negligible_strong [where a=a])
        fix c::real and x
        have eq: a + c *R x = (a + x) − (1 − c) *R ((a + x) − a)
          by (simp add: algebra_simps)
        assume 0 ≤ c c < 1 a + x ∈ frontier S
        then show a + c *R x ∉ frontier S
          using eq mem_interior_closure_convex_shrink [OF ‹convex S› a, of _ 1−c]
          unfolding frontier_def
          by (metis Diff_iff add_diff_cancel_left' add_diff_eq diff_gt_0_iff_gt group_cancel.rule0 not_le)
      qed auto
    qed
  qed
  show ?thesis
  proof (cases S = {})
    case True then show ?thesis by auto
  next
    case False
    then obtain a where a ∈ S by auto
    show ?thesis
      using nf [of (λx. −a + x) ‘ S]
    by (metis ‹a ∈ S› add.left_inverse assms convex_translation_eq frontier_translation
              image_eqI negligible_translation_rev)
  qed
qed

corollary negligible_sphere: negligible (sphere a e)
  using frontier_cball negligible_convex_frontier convex_cball
  by (blast intro: negligible_subset)

lemma non_negligible_UNIV [simp]: ¬ negligible UNIV
  unfolding negligible_iff_null_sets by (auto simp: null_sets_def)

lemma negligible_interval:
```

negligible (*cbox a b*) ⟷ *box a b* = {} *negligible* (*box a b*) ⟷ *box a b* = {}
by (*auto simp*: *negligible_iff_null_sets null_sets_def prod_nonneg inner_diff_left box_eq_empty*
not_le emeasure_lborel_cbox_eq emeasure_lborel_box_eq
intro: *eq_refl antisym less_imp_le*)

proposition *open_not_negligible*:
assumes *open S S* ≠ {}
shows ¬ *negligible S*
proof
assume *neg*: *negligible S*
obtain *a* **where** *a* ∈ *S*
using ‹*S* ≠ {}› **by** *blast*
then obtain *e* **where** *e* > *0 cball a e* ⊆ *S*
using ‹*open S*› *open_contains_cball_eq* **by** *blast*
let *?p* = *a* − (*e* / *DIM*('*a*)) ∗$_R$ *One* **let** *?q* = *a* + (*e* / *DIM*('*a*)) ∗$_R$ *One*
have *cbox ?p ?q* ⊆ *cball a e*
proof (*clarsimp simp*: *mem_box dist_norm*)
fix *x*
assume ∀ *i*∈*Basis*. *?p* · *i* ≤ *x* · *i* ∧ *x* · *i* ≤ *?q* · *i*
then have *ax*: |(*a* − *x*) · *i*| ≤ *e* / *real DIM*('*a*) **if** *i* ∈ *Basis* **for** *i*
using *that* **by** (*auto simp*: *algebra_simps*)
have *norm* (*a* − *x*) ≤ (∑ *i*∈*Basis*. |(*a* − *x*) · *i*|)
by (*rule norm_le_l1*)
also have ... ≤ *DIM*('*a*) ∗ (*e* / *real DIM*('*a*))
by (*intro sum_bounded_above ax*)
also have ... = *e*
by *simp*
finally show *norm* (*a* − *x*) ≤ *e* **.**
qed
then have *negligible* (*cbox ?p ?q*)
by (*meson* ‹*cball a e* ⊆ *S*› *neg negligible_subset*)
with ‹*e* > *0*› **show** *False*
by (*simp add*: *negligible_interval box_eq_empty algebra_simps field_split_simps mult_le_0_iff*)
qed

lemma *negligible_convex_interior*:
convex S ⟹ (*negligible S* ⟷ *interior S* = {})
by (*metis Diff_empty closure_subset frontier_def interior_subset negligible_convex_frontier negligible_subset open_interior open_not_negligible*)

lemma *measure_eq_0_null_sets*: *S* ∈ *null_sets M* ⟹ *measure M S* = *0*
by (*auto simp*: *measure_def null_sets_def*)

lemma *negligible_imp_measure0*: *negligible S* ⟹ *measure lebesgue S* = *0*
by (*simp add*: *measure_eq_0_null_sets negligible_iff_null_sets*)

lemma *negligible_iff_emeasure0*: *S* ∈ *sets lebesgue* ⟹ (*negligible S* ⟷ *emeasure*

lebesgue S = 0)
by (*auto simp*: *measure_eq_0_null_sets negligible_iff_null_sets*)

lemma *negligible_iff_measure0*: *S ∈ lmeasurable ⟹* (*negligible S ⟷ measure lebesgue S = 0*)
by (*metis* (*full_types*) *completion.null_sets_outer negligible_iff_null_sets negligible_imp_measure0 order_refl*)

lemma *negligible_imp_sets*: *negligible S ⟹ S ∈ sets lebesgue*
by (*simp add*: *negligible_iff_null_sets null_setsD2*)

lemma *negligible_imp_measurable*: *negligible S ⟹ S ∈ lmeasurable*
by (*simp add*: *fmeasurableI_null_sets negligible_iff_null_sets*)

lemma *negligible_iff_measure*: *negligible S ⟷ S ∈ lmeasurable ∧ measure lebesgue S = 0*
by (*fastforce simp*: *negligible_iff_measure0 negligible_imp_measurable dest*: *negligible_imp_measure0*)

lemma *negligible_outer*:
negligible S ⟷ (*∀ e>0. ∃ T. S ⊆ T ∧ T ∈ lmeasurable ∧ measure lebesgue T < e*) (**is** *_ = ?rhs*)
proof
assume *negligible S* **then show** *?rhs*
by (*metis negligible_iff_measure order_refl*)
next
assume *?rhs* **then show** *negligible S*
by (*meson completion.null_sets_outer negligible_iff_null_sets*)
qed

lemma *negligible_outer_le*:
negligible S ⟷ (*∀ e>0. ∃ T. S ⊆ T ∧ T ∈ lmeasurable ∧ measure lebesgue T ≤ e*) (**is** *_ = ?rhs*)
proof
assume *negligible S* **then show** *?rhs*
by (*metis dual_order.strict_implies_order negligible_imp_measurable negligible_imp_measure0 order_refl*)
next
assume *?rhs* **then show** *negligible S*
by (*metis le_less_trans negligible_outer field_lbound_gt_zero*)
qed

lemma *negligible_UNIV*: *negligible S ⟷* (*indicat_real S has_integral 0*) *UNIV* (**is** *_=?rhs*)
by (*metis lmeasurable_iff_indicator_has_integral negligible_iff_measure*)

lemma *sets_negligible_symdiff*:
⟦*S ∈ sets lebesgue*; *negligible*((*S − T*) ∪ (*T − S*))⟧ *⟹ T ∈ sets lebesgue*
by (*metis Diff_Diff_Int Int_Diff_Un inf_commute negligible_Un_eq negligi-*

ble_imp_sets sets.Diff sets.Un)

lemma *lmeasurable_negligible_symdiff*:
  ⟦$S \in$ *lmeasurable*; *negligible*$((S - T) \cup (T - S))$⟧ $\Longrightarrow$ $T \in$ *lmeasurable*
 using *integrable_spike_set_eq lmeasurable_iff_integrable_on* **by** *blast*

lemma *measure_Un3_negligible*:
 assumes *meas*: $S \in$ *lmeasurable* $T \in$ *lmeasurable* $U \in$ *lmeasurable*
 and *neg*: *negligible*$(S \cap T)$ *negligible*$(S \cap U)$ *negligible*$(T \cap U)$ **and** V: $S \cup T \cup U = V$
shows *measure lebesgue* V = *measure lebesgue* S + *measure lebesgue* T + *measure lebesgue* U
proof −
 have [*simp*]: *measure lebesgue* $(S \cap T) = 0$
  using *neg*(*1*) *negligible_imp_measure0* **by** *blast*
 have [*simp*]: *measure lebesgue* $(S \cap U \cup T \cap U) = 0$
  using *neg*(*2*) *neg*(*3*) *negligible_Un negligible_imp_measure0* **by** *blast*
 have *measure lebesgue* V = *measure lebesgue* $(S \cup T \cup U)$
  using V **by** *simp*
 also have ... = *measure lebesgue* S + *measure lebesgue* T + *measure lebesgue* U
  by (*simp add*: *measure_Un3 meas fmeasurable.Un Int_Un_distrib2*)
 finally show *?thesis* **.**
qed

lemma *measure_translate_add*:
 assumes *meas*: $S \in$ *lmeasurable* $T \in$ *lmeasurable*
  and U: $S \cup ((+)a$ ‘ $T) = U$ **and** *neg*: *negligible*$(S \cap ((+)a$ ‘ $T))$
 shows *measure lebesgue* S + *measure lebesgue* T = *measure lebesgue* U
proof −
 have [*simp*]: *measure lebesgue* $(S \cap (+)\ a$ ‘ $T) = 0$
  using *neg negligible_imp_measure0* **by** *blast*
 have *measure lebesgue* $(S \cup ((+)a$ ‘ $T))$ = *measure lebesgue* S + *measure lebesgue* T
  by (*simp add*: *measure_Un3 meas measurable_translation measure_translation fmeasurable.Un*)
 then show *?thesis*
  using U **by** *auto*
qed

lemma *measure_negligible_symdiff*:
 assumes S: $S \in$ *lmeasurable*
  and *neg*: *negligible* $(S - T \cup (T - S))$
 shows *measure lebesgue* T = *measure lebesgue* S
proof −
 have *measure lebesgue* $(S - T) = 0$
  using *neg negligible_Un_eq negligible_imp_measure0* **by** *blast*
 then show *?thesis*

by (*metis S Un_commute add.right_neutral lmeasurable_negligible_symdiff measure_Un2 neg negligible_Un_eq negligible_imp_measure0*)
qed

lemma *measure_closure*:
assumes *bounded S* **and** *neg*: *negligible* (*frontier S*)
shows *measure lebesgue* (*closure S*) = *measure lebesgue S*
proof −
have *measure lebesgue* (*frontier S*) = *0*
by (*metis neg negligible_imp_measure0*)
then show *?thesis*
by (*metis assms lmeasurable_iff_integrable_on eq_iff_diff_eq_0 has_integral_interior integrable_on_def integral_unique lmeasurable_interior lmeasure_integral measure_frontier*)
qed

lemma *measure_interior*:
⟦*bounded S*; *negligible*(*frontier S*)⟧ ⟹ *measure lebesgue* (*interior S*) = *measure lebesgue S*
using *measure_closure measure_frontier negligible_imp_measure0* **by** *fastforce*

lemma *measurable_Jordan*:
assumes *bounded S* **and** *neg*: *negligible* (*frontier S*)
shows *S* ∈ *lmeasurable*
proof −
have *closure S* ∈ *lmeasurable*
by (*metis lmeasurable_closure* ‹*bounded S*›)
moreover have *interior S* ∈ *lmeasurable*
by (*simp add*: *lmeasurable_interior* ‹*bounded S*›)
moreover have *interior S* ⊆ *S*
by (*simp add*: *interior_subset*)
ultimately show *?thesis*
using *assms* **by** (*metis* (*full_types*) *closure_subset completion.complete_sets_sandwich_fmeasurable measure_closure measure_interior*)
qed

lemma *measurable_convex*: ⟦*convex S*; *bounded S*⟧ ⟹ *S* ∈ *lmeasurable*
by (*simp add*: *measurable_Jordan negligible_convex_frontier*)

lemma *content_cball_conv_ball*: *content* (*cball c r*) = *content* (*ball c r*)
proof −
have *ball c r* − *cball c r* ∪ (*cball c r* − *ball c r*) = *sphere c r*
by *auto*
hence *measure lebesgue* (*cball c r*) = *measure lebesgue* (*ball c r*)
using *negligible_sphere*[*of c r*] **by** (*intro measure_negligible_symdiff*) *simp_all*
thus *?thesis* **by** *simp*
qed

6.16.10 Negligibility of image under non-injective linear map

lemma *negligible_Union_nat*:
assumes $\bigwedge n$::*nat*. *negligible*(S n)
shows *negligible*($\bigcup n$. S n)
proof −
have *negligible* ($\bigcup m{\leq}k$. S m) **for** k
using *assms* **by** *blast*
then have *0*: *integral UNIV* (*indicat_real* ($\bigcup m{\leq}k$. S m)) = *0*
and *1*: (*indicat_real* ($\bigcup m{\leq}k$. S m)) *integrable_on UNIV* **for** k
by (*auto simp*: *negligible has_integral_iff*)
have *2*: $\bigwedge k$ x. *indicat_real* ($\bigcup m{\leq}k$. S m) $x \leq$ (*indicat_real* ($\bigcup m{\leq}$*Suc* k. S m) x)
by (*auto simp add*: *indicator_def*)
have *3*: $\bigwedge x$. (λk. *indicat_real* ($\bigcup m{\leq}k$. S m) x) ⟶ (*indicat_real* ($\bigcup n$. S n) x)
by (*force simp*: *indicator_def eventually_sequentially intro*: *tendsto_eventually*)
have *4*: *bounded* (*range* (λk. *integral UNIV* (*indicat_real* ($\bigcup m{\leq}k$. S m))))
by (*simp add*: *0*)
have ∗: *indicat_real* ($\bigcup n$. S n) *integrable_on UNIV* ∧
(λk. *integral UNIV* (*indicat_real* ($\bigcup m{\leq}k$. S m))) ⟶ (*integral UNIV* (*indicat_real* ($\bigcup n$. S n)))
by (*intro monotone_convergence_increasing 1 2 3 4*)
then have *integral UNIV* (*indicat_real* ($\bigcup n$. S n)) = (*0*::*real*)
using *LIMSEQ_unique* **by** (*auto simp*: *0*)
then show *?thesis*
using ∗ **by** (*simp add*: *negligible_UNIV has_integral_iff*)
qed

lemma *negligible_linear_singular_image*:
fixes f :: 'n::*euclidean_space* ⇒ 'n
assumes *linear* f ¬ *inj* f
shows *negligible* (f ‘ S)
proof −
obtain a **where** $a \neq 0$ $\bigwedge S$. f ‘ $S \subseteq \{x.\ a \cdot x = 0\}$
using *assms linear_singular_image_hyperplane* **by** *blast*
then show *negligible* (f ‘ S)
by (*metis negligible_hyperplane negligible_subset*)
qed

lemma *measure_negligible_finite_Union*:
assumes *finite* $\mathcal{F}$
and *meas*: $\bigwedge S$. $S \in \mathcal{F} \Longrightarrow S \in$ *lmeasurable*
and *djointish*: *pairwise* (λS T. *negligible* ($S \cap T$)) $\mathcal{F}$
shows *measure lebesgue* ($\bigcup \mathcal{F}$) = ($\sum S{\in}\mathcal{F}$. *measure lebesgue* S)
using *assms*
proof (*induction*)
case *empty*
then show *?case*

```
    by auto
next
  case (insert S \<F>)
  then have S \<in> lmeasurable \<Union>\<F> \<in> lmeasurable pairwise (\<lambda>S T. negligible (S \<inter>
T)) \<F>
      by (simp_all add: fmeasurable.finite_Union insert.hyps(1) insert.prems(1)
pairwise_insert subsetI)
  then show ?case
  proof (simp add: measure_Un3 insert)
    have *: \<And>T. T \<in> (\<inter>) S ` \<F> \<Longrightarrow> negligible T
      using insert by (force simp: pairwise_def)
    have negligible(S \<inter> \<Union>\<F>)
      unfolding Int_Union
      by (rule negligible_Union) (simp_all add: * insert.hyps(1))
    then show measure lebesgue (S \<inter> \<Union>\<F>) = 0
      using negligible_imp_measure0 by blast
  qed
qed

lemma measure_negligible_finite_Union_image:
  assumes finite S
    and meas: \<And>x. x \<in> S \<Longrightarrow> f x \<in> lmeasurable
    and djointish: pairwise (\<lambda>x y. negligible (f x \<inter> f y)) S
  shows measure lebesgue (\<Union>(f ` S)) = (\<Sum>x\<in>S. measure lebesgue (f x))
proof -
  have measure lebesgue (\<Union>(f ` S)) = sum (measure lebesgue) (f ` S)
   using assms by (auto simp: pairwise_mono pairwise_image intro: measure_negligible_finite_Union)
  also have ... = sum (measure lebesgue \<circ> f) S
   using djointish [unfolded pairwise_def] by (metis inf.idem negligible_imp_measure0
sum.reindex_nontrivial [OF \<open>finite S\<close>])
  also have ... = (\<Sum>x\<in>S. measure lebesgue (f x))
    by simp
  finally show ?thesis .
qed
```

6.16.11 Negligibility of a Lipschitz image of a negligible set

The bound will be eliminated by a sort of onion argument

```
lemma locally_Lipschitz_negl_bounded:
  fixes f :: 'M::euclidean_space \<Rightarrow> 'N::euclidean_space
  assumes MleN: DIM('M) \<le> DIM('N) 0 < B bounded S negligible S
      and lips: \<And>x. x \<in> S
                  \<Longrightarrow> \<exists>T. open T \<and> x \<in> T \<and>
                        (\<forall>y \<in> S \<inter> T. norm(f y - f x) \<le> B * norm(y - x))
  shows negligible (f ` S)
  unfolding negligible_iff_null_sets
proof (clarsimp simp: completion.null_sets_outer)
  fix e::real
  assume 0 < e
```

have $S \in lmeasurable$
using ‹*negligible S*› **by** (*simp add*: *negligible_iff_null_sets fmeasurableI_null_sets*)
then have $S \in sets\ lebesgue$
by *blast*
have *e22*: $0 < e/2 \ / \ (2 * B * real\ DIM('M)) \ \hat{}\ DIM('N)$
using ‹*0 < e*› ‹*0 < B*› **by** (*simp add*: *field_split_simps*)
obtain *T* **where** *open T S* $\subseteq T$ $(T - S) \in lmeasurable$
$measure\ lebesgue\ (T - S) < e/2 \ / \ (2 * B * DIM('M)) \ \hat{}\ DIM('N)$
using *sets_lebesgue_outer_open* [*OF* ‹*S* $\in$ *sets lebesgue*› *e22*]
by (*metis emeasure_eq_measure2 ennreal_leI linorder_not_le*)
then have *T*: $measure\ lebesgue\ T \leq e/2 \ / \ (2 * B * DIM('M)) \ \hat{}\ DIM('N)$
using ‹*negligible S*› **by** (*simp add*: *measure_Diff_null_set negligible_iff_null_sets*)
have $\exists r.\ 0 < r \wedge r \leq 1/2 \wedge$
$(x \in S \longrightarrow (\forall y.\ norm(y - x) < r$
$\longrightarrow y \in T \wedge (y \in S \longrightarrow norm(f\ y - f\ x) \leq B * norm(y - x))))$
for *x*
proof (*cases* $x \in S$)
case *True*
obtain *U* **where** *open U* $x \in U$ **and** *U*: $\bigwedge y.\ y \in S \cap U \Longrightarrow norm(f\ y - f\ x) \leq B * norm(y - x)$
using *lips* [*OF* ‹$x \in S$›] **by** *auto*
have $x \in T \cap U$
using ‹$S \subseteq T$› ‹$x \in U$› ‹$x \in S$› **by** *auto*
then obtain ε **where** $0 < \varepsilon$ *ball x* $\varepsilon \subseteq T \cap U$
by (*metis* ‹*open T*› ‹*open U*› *openE open_Int*)
then show *?thesis*
by (*rule_tac x=min* (*1/2*) ε **in** *exI*) (*simp add*: *U dist_norm norm_minus_commute subset_iff*)
next
case *False*
then show *?thesis*
by (*rule_tac x=1/4* **in** *exI*) *auto*
qed
then obtain *R* **where** *R12*: $\bigwedge x.\ 0 < R\ x \wedge R\ x \leq 1/2$
and *RT*: $\bigwedge x\ y.\ [\![x \in S;\ norm(y - x) < R\ x]\!] \Longrightarrow y \in T$
and *RB*: $\bigwedge x\ y.\ [\![x \in S;\ y \in S;\ norm(y - x) < R\ x]\!] \Longrightarrow norm(f\ y - f\ x) \leq B * norm(y - x)$
by *metis+*
then have *gaugeR*: $gauge\ (\lambda x.\ ball\ x\ (R\ x))$
by (*simp add*: *gauge_def*)
obtain *c* **where** *c*: $S \subseteq cbox\ (-c *_R One)\ (c *_R One)\ box\ (-c *_R One::'M)\ (c *_R One) \neq \{\}$
proof −
obtain *B* **where** *B*: $\bigwedge x.\ x \in S \Longrightarrow norm\ x \leq B$
using ‹*bounded S*› *bounded_iff* **by** *blast*
show *?thesis*
proof (*rule_tac c* = *abs B* + *1* **in** *that*)
show $S \subseteq cbox\ (-\ (|B| + 1) *_R One)\ ((|B| + 1) *_R One)$
using *norm_bound_Basis_le Basis_le_norm*

```
      by (fastforce simp: mem_box dest!: B intro: order_trans)
    show box (- (|B| + 1) *R One) ((|B| + 1) *R One) ≠ {}
      by (simp add: box_eq_empty(1))
  qed
 qed
 obtain 𝒟 where countable 𝒟
   and Dsub: ⋃𝒟 ⊆ cbox (-c *R One) (c *R One)
   and cbox: ⋀K. K ∈ 𝒟 ⟹ interior K ≠ {} ∧ (∃c d. K = cbox c d)
   and pw:   pairwise (λA B. interior A ∩ interior B = {}) 𝒟
   and Ksub: ⋀K. K ∈ 𝒟 ⟹ ∃x∈S ∩ K. K ⊆ (λx. ball x (R x)) x
   and exN:  ⋀u v. cbox u v ∈ 𝒟 ⟹ ∃n. ∀i∈Basis. v · i - u · i = (2*c) /
2^n
   and S ⊆ ⋃𝒟
  using covering_lemma [OF c gaugeR] by force
 have ∃u v z. K = cbox u v ∧ box u v ≠ {} ∧ z ∈ S ∧ z ∈ cbox u v ∧
            cbox u v ⊆ ball z (R z) if K ∈ 𝒟 for K
 proof -
  obtain u v where K = cbox u v
   using ‹K ∈ 𝒟› cbox by blast
  with that show ?thesis
   by (metis Int_iff interior_cbox cbox Ksub)
 qed
 then obtain uf vf zf
  where uvz: ⋀K. K ∈ 𝒟 ⟹
            K = cbox (uf K) (vf K) ∧ box (uf K) (vf K) ≠ {} ∧ zf K ∈ S ∧
            zf K ∈ cbox (uf K) (vf K) ∧ cbox (uf K) (vf K) ⊆ ball (zf K) (R (zf
K))
   by metis
 define prj1 where prj1 ≡ λx::'M. x · (SOME i. i ∈ Basis)
 define fbx where fbx ≡ λD. cbox (f(zf D) - (B * DIM('M) * (prj1(vf D - uf
D))) *R One::'N)
                          (f(zf D) + (B * DIM('M) * prj1(vf D - uf D)) *R
One)
 have vu_pos: 0 < prj1 (vf X - uf X) if X ∈ 𝒟 for X
   using uvz [OF that] by (simp add: prj1_def box_ne_empty SOME_Basis
inner_diff_left)
 have prj1_idem: prj1 (vf X - uf X) = (vf X - uf X) · i if X ∈ 𝒟 i ∈ Basis
for X i
 proof -
  have cbox (uf X) (vf X) ∈ 𝒟
   using uvz ‹X ∈ 𝒟› by auto
  with exN obtain n where ⋀i. i ∈ Basis ⟹ vf X · i - uf X · i = (2*c) /
2^n
   by blast
  then show ?thesis
   by (simp add: ‹i ∈ Basis› SOME_Basis inner_diff prj1_def)
 qed
 have countbl: countable (fbx ` 𝒟)
  using ‹countable 𝒟› by blast
```

```
  have (∑ k∈fbx`𝒟'. measure lebesgue k) ≤ e/2 if 𝒟' ⊆ 𝒟 finite 𝒟' for 𝒟'
  proof −
    have BM_ge0: 0 ≤ B * (DIM('M) * prj1 (vf X − uf X)) if X ∈ 𝒟' for X
      using ‹0 < B› ‹𝒟' ⊆ 𝒟› that vu_pos by fastforce
    have {} ∉ 𝒟'
      using cbox ‹𝒟' ⊆ 𝒟› interior_empty by blast
    have (∑ k∈fbx`𝒟'. measure lebesgue k) ≤ sum (measure lebesgue o fbx) 𝒟'
      by (rule sum_image_le [OF ‹finite 𝒟'›]) (force simp: fbx_def)
   also have ... ≤ (∑ X∈𝒟'. (2 * B * DIM('M)) ^ DIM('N) * measure lebesgue
X)
    proof (rule sum_mono)
      fix X assume X ∈ 𝒟'
      then have X ∈ 𝒟 using ‹𝒟' ⊆ 𝒟› by blast
      then have ufvf: cbox (uf X) (vf X) = X
        using uvz by blast
      have prj1 (vf X − uf X) ^ DIM('M) = (∏ i::'M ∈ Basis. prj1 (vf X − uf
X))
        by (rule prod_constant [symmetric])
      also have ... = (∏ i∈Basis. vf X • i − uf X • i)
        by (simp add: ‹X ∈ 𝒟› inner_diff_left prj1_idem cong: prod.cong)
      finally have prj1_eq: prj1 (vf X − uf X) ^ DIM('M) = (∏ i∈Basis. vf X •
i − uf X • i) .
      have uf X ∈ cbox (uf X) (vf X) vf X ∈ cbox (uf X) (vf X)
        using uvz [OF ‹X ∈ 𝒟›] by (force simp: mem_box)+
      moreover have cbox (uf X) (vf X) ⊆ ball (zf X) (1/2)
        by (meson R12 order_trans subset_ball uvz [OF ‹X ∈ 𝒟›])
      ultimately have uf X ∈ ball (zf X) (1/2)  vf X ∈ ball (zf X) (1/2)
        by auto
      then have dist (vf X) (uf X) ≤ 1
        unfolding mem_ball
        by (metis dist_commute dist_triangle_half_l dual_order.order_iff_strict)
      then have 1: prj1 (vf X − uf X) ≤ 1
        unfolding prj1_def dist_norm using Basis_le_norm SOME_Basis or-
der_trans by fastforce
      have 0: 0 ≤ prj1 (vf X − uf X)
        using ‹X ∈ 𝒟› prj1_def vu_pos by fastforce
     have (measure lebesgue ∘ fbx) X ≤ (2 * B * DIM('M)) ^ DIM('N) * content
(cbox (uf X) (vf X))
       apply (simp add: fbx_def content_cbox_cases algebra_simps BM_ge0 ‹X
∈ 𝒟'› ‹0 < B› flip: prj1_eq)
        using MleN 0 1 uvz ‹X ∈ 𝒟›
        by (fastforce simp add: box_ne_empty power_decreasing)
      also have ... = (2 * B * DIM('M)) ^ DIM('N) * measure lebesgue X
        by (subst (3) ufvf[symmetric]) simp
      finally show (measure lebesgue ∘ fbx) X ≤ (2 * B * DIM('M)) ^ DIM('N)
* measure lebesgue X .
    qed
    also have ... = (2 * B * DIM('M)) ^ DIM('N) * sum (measure lebesgue) 𝒟'
      by (simp add: sum_distrib_left)
```

```
    also have ... ≤ e/2
    proof −
      have ⋀K. K ∈ 𝒟' ⟹ ∃ a b. K = cbox a b
        using cbox that by blast
      then have div: 𝒟' division_of ⋃𝒟'
        using pairwise_subset [OF pw ‹𝒟' ⊆ 𝒟›] unfolding pairwise_def
        by (force simp: ‹finite 𝒟'› ‹{} ∉ 𝒟'› division_of_def)
      have le_meaT: measure lebesgue (⋃𝒟') ≤ measure lebesgue T
      proof (rule measure_mono_fmeasurable)
        show (⋃𝒟') ∈ sets lebesgue
          using div lmeasurable_division by auto
        have ⋃𝒟' ⊆ ⋃𝒟
          using ‹𝒟' ⊆ 𝒟› by blast
        also have ... ⊆ T
        proof (clarify)
          fix x D
          assume x ∈ D D ∈ 𝒟
          show x ∈ T
            using Ksub [OF ‹D ∈ 𝒟›]
            by (metis ‹x ∈ D› Int_iff dist_norm mem_ball norm_minus_commute
subsetD RT)
        qed
        finally show ⋃𝒟' ⊆ T .
        show T ∈ lmeasurable
            using ‹S ∈ lmeasurable› ‹S ⊆ T› ‹T − S ∈ lmeasurable› fmeasur-
able_Diff_D by blast
      qed
      have sum (measure lebesgue) 𝒟' = sum content 𝒟'
        using ‹𝒟' ⊆ 𝒟› cbox by (force intro: sum.cong)
      then have (2 * B * DIM('M)) ^ DIM('N) * sum (measure lebesgue) 𝒟' =
                (2 * B * real DIM('M)) ^ DIM('N) * measure lebesgue (⋃𝒟')
        using content_division [OF div] by auto
      also have ... ≤ (2 * B * real DIM('M)) ^ DIM('N) * measure lebesgue T
        using ‹0 < B›
        by (intro mult_left_mono [OF le_meaT]) (force simp add: algebra_simps)
      also have ... ≤ e/2
        using T ‹0 < B› by (simp add: field_simps)
      finally show ?thesis .
    qed
    finally show ?thesis .
  qed
  then have e2: sum (measure lebesgue) 𝒢 ≤ e/2 if 𝒢 ⊆ fbx ` 𝒟 finite 𝒢 for 𝒢
    by (metis finite_subset_image that)
  show ∃ W∈lmeasurable. f ` S ⊆ W ∧ measure lebesgue W < e
  proof (intro bexI conjI)
    have ∃ X∈𝒟. f y ∈ fbx X if y ∈ S for y
    proof −
      obtain X where y ∈ X X ∈ 𝒟
        using ‹S ⊆ ⋃𝒟› ‹y ∈ S› by auto
```

```
    then have y: y ∈ ball(zf X) (R(zf X))
      using uvz by fastforce
    have conj_le_eq: z − b ≤ y ∧ y ≤ z + b ⟷ abs(y − z) ≤ b for z y b::real
      by auto
    have yin: y ∈ cbox (uf X) (vf X) and zin: (zf X) ∈ cbox (uf X) (vf X)
      using uvz ‹X ∈ 𝒟› ‹y ∈ X› by auto
    have norm (y − zf X) ≤ (∑ i∈Basis. |(y − zf X) · i|)
      by (rule norm_le_l1)
    also have ... ≤ real DIM('M) * prj1 (vf X − uf X)
    proof (rule sum_bounded_above)
      fix j::'M assume j: j ∈ Basis
      show |(y − zf X) · j| ≤ prj1 (vf X − uf X)
        using yin zin j
     by (fastforce simp add: mem_box prj1_idem [OF ‹X ∈ 𝒟› j] inner_diff_left)
    qed
    finally have nole: norm (y − zf X) ≤ DIM('M) * prj1 (vf X − uf X)
      by simp
    have fle: |f y · i − f(zf X) · i| ≤ B * DIM('M) * prj1 (vf X − uf X) if i ∈
Basis for i
    proof −
      have |f y · i − f (zf X) · i| = |(f y − f (zf X)) · i|
        by (simp add: algebra_simps)
      also have ... ≤ norm (f y − f (zf X))
        by (simp add: Basis_le_norm that)
      also have ... ≤ B * norm(y − zf X)
        by (metis uvz RB ‹X ∈ 𝒟› dist_commute dist_norm mem_ball ‹y ∈ S›
y)
      also have ... ≤ B * real DIM('M) * prj1 (vf X − uf X)
        using ‹0 < B› by (simp add: nole)
      finally show ?thesis .
    qed
    show ?thesis
      by (rule_tac x=X in bexI)
       (auto simp: fbx_def prj1_idem mem_box conj_le_eq inner_add inner_diff
fle ‹X ∈ 𝒟›)
  qed
  then show f ‘ S ⊆ (⋃ D∈𝒟. fbx D) by auto
next
  have 1: ⋀D. D ∈ 𝒟 ⟹ fbx D ∈ lmeasurable
    by (auto simp: fbx_def)
  have 2: I' ⊆ 𝒟 ⟹ finite I' ⟹ measure lebesgue (⋃ D∈I'. fbx D) ≤ e/2 for
I'
    by (rule order_trans[OF measure_Union_le e2]) (auto simp: fbx_def)
  show (⋃ D∈𝒟. fbx D) ∈ lmeasurable
    by (intro fmeasurable_UN_bound[OF ‹countable 𝒟› 1 2])
  have measure lebesgue (⋃ D∈𝒟. fbx D) ≤ e/2
    by (intro measure_UN_bound[OF ‹countable 𝒟› 1 2])
  then show measure lebesgue (⋃ D∈𝒟. fbx D) < e
    using ‹0 < e› by linarith
```

```
  qed
qed

proposition negligible_locally_Lipschitz_image:
  fixes f :: 'M::euclidean_space ⇒ 'N::euclidean_space
  assumes MleN: DIM('M) ≤ DIM('N) negligible S
      and lips: ⋀x. x ∈ S
                  ⟹ ∃ T B. open T ∧ x ∈ T ∧
                       (∀ y ∈ S ∩ T. norm(f y − f x) ≤ B * norm(y − x))
    shows negligible (f ‘ S)
proof −
  let ?S = λn. ({x ∈ S. norm x ≤ n ∧
                       (∃ T. open T ∧ x ∈ T ∧
                            (∀ y∈S ∩ T. norm (f y − f x) ≤ (real n + 1) * norm (y
− x)))})
  have negfn: f ‘ ?S n ∈ null_sets lebesgue for n::nat
    unfolding negligible_iff_null_sets[symmetric]
    apply (rule_tac B = real n + 1 in locally_Lipschitz_negl_bounded)
    by (auto simp: MleN bounded_iff intro: negligible_subset [OF ‹negligible S›])
  have S = (⋃ n. ?S n)
  proof (intro set_eqI iffI)
    fix x assume x ∈ S
    with lips obtain T B where T: open T x ∈ T
                          and B: ⋀y. y ∈ S ∩ T ⟹ norm(f y − f x) ≤ B * norm(y
− x)
      by metis+
    have no: norm (f y − f x) ≤ (nat ⌈max B (norm x)⌉ + 1) * norm (y − x) if
y ∈ S ∩ T for y
    proof −
      have B * norm(y − x) ≤ (nat ⌈max B (norm x)⌉ + 1) * norm (y − x)
      by (meson max.cobounded1 mult_right_mono nat_ceiling_le_eq nat_le_iff_add
norm_ge_zero order_trans)
      then show ?thesis
        using B order_trans that by blast
    qed
    have norm x ≤ real (nat ⌈max B (norm x)⌉)
      by linarith
    then have x ∈ ?S (nat (ceiling (max B (norm x))))
      using T no by (force simp: ‹x ∈ S› algebra_simps)
    then show x ∈ (⋃ n. ?S n) by force
  qed auto
  then show ?thesis
   by (rule ssubst) (auto simp: image_Union negligible_iff_null_sets intro: negfn)
qed

corollary negligible_differentiable_image_negligible:
  fixes f :: 'M::euclidean_space ⇒ 'N::euclidean_space
  assumes MleN: DIM('M) ≤ DIM('N) negligible S
      and diff_f: f differentiable_on S
```

```
    shows negligible (f ‘ S)
proof −
  have ∃ T B. open T ∧ x ∈ T ∧ (∀ y ∈ S ∩ T. norm(f y − f x) ≤ B ∗ norm(y
− x))
        if x ∈ S for x
  proof −
    obtain f′ where linear f′
      and f′: ⋀e. e>0 ⟹
                    ∃ d>0. ∀ y∈S. norm (y − x) < d ⟶
                                 norm (f y − f x − f′ (y − x)) ≤ e ∗ norm (y − x)
      using diff_f ‹x ∈ S›
    by (auto simp: linear_linear differentiable_on_def differentiable_def has_derivative_within_alt)
    obtain B where B > 0 and B: ∀ x. norm (f′ x) ≤ B ∗ norm x
      using linear_bounded_pos ‹linear f′› by blast
    obtain d where d>0
            and d: ⋀y. ⟦y ∈ S; norm (y − x) < d⟧ ⟹
                       norm (f y − f x − f′ (y − x)) ≤ norm (y − x)
      using f′ [of 1] by (force simp:)
    show ?thesis
    proof (intro exI conjI ballI)
      show norm (f y − f x) ≤ (B + 1) ∗ norm (y − x)
        if y ∈ S ∩ ball x d for y
      proof −
       have norm (f y − f x) − B ∗ norm (y − x) ≤ norm (f y − f x) − norm (f′
(y − x))
         by (simp add: B)
       also have ... ≤ norm (f y − f x − f′ (y − x))
         by (rule norm_triangle_ineq2)
       also have ... ≤ norm (y − x)
         by (metis IntE d dist_norm mem_ball norm_minus_commute that)
       finally show ?thesis
         by (simp add: algebra_simps)
      qed
    qed (use ‹d>0› in auto)
  qed
  with negligible_locally_Lipschitz_image assms show ?thesis by metis
qed

corollary negligible_differentiable_image_lowdim:
  fixes f :: 'M::euclidean_space ⇒ 'N::euclidean_space
  assumes MlessN: DIM('M) < DIM('N) and diff_f: f differentiable_on S
    shows negligible (f ‘ S)
proof −
  have x ≤ DIM('M) ⟹ x ≤ DIM('N) for x
    using MlessN by linarith
  obtain lift :: 'M ∗ real ⇒ 'N and drop :: 'N ⇒ 'M ∗ real and j :: 'N
    where linear lift linear drop and dropl [simp]: ⋀z. drop (lift z) = z
      and j ∈ Basis and j: ⋀x. lift(x,0) • j = 0
    using lowerdim_embeddings [OF MlessN] by metis
```

have *negligible* ((λ*x*. *lift* (*x*, *0*)) ‘ *S*)
proof −
have *negligible* {*x*. *x*•*j* = *0*}
by (*metis* ‹*j* ∈ *Basis*› *negligible_standard_hyperplane*)
moreover have (λ*m*. *lift* (*m*, *0*)) ‘ *S* ⊆ {*n*. *n* • *j* = *0*}
using *j* **by** *force*
ultimately show *?thesis*
using *negligible_subset* **by** *auto*
qed
moreover
have *f* ∘ *fst* ∘ *drop differentiable_on* (λ*x*. *lift* (*x*, *0*)) ‘ *S*
using *diff_f*
apply (*clarsimp simp add*: *differentiable_on_def*)
apply (*intro differentiable_chain_within linear_imp_differentiable* [*OF* ‹*linear drop*›]
linear_imp_differentiable [*OF linear_fst*])
apply (*force simp*: *image_comp o_def*)
done
moreover
have *f* = *f* ∘ *fst* ∘ *drop* ∘ (λ*x*. *lift* (*x*, *0*))
by (*simp add*: *o_def*)
ultimately show *?thesis*
by (*metis* (*no_types*) *image_comp negligible_differentiable_image_negligible order_refl*)
qed

6.16.12 Measurability of countable unions and intersections of various kinds.

lemma
assumes *S*: ⋀*n*. *S n* ∈ *lmeasurable*
and *leB*: ⋀*n*. *measure lebesgue* (*S n*) ≤ *B*
and *nest*: ⋀*n*. *S n* ⊆ *S*(*Suc n*)
shows *measurable_nested_Union*: (⋃ *n*. *S n*) ∈ *lmeasurable*
and *measure_nested_Union*: (λ*n*. *measure lebesgue* (*S n*)) ⟶ *measure lebesgue* (⋃ *n*. *S n*) (**is** *?Lim*)
proof −
have *indicat_real* (⋃ (*range S*)) *integrable_on UNIV* ∧
(λ*n*. *integral UNIV* (*indicat_real* (*S n*)))
⟶ *integral UNIV* (*indicat_real* (⋃ (*range S*)))
proof (*rule monotone_convergence_increasing*)
show ⋀*n*. (*indicat_real* (*S n*)) *integrable_on UNIV*
using *S measurable_integrable* **by** *blast*
show ⋀*n x*::*'a*. *indicat_real* (*S n*) *x* ≤ (*indicat_real* (*S* (*Suc n*)) *x*)
by (*simp add*: *indicator_leI nest rev_subsetD*)
have ⋀*x*. (∃ *n*. *x* ∈ *S n*) ⟶ (∀ *F n in sequentially*. *x* ∈ *S n*)
by (*metis eventually_sequentiallyI lift_Suc_mono_le nest subsetCE*)
then
show ⋀*x*. (λ*n*. *indicat_real* (*S n*) *x*) ⟶ (*indicat_real* (⋃(*S* ‘ *UNIV*)) *x*)

```
    by (simp add: indicator_def tendsto_eventually)
   show bounded (range (λn. integral UNIV (indicat_real (S n))))
    using leB by (auto simp: lmeasure_integral_UNIV [symmetric] S bounded_iff)
  qed
  then have (⋃ n. S n) ∈ lmeasurable ∧ ?Lim
    by (simp add: lmeasure_integral_UNIV S cong: conj_cong) (simp add: measurable_integrable)
  then show (⋃ n. S n) ∈ lmeasurable ?Lim
    by auto
qed

lemma
  assumes S: ⋀n. S n ∈ lmeasurable
    and djointish: pairwise (λm n. negligible (S m ∩ S n)) UNIV
    and leB: ⋀n. (∑ k≤n. measure lebesgue (S k)) ≤ B
  shows measurable_countable_negligible_Union: (⋃ n. S n) ∈ lmeasurable
  and   measure_countable_negligible_Union:   (λn. (measure lebesgue (S n)))
sums measure lebesgue (⋃ n. S n) (is ?Sums)
proof −
  have 1: ⋃ (S ‘ {..n}) ∈ lmeasurable for n
    using S by blast
  have 2: measure lebesgue (⋃ (S ‘ {..n})) ≤ B for n
  proof −
    have measure lebesgue (⋃ (S ‘ {..n})) ≤ (∑ k≤n. measure lebesgue (S k))
      by (simp add: S fmeasurableD measure_UNION_le)
    with leB show ?thesis
      using order_trans by blast
  qed
  have 3: ⋀n. ⋃ (S ‘ {..n}) ⊆ ⋃ (S ‘ {..Suc n})
    by (simp add: SUP_subset_mono)
  have eqS: (⋃ n. S n) = (⋃ n. ⋃ (S ‘ {..n}))
    using atLeastAtMost_iff by blast
  also have (⋃ n. ⋃ (S ‘ {..n})) ∈ lmeasurable
    by (intro measurable_nested_Union [OF 1 2] 3)
  finally show (⋃ n. S n) ∈ lmeasurable .
  have eqm: (∑ i≤n. measure lebesgue (S i)) = measure lebesgue (⋃ (S ‘ {..n}))
for n
    using assms by (simp add: measure_negligible_finite_Union_image pairwise_mono)
  have (λn. (measure lebesgue (S n))) sums measure lebesgue (⋃ n. ⋃ (S ‘ {..n}))
    by (simp add: sums_def' eqm atLeast0AtMost) (intro measure_nested_Union
[OF 1 2] 3)
  then show ?Sums
    by (simp add: eqS)
qed

lemma negligible_countable_Union [intro]:
  assumes countable ℱ and meas: ⋀S. S ∈ ℱ ⟹ negligible S
  shows negligible (⋃ ℱ)
```

proof (*cases* $\mathcal{F} = \{\}$)
case *False*
then show *?thesis*
by (*metis from_nat_into range_from_nat_into assms negligible_Union_nat*)
qed *simp*

lemma
assumes *S*: $\bigwedge n.\ (S\ n) \in$ *lmeasurable*
and *djointish*: *pairwise* $(\lambda m\ n.\ \textit{negligible}\ (S\ m \cap S\ n))$ *UNIV*
and *bo*: *bounded* $(\bigcup n.\ S\ n)$
shows *measurable_countable_negligible_Union_bounded*: $(\bigcup n.\ S\ n) \in$ *lmeasurable*
and *measure_countable_negligible_Union_bounded*: $(\lambda n.\ (\textit{measure lebesgue}\ (S\ n)))$ *sums measure lebesgue* $(\bigcup n.\ S\ n)$ (**is** *?Sums*)
proof −
obtain *a b* **where** *ab*: $(\bigcup n.\ S\ n) \subseteq \textit{cbox}\ a\ b$
using *bo bounded_subset_cbox_symmetric* **by** *metis*
then have *B*: $(\sum k{\le}n.\ \textit{measure lebesgue}\ (S\ k)) \le \textit{measure lebesgue}\ (\textit{cbox}\ a\ b)$
for *n*
proof −
have $(\sum k{\le}n.\ \textit{measure lebesgue}\ (S\ k)) = \textit{measure lebesgue}\ (\bigcup\ (S\ `\ \{..n\}))$
using *measure_negligible_finite_Union_image* [*OF* _ _ *pairwise_subset*] *djointish*
by (*metis S finite_atMost subset_UNIV*)
also have $\ldots \le \textit{measure lebesgue}\ (\textit{cbox}\ a\ b)$
proof (*rule measure_mono_fmeasurable*)
show $\bigcup\ (S\ `\ \{..n\}) \in$ *sets lebesgue* **using** *S* **by** *blast*
qed (*use ab* **in** *auto*)
finally show *?thesis* **.**
qed
show $(\bigcup n.\ S\ n) \in$ *lmeasurable*
by (*rule measurable_countable_negligible_Union* [*OF S djointish B*])
show *?Sums*
by (*rule measure_countable_negligible_Union* [*OF S djointish B*])
qed

lemma *measure_countable_Union_approachable*:
assumes *countable* $\mathcal{D}$ $e > 0$ **and** *measD*: $\bigwedge d.\ d \in \mathcal{D} \Longrightarrow d \in$ *lmeasurable*
and *B*: $\bigwedge D'.\ [\![D' \subseteq \mathcal{D};\ \textit{finite}\ D']\!] \Longrightarrow \textit{measure lebesgue}\ (\bigcup D') \le B$
obtains D' **where** $D' \subseteq \mathcal{D}$ *finite* D' *measure lebesgue* $(\bigcup \mathcal{D}) - e < \textit{measure lebesgue}\ (\bigcup D')$
proof (*cases* $\mathcal{D} = \{\}$)
case *True*
then show *?thesis*
by (*simp add*: ‹$e > 0$› *that*)
next
case *False*
let *?S* $= \lambda n.\ \bigcup k \le n.\ \textit{from\_nat\_into}\ \mathcal{D}\ k$
have $(\lambda n.\ \textit{measure lebesgue}\ (?S\ n)) \longrightarrow \textit{measure lebesgue}\ (\bigcup n.\ ?S\ n)$

proof (*rule measure_nested_Union*)
show *?S n ∈ lmeasurable* **for** *n*
by (*simp add*: *False fmeasurable.finite_UN from_nat_into measD*)
show *measure lebesgue* (*?S n*) ≤ *B* **for** *n*
by (*metis* (*mono_tags*, *lifting*) *B False finite_atMost finite_imageI from_nat_into image_iff subsetI*)
show *?S n* ⊆ *?S* (*Suc n*) **for** *n*
by *force*
qed
then obtain *N* **where** *N*: ⋀*n*. *n* ≥ *N* ⟹ *dist* (*measure lebesgue* (*?S n*)) (*measure lebesgue* (⋃ *n*. *?S n*)) < *e*
using *metric_LIMSEQ_D* ‹*e* > *0*› **by** *blast*
show *?thesis*
proof
show *from_nat_into* 𝒟 ‘ {..*N*} ⊆ 𝒟
by (*auto simp*: *False from_nat_into*)
have *eq*: (⋃ *n*. ⋃ *k*≤*n*. *from_nat_into* 𝒟 *k*) = (⋃ 𝒟)
using ‹*countable* 𝒟› *False*
by (*auto intro*: *from_nat_into dest*: *from_nat_into_surj* [*OF* ‹*countable* 𝒟›])
show *measure lebesgue* (⋃ 𝒟) − *e* < *measure lebesgue* (⋃ (*from_nat_into* 𝒟 ‘ {..*N*}))
using *N* [*OF order_refl*]
by (*auto simp*: *eq algebra_simps dist_norm*)
qed *auto*
qed

6.16.13 Negligibility is a local property

lemma *locally_negligible_alt*:
negligible S ⟷ (∀ *x* ∈ *S*. ∃ *U*. *openin* (*top_of_set S*) *U* ∧ *x* ∈ *U* ∧ *negligible U*)
(**is** *_* = *?rhs*)
proof
assume *negligible S*
then show *?rhs*
using *openin_subtopology_self* **by** *blast*
next
assume *?rhs*
then obtain *U* **where** *ope*: ⋀*x*. *x* ∈ *S* ⟹ *openin* (*top_of_set S*) (*U x*)
and *cov*: ⋀*x*. *x* ∈ *S* ⟹ *x* ∈ *U x*
and *neg*: ⋀*x*. *x* ∈ *S* ⟹ *negligible* (*U x*)
by *metis*
obtain ℱ **where** ℱ ⊆ *U* ‘ *S countable* ℱ **and** *eq*: ⋃ ℱ = ⋃ (*U* ‘ *S*)
using *ope* **by** (*force intro*: *Lindelof_openin* [*of U* ‘ *S S*])
then have *negligible* (⋃ ℱ)
by (*metis imageE neg negligible_countable_Union subset_eq*)
with *eq* **have** *negligible* (⋃ (*U* ‘ *S*))
by *metis*
moreover have *S* ⊆ ⋃ (*U* ‘ *S*)

using *cov* **by** *blast*
ultimately show *negligible S*
using *negligible_subset* **by** *blast*
qed

lemma *locally_negligible*: *locally negligible S* ⟷ *negligible S*
unfolding *locally_def*
by (*metis locally_negligible_alt negligible_subset openin_imp_subset openin_subtopology_self*)

6.16.14 Integral bounds

lemma *set_integral_norm_bound*:
fixes *f* :: _ ⇒ 'a :: {*banach*, *second_countable_topology*}
shows *set_integrable M k f* ⟹ *norm* (*LINT x*:*k*|*M*. *f x*) ≤ *LINT x*:*k*|*M*. *norm* (*f x*)
using *integral_norm_bound*[*of M* λ*x*. *indicator k x* ∗$_R$ *f x*] **by** (*simp add*: *set_lebesgue_integral_def*)

lemma *set_integral_finite_UN_AE*:
fixes *f* :: _ ⇒ _ :: {*banach*, *second_countable_topology*}
assumes *finite I*
and *ae*: ⋀*i j*. *i* ∈ *I* ⟹ *j* ∈ *I* ⟹ *AE x in M*. (*x* ∈ *A i* ∧ *x* ∈ *A j*) ⟶ *i* = *j*
and [*measurable*]: ⋀*i*. *i* ∈ *I* ⟹ *A i* ∈ *sets M*
and *f*: ⋀*i*. *i* ∈ *I* ⟹ *set_integrable M* (*A i*) *f*
shows *LINT x*:(⋃ *i*∈*I*. *A i*)|*M*. *f x* = (∑ *i*∈*I*. *LINT x*:*A i*|*M*. *f x*)
using ‹*finite I*› *order_refl*[*of I*]
proof (*induction I rule*: *finite_subset_induct'*)
case (*insert i I'*)
have *AE x in M*. (∀ *j*∈*I'*. *x* ∈ *A i* ⟶ *x* ∉ *A j*)
proof (*intro AE_ball_countable*[*THEN iffD2*] *ballI*)
fix *j* **assume** *j* ∈ *I'*
with ‹*I'* ⊆ *I*› ‹*i* ∉ *I'*› **have** *i* ≠ *j j* ∈ *I*
by *auto*
then show *AE x in M*. *x* ∈ *A i* ⟶ *x* ∉ *A j*
using *ae*[*of i j*] ‹*i* ∈ *I*› **by** *auto*
qed (*use* ‹*finite I'*› **in** ‹*rule countable_finite*›)
then have *AE x*∈*A i in M*. ∀ *xa*∈*I'*. *x* ∉ *A xa*
by *auto*
with *insert.hyps insert.IH*[*symmetric*]
show *?case*
by (*auto intro*!: *set_integral_Un_AE sets.finite_UN f set_integrable_UN*)
qed (*simp add*: *set_lebesgue_integral_def*)

lemma *set_integrable_norm*:
fixes *f* :: 'a ⇒ 'b::{*banach*, *second_countable_topology*}
assumes *f*: *set_integrable M k f* **shows** *set_integrable M k* (λ*x*. *norm* (*f x*))
using *integrable_norm f* **by** (*force simp add*: *set_integrable_def*)

lemma *absolutely_integrable_bounded_variation*:

fixes f :: $'a$::*euclidean_space* ⇒ $'b$::*euclidean_space*
assumes f: f *absolutely_integrable_on UNIV*
obtains B **where** ∀ d. d *division_of* (⋃ d) ⟶ *sum* (λk. *norm* (*integral* k f)) d ≤ B
proof (*rule that*[*of integral UNIV* (λx. *norm* (f x))]; *safe*)
fix d :: $'a$ *set set* **assume** d: d *division_of* ⋃ d
have ∗: k ∈ d ⟹ f *absolutely_integrable_on* k **for** k
using f[*THEN set_integrable_subset*, *of* k] *division_ofD*(*2*,*4*)[*OF* d, *of* k] **by** *auto*
note d' = *division_ofD*[*OF* d]
have (∑ k∈d. *norm* (*integral* k f)) = (∑ k∈d. *norm* (*LINT* x:k|*lebesgue*. f x))
by (*intro sum.cong refl arg_cong*[**where** f=*norm*] *set_lebesgue_integral_eq_integral*(*2*)[*symmetric*] ∗)
also have ... ≤ (∑ k∈d. *LINT* x:k|*lebesgue*. *norm* (f x))
by (*intro sum_mono set_integral_norm_bound* ∗)
also have ... = (∑ k∈d. *integral* k (λx. *norm* (f x)))
by (*intro sum.cong refl set_lebesgue_integral_eq_integral*(*2*) *set_integrable_norm* ∗)
also have ... ≤ *integral* (⋃ d) (λx. *norm* (f x))
using *integrable_on_subdivision*[*OF* d] *assms* f **unfolding** *absolutely_integrable_on_def*
by (*subst integral_combine_division_topdown*[*OF* _ d]) *auto*
also have ... ≤ *integral UNIV* (λx. *norm* (f x))
using *integrable_on_subdivision*[*OF* d] *assms* **unfolding** *absolutely_integrable_on_def*
by (*intro integral_subset_le*) *auto*
finally show (∑ k∈d. *norm* (*integral* k f)) ≤ *integral UNIV* (λx. *norm* (f x)) **.**
qed

lemma *absdiff_norm_less*:
assumes *sum* (λx. *norm* (f x − g x)) S < e
shows |*sum* (λx. *norm*(f x)) S − *sum* (λx. *norm*(g x)) S| < e (**is** *?lhs* < e)
proof −
have *?lhs* ≤ (∑ i∈S. |*norm* (f i) − *norm* (g i)|)
by (*metis* (*no_types*) *sum_abs sum_subtractf*)
also have ... ≤ (∑ x∈S. *norm* (f x − g x))
by (*simp add*: *norm_triangle_ineq3 sum_mono*)
also have ... < e
using *assms*(*1*) **by** *blast*
finally show *?thesis* **.**
qed

proposition *bounded_variation_absolutely_integrable_interval*:
fixes f :: $'n$::*euclidean_space* ⇒ $'m$::*euclidean_space*
assumes f: f *integrable_on cbox a b*
and ∗: ⋀d. d *division_of* (*cbox a b*) ⟹ *sum* (λK. *norm*(*integral* K f)) d ≤ B
shows f *absolutely_integrable_on cbox a b*
proof −
let *?f* = λd. ∑ K∈d. *norm* (*integral* K f) **and** *?D* = {d. d *division_of* (*cbox a b*)}
have *D_1*: *?D* ≠ {}

```
    by (rule elementary_interval[of a b]) auto
  have D_2: bdd_above (?f`?D)
    by (metis * mem_Collect_eq bdd_aboveI2)
  note D = D_1 D_2
  let ?S = SUP x∈?D. ?f x
  have *: ∃γ. gauge γ ∧
            (∀p. p tagged_division_of cbox a b ∧
                 γ fine p ⟶
                 norm ((∑(x,k) ∈ p. content k *R norm (f x)) − ?S) < e)
    if e: e > 0 for e
  proof −
    have ?S − e/2 < ?S using ‹e > 0› by simp
    then obtain d where d: d division_of (cbox a b) ?S − e/2 < (∑k∈d. norm
(integral k f))
      unfolding less_cSUP_iff[OF D] by auto
    note d' = division_ofD[OF this(1)]

    have ∃e>0. ∀i∈d. x ∉ i ⟶ ball x e ∩ i = {} for x
    proof −
      have ∃d'>0. ∀x'∈⋃{i ∈ d. x ∉ i}. d' ≤ dist x x'
      proof (rule separate_point_closed)
        show closed (⋃{i ∈ d. x ∉ i})
          using d' by force
        show x ∉ ⋃{i ∈ d. x ∉ i}
          by auto
      qed
      then show ?thesis
        by force
    qed
    then obtain k where k: ⋀x. 0 < k x ⋀i x. ⟦i ∈ d; x ∉ i⟧ ⟹ ball x (k x) ∩
i = {}
      by metis
    have e/2 > 0
      using e by auto
    with Henstock_lemma[OF f]
    obtain γ where g: gauge γ
      ⋀p. ⟦p tagged_partial_division_of cbox a b; γ fine p⟧
            ⟹ (∑(x,k) ∈ p. norm (content k *R f x − integral k f)) < e/2
      by (metis (no_types, lifting))
    let ?g = λx. γ x ∩ ball x (k x)
    show ?thesis
    proof (intro exI conjI allI impI)
      show gauge ?g
        using g(1) k(1) by (auto simp: gauge_def)
    next
      fix p
      assume p tagged_division_of (cbox a b) ∧ ?g fine p
      then have p: p tagged_division_of cbox a b γ fine p (λx. ball x (k x)) fine p
        by (auto simp: fine_Int)
```

```
    note p' = tagged_division_ofD[OF p(1)]
    define p' where p' = {(x,k) | x k. ∃i l. x ∈ i ∧ i ∈ d ∧ (x,l) ∈ p ∧ k = i ∩
l}
    have gp': γ fine p'
      using p(2) by (auto simp: p'_def fine_def)
    have p'': p' tagged_division_of (cbox a b)
    proof (rule tagged_division_ofI)
      show finite p'
      proof (rule finite_subset)
        show p' ⊆ (λ(k, x, l). (x, k ∩ l)) ` (d × p)
          by (force simp: p'_def image_iff)
        show finite ((λ(k, x, l). (x, k ∩ l)) ` (d × p))
          by (simp add: d'(1) p'(1))
      qed
    next
      fix x K
      assume (x, K) ∈ p'
      then have ∃i l. x ∈ i ∧ i ∈ d ∧ (x, l) ∈ p ∧ K = i ∩ l
        unfolding p'_def by auto
      then obtain i l where il: x ∈ i i ∈ d (x, l) ∈ p K = i ∩ l by blast
      show x ∈ K and K ⊆ cbox a b
        using p'(2−3)[OF il(3)] il by auto
      show ∃a b. K = cbox a b
      unfolding il using p'(4)[OF il(3)] d'(4)[OF il(2)] by (meson Int_interval)
    next
      fix x1 K1
      assume (x1, K1) ∈ p'
      then have ∃i l. x1 ∈ i ∧ i ∈ d ∧ (x1, l) ∈ p ∧ K1 = i ∩ l
        unfolding p'_def by auto
       then obtain i1 l1 where il1: x1 ∈ i1 i1 ∈ d (x1, l1) ∈ p K1 = i1 ∩ l1
by blast
      fix x2 K2
      assume (x2,K2) ∈ p'
      then have ∃i l. x2 ∈ i ∧ i ∈ d ∧ (x2, l) ∈ p ∧ K2 = i ∩ l
        unfolding p'_def by auto
       then obtain i2 l2 where il2: x2 ∈ i2 i2 ∈ d (x2, l2) ∈ p K2 = i2 ∩ l2
by blast
      assume (x1, K1) ≠ (x2, K2)
      then have interior i1 ∩ interior i2 = {} ∨ interior l1 ∩ interior l2 = {}
        using d'(5)[OF il1(2) il2(2)] p'(5)[OF il1(3) il2(3)]  by (auto simp: il1
il2)
      then show interior K1 ∩ interior K2 = {}
        unfolding il1 il2 by auto
    next
      have ∗: ∀(x, X) ∈ p'. X ⊆ cbox a b
        unfolding p'_def using d' by blast
      show ⋃{K. ∃x. (x, K) ∈ p'} = cbox a b
      proof
        show ⋃{k. ∃x. (x, k) ∈ p'} ⊆ cbox a b
```

```
        using ∗ by auto
      next
        show cbox a b ⊆ ⋃{k. ∃x. (x, k) ∈ p'}
        proof
          fix y
          assume y: y ∈ cbox a b
          obtain x L where xl: (x, L) ∈ p y ∈ L
            using y unfolding p'(6)[symmetric] by auto
          obtain I where i: I ∈ d y ∈ I
            using y unfolding d'(6)[symmetric] by auto
          have x ∈ I
            using fineD[OF p(3) xl(1)] using k(2) i xl by auto
          then show y ∈ ⋃{K. ∃x. (x, K) ∈ p'}
          proof −
            obtain x l where xl: (x, l) ∈ p y ∈ l
              using y unfolding p'(6)[symmetric] by auto
            obtain i where i: i ∈ d y ∈ i
              using y unfolding d'(6)[symmetric] by auto
            have x ∈ i
              using fineD[OF p(3) xl(1)] using k(2) i xl by auto
            then show ?thesis
            unfolding p'_def by (rule_tac X=i ∩ l in UnionI) (use i xl in auto)
          qed
        qed
      qed
    qed
    then have sum_less_e2: (∑(x,K) ∈ p'. norm (content K ∗R f x − integral
K f)) < e/2
      using g(2) gp' tagged_division_of_def by blast

    have in_p': (x, I ∩ L) ∈ p' if x: (x, L) ∈ p I ∈ d and y: y ∈ I y ∈ L
      for x I L y
    proof −
      have x ∈ I
        using fineD[OF p(3) that(1)] k(2)[OF ‹I ∈ d›] y by auto
      with x have (∃i l. x ∈ i ∧ i ∈ d ∧ (x, l) ∈ p ∧ I ∩ L = i ∩ l)
        by blast
      then have (x, I ∩ L) ∈ p'
        by (simp add: p'_def)
      with y show ?thesis by auto
    qed
    moreover
    have Ex_p_p': ∃y i l. (x, K) = (y, i ∩ l) ∧ (y, l) ∈ p ∧ i ∈ d ∧ i ∩ l ≠ {}
      if xK: (x,K) ∈ p' for x K
    proof −
      obtain i l where il: x ∈ i i ∈ d (x, l) ∈ p K = i ∩ l
        using xK unfolding p'_def by auto
      then show ?thesis
        using p'(2) by fastforce
```

qed
ultimately have *p'alt*: $p' = \{(x,\ I \cap L) \mid x\ I\ L.\ (x,L) \in p \wedge I \in d \wedge I \cap L \neq \{\}\}$
by *auto*
have *sum_p'*: $(\sum (x,K) \in p'.\ norm\ (integral\ K\ f)) = (\sum k \in snd\ `\ p'.\ norm\ (integral\ k\ f))$
proof (*rule sum.over_tagged_division_lemma*[*OF p''*])
show $\bigwedge u\ v.\ box\ u\ v = \{\} \Longrightarrow norm\ (integral\ (cbox\ u\ v)\ f) = 0$
by (*auto intro*: *integral_null simp*: *content_eq_0_interior*)
qed
have *snd_p_div*: *snd ` p division_of cbox a b*
by (*rule division_of_tagged_division*[*OF p(1)*])
note *snd_p* = *division_ofD*[*OF snd_p_div*]
have *fin_d_sndp*: *finite* $(d \times snd\ `\ p)$
by (*simp add*: *d'(1) snd_p(1)*)

have $*$: $\bigwedge sni\ sni'\ sf\ sf'.\ [\![|sf' - sni'| < e/2;\ ?S - e/2 < sni;\ sni' \leq ?S;\ sni \leq sni';\ sf' = sf]\!] \Longrightarrow |sf - ?S| < e$
by *arith*
show $norm\ ((\sum (x,k) \in p.\ content\ k *_R norm\ (f\ x)) - ?S) < e$
unfolding *real_norm_def*
proof (*rule* $*$)
show $|(\sum (x,K) \in p'.\ norm\ (content\ K *_R f\ x)) - (\sum (x,k) \in p'.\ norm\ (integral\ k\ f))| < e/2$
using *p'' sum_less_e2* **unfolding** *split_def* **by** (*force intro*!: *absdiff_norm_less*)
show $(\sum (x,k) \in p'.\ norm\ (integral\ k\ f)) \leq ?S$
by (*auto simp*: *sum_p' division_of_tagged_division*[*OF p''*] *D intro*!: *cSUP_upper*)
show $(\sum k \in d.\ norm\ (integral\ k\ f)) \leq (\sum (x,k) \in p'.\ norm\ (integral\ k\ f))$
proof −
have $*$: $\{k \cap l \mid k\ l.\ k \in d \wedge l \in snd\ `\ p\} = (\lambda(k,l).\ k \cap l)\ `\ (d \times snd\ `\ p)$
by *auto*
have $(\sum K \in d.\ norm\ (integral\ K\ f)) \leq (\sum i \in d.\ \sum l \in snd\ `\ p.\ norm\ (integral\ (i \cap l)\ f))$
proof (*rule sum_mono*)
fix *K* **assume** *k*: $K \in d$
from *d'(4)*[*OF this*] **obtain** *u v* **where** *uv*: $K = cbox\ u\ v$ **by** *metis*
define *d'* **where** $d' = \{cbox\ u\ v \cap l \mid l.\ l \in snd\ `\ p \wedge\ cbox\ u\ v \cap l \neq \{\}\}$
have *uvab*: $cbox\ u\ v \subseteq cbox\ a\ b$
using *d(1) k uv* **by** *blast*
have *d'_div*: *d' division_of cbox u v*
unfolding *d'_def* **by** (*rule division_inter_1* [*OF snd_p_div uvab*])
moreover have $norm\ (\sum i \in d'.\ integral\ i\ f) \leq (\sum k \in d'.\ norm\ (integral\ k\ f))$
by (*simp add*: *sum_norm_le*)
moreover have *f integrable_on K*
using *f integrable_on_subcbox uv uvab* **by** *blast*
moreover have *d' division_of K*

```
      using d'_div uv by blast
    ultimately have norm (integral K f) ≤ sum (λk. norm (integral k f)) d'
      by (simp add: integral_combine_division_topdown)
    also have ... = (∑ I∈{K ∩ L |L. L ∈ snd ` p}. norm (integral I f))
    proof (rule sum.mono_neutral_left)
      show finite {K ∩ L |L. L ∈ snd ` p}
        by (simp add: snd_p(1))
      show ∀ i∈{K ∩ L |L. L ∈ snd ` p} − d'. norm (integral i f) = 0
        d' ⊆ {K ∩ L |L. L ∈ snd ` p}
        using d'_def image_eqI uv by auto
    qed
    also have ... = (∑ l∈snd ` p. norm (integral (K ∩ l) f))
      unfolding Setcompr_eq_image
    proof (rule sum.reindex_nontrivial [unfolded o_def])
      show finite (snd ` p)
        using snd_p(1) by blast
      show norm (integral (K ∩ l) f) = 0
        if l ∈ snd ` p y ∈ snd ` p l ≠ y K ∩ l = K ∩ y for l y
      proof −
        have interior (K ∩ l) ⊆ interior (l ∩ y)
          by (metis Int_lower2 interior_mono le_inf_iff that(4))
        then have interior (K ∩ l) = {}
          by (simp add: snd_p(5) that)
        moreover from d'(4)[OF k] snd_p(4)[OF that(1)]
        obtain u1 v1 u2 v2
          where uv: K = cbox u1 u2 l = cbox v1 v2 by metis
        ultimately show ?thesis
          using that integral_null
          unfolding uv Int_interval content_eq_0_interior
          by (metis (mono_tags, lifting) norm_eq_zero)
      qed
    qed
    finally show norm (integral K f) ≤ (∑ l∈snd ` p. norm (integral (K ∩
l) f)) .
  qed
  also have ... = (∑ (i,l) ∈ d × snd ` p. norm (integral (i∩l) f))
    by (simp add: sum.cartesian_product)
  also have ... = (∑ x ∈ d × snd ` p. norm (integral (case_prod (∩) x) f))
    by (force simp: split_def intro!: sum.cong)
  also have ... = (∑ k∈{i ∩ l |i l. i ∈ d ∧ l ∈ snd ` p}. norm (integral k
f))
  proof −
    have eq0: (integral (l1 ∩ k1) f) = 0
      if l1 ∩ k1 = l2 ∩ k2 (l1, k1) ≠ (l2, k2)
         l1 ∈ d (j1,k1) ∈ p l2 ∈ d (j2,k2) ∈ p
      for l1 l2 k1 k2 j1 j2
    proof −
      obtain u1 v1 u2 v2 where uv: l1 = cbox u1 u2 k1 = cbox v1 v2
        using ‹(j1, k1) ∈ p› ‹l1 ∈ d› d'(4) p'(4) by blast
```

```
            have l1 ≠ l2 ∨ k1 ≠ k2
              using that by auto
             then have interior k1 ∩ interior k2 = {} ∨ interior l1 ∩ interior l2
= {}
             by (meson d'(5) old.prod.inject p'(5) that(3) that(4) that(5) that(6))
            moreover have interior (l1 ∩ k1) = interior (l2 ∩ k2)
              by (simp add: that(1))
            ultimately have interior(l1 ∩ k1) = {}
              by auto
            then show ?thesis
            unfolding uv Int_interval content_eq_0_interior[symmetric] by auto
          qed
          show ?thesis
            unfolding *
                apply (rule sum.reindex_nontrivial [OF fin_d_sndp, symmetric,
unfolded o_def])
            apply clarsimp
            by (metis eq0 fst_conv snd_conv)
        qed
        also have ... = (∑(x,k) ∈ p'. norm (integral k f))
          unfolding sum_p'
        proof (rule sum.mono_neutral_right)
          show finite {i ∩ l |i l. i ∈ d ∧ l ∈ snd ` p}
            by (metis * finite_imageI[OF fin_d_sndp])
          show snd ` p' ⊆ {i ∩ l |i l. i ∈ d ∧ l ∈ snd ` p}
            by (clarsimp simp: p'_def) (metis image_eqI snd_conv)
          show ∀ i∈{i ∩ l |i l. i ∈ d ∧ l ∈ snd ` p} − snd ` p'. norm (integral i f)
= 0
              by clarsimp (metis Henstock_Kurzweil_Integration.integral_empty
disjoint_iff image_eqI in_p' snd_conv)
        qed
        finally show ?thesis .
      qed
      show (∑(x,k) ∈ p'. norm (content k *R f x)) = (∑(x,k) ∈ p. content k *R
norm (f x))
      proof −
        let ?S = {(x, i ∩ l) |x i l. (x, l) ∈ p ∧ i ∈ d}
        have *: ?S = (λ(xl,i). (fst xl, snd xl ∩ i)) ` (p × d)
          by force
        have fin_pd: finite (p × d)
          using finite_cartesian_product[OF p'(1) d'(1)] by metis
        have (∑(x,k) ∈ p'. norm (content k *R f x)) = (∑(x,k) ∈ ?S. |content
k| * norm (f x))
          unfolding norm_scaleR
        proof (rule sum.mono_neutral_left)
          show finite {(x, i ∩ l) |x i l. (x, l) ∈ p ∧ i ∈ d}
            by (simp add: * fin_pd)
        qed (use p'alt in ‹force+›)
        also have ... = (∑((x,l),i)∈p × d. |content (l ∩ i)| * norm (f x))
```

```
      proof −
        have |content (l1 ∩ k1)| ∗ norm (f x1) = 0
          if (x1, l1) ∈ p (x2, l2) ∈ p k1 ∈ d k2 ∈ d
            x1 = x2 l1 ∩ k1 = l2 ∩ k2 x1 ≠ x2 ∨ l1 ≠ l2 ∨ k1 ≠ k2
          for x1 l1 k1 x2 l2 k2
        proof −
          obtain u1 v1 u2 v2 where uv: k1 = cbox u1 u2 l1 = cbox v1 v2
            by (meson ‹(x1, l1) ∈ p› ‹k1 ∈ d› d(1) division_ofD(4) p′(4))
          have l1 ≠ l2 ∨ k1 ≠ k2
            using that by auto
           then have interior k1 ∩ interior k2 = {} ∨ interior l1 ∩ interior l2
= {}
            using that p′(5) d′(5) by (metis snd_conv)
          moreover have interior (l1 ∩ k1) = interior (l2 ∩ k2)
            unfolding that ..
          ultimately have interior (l1 ∩ k1) = {}
            by auto
          then show |content (l1 ∩ k1)| ∗ norm (f x1) = 0
          unfolding uv Int_interval content_eq_0_interior[symmetric] by auto
        qed
        then show ?thesis
          unfolding ∗
          apply (subst sum.reindex_nontrivial [OF fin_pd])
          unfolding split_paired_all o_def split_def prod.inject
          by force+
      qed
      also have ... = (∑(x,k) ∈ p. content k ∗R norm (f x))
      proof −
        have sumeq: (∑i∈d. content (l ∩ i) ∗ norm (f x)) = content l ∗ norm
(f x)
          if (x, l) ∈ p for x l
        proof −
          note xl = p′(2−4)[OF that]
          then obtain u v where uv: l = cbox u v by blast
          have (∑i∈d. |content (l ∩ i)|) = (∑k∈d. content (k ∩ cbox u v))
            by (simp add: Int_commute uv)
          also have ... = sum content {k ∩ cbox u v| k. k ∈ d}
          proof −
            have eq0: content (k ∩ cbox u v) = 0
              if k ∈ d y ∈ d k ≠ y and eq: k ∩ cbox u v = y ∩ cbox u v for k y
            proof −
              from d′(4)[OF that(1)] d′(4)[OF that(2)]
              obtain α β where α: k ∩ cbox u v = cbox α β
                by (meson Int_interval)
              have {} = interior ((k ∩ y) ∩ cbox u v)
                by (simp add: d′(5) that)
              also have ... = interior (y ∩ (k ∩ cbox u v))
                by auto
              also have ... = interior (k ∩ cbox u v)
```

```
                unfolding eq by auto
              finally show ?thesis
                unfolding α content_eq_0_interior ..
            qed
            then show ?thesis
              unfolding Setcompr_eq_image
              by (fastforce intro: sum.reindex_nontrivial [OF ‹finite d›, unfolded
o_def, symmetric])
          qed
          also have ... = sum content {cbox u v ∩ k |k. k ∈ d ∧ cbox u v ∩ k
≠ {}}
          proof (rule sum.mono_neutral_right)
            show finite {k ∩ cbox u v |k. k ∈ d}
              by (simp add: d'(1))
          qed (fastforce simp: inf.commute)+
          finally have (∑ i∈d. |content (l ∩ i)|) = content (cbox u v)
            using additive_content_division[OF division_inter_1[OF d(1)]] uv
xl(2) by auto
          then show (∑ i∈d. content (l ∩ i) * norm (f x)) = content l * norm
(f x)
            unfolding sum_distrib_right[symmetric] using uv by auto
        qed
        show ?thesis
        by (subst sum_Sigma_product[symmetric]) (auto intro!: sumeq sum.cong
p' d')
      qed
      finally show ?thesis .
    qed
  qed (rule d)
 qed
qed
 qed
 then show ?thesis
  using absolutely_integrable_onI [OF f has_integral_integrable] has_integral[of
_ ?S]
  by blast
qed


lemma bounded_variation_absolutely_integrable:
 fixes f :: 'n::euclidean_space ⇒ 'm::euclidean_space
 assumes f integrable_on UNIV
  and ∀ d. d division_of (⋃ d) ⟶ sum (λk. norm (integral k f)) d ≤ B
 shows f absolutely_integrable_on UNIV
proof (rule absolutely_integrable_onI, fact)
 let ?f = λD. ∑ k∈D. norm (integral k f) and ?D = {d. d division_of (⋃ d)}
 define SDF where SDF ≡ SUP d∈?D. ?f d
 have D_1: ?D ≠ {}
  by (rule elementary_interval) auto
 have D_2: bdd_above (?f`?D)
```

```
    using assms(2) by auto
  have f_int: ⋀a b. f absolutely_integrable_on cbox a b
    using assms integrable_on_subcbox
    by (blast intro!: bounded_variation_absolutely_integrable_interval)
  have ∃B>0. ∀a b. ball 0 B ⊆ cbox a b ⟶
                  |integral (cbox a b) (λx. norm (f x)) − SDF| < e
    if 0 < e for e
  proof −
    have ∃y ∈ ?f ‘ ?D. ¬ y ≤ SDF − e
    proof (rule ccontr)
      assume ¬ ?thesis
      then have SDF ≤ SDF − e
        unfolding SDF_def
        by (metis (mono_tags) D_1 cSUP_least image_eqI)
      then show False
        using that by auto
    qed
    then obtain d K where ddiv: d division_of ⋃d and K = ?f d SDF − e < K
      by (auto simp add: image_iff not_le)
    then have d: SDF − e < ?f d
      by auto
    note d'=division_ofD[OF ddiv]
    have bounded (⋃d)
      using ddiv by blast
    then obtain K where K: 0 < K ∀x∈⋃d. norm x ≤ K
      using bounded_pos by blast
    show ?thesis
    proof (intro conjI impI allI exI)
      fix a b :: 'n
      assume ab: ball 0 (K + 1) ⊆ cbox a b
      have ∗: ⋀s s1. ⟦SDF − e < s1; s1 ≤ s; s < SDF + e⟧ ⟹ |s − SDF| < e
        by arith
      show |integral (cbox a b) (λx. norm (f x)) − SDF| < e
        unfolding real_norm_def
      proof (rule ∗ [OF d])
        have ?f d ≤ sum (λk. integral k (λx. norm (f x))) d
        proof (intro sum_mono)
          fix k assume k ∈ d
          with d'(4) f_int show norm (integral k f) ≤ integral k (λx. norm (f x))
         by (force simp: absolutely_integrable_on_def integral_norm_bound_integral)
        qed
        also have ... = integral (⋃d) (λx. norm (f x))
            by (metis (full_types) absolutely_integrable_on_def d'(4) ddiv f_int
integral_combine_division_bottomup)
        also have ... ≤ integral (cbox a b) (λx. norm (f x))
        proof −
          have ⋃d ⊆ cbox a b
            using K(2) ab by fastforce
          then show ?thesis
```

using *integrable_on_subdivision*[*OF ddiv*] *f_int*[*of a b*] **unfolding** *absolutely_integrable_on_def*
by (*auto intro*!: *integral_subset_le*)
qed
finally show *?f d* ≤ *integral* (*cbox a b*) (λ*x*. *norm* (*f x*)) **.**
next
have *e/2>0*
using ‹*e* > *0*› **by** *auto*
moreover
have *f*: *f integrable_on cbox a b* (λ*x*. *norm* (*f x*)) *integrable_on cbox a b*
using *f_int* **by** (*auto simp*: *absolutely_integrable_on_def*)
ultimately obtain *d1* **where** *gauge d1*
and *d1*: ⋀*p*. ⟦*p tagged_division_of* (*cbox a b*); *d1 fine p*⟧ ⟹
norm ((∑ (*x*,*k*) ∈ *p*. *content k* ∗*R norm* (*f x*)) − *integral* (*cbox a b*) (λ*x*. *norm* (*f x*))) < *e/2*
unfolding *has_integral_integral has_integral* **by** *meson*
obtain *d2* **where** *gauge d2*
and *d2*: ⋀*p*. ⟦*p tagged_partial_division_of* (*cbox a b*); *d2 fine p*⟧ ⟹
(∑ (*x*,*k*) ∈ *p*. *norm* (*content k* ∗*R f x* − *integral k f*)) < *e/2*
by (*blast intro*: *Henstock_lemma* [*OF f*(*1*) ‹*e/2>0*›])
obtain *p* **where**
p: *p tagged_division_of* (*cbox a b*) *d1 fine p d2 fine p*
by (*rule fine_division_exists* [*OF gauge_Int* [*OF* ‹*gauge d1*› ‹*gauge d2*›], *of a b*])
(*auto simp add*: *fine_Int*)
have ∗: ⋀*sf sf*′ *si di*. ⟦*sf*′ = *sf*; *si* ≤ *SDF*; |*sf* − *si*| < *e/2*;
|*sf*′ − *di*| < *e/2*⟧ ⟹ *di* < *SDF* + *e*
by *arith*
have *integral* (*cbox a b*) (λ*x*. *norm* (*f x*)) < *SDF* + *e*
proof (*rule* ∗)
show |(∑ (*x*,*k*)∈*p*. *norm* (*content k* ∗*R f x*)) − (∑ (*x*,*k*)∈*p*. *norm* (*integral k f*))| < *e/2*
unfolding *split_def*
proof (*rule absdiff_norm_less*)
show (∑ *p*∈*p*. *norm* (*content* (*snd p*) ∗*R f* (*fst p*) − *integral* (*snd p*) *f*)) < *e/2*
using *d2*[*of p*] *p*(*1*,*3*) **by** (*auto simp*: *tagged_division_of_def split_def*)
qed
show |(∑ (*x*,*k*) ∈ *p*. *content k* ∗*R norm* (*f x*)) − *integral* (*cbox a b*) (λ*x*. *norm*(*f x*))| < *e/2*
using *d1*[*OF p*(*1*,*2*)] **by** (*simp only*: *real_norm_def*)
show (∑ (*x*,*k*) ∈ *p*. *content k* ∗*R norm* (*f x*)) = (∑ (*x*,*k*) ∈ *p*. *norm* (*content k* ∗*R f x*))
by (*auto simp*: *split_paired_all sum.cong* [*OF refl*])
have (∑ (*x*,*k*) ∈ *p*. *norm* (*integral k f*)) = (∑ *k*∈*snd* ‘ *p*. *norm* (*integral k f*))
apply (*rule sum.over_tagged_division_lemma*[*OF p*(*1*)])
by (*metis Henstock_Kurzweil_Integration.integral_empty integral_open_interval norm_zero*)

```
    also have ... ≤ SDF
      using partial_division_of_tagged_division[of p cbox a b] p(1)
    by (auto simp: SDF_def tagged_partial_division_of_def intro!: cSUP_upper2
D_1 D_2)
    finally show (∑(x,k) ∈ p. norm (integral k f)) ≤ SDF .
   qed
   then show integral (cbox a b) (λx. norm (f x)) < SDF + e
    by simp
  qed
 qed (use K in auto)
qed
moreover have ⋀a b. (λx. norm (f x)) integrable_on cbox a b
 using absolutely_integrable_on_def f_int by auto
ultimately
have ((λx. norm (f x)) has_integral SDF) UNIV
 by (auto simp: has_integral_alt')
then show (λx. norm (f x)) integrable_on UNIV
 by blast
qed
```

6.16.15 Outer and inner approximation of measurable sets by well-behaved sets.

```
proposition measurable_outer_intervals_bounded:
 assumes S ∈ lmeasurable S ⊆ cbox a b e > 0
 obtains 𝒟
 where countable 𝒟
       ⋀K. K ∈ 𝒟 ⟹ K ⊆ cbox a b ∧ K ≠ {} ∧ (∃c d. K = cbox c d)
       pairwise (λA B. interior A ∩ interior B = {}) 𝒟
       ⋀u v. cbox u v ∈ 𝒟 ⟹ ∃n. ∀i ∈ Basis. v · i − u · i = (b · i − a · i)/2^n
       ⋀K. ⟦K ∈ 𝒟; box a b ≠ {}⟧ ⟹ interior K ≠ {}
       S ⊆ ⋃𝒟 ⋃𝒟 ∈ lmeasurable measure lebesgue (⋃𝒟) ≤ measure lebesgue S
+ e
proof (cases box a b = {})
 case True
 show ?thesis
 proof (cases cbox a b = {})
  case True
  with assms have [simp]: S = {}
   by auto
  show ?thesis
  proof
   show countable {}
    by simp
  qed (use ‹e > 0› in auto)
 next
  case False
  show ?thesis
  proof
```

```
      show countable {cbox a b}
        by simp
      show ⋀u v. cbox u v ∈ {cbox a b} ⟹ ∃n. ∀i∈Basis. v • i − u • i = (b • i
− a • i)/2 ^ n
        using False by (force simp: eq_cbox intro: exI [where x=0])
      show measure lebesgue (⋃{cbox a b}) ≤ measure lebesgue S + e
        using assms by (simp add: sum_content.box_empty_imp [OF True])
    qed (use assms ‹cbox a b ≠ {}› in auto)
  qed
next
  case False
  let ?μ = measure lebesgue
  have S ∩ cbox a b ∈ lmeasurable
    using ‹S ∈ lmeasurable› by blast
  then have indS_int: (indicator S has_integral (?μ S)) (cbox a b)
    by (metis integral_indicator ‹S ⊆ cbox a b› has_integral_integrable_integral
inf.orderE integrable_on_indicator)
  with ‹e > 0› obtain γ where gauge γ and γ:
    ⋀𝒟. ⟦𝒟 tagged_division_of (cbox a b); γ fine 𝒟⟧ ⟹ norm ((∑(x,K)∈𝒟.
content(K) *R indicator S x) − ?μ S) < e
    by (force simp: has_integral)
  have inteq: integral (cbox a b) (indicat_real S) = integral UNIV (indicator S)
    using assms by (metis has_integral_iff indS_int lmeasure_integral_UNIV)
  obtain 𝒟 where 𝒟: countable 𝒟 ⋃𝒟 ⊆ cbox a b
          and cbox: ⋀K. K ∈ 𝒟 ⟹ interior K ≠ {} ∧ (∃c d. K = cbox c d)
          and djointish: pairwise (λA B. interior A ∩ interior B = {}) 𝒟
          and covered: ⋀K. K ∈ 𝒟 ⟹ ∃x ∈ S ∩ K. K ⊆ γ x
          and close: ⋀u v. cbox u v ∈ 𝒟 ⟹ ∃n. ∀i ∈ Basis. v•i − u•i = (b•i −
a•i)/2^n
          and covers: S ⊆ ⋃𝒟
    using covering_lemma [of S a b γ] ‹gauge γ› ‹box a b ≠ {}› assms by force
  show ?thesis
  proof
    show ⋀K. K ∈ 𝒟 ⟹ K ⊆ cbox a b ∧ K ≠ {} ∧ (∃c d. K = cbox c d)
      by (meson Sup_le_iff 𝒟(2) cbox interior_empty)
    have negl_int: negligible(K ∩ L) if K ∈ 𝒟 L ∈ 𝒟 K ≠ L for K L
    proof −
      have interior K ∩ interior L = {}
        using djointish pairwiseD that by fastforce
      moreover obtain u v x y where K = cbox u v L = cbox x y
        using cbox ‹K ∈ 𝒟› ‹L ∈ 𝒟› by blast
      ultimately show ?thesis
        by (simp add: Int_interval box_Int_box negligible_interval(1))
    qed
    have fincase: ⋃ℱ ∈ lmeasurable ∧ ?μ (⋃ℱ) ≤ ?μ S + e if finite ℱ ℱ ⊆ 𝒟
for ℱ
    proof −
      obtain t where t: ⋀K. K ∈ ℱ ⟹ t K ∈ S ∩ K ∧ K ⊆ γ(t K)
        using covered ‹ℱ ⊆ 𝒟› subsetD by metis
```

have $\forall K \in \mathcal{F}.\ \forall L \in \mathcal{F}.\ K \neq L \longrightarrow \mathit{interior}\ K \cap \mathit{interior}\ L = \{\}$
using *that djointish* **by** (*simp add*: *pairwise_def*) (*metis subsetD*)
with *cbox that* $\mathcal{D}$ **have** $\mathcal{F}$*div*: $\mathcal{F}$ *division_of* $(\bigcup \mathcal{F})$
by (*fastforce simp*: *division_of_def dest*: *cbox*)
then have *1*: $\bigcup \mathcal{F} \in \mathit{lmeasurable}$
by *blast*
have *norme*: $\bigwedge p.\ [\![p\ \mathit{tagged\_division\_of}\ \mathit{cbox}\ a\ b;\ \gamma\ \mathit{fine}\ p]\!]$
$\Longrightarrow \mathit{norm}\ ((\sum (x,K) \in p.\ \mathit{content}\ K * \mathit{indicator}\ S\ x) - \mathit{integral}\ (\mathit{cbox}\ a\ b)\ (\mathit{indicator}\ S)) < e$
by (*auto simp*: *lmeasure_integral_UNIV assms inteq dest*: γ)
have $\forall x\ K\ y\ L.\ (x,K) \in (\lambda K.\ (t\ K,K))\ `\ \mathcal{F} \wedge (y,L) \in (\lambda K.\ (t\ K,K))\ `\ \mathcal{F} \wedge (x,K) \neq (y,L) \longrightarrow \mathit{interior}\ K \cap \mathit{interior}\ L = \{\}$
using *that djointish* **by** (*clarsimp simp*: *pairwise_def*) (*metis subsetD*)
with *that* $\mathcal{D}$ **have** *tagged*: $(\lambda K.\ (t\ K,\ K))\ `\ \mathcal{F}$ *tagged_partial_division_of cbox a b*
by (*auto simp*: *tagged_partial_division_of_def dest*: *t cbox*)
have *fine*: γ *fine* $(\lambda K.\ (t\ K,\ K))\ `\ \mathcal{F}$
using *t* **by** (*auto simp*: *fine_def*)
have $*$: $y \leq ?\mu\ S \Longrightarrow |x - y| \leq e \Longrightarrow x \leq ?\mu\ S + e$ **for** *x y*
by *arith*
have $?\mu\ (\bigcup \mathcal{F}) \leq ?\mu\ S + e$
proof (*rule* $*$)
have $(\sum K \in \mathcal{F}.\ ?\mu\ (K \cap S)) = ?\mu\ (\bigcup C \in \mathcal{F}.\ C \cap S)$
proof (*rule measure_negligible_finite_Union_image* [*OF* ‹*finite* $\mathcal{F}$›, *symmetric*])
show $\bigwedge K.\ K \in \mathcal{F} \Longrightarrow K \cap S \in \mathit{lmeasurable}$
using $\mathcal{F}$*div* ‹$S \in \mathit{lmeasurable}$› **by** *blast*
show *pairwise* $(\lambda K\ y.\ \mathit{negligible}\ (K \cap S \cap (y \cap S)))\ \mathcal{F}$
unfolding *pairwise_def*
by (*metis inf.commute inf_sup_aci*(*3*) *negligible_Int subsetCE negl_int* ‹$\mathcal{F} \subseteq \mathcal{D}$›)
qed
also have $\ldots = ?\mu\ (\bigcup \mathcal{F} \cap S)$
by *simp*
also have $\ldots \leq ?\mu\ S$
by (*simp add*: *1* ‹$S \in \mathit{lmeasurable}$› *fmeasurableD measure_mono_fmeasurable sets.Int*)
finally show $(\sum K \in \mathcal{F}.\ ?\mu\ (K \cap S)) \leq ?\mu\ S$ **.**
next
have $?\mu\ (\bigcup \mathcal{F}) = \mathit{sum}\ ?\mu\ \mathcal{F}$
by (*metis* $\mathcal{F}$*div content_division*)
also have $\ldots = (\sum K \in \mathcal{F}.\ \mathit{content}\ K)$
using $\mathcal{F}$*div* **by** (*force intro*: *sum.cong*)
also have $\ldots = (\sum x \in \mathcal{F}.\ \mathit{content}\ x * \mathit{indicator}\ S\ (t\ x))$
using *t* **by** *auto*
finally have *eq1*: $?\mu\ (\bigcup \mathcal{F}) = (\sum x \in \mathcal{F}.\ \mathit{content}\ x * \mathit{indicator}\ S\ (t\ x))$ **.**
have *eq2*: $(\sum K \in \mathcal{F}.\ ?\mu\ (K \cap S)) = (\sum K \in \mathcal{F}.\ \mathit{integral}\ K\ (\mathit{indicator}\ S))$
apply (*rule sum.cong* [*OF refl*])
by (*metis integral_indicator* $\mathcal{F}$*div* ‹$S \in \mathit{lmeasurable}$› *division_ofD*(*4*)

```
fmeasurable.Int inf.commute lmeasurable_cbox)
      have |∑(x,K)∈(λK. (t K, K)) ` F. content K * indicator S x − integral K
(indicator S)| ≤ e
        using Henstock_lemma_part1 [of indicator S::'a⇒real, OF _ ‹e > 0›
‹gauge γ› _ tagged fine]
          indS_int norme by auto
      then show |?μ (⋃F) − (∑ K∈F. ?μ (K ∩ S))| ≤ e
       by (simp add: eq1 eq2 comm_monoid_add_class.sum.reindex inj_on_def
sum_subtractf)
    qed
    with 1 show ?thesis by blast
  qed
  have ⋃D ∈ lmeasurable ∧ ?μ (⋃D) ≤ ?μ S + e
  proof (cases finite D)
    case True
    with fincase show ?thesis
      by blast
  next
    case False
    let ?T = from_nat_into D
    have T: bij_betw ?T UNIV D
      by (simp add: False D(1) bij_betw_from_nat_into)
    have TM: ⋀n. ?T n ∈ lmeasurable
      by (metis False cbox finite.emptyI from_nat_into lmeasurable_cbox)
    have TN: ⋀m n. m ≠ n ⟹ negligible (?T m ∩ ?T n)
     by (simp add: False D(1) from_nat_into infinite_imp_nonempty negl_int)
    have TB: (∑ k≤n. ?μ (?T k)) ≤ ?μ S + e for n
    proof −
      have (∑ k≤n. ?μ (?T k)) = ?μ (⋃ (?T ` {..n}))
     by (simp add: pairwise_def TM TN measure_negligible_finite_Union_image)
      also have ?μ (⋃ (?T ` {..n})) ≤ ?μ S + e
        using fincase [of ?T ` {..n}] T by (auto simp: bij_betw_def)
      finally show ?thesis .
    qed
    have ⋃D ∈ lmeasurable
      by (metis lmeasurable_compact T D(2) bij_betw_def cbox compact_cbox
countable_Un_Int(1) fmeasurableD fmeasurableI2 rangeI)
    moreover
    have ?μ (⋃x. from_nat_into D x) ≤ ?μ S + e
    proof (rule measure_countable_Union_le [OF TM])
      show ?μ (⋃x≤n. from_nat_into D x) ≤ ?μ S + e for n
        by (metis (mono_tags, lifting) False fincase finite.emptyI finite_atMost
finite_imageI from_nat_into imageE subsetI)
    qed
    ultimately show ?thesis by (metis T bij_betw_def)
  qed
  then show ⋃D ∈ lmeasurable measure lebesgue (⋃D) ≤ ?μ S + e by blast+
 qed (use D cbox djointish close covers in auto)
qed
```

6.16.16 Transformation of measure by linear maps

lemma *emeasure_lebesgue_ball_conv_unit_ball*:
fixes $c :: 'a :: euclidean\_space$
assumes $r \geq 0$
shows *emeasure lebesgue* (*ball c r*) =
ennreal ($r \mathbin{\hat{}} DIM('a)$) $*$ *emeasure lebesgue* (*ball* (*0* :: $'a$) *1*)
proof (*cases r = 0*)
case *False*
with *assms* **have** *r*: $r > 0$ **by** *auto*
have *emeasure lebesgue* ((λ*x*. *c* + *x*) ‘ (λ*x*. *r* $*_R$ *x*) ‘ *ball* (*0* :: $'a$) *1*) =
$r \mathbin{\hat{}} DIM('a)$ $*$ *emeasure lebesgue* (*ball* (*0* :: $'a$) *1*)
unfolding *image_image* **using** *emeasure_lebesgue_affine*[*of r c ball 0 1*] *assms*
by (*simp add*: *add_ac*)
also have (λ*x*. *r* $*_R$ *x*) ‘ *ball 0 1* = *ball* (*0* :: $'a$) *r*
using *r* **by** (*subst ball_scale*) *auto*
also have (λ*x*. *c* + *x*) ‘ ... = *ball c r*
by (*subst image_add_ball*) (*simp_all add*: *algebra_simps*)
finally show *?thesis* **by** *simp*
qed *auto*

lemma *content_ball_conv_unit_ball*:
fixes $c :: 'a :: euclidean\_space$
assumes $r \geq 0$
shows *content* (*ball c r*) = $r \mathbin{\hat{}} DIM('a)$ $*$ *content* (*ball* (*0* :: $'a$) *1*)
proof −
have *ennreal* (*content* (*ball c r*)) = *emeasure lebesgue* (*ball c r*)
using *emeasure_lborel_ball_finite*[*of c r*] **by** (*subst emeasure_eq_ennreal_measure*)
auto
also have ... = *ennreal* ($r \mathbin{\hat{}} DIM('a)$) $*$ *emeasure lebesgue* (*ball* (*0* :: $'a$) *1*)
using *assms* **by** (*intro emeasure_lebesgue_ball_conv_unit_ball*) *auto*
also have ... = *ennreal* ($r \mathbin{\hat{}} DIM('a)$ $*$ *content* (*ball* (*0*::$'a$) *1*))
using *emeasure_lborel_ball_finite*[*of 0*::$'a$ *1*] *assms*
by (*subst emeasure_eq_ennreal_measure*) (*auto simp*: *ennreal_mult'*)
finally show *?thesis*
using *assms* **by** (*subst* (*asm*) *ennreal_inj*) *auto*
qed

lemma *measurable_linear_image_interval*:
linear f ⟹ *f* ‘ (*cbox a b*) ∈ *lmeasurable*
by (*metis bounded_linear_image linear_linear bounded_cbox closure_bounded_linear_image closure_cbox compact_closure lmeasurable_compact*)

proposition *measure_linear_sufficient*:
fixes $f :: 'n::euclidean\_space \Rightarrow 'n$
assumes *linear f* **and** *S*: *S* ∈ *lmeasurable*
and *im*: ⋀*a b*. *measure lebesgue* (*f* ‘ (*cbox a b*)) = *m* $*$ *measure lebesgue* (*cbox a b*)
shows *f* ‘ *S* ∈ *lmeasurable* ∧ *m* $*$ *measure lebesgue S* = *measure lebesgue* (*f* ‘ *S*)
using *le_less_linear* [*of 0 m*]

proof
assume $m < 0$
then show *?thesis*
using *im* [*of 0 One*] **by** *auto*
next
assume $m \geq 0$
let $?\mu$ = *measure lebesgue*
show *?thesis*
proof (*cases inj f*)
case *False*
then have $?\mu\ (f\ `\ S) = 0$
using ‹*linear f*› *negligible_imp_measure0 negligible_linear_singular_image*
by *blast*
then have $m * ?\mu\ (cbox\ 0\ (One)) = 0$
by (*metis False* ‹*linear f*› *cbox_borel content_unit im measure_completion negligible_imp_measure0 negligible_linear_singular_image sets_lborel*)
then show *?thesis*
using ‹*linear f*› *negligible_linear_singular_image negligible_imp_measure0 False*
by (*auto simp*: *lmeasurable_iff_has_integral negligible_UNIV*)
next
case *True*
then obtain *h* **where** *linear h* **and** *hf*: $\bigwedge x.\ h\ (f\ x) = x$ **and** *fh*: $\bigwedge x.\ f\ (h\ x) = x$
using ‹*linear f*› *linear_injective_isomorphism* **by** *blast*
have *fBS*: $(f\ `\ S) \in lmeasurable \wedge m * ?\mu\ S = ?\mu\ (f\ `\ S)$
if *bounded S S* $\in$ *lmeasurable* **for** *S*
proof −
obtain *a b* **where** $S \subseteq cbox\ a\ b$
using ‹*bounded S*› *bounded_subset_cbox_symmetric* **by** *metis*
have *fUD*: $(f\ `\ \bigcup \mathcal{D}) \in lmeasurable \wedge\ ?\mu\ (f\ `\ \bigcup \mathcal{D}) = (m * ?\mu\ (\bigcup \mathcal{D}))$
if *countable* $\mathcal{D}$
and *cbox*: $\bigwedge K.\ K \in \mathcal{D} \Longrightarrow K \subseteq cbox\ a\ b \wedge K \neq \{\} \wedge (\exists\, c\ d.\ K = cbox\ c\ d)$
and *intint*: *pairwise* $(\lambda A\ B.\ interior\ A \cap interior\ B = \{\})\ \mathcal{D}$
for $\mathcal{D}$
proof −
have *conv*: $\bigwedge K.\ K \in \mathcal{D} \Longrightarrow convex\ K$
using *cbox convex_box*(*1*) **by** *blast*
have *neg*: *negligible* $(g\ `\ K \cap g\ `\ L)$ **if** *linear g K* $\in \mathcal{D}$ *L* $\in \mathcal{D}$ $K \neq L$
for *K L* **and** *g* :: $'n \Rightarrow 'n$
proof (*cases inj g*)
case *True*
have *negligible* (*frontier*$(g\ `\ K \cap g\ `\ L) \cup interior(g\ `\ K \cap g\ `\ L)$)
proof (*rule negligible_Un*)
show *negligible* (*frontier* $(g\ `\ K \cap g\ `\ L)$)
by (*simp add*: *negligible_convex_frontier convex_Int conv convex_linear_image that*)
next

have $\forall\, p\ N.$ *pairwise* $p\ N = (\forall\, Na.\ (Na::'n\ set) \in N \longrightarrow (\forall\, Nb.\ Nb \in N \wedge Na \neq Nb \longrightarrow p\ Na\ Nb))$
by (*metis pairwise_def*)
then have *interior* $K \cap$ *interior* $L = \{\}$
using *intint that*(*2*) *that*(*3*) *that*(*4*) **by** *presburger*
then show *negligible* (*interior* $(g\ `\ K \cap g\ `\ L)$)
by (*metis True empty_imp_negligible image_Int image_empty interior_Int interior_injective_linear_image that*(*1*))
qed
moreover have $g\ `\ K \cap g\ `\ L \subseteq$ *frontier* $(g\ `\ K \cap g\ `\ L) \cup$ *interior* $(g\ `\ K \cap g\ `\ L)$
by (*metis Diff_partition Int_commute calculation closure_Un_frontier frontier_def inf.absorb_iff2 inf_bot_right inf_sup_absorb negligible_Un_eq open_interior open_not_negligible sup_commute*)
ultimately show *?thesis*
by (*rule negligible_subset*)
next
case *False*
then show *?thesis*
by (*simp add*: *negligible_Int negligible_linear_singular_image* ‹*linear g*›)
qed
have *negf*: *negligible* $((f\ `\ K) \cap (f\ `\ L))$
and *negid*: *negligible* $(K \cap L)$ **if** $K \in \mathcal{D}\ L \in \mathcal{D}\ K \neq L$ **for** $K\ L$
using *neg* [*OF* ‹*linear f*›] *neg* [*OF linear_id*] *that* **by** *auto*
show *?thesis*
proof (*cases finite* $\mathcal{D}$)
case *True*
then have $?\mu\ (\bigcup x{\in}\mathcal{D}.\ f\ `\ x) = (\sum x{\in}\mathcal{D}.\ ?\mu\ (f\ `\ x))$
using ‹*linear f*› *cbox measurable_linear_image_interval negf*
by (*blast intro*: *measure_negligible_finite_Union_image* [*unfolded pairwise_def*])
also have $\ldots = (\sum k{\in}\mathcal{D}.\ m * ?\mu\ k)$
by (*metis* (*no_types*, *lifting*) *cbox im sum.cong*)
also have $\ldots = m * ?\mu\ (\bigcup \mathcal{D})$
unfolding *sum_distrib_left* [*symmetric*]
by (*metis True cbox lmeasurable_cbox measure_negligible_finite_Union* [*unfolded pairwise_def*] *negid*)
finally show *?thesis*
by (*metis True* ‹*linear f*› *cbox image_Union fmeasurable.finite_UN measurable_linear_image_interval*)
next
case *False*
with ‹*countable* $\mathcal{D}$› **obtain** $X :: nat \Rightarrow 'n\ set$ **where** S: *bij_betw X UNIV* $\mathcal{D}$
using *bij_betw_from_nat_into* **by** *blast*
then have *eq*: $(\bigcup \mathcal{D}) = (\bigcup n.\ X\ n)$ $(f\ `\ \bigcup \mathcal{D}) = (\bigcup n.\ f\ `\ X\ n)$
by (*auto simp*: *bij_betw_def*)
have *meas*: $\bigwedge K.\ K \in \mathcal{D} \Longrightarrow K \in$ *lmeasurable*
using *cbox* **by** *blast*

```
        with S have 1: ⋀n. X n ∈ lmeasurable
          by (auto simp: bij_betw_def)
        have 2: pairwise (λm n. negligible (X m ∩ X n)) UNIV
           using S unfolding bij_betw_def pairwise_def by (metis injD negid
range_eqI)
        have bounded (⋃ 𝒟)
          by (meson Sup_least bounded_cbox bounded_subset cbox)
        then have 3: bounded (⋃ n. X n)
          using S unfolding bij_betw_def by blast
        have (⋃ n. X n) ∈ lmeasurable
          by (rule measurable_countable_negligible_Union_bounded [OF 1 2 3])
        with S have f1: ⋀n. f ‘ (X n) ∈ lmeasurable
       unfolding bij_betw_def by (metis assms(1) cbox measurable_linear_image_interval
rangeI)
        have f2: pairwise (λm n. negligible (f ‘ (X m) ∩ f ‘ (X n))) UNIV
           using S unfolding bij_betw_def pairwise_def by (metis injD negf
rangeI)
        have bounded (⋃ 𝒟)
          by (meson Sup_least bounded_cbox bounded_subset cbox)
        then have f3: bounded (⋃ n. f ‘ X n)
          using S unfolding bij_betw_def
          by (metis bounded_linear_image linear_linear assms(1) image_Union
range_composition)
        have (λn. ?μ (X n)) sums ?μ (⋃ n. X n)
          by (rule measure_countable_negligible_Union_bounded [OF 1 2 3])
        have meq: ?μ (⋃ n. f ‘ X n) = m * ?μ (⋃ (X ‘ UNIV))
      proof (rule sums_unique2 [OF measure_countable_negligible_Union_bounded
[OF f1 f2 f3]])
          have m: ⋀n. ?μ (f ‘ X n) = (m * ?μ (X n))
            using S unfolding bij_betw_def by (metis cbox im rangeI)
          show (λn. ?μ (f ‘ X n)) sums (m * ?μ (⋃ (X ‘ UNIV)))
            unfolding m
               using measure_countable_negligible_Union_bounded [OF 1 2 3]
sums_mult by blast
        qed
        show ?thesis
          using measurable_countable_negligible_Union_bounded [OF f1 f2 f3]
meq
          by (auto simp: eq [symmetric])
      qed
    qed
    show ?thesis
      unfolding completion.fmeasurable_measure_inner_outer_le
    proof (intro conjI allI impI)
      fix e :: real
      assume e > 0
      have 1: cbox a b − S ∈ lmeasurable
        by (simp add: fmeasurable.Diff that)
      have 2: 0 < e / (1 + |m|)
```

```
      using ‹e > 0› by (simp add: field_split_simps abs_add_one_gt_zero)
    obtain 𝒟
      where countable 𝒟
       and cbox: ⋀K. K ∈ 𝒟 ⟹ K ⊆ cbox a b ∧ K ≠ {} ∧ (∃c d. K = cbox
c d)
       and intdisj: pairwise (λA B. interior A ∩ interior B = {}) 𝒟
       and DD: cbox a b − S ⊆ ⋃𝒟 ⋃𝒟 ∈ lmeasurable
       and le: ?μ (⋃𝒟) ≤ ?μ (cbox a b − S) + e/(1 + |m|)
      by (rule measurable_outer_intervals_bounded [of cbox a b − S a b e/(1
+ |m|)]; use 1 2 pairwise_def in force)
    show ∃T ∈ lmeasurable. T ⊆ f ‘ S ∧ m ∗ ?μ S − e ≤ ?μ T
    proof (intro bexI conjI)
      show f ‘ (cbox a b) − f ‘ (⋃𝒟) ⊆ f ‘ S
        using ‹cbox a b − S ⊆ ⋃𝒟› by force
      have m ∗ ?μ S − e ≤ m ∗ (?μ S − e / (1 + |m|))
        using ‹m ≥ 0› ‹e > 0› by (simp add: field_simps)
      also have ... ≤ ?μ (f ‘ cbox a b) − ?μ (f ‘ (⋃𝒟))
      proof −
        have ?μ (cbox a b − S) = ?μ (cbox a b) − ?μ S
          by (simp add: measurable_measure_Diff ‹S ⊆ cbox a b› fmeasurableD
that(2))
        then have (?μ S − e / (1 + m)) ≤ (content (cbox a b) − ?μ (⋃ 𝒟))
          using ‹m ≥ 0› le by auto
        then show ?thesis
          using ‹m ≥ 0› ‹e > 0›
         by (simp add: mult_left_mono im fUD [OF ‹countable 𝒟› cbox intdisj]
flip: right_diff_distrib)
      qed
      also have ... = ?μ (f ‘ cbox a b − f ‘ ⋃𝒟)
      proof (rule measurable_measure_Diff [symmetric])
        show f ‘ cbox a b ∈ lmeasurable
          by (simp add: assms(1) measurable_linear_image_interval)
        show f ‘ ⋃ 𝒟 ∈ sets lebesgue
          by (simp add: ‹countable 𝒟› cbox fUD fmeasurableD intdisj)
        show f ‘ ⋃ 𝒟 ⊆ f ‘ cbox a b
          by (simp add: Sup_le_iff cbox image_mono)
      qed
      finally show m ∗ ?μ S − e ≤ ?μ (f ‘ cbox a b − f ‘ ⋃𝒟) .
      show f ‘ cbox a b − f ‘ ⋃𝒟 ∈ lmeasurable
        by (simp add: fUD ‹countable 𝒟› ‹linear f› cbox fmeasurable.Diff intdisj
measurable_linear_image_interval)
    qed
  next
    fix e :: real
    assume e > 0
    have em: 0 < e / (1 + |m|)
      using ‹e > 0› by (simp add: field_split_simps abs_add_one_gt_zero)
    obtain 𝒟
      where countable 𝒟
```

```
          and cbox: ⋀K. K ∈ 𝒟 ⟹ K ⊆ cbox a b ∧ K ≠ {} ∧ (∃ c d. K = cbox
c d)
          and intdisj: pairwise (λA B. interior A ∩ interior B = {}) 𝒟
          and DD: S ⊆ ⋃𝒟 ⋃𝒟 ∈ lmeasurable
          and le: ?μ (⋃𝒟) ≤ ?μ S + e/(1 + |m|)
        by (rule measurable_outer_intervals_bounded [of S a b e/(1 + |m|)]; use
‹S ∈ lmeasurable› ‹S ⊆ cbox a b› em in force)
      show ∃ U ∈ lmeasurable. f ‘ S ⊆ U ∧ ?μ U ≤ m * ?μ S + e
      proof (intro bexI conjI)
        show f ‘ S ⊆ f ‘ (⋃𝒟)
          by (simp add: DD(1) image_mono)
        have ?μ (f ‘ ⋃𝒟) ≤ m * (?μ S + e / (1 + |m|))
          using ‹m ≥ 0› le mult_left_mono
         by (auto simp: fUD ‹countable 𝒟› ‹linear f› cbox fmeasurable.Diff intdisj
measurable_linear_image_interval)
        also have ... ≤ m * ?μ S + e
           using ‹m ≥ 0› ‹e > 0› by (simp add: fUD [OF ‹countable 𝒟› cbox
intdisj] field_simps)
        finally show ?μ (f ‘ ⋃𝒟) ≤ m * ?μ S + e .
        show f ‘ ⋃𝒟 ∈ lmeasurable
          by (simp add: ‹countable 𝒟› cbox fUD intdisj)
      qed
    qed
  qed
  show ?thesis
    unfolding has_measure_limit_iff
  proof (intro allI impI)
    fix e :: real
    assume e > 0
    obtain B where B > 0 and B:
      ⋀a b. ball 0 B ⊆ cbox a b ⟹ |?μ (S ∩ cbox a b) − ?μ S| < e / (1 + |m|)
     using has_measure_limit [OF S] ‹e > 0› by (metis abs_add_one_gt_zero
zero_less_divide_iff)
    obtain c d::'n where cd: ball 0 B ⊆ cbox c d
      by (metis bounded_subset_cbox_symmetric bounded_ball)
    with B have less: |?μ (S ∩ cbox c d) − ?μ S| < e / (1 + |m|) .
    obtain D where D > 0 and D: cbox c d ⊆ ball 0 D
      by (metis bounded_cbox bounded_subset_ballD)
    obtain C where C > 0 and C: ⋀x. norm (f x) ≤ C * norm x
      using linear_bounded_pos ‹linear f› by blast
    have f ‘ S ∩ cbox a b ∈ lmeasurable ∧
          |?μ (f ‘ S ∩ cbox a b) − m * ?μ S| < e
      if ball 0 (D*C) ⊆ cbox a b for a b
    proof −
      have bounded (S ∩ h ‘ cbox a b)
      by (simp add: bounded_linear_image linear_linear ‹linear h› bounded_Int)
      moreover have Shab: S ∩ h ‘ cbox a b ∈ lmeasurable
     by (simp add: S ‹linear h› fmeasurable.Int measurable_linear_image_interval)
      moreover have fim: f ‘ (S ∩ h ‘ (cbox a b)) = (f ‘ S) ∩ cbox a b
```

```
        by (auto simp: hf rev_image_eqI fh)
      ultimately have 1: (f ‘ S) ∩ cbox a b ∈ lmeasurable
          and 2: ?μ ((f ‘ S) ∩ cbox a b) = m * ?μ (S ∩ h ‘ cbox a b)
        using fBS [of S ∩ (h ‘ (cbox a b))] by auto
      have *: ⟦|z − m| < e; z ≤ w; w ≤ m⟧ ⟹ |w − m| ≤ e
        for w z m and e::real by auto
      have meas_adiff: |?μ (S ∩ h ‘ cbox a b) − ?μ S| ≤ e / (1 + |m|)
      proof (rule * [OF less])
        show ?μ (S ∩ cbox c d) ≤ ?μ (S ∩ h ‘ cbox a b)
        proof (rule measure_mono_fmeasurable [OF _ _ Shab])
          have f ‘ ball 0 D ⊆ ball 0 (C * D)
            using C ‹C > 0›
            apply (clarsimp simp: algebra_simps)
          by (meson le_less_trans linordered_comm_semiring_strict_class.comm_mult_strict_left_mono)
          then have f ‘ ball 0 D ⊆ cbox a b
            by (metis mult.commute order_trans that)
          have ball 0 D ⊆ h ‘ cbox a b
            by (metis ‹f ‘ ball 0 D ⊆ cbox a b› hf image_subset_iff subsetI)
          then show S ∩ cbox c d ⊆ S ∩ h ‘ cbox a b
            using D by blast
        next
          show S ∩ cbox c d ∈ sets lebesgue
            using S fmeasurable_cbox by blast
        qed
      next
        show ?μ (S ∩ h ‘ cbox a b) ≤ ?μ S
          by (simp add: S Shab fmeasurableD measure_mono_fmeasurable)
      qed
      have |?μ (f ‘ S ∩ cbox a b) − m * ?μ S| ≤ |?μ S − ?μ (S ∩ h ‘ cbox a b)|
* m
        by (metis 2 ‹m ≥ 0› abs_minus_commute abs_mult_pos mult.commute
order_refl right_diff_distrib')
      also have ... ≤ e / (1 + m) * m
          by (metis ‹m ≥ 0› abs_minus_commute abs_of_nonneg meas_adiff
mult.commute mult_left_mono)
      also have ... < e
        using ‹e > 0› ‹m ≥ 0› by (simp add: field_simps)
      finally have |?μ (f ‘ S ∩ cbox a b) − m * ?μ S| < e .
      with 1 show ?thesis by auto
    qed
    then show ∃B>0. ∀a b. ball 0 B ⊆ cbox a b ⟶
                  f ‘ S ∩ cbox a b ∈ lmeasurable ∧
                  |?μ (f ‘ S ∩ cbox a b) − m * ?μ S| < e
      using ‹C>0› ‹D>0› by (metis mult_zero_left mult_less_iff1)
  qed
 qed
qed
```

6.16.17 Lemmas about absolute integrability

lemma *absolutely_integrable_linear*:
fixes $f :: 'm::euclidean\_space \Rightarrow 'n::euclidean\_space$
and $h :: 'n::euclidean\_space \Rightarrow 'p::euclidean\_space$
shows f *absolutely_integrable_on* $s \Longrightarrow$ *bounded_linear* $h \Longrightarrow (h \circ f)$ *absolutely_integrable_on* s
using *integrable_bounded_linear*[*of* h *lebesgue* $\lambda x.$ *indicator* $s\ x *_R f\ x$]
by (*simp add*: *linear_simps*[*of* h] *set_integrable_def*)

lemma *absolutely_integrable_sum*:
fixes $f :: 'a \Rightarrow 'n::euclidean\_space \Rightarrow 'm::euclidean\_space$
assumes *finite* T **and** $\bigwedge a.\ a \in T \Longrightarrow (f\ a)$ *absolutely_integrable_on* S
shows $(\lambda x.\ sum\ (\lambda a.\ f\ a\ x)\ T)$ *absolutely_integrable_on* S
using *assms* **by** *induction auto*

lemma *absolutely_integrable_integrable_bound*:
fixes $f :: 'n::euclidean\_space \Rightarrow 'm::euclidean\_space$
assumes *le*: $\bigwedge x.\ x \in S \Longrightarrow norm\ (f\ x) \leq g\ x$ **and** f: f *integrable_on* S **and** g: g *integrable_on* S
shows f *absolutely_integrable_on* S
unfolding *set_integrable_def*
proof (*rule Bochner_Integration.integrable_bound*)
have g *absolutely_integrable_on* S
unfolding *absolutely_integrable_on_def*
proof
show $(\lambda x.\ norm\ (g\ x))$ *integrable_on* S
using *le norm_ge_zero*[*of* f _]
by (*intro integrable_spike_finite*[*OF* _ _ g, *of* {}])
(*auto intro*!: *abs_of_nonneg intro*: *order_trans simp del*: *norm_ge_zero*)
qed *fact*
then show *integrable lebesgue* $(\lambda x.\ indicat\_real\ S\ x *_R g\ x)$
by (*simp add*: *set_integrable_def*)
show $(\lambda x.\ indicat\_real\ S\ x *_R f\ x) \in borel\_measurable\ lebesgue$
using f **by** (*auto intro*: *has_integral_implies_lebesgue_measurable simp*: *integrable_on_def*)
qed (*use le* **in** ‹*force intro*!: *always_eventually split*: *split_indicator*›)

corollary *absolutely_integrable_on_const* [*simp*]:
fixes $c :: 'a::euclidean\_space$
assumes $S \in lmeasurable$
shows $(\lambda x.\ c)$ *absolutely_integrable_on* S
by (*metis* (*full_types*) *assms absolutely_integrable_integrable_bound integrable_on_const order_refl*)

lemma *absolutely_integrable_continuous*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
shows *continuous_on* (*cbox a b*) $f \Longrightarrow f$ *absolutely_integrable_on cbox a b*
using *absolutely_integrable_integrable_bound*
by (*simp add*: *absolutely_integrable_on_def continuous_on_norm integrable_continuous*)

lemma *absolutely_integrable_continuous_real*:
fixes $f :: real \Rightarrow 'b::euclidean\_space$
shows *continuous_on* $\{a..b\}$ $f \Longrightarrow f$ *absolutely_integrable_on* $\{a..b\}$
by (*metis absolutely_integrable_continuous box_real*(2))

lemma *continuous_imp_integrable*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes *continuous_on* (*cbox a b*) *f*
shows *integrable* (*lebesgue_on* (*cbox a b*)) *f*
proof −
have *f absolutely_integrable_on cbox a b*
by (*simp add*: *absolutely_integrable_continuous assms*)
then show *?thesis*
by (*simp add*: *integrable_restrict_space set_integrable_def*)
qed

lemma *continuous_imp_integrable_real*:
fixes $f :: real \Rightarrow 'b::euclidean\_space$
assumes *continuous_on* $\{a..b\}$ *f*
shows *integrable* (*lebesgue_on* $\{a..b\}$) *f*
by (*metis assms continuous_imp_integrable interval_cbox*)

6.16.18 Componentwise

proposition *absolutely_integrable_componentwise_iff*:
shows *f absolutely_integrable_on A* $\longleftrightarrow$ ($\forall$ *b*$\in$*Basis*. (λx. *f x* ⋅ *b*) *absolutely_integrable_on A*)
proof −
have ∗: (λx. *norm* (*f x*)) *integrable_on A* $\longleftrightarrow$ ($\forall$ *b*$\in$*Basis*. (λx. *norm* (*f x* ⋅ *b*)) *integrable_on A*) (**is** *?lhs* = *?rhs*)
if *f integrable_on A*
proof
assume *?lhs*
then show *?rhs*
by (*metis absolutely_integrable_on_def Topology_Euclidean_Space.norm_nth_le absolutely_integrable_integrable_bound integrable_component that*)
next
assume *R*: *?rhs*
have *f absolutely_integrable_on A*
proof (*rule absolutely_integrable_integrable_bound*)
show (λx. $\sum$ *i*$\in$*Basis*. *norm* (*f x* ⋅ *i*)) *integrable_on A*
using *R* **by** (*force intro*: *integrable_sum*)
qed (*use that norm_le_l1* **in** *auto*)
then show *?lhs*
using *absolutely_integrable_on_def* **by** *auto*
qed
show *?thesis*
unfolding *absolutely_integrable_on_def*

by (*simp add*: *integrable_componentwise_iff* [*symmetric*] *ball_conj_distrib* ∗ *cong*: *conj_cong*)
qed

lemma *absolutely_integrable_componentwise*:
shows (⋀*b*. *b* ∈ *Basis* ⟹ (λ*x*. *f x* ⋅ *b*) *absolutely_integrable_on A*) ⟹ *f absolutely_integrable_on A*
using *absolutely_integrable_componentwise_iff* **by** *blast*

lemma *absolutely_integrable_component*:
f absolutely_integrable_on A ⟹ (λ*x*. *f x* ⋅ (*b* :: *'b* :: *euclidean_space*)) *absolutely_integrable_on A*
by (*drule absolutely_integrable_linear*[*OF* _ *bounded_linear_inner_left*[*of b*]]) (*simp add*: *o_def*)

lemma *absolutely_integrable_scaleR_left*:
fixes *f* :: *'n*::*euclidean_space* ⇒ *'m*::*euclidean_space*
assumes *f absolutely_integrable_on S*
shows (λ*x*. *c* ∗*R f x*) *absolutely_integrable_on S*
proof −
have (λ*x*. *c* ∗*R x*) *o f absolutely_integrable_on S*
by (*simp add*: *absolutely_integrable_linear assms bounded_linear_scaleR_right*)
then show *?thesis*
using *assms* **by** *blast*
qed

lemma *absolutely_integrable_scaleR_right*:
assumes *f absolutely_integrable_on S*
shows (λ*x*. *f x* ∗*R c*) *absolutely_integrable_on S*
using *assms* **by** *blast*

lemma *absolutely_integrable_norm*:
fixes *f* :: *'a* :: *euclidean_space* ⇒ *'b* :: *euclidean_space*
assumes *f absolutely_integrable_on S*
shows (*norm o f*) *absolutely_integrable_on S*
using *assms* **by** (*simp add*: *absolutely_integrable_on_def o_def*)

lemma *absolutely_integrable_abs*:
fixes *f* :: *'a* :: *euclidean_space* ⇒ *'b* :: *euclidean_space*
assumes *f absolutely_integrable_on S*
shows (λ*x*. ∑*i*∈*Basis*. |*f x* ⋅ *i*| ∗*R i*) *absolutely_integrable_on S*
(**is** *?g absolutely_integrable_on S*)
proof −
have ∗: (λ*y*. ∑*j*∈*Basis*. *if j* = *i then y* ∗*R j else* 0) ∘
(λ*x*. *norm* (∑*j*∈*Basis*. *if j* = *i then* (*x* ⋅ *i*) ∗*R j else* 0)) ∘ *f*
absolutely_integrable_on S
if *i* ∈ *Basis* **for** *i*
proof −

```
    have bounded_linear (λy. ∑j∈Basis. if j = i then y *R j else 0)
      by (simp add: linear_linear algebra_simps linearI)
    moreover have (λx. norm (∑j∈Basis. if j = i then (x • i) *R j else 0)) ∘ f
                  absolutely_integrable_on S
      using assms ‹i ∈ Basis›
      unfolding o_def
      by (intro absolutely_integrable_norm [unfolded o_def])
        (auto simp: algebra_simps dest: absolutely_integrable_component)
    ultimately show ?thesis
      by (subst comp_assoc) (blast intro: absolutely_integrable_linear)
  qed
  have eq: ?g =
        (λx. ∑i∈Basis. ((λy. ∑j∈Basis. if j = i then y *R j else 0) ∘
              (λx. norm(∑j∈Basis. if j = i then (x • i) *R j else 0)) ∘ f) x)
    by (simp)
  show ?thesis
    unfolding eq
    by (rule absolutely_integrable_sum) (force simp: intro!: *)+
qed

lemma abs_absolutely_integrableI_1:
  fixes f :: 'a :: euclidean_space ⇒ real
  assumes f: f integrable_on A and (λx. |f x|) integrable_on A
  shows f absolutely_integrable_on A
  by (rule absolutely_integrable_integrable_bound [OF _ assms]) auto


lemma abs_absolutely_integrableI:
  assumes f: f integrable_on S and fcomp: (λx. ∑i∈Basis. |f x • i| *R i) inte-
grable_on S
  shows f absolutely_integrable_on S
proof −
  have (λx. (f x • i) *R i) absolutely_integrable_on S if i ∈ Basis for i
  proof −
    have (λx. |f x • i|) integrable_on S
      using assms integrable_component [OF fcomp, where y=i] that by simp
    then have (λx. f x • i) absolutely_integrable_on S
      using abs_absolutely_integrableI_1 f integrable_component by blast
    then show ?thesis
      by (rule absolutely_integrable_scaleR_right)
  qed
  then have (λx. ∑i∈Basis. (f x • i) *R i) absolutely_integrable_on S
    by (simp add: absolutely_integrable_sum)
  then show ?thesis
    by (simp add: euclidean_representation)
qed


lemma absolutely_integrable_abs_iff:
```

f absolutely_integrable_on S $\longleftrightarrow$
f integrable_on S $\wedge$ $(\lambda x.\ \sum i{\in}Basis.\ |f\ x \cdot i| *_R i)$ *integrable_on S*
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs* **then show** *?rhs*
using *absolutely_integrable_abs absolutely_integrable_on_def* **by** *blast*
next
assume *?rhs*
moreover
have $(\lambda x.\ if\ x \in S\ then\ \sum i{\in}Basis.\ |f\ x \cdot i| *_R i\ else\ 0) = (\lambda x.\ \sum i{\in}Basis.\ |(if\ x \in S\ then\ f\ x\ else\ 0) \cdot i| *_R i)$
by *force*
ultimately show *?lhs*
by (*simp only*: *absolutely_integrable_restrict_UNIV* [*of S*, *symmetric*] *integrable_restrict_UNIV* [*of S*, *symmetric*] *abs_absolutely_integrableI*)
qed

lemma *absolutely_integrable_max*:
fixes *f* :: *'n::euclidean_space* $\Rightarrow$ *'m::euclidean_space*
assumes *f absolutely_integrable_on S g absolutely_integrable_on S*
shows $(\lambda x.\ \sum i{\in}Basis.\ max\ (f\ x \cdot i)\ (g\ x \cdot i) *_R i)$
absolutely_integrable_on S
proof −
have $(\lambda x.\ \sum i{\in}Basis.\ max\ (f\ x \cdot i)\ (g\ x \cdot i) *_R i) =$
$(\lambda x.\ (1/2) *_R (f\ x + g\ x + (\sum i{\in}Basis.\ |f\ x \cdot i - g\ x \cdot i| *_R i)))$
proof (*rule ext*)
fix *x*
have $(\sum i{\in}Basis.\ max\ (f\ x \cdot i)\ (g\ x \cdot i) *_R i) = (\sum i{\in}Basis.\ ((f\ x \cdot i + g\ x \cdot i + |f\ x \cdot i - g\ x \cdot i|)\ /\ 2) *_R i)$
by (*force intro*: *sum.cong*)
also have ... $= (1\ /\ 2) *_R (\sum i{\in}Basis.\ (f\ x \cdot i + g\ x \cdot i + |f\ x \cdot i - g\ x \cdot i|) *_R i)$
by (*simp add*: *scaleR_right.sum*)
also have ... $= (1\ /\ 2) *_R (f\ x + g\ x + (\sum i{\in}Basis.\ |f\ x \cdot i - g\ x \cdot i| *_R i))$
by (*simp add*: *sum.distrib algebra_simps euclidean_representation*)
finally
show $(\sum i{\in}Basis.\ max\ (f\ x \cdot i)\ (g\ x \cdot i) *_R i) =$
$(1\ /\ 2) *_R (f\ x + g\ x + (\sum i{\in}Basis.\ |f\ x \cdot i - g\ x \cdot i| *_R i))$.
qed
moreover have $(\lambda x.\ (1\ /\ 2) *_R (f\ x + g\ x + (\sum i{\in}Basis.\ |f\ x \cdot i - g\ x \cdot i| *_R i)))$
absolutely_integrable_on S
using *absolutely_integrable_abs* [*OF set_integral_diff*(*1*) [*OF assms*]]
by (*intro set_integral_add absolutely_integrable_scaleR_left assms*) (*simp add*: *algebra_simps*)
ultimately show *?thesis* **by** *metis*
qed

corollary *absolutely_integrable_max_1*:

fixes $f :: 'n::euclidean\_space \Rightarrow real$
assumes *f absolutely_integrable_on S g absolutely_integrable_on S*
shows $(\lambda x.\ max\ (f\ x)\ (g\ x))$ *absolutely_integrable_on S*
using *absolutely_integrable_max* [*OF assms*] **by** *simp*

lemma *absolutely_integrable_min*:
fixes $f :: 'n::euclidean\_space \Rightarrow 'm::euclidean\_space$
assumes *f absolutely_integrable_on S g absolutely_integrable_on S*
shows $(\lambda x.\ \sum i{\in}Basis.\ min\ (f\ x \cdot i)\ (g\ x \cdot i) *_R i)$
absolutely_integrable_on S
proof −
have $(\lambda x.\ \sum i{\in}Basis.\ min\ (f\ x \cdot i)\ (g\ x \cdot i) *_R i) =$
$(\lambda x.\ (1/2) *_R (f\ x + g\ x - (\sum i{\in}Basis.\ |f\ x \cdot i - g\ x \cdot i| *_R i)))$
proof (*rule ext*)
fix *x*
have $(\sum i{\in}Basis.\ min\ (f\ x \cdot i)\ (g\ x \cdot i) *_R i) = (\sum i{\in}Basis.\ ((f\ x \cdot i + g\ x \cdot i - |f\ x \cdot i - g\ x \cdot i|)\ /\ 2) *_R i)$
by (*force intro*: *sum.cong*)
also have $... = (1\ /\ 2) *_R (\sum i{\in}Basis.\ (f\ x \cdot i + g\ x \cdot i - |f\ x \cdot i - g\ x \cdot i|) *_R i)$
by (*simp add*: *scaleR_right.sum*)
also have $... = (1\ /\ 2) *_R (f\ x + g\ x - (\sum i{\in}Basis.\ |f\ x \cdot i - g\ x \cdot i| *_R i))$
by (*simp add*: *sum.distrib sum_subtractf algebra_simps euclidean_representation*)
finally
show $(\sum i{\in}Basis.\ min\ (f\ x \cdot i)\ (g\ x \cdot i) *_R i) =$
$(1\ /\ 2) *_R (f\ x + g\ x - (\sum i{\in}Basis.\ |f\ x \cdot i - g\ x \cdot i| *_R i))$.
qed
moreover have $(\lambda x.\ (1\ /\ 2) *_R (f\ x + g\ x - (\sum i{\in}Basis.\ |f\ x \cdot i - g\ x \cdot i| *_R i)))$
absolutely_integrable_on S
using *absolutely_integrable_abs* [*OF set_integral_diff*(*1*) [*OF assms*]]
by (*intro set_integral_add set_integral_diff absolutely_integrable_scaleR_left assms*)
(*simp add*: *algebra_simps*)
ultimately show *?thesis* **by** *metis*
qed

corollary *absolutely_integrable_min_1*:
fixes $f :: 'n::euclidean\_space \Rightarrow real$
assumes *f absolutely_integrable_on S g absolutely_integrable_on S*
shows $(\lambda x.\ min\ (f\ x)\ (g\ x))$ *absolutely_integrable_on S*
using *absolutely_integrable_min* [*OF assms*] **by** *simp*

lemma *nonnegative_absolutely_integrable*:
fixes $f :: 'a :: euclidean\_space \Rightarrow 'b :: euclidean\_space$
assumes *f integrable_on A* **and** *comp*: $\bigwedge x\ b.\ [\![x \in A;\ b \in Basis]\!] \Longrightarrow 0 \le f\ x \cdot b$
shows *f absolutely_integrable_on A*
proof −
have $(\lambda x.\ (f\ x \cdot i) *_R i)$ *absolutely_integrable_on A* **if** $i \in Basis$ **for** *i*

```
  proof −
    have (λx. f x · i) integrable_on A
      by (simp add: assms(1) integrable_component)
    then have (λx. f x · i) absolutely_integrable_on A
      by (metis that comp nonnegative_absolutely_integrable_1)
    then show ?thesis
      by (rule absolutely_integrable_scaleR_right)
  qed
  then have (λx. ∑ i∈Basis. (f x · i) *R i) absolutely_integrable_on A
    by (simp add: absolutely_integrable_sum)
  then show ?thesis
    by (simp add: euclidean_representation)
qed

lemma absolutely_integrable_component_ubound:
  fixes f :: 'a :: euclidean_space ⇒ 'b :: euclidean_space
  assumes f: f integrable_on A and g: g absolutely_integrable_on A
      and comp: ⋀x b. ⟦x ∈ A; b ∈ Basis⟧ ⟹ f x · b ≤ g x · b
  shows f absolutely_integrable_on A
proof −
  have (λx. g x − (g x − f x)) absolutely_integrable_on A
  proof (rule set_integral_diff [OF g nonnegative_absolutely_integrable])
    show (λx. g x − f x) integrable_on A
    using Henstock_Kurzweil_Integration.integrable_diff absolutely_integrable_on_def
f g by blast
  qed (simp add: comp inner_diff_left)
  then show ?thesis
    by simp
qed

lemma absolutely_integrable_component_lbound:
  fixes f :: 'a :: euclidean_space ⇒ 'b :: euclidean_space
  assumes f: f absolutely_integrable_on A and g: g integrable_on A
      and comp: ⋀x b. ⟦x ∈ A; b ∈ Basis⟧ ⟹ f x · b ≤ g x · b
  shows g absolutely_integrable_on A
proof −
  have (λx. f x + (g x − f x)) absolutely_integrable_on A
  proof (rule set_integral_add [OF f nonnegative_absolutely_integrable])
    show (λx. g x − f x) integrable_on A
    using Henstock_Kurzweil_Integration.integrable_diff absolutely_integrable_on_def
f g by blast
  qed (simp add: comp inner_diff_left)
  then show ?thesis
    by simp
qed

lemma integrable_on_1_iff:
  fixes f :: 'a::euclidean_space ⇒ real^1
  shows f integrable_on S ⟷ (λx. f x $ 1) integrable_on S
```

by (*auto simp*: *integrable_componentwise_iff* [*of f*] *Basis_vec_def cart_eq_inner_axis*)

lemma *integral_on_1_eq*:
fixes *f* :: *'a::euclidean_space* ⇒ *real^1*
shows *integral S f* = *vec* (*integral S* (λ*x*. *f x* $ *1*))
by (*cases f integrable_on S*) (*simp_all add*: *integrable_on_1_iff vec_eq_iff not_integrable_integral*)

lemma *absolutely_integrable_on_1_iff*:
fixes *f* :: *'a::euclidean_space* ⇒ *real^1*
shows *f absolutely_integrable_on S* ⟷ (λ*x*. *f x* $ *1*) *absolutely_integrable_on S*
unfolding *absolutely_integrable_on_def*
by (*auto simp*: *integrable_on_1_iff norm_real*)

lemma *absolutely_integrable_absolutely_integrable_lbound*:
fixes *f* :: *'m::euclidean_space* ⇒ *real*
assumes *f*: *f integrable_on S* **and** *g*: *g absolutely_integrable_on S*
and ∗: ⋀*x*. *x* ∈ *S* ⟹ *g x* ≤ *f x*
shows *f absolutely_integrable_on S*
by (*rule absolutely_integrable_component_lbound* [*OF g f*]) (*simp add*: ∗)

lemma *absolutely_integrable_absolutely_integrable_ubound*:
fixes *f* :: *'m::euclidean_space* ⇒ *real*
assumes *fg*: *f integrable_on S g absolutely_integrable_on S*
and ∗: ⋀*x*. *x* ∈ *S* ⟹ *f x* ≤ *g x*
shows *f absolutely_integrable_on S*
by (*rule absolutely_integrable_component_ubound* [*OF fg*]) (*simp add*: ∗)

lemma *has_integral_vec1_I_cbox*:
fixes *f* :: *real^1* ⇒ *'a::real_normed_vector*
assumes (*f has_integral y*) (*cbox a b*)
shows ((*f* ∘ *vec*) *has_integral y*) {*a$1..b$1*}
proof −
have ((λ*x*. *f*(*vec x*)) *has_integral* (*1* / *1*) ∗$_R$ *y*) ((λ*x*. *x* $ *1*) ‘ *cbox a b*)
proof (*rule has_integral_twiddle*)
show ∃ *w z*::*real^1*. *vec* ‘ *cbox u v* = *cbox w z*
content (*vec* ‘ *cbox u v* :: (*real^1*) *set*) = *1* ∗ *content* (*cbox u v*) **for** *u v*
unfolding *vec_cbox_1_eq*
by (*auto simp*: *content_cbox_if_cart interval_eq_empty_cart*)
show ∃ *w z*. (λ*x*. *x* $ *1*) ‘ *cbox u v* = *cbox w z* **for** *u v* :: *real^1*
using *vec_nth_cbox_1_eq* **by** *blast*
qed (*auto simp*: *continuous_vec assms*)
then show *?thesis*
by (*simp add*: *o_def*)
qed

lemma *has_integral_vec1_I*:
fixes *f* :: *real^1* ⇒ *'a::real_normed_vector*
assumes (*f has_integral y*) *S*

```
  shows (f ∘ vec has_integral y) ((λx. x $ 1) ‘ S)
proof −
  have ∗: ∃ z. ((λx. if x ∈ (λx. x $ 1) ‘ S then (f ∘ vec) x else 0) has_integral z)
{a..b} ∧ norm (z − y) < e
    if int: ⋀a b. ball 0 B ⊆ cbox a b ⟹
                  (∃ z. ((λx. if x ∈ S then f x else 0) has_integral z) (cbox a b) ∧
norm (z − y) < e)
      and B: ball 0 B ⊆ {a..b} for e B a b
  proof −
    have [simp]: (∃ y∈S. x = y $ 1) ⟷ vec x ∈ S for x
      by force
    have B′: ball (0::real^1) B ⊆ cbox (vec a) (vec b)
        using B by (simp add: Basis_vec_def cart_eq_inner_axis [symmetric]
mem_box norm_real subset_iff)
    show ?thesis
        using int [OF B′] by (auto simp: image_iff o_def cong: if_cong dest!:
has_integral_vec1_I_cbox)
  qed
  show ?thesis
    using assms
    apply (subst has_integral_alt)
    apply (subst (asm) has_integral_alt)
    apply (simp add: has_integral_vec1_I_cbox split: if_split_asm)
    subgoal by (metis vector_one_nth)
    subgoal
      apply (erule all_forward imp_forward ex_forward asm_rl)+
      by (blast intro!: ∗)+
    done
qed

lemma has_integral_vec1_nth_cbox:
  fixes f :: real ⇒ 'a::real_normed_vector
  assumes (f has_integral y) {a..b}
  shows ((λx::real^1. f(x$1)) has_integral y) (cbox (vec a) (vec b))
proof −
  have ((λx::real^1. f(x$1)) has_integral (1 / 1) ∗R y) (vec ‘ cbox a b)
  proof (rule has_integral_twiddle)
    show ∃ w z::real. (λx. x $ 1) ‘ cbox u v = cbox w z
         content ((λx. x $ 1) ‘ cbox u v) = 1 ∗ content (cbox u v) for u v::real^1
     unfolding vec_cbox_1_eq by (auto simp: content_cbox_if_cart interval_eq_empty_cart)
    show ∃ w z::real^1. vec ‘ cbox u v = cbox w z for u v :: real
      using vec_cbox_1_eq by auto
  qed (auto simp: continuous_vec assms)
  then show ?thesis
    using vec_cbox_1_eq by auto
qed

lemma has_integral_vec1_D_cbox:
  fixes f :: real^1 ⇒ 'a::real_normed_vector
```

assumes ((*f* ∘ *vec*) *has_integral y*) {*a$1..b$1*}
shows (*f has_integral y*) (*cbox a b*)
by (*metis* (*mono_tags*, *lifting*) *assms comp_apply has_integral_eq has_integral_vec1_nth_cbox vector_one_nth*)

lemma *has_integral_vec1_D*:
fixes *f* :: *real^1* ⇒ *'a*::*real_normed_vector*
assumes ((*f* ∘ *vec*) *has_integral y*) ((λ*x*. *x* $ *1*) ‘ *S*)
shows (*f has_integral y*) *S*
proof −
have ∗: ∃ *z*. ((λ*x*. *if x* ∈ *S then f x else 0*) *has_integral z*) (*cbox a b*) ∧ *norm* (*z* − *y*) < *e*
if *int*: ⋀*a b*. *ball 0 B* ⊆ {*a..b*} ⟹ (∃ *z*. ((λ*x*. *if x* ∈ (λ*x*. *x* $ *1*) ‘ *S then* (*f* ∘ *vec*) *x else 0*) *has_integral z*) {*a..b*} ∧ *norm* (*z* − *y*) < *e*)
and *B*: *ball 0 B* ⊆ *cbox a b* **for** *e B* **and** *a b*::*real^1*
proof −
have *B'*: *ball 0 B* ⊆ {*a$1..b$1*}
proof (*clarsimp*)
fix *t*
assume |*t*| < *B* **then show** *a* $ *1* ≤ *t* ∧ *t* ≤ *b* $ *1*
using *subsetD* [*OF B*]
by (*metis* (*mono_tags*, *opaque_lifting*) *mem_ball_0 mem_box_cart*(*2*) *norm_real vec_component*)
qed
have *eq*: (λ*x*. *if vec x* ∈ *S then f* (*vec x*) *else 0*) = (λ*x*. *if x* ∈ *S then f x else 0*) ∘ *vec*
by *force*
have [*simp*]: (∃ *y*∈*S*. *x* = *y* $ *1*) ⟷ *vec x* ∈ *S* **for** *x*
by *force*
show *?thesis*
using *int* [*OF B'*] **by** (*auto simp*: *image_iff eq cong*: *if_cong dest*!: *has_integral_vec1_D_cbox*)
qed
show *?thesis*
using *assms*
apply (*subst has_integral_alt*)
apply (*subst* (*asm*) *has_integral_alt*)
apply (*simp add*: *has_integral_vec1_D_cbox eq_cbox split*: *if_split_asm*, *blast*)
apply (*intro conjI impI*)
subgoal by (*metis vector_one_nth*)
apply (*erule thin_rl*)
apply (*erule all_forward ex_forward conj_forward*)+
by (*blast intro*!: ∗)+
qed

lemma *integral_vec1_eq*:
fixes *f* :: *real^1* ⇒ *'a*::*real_normed_vector*
shows *integral S f* = *integral* ((λ*x*. *x* $ *1*) ‘ *S*) (*f* ∘ *vec*)

using *has_integral_vec1_I* [*of f*] *has_integral_vec1_D* [*of f*]
by (*metis has_integral_iff not_integrable_integral*)

lemma *absolutely_integrable_drop*:
fixes *f* :: *real^1* ⇒ *'b*::*euclidean_space*
shows *f absolutely_integrable_on S* ⟷ (*f* ∘ *vec*) *absolutely_integrable_on* (λ*x*. *x* $ *1*) ‘ *S*
unfolding *absolutely_integrable_on_def integrable_on_def*
proof *safe*
fix *y r*
assume (*f has_integral y*) *S* ((λ*x*. *norm* (*f x*)) *has_integral r*) *S*
then show ∃ *y*. (*f* ∘ *vec has_integral y*) ((λ*x*. *x* $ *1*) ‘ *S*)
∃ *y*. ((λ*x*. *norm* ((*f* ∘ *vec*) *x*)) *has_integral y*) ((λ*x*. *x* $ *1*) ‘ *S*)
by (*force simp*: *o_def dest*!: *has_integral_vec1_I*)+
next
fix *y* :: *'b* **and** *r* :: *real*
assume (*f* ∘ *vec has_integral y*) ((λ*x*. *x* $ *1*) ‘ *S*)
((λ*x*. *norm* ((*f* ∘ *vec*) *x*)) *has_integral r*) ((λ*x*. *x* $ *1*) ‘ *S*)
then show ∃ *y*. (*f has_integral y*) *S* ∃ *y*. ((λ*x*. *norm* (*f x*)) *has_integral y*) *S*
by (*force simp*: *o_def intro*: *has_integral_vec1_D*)+
qed

6.16.19 Dominated convergence

lemma *dominated_convergence*:
fixes *f* :: *nat* ⇒ *'n*::*euclidean_space* ⇒ *'m*::*euclidean_space*
assumes *f*: ⋀*k*. (*f k*) *integrable_on S* **and** *h*: *h integrable_on S*
and *le*: ⋀*k x*. *x* ∈ *S* ⟹ *norm* (*f k x*) ≤ *h x*
and *conv*: ⋀*x*. *x* ∈ *S* ⟹ (λ*k*. *f k x*) ⟶ *g x*
shows *g integrable_on S* (λ*k*. *integral S* (*f k*)) ⟶ *integral S g*
proof −
have *3*: *h absolutely_integrable_on S*
unfolding *absolutely_integrable_on_def*
proof
show (λ*x*. *norm* (*h x*)) *integrable_on S*
proof (*intro integrable_spike_finite*[*OF* _ _ *h*, *of* {}] *ballI*)
fix *x* **assume** *x* ∈ *S* − {} **then show** *norm* (*h x*) = *h x*
by (*metis Diff_empty abs_of_nonneg bot_set_def le norm_ge_zero order_trans real_norm_def*)
qed *auto*
qed *fact*
have *2*: *set_borel_measurable lebesgue S* (*f k*) **for** *k*
unfolding *set_borel_measurable_def*
using *f* **by** (*auto intro*: *has_integral_implies_lebesgue_measurable simp*: *integrable_on_def*)
then have *1*: *set_borel_measurable lebesgue S g*
unfolding *set_borel_measurable_def*
by (*rule borel_measurable_LIMSEQ_metric*) (*use conv* **in** ‹*auto split*: *split_indicator*›)
have *4*: *AE x in lebesgue*. (λ*i*. *indicator S x* ∗$_R$ *f i x*) ⟶ *indicator S x* ∗$_R$ *g*

x
AE x in lebesgue. norm (indicator S x ∗R f k x) ≤ indicator S x ∗R h x **for** *k*
using *conv le* **by** (*auto intro*!: *always_eventually split*: *split_indicator*)
have *g*: *g absolutely_integrable_on S*
using *1 2 3 4* **unfolding** *set_borel_measurable_def set_integrable_def*
by (*rule integrable_dominated_convergence*)
then show *g integrable_on S*
by (*auto simp*: *absolutely_integrable_on_def*)
have (λ*k*. (*LINT x*:*S*|*lebesgue*. *f k x*)) ⟶ (*LINT x*:*S*|*lebesgue*. *g x*)
unfolding *set_borel_measurable_def set_lebesgue_integral_def*
using *1 2 3 4* **unfolding** *set_borel_measurable_def set_lebesgue_integral_def set_integrable_def*
by (*rule integral_dominated_convergence*)
then show (λ*k*. *integral S* (*f k*)) ⟶ *integral S g*
using *g absolutely_integrable_integrable_bound*[*OF le f h*]
by (*subst* (*asm*) (*1 2*) *set_lebesgue_integral_eq_integral*) *auto*
qed

lemma *has_integral_dominated_convergence*:
fixes *f* :: *nat* ⇒ *'n*::*euclidean_space* ⇒ *'m*::*euclidean_space*
assumes ⋀*k*. (*f k has_integral y k*) *S h integrable_on S*
⋀*k*. ∀*x*∈*S*. *norm* (*f k x*) ≤ *h x* ∀*x*∈*S*. (λ*k*. *f k x*) ⟶ *g x*
and *x*: *y* ⟶ *x*
shows (*g has_integral x*) *S*
proof −
have *int_f*: ⋀*k*. (*f k*) *integrable_on S*
using *assms* **by** (*auto simp*: *integrable_on_def*)
have (*g has_integral* (*integral S g*)) *S*
by (*metis assms*(*2*−*4*) *dominated_convergence*(*1*) *has_integral_integral int_f*)
moreover have *integral S g* = *x*
proof (*rule LIMSEQ_unique*)
show (λ*i*. *integral S* (*f i*)) ⟶ *x*
using *integral_unique*[*OF assms*(*1*)] *x* **by** *simp*
show (λ*i*. *integral S* (*f i*)) ⟶ *integral S g*
by (*metis assms*(*2*) *assms*(*3*) *assms*(*4*) *dominated_convergence*(*2*) *int_f*)
qed
ultimately show *?thesis*
by *simp*
qed

lemma *dominated_convergence_integrable_1*:
fixes *f* :: *nat* ⇒ *'n*::*euclidean_space* ⇒ *real*
assumes *f*: ⋀*k*. *f k absolutely_integrable_on S*
and *h*: *h integrable_on S*
and *normg*: ⋀*x*. *x* ∈ *S* ⟹ *norm*(*g x*) ≤ (*h x*)
and *lim*: ⋀*x*. *x* ∈ *S* ⟹ (λ*k*. *f k x*) ⟶ *g x*
shows *g integrable_on S*
proof −
have *habs*: *h absolutely_integrable_on S*

```
  using h normg nonnegative_absolutely_integrable_1 norm_ge_zero order_trans
by blast
 let ?f = λn x. (min (max (− h x) (f n x)) (h x))
 have h0: h x ≥ 0 if x ∈ S for x
  using normg that by force
 have leh: norm (?f k x) ≤ h x if x ∈ S for k x
  using h0 that by force
 have limf: (λk. ?f k x) ⟶ g x if x ∈ S for x
 proof −
  have ⋀e y. |f y x − g x| < e ⟹ |min (max (− h x) (f y x)) (h x) − g x| < e
   using h0 [OF that] normg [OF that] by simp
  then show ?thesis
   using lim [OF that] by (auto simp add: tendsto_iff dist_norm elim!: eventu-
ally_mono)
 qed
 show ?thesis
 proof (rule dominated_convergence [of ?f S h g])
  have (λx. − h x) absolutely_integrable_on S
   using habs unfolding set_integrable_def by auto
  then show ?f k integrable_on S for k
   by (intro set_lebesgue_integral_eq_integral absolutely_integrable_min_1 ab-
solutely_integrable_max_1 f habs)
 qed (use assms leh limf in auto)
qed

lemma dominated_convergence_integrable:
 fixes f :: nat ⇒ 'n::euclidean_space ⇒ 'm::euclidean_space
 assumes f: ⋀k. f k absolutely_integrable_on S
  and h: h integrable_on S
  and normg: ⋀x. x ∈ S ⟹ norm(g x) ≤ (h x)
  and lim: ⋀x. x ∈ S ⟹ (λk. f k x) ⟶ g x
 shows g integrable_on S
 using f
 unfolding integrable_componentwise_iff [of g] absolutely_integrable_componentwise_iff
[where f = f k for k]
proof clarify
 fix b :: 'm
 assume fb [rule_format]: ⋀k. ∀ b∈Basis. (λx. f k x • b) absolutely_integrable_on
S and b: b ∈ Basis
 show (λx. g x • b) integrable_on S
 proof (rule dominated_convergence_integrable_1 [OF fb h])
  fix x
  assume x ∈ S
  show norm (g x • b) ≤ h x
   using norm_nth_le ‹x ∈ S› b normg order.trans by blast
  show (λk. f k x • b) ⟶ g x • b
   using ‹x ∈ S› b lim tendsto_componentwise_iff by fastforce
 qed (use b in auto)
qed
```

lemma *dominated_convergence_absolutely_integrable*:
fixes $f :: nat \Rightarrow 'n::euclidean\_space \Rightarrow 'm::euclidean\_space$
assumes f: $\bigwedge k.\ f\ k$ *absolutely_integrable_on* S
and h: h *integrable_on* S
and *normg*: $\bigwedge x.\ x \in S \Longrightarrow norm(g\ x) \leq (h\ x)$
and *lim*: $\bigwedge x.\ x \in S \Longrightarrow (\lambda k.\ f\ k\ x) \longrightarrow g\ x$
shows g *absolutely_integrable_on* S
proof −
have g *integrable_on* S
by (*rule dominated_convergence_integrable* [*OF assms*])
with *assms* **show** *?thesis*
by (*blast intro*: *absolutely_integrable_integrable_bound* [**where** $g{=}h$])
qed

proposition *integral_countable_UN*:
fixes $f :: real^{\frown}'m \Rightarrow real^{\frown}'n$
assumes f: f *absolutely_integrable_on* $(\bigcup (range\ s))$
and s: $\bigwedge m.\ s\ m \in sets\ lebesgue$
shows $\bigwedge n.\ f$ *absolutely_integrable_on* $(\bigcup m{\leq}n.\ s\ m)$
and $(\lambda n.\ integral\ (\bigcup m{\leq}n.\ s\ m)\ f) \longrightarrow integral\ (\bigcup (s\ `\ UNIV))\ f$ (**is** *?F* $\longrightarrow$ *?I*)
proof −
show *fU*: f *absolutely_integrable_on* $(\bigcup m{\leq}n.\ s\ m)$ **for** n
using *assms* **by** (*blast intro*: *set_integrable_subset* [*OF f*])
have *fint*: f *integrable_on* $(\bigcup\ (range\ s))$
using *absolutely_integrable_on_def f* **by** *blast*
let *?h* $= \lambda x.$ *if* $x \in \bigcup (s\ `\ UNIV)$ *then* $norm(f\ x)$ *else* 0
have $(\lambda n.\ integral\ UNIV\ (\lambda x.\ if\ x \in (\bigcup m{\leq}n.\ s\ m)\ then\ f\ x\ else\ 0))$
$\longrightarrow integral\ UNIV\ (\lambda x.\ if\ x \in \bigcup (s\ `\ UNIV)\ then\ f\ x\ else\ 0)$
proof (*rule dominated_convergence*)
show $(\lambda x.\ if\ x \in (\bigcup m{\leq}n.\ s\ m)\ then\ f\ x\ else\ 0)$ *integrable_on UNIV* **for** n
unfolding *integrable_restrict_UNIV*
using *fU absolutely_integrable_on_def* **by** *blast*
show $(\lambda x.\ if\ x \in \bigcup (s\ `\ UNIV)\ then\ norm(f\ x)\ else\ 0)$ *integrable_on UNIV*
by (*metis* (*no_types*) *absolutely_integrable_on_def f integrable_restrict_UNIV*)
show $\bigwedge x.\ (\lambda n.\ if\ x \in (\bigcup m{\leq}n.\ s\ m)\ then\ f\ x\ else\ 0)$
$\longrightarrow (if\ x \in \bigcup (s\ `\ UNIV)\ then\ f\ x\ else\ 0)$
by (*force intro*: *tendsto_eventually eventually_sequentiallyI*)
qed *auto*
then show *?F* $\longrightarrow$ *?I*
by (*simp only*: *integral_restrict_UNIV*)
qed

6.16.20 Fundamental Theorem of Calculus for the Lebesgue integral

For the positive integral we replace continuity with Borel-measurability.

lemma
fixes $f :: real \Rightarrow real$
assumes [*measurable*]: $f \in borel\_measurable\ borel$
assumes f: $\bigwedge x.\ x \in \{a..b\} \Longrightarrow DERIV\ F\ x :> f\ x$ $\bigwedge x.\ x \in \{a..b\} \Longrightarrow 0 \leq f\ x$ **and** $a \leq b$
shows *nn_integral_FTC_Icc*: $(\int^+ x.\ ennreal\ (f\ x) * indicator\ \{a\ ..\ b\}\ x\ \partial lborel) = F\ b - F\ a$ (**is** *?nn*)
and *has_bochner_integral_FTC_Icc_nonneg*:
has_bochner_integral lborel $(\lambda x.\ f\ x * indicator\ \{a\ ..\ b\}\ x)\ (F\ b - F\ a)$ (**is** *?has*)
and *integral_FTC_Icc_nonneg*: $(\int x.\ f\ x * indicator\ \{a\ ..\ b\}\ x\ \partial lborel) = F\ b - F\ a$ (**is** *?eq*)
and *integrable_FTC_Icc_nonneg*: *integrable lborel* $(\lambda x.\ f\ x * indicator\ \{a\ ..\ b\}\ x)$ (**is** *?int*)
proof −
have $*$: $(\lambda x.\ f\ x * indicator\ \{a..b\}\ x) \in borel\_measurable\ borel$ $\bigwedge x.\ 0 \leq f\ x * indicator\ \{a..b\}\ x$
using *f*(*2*) **by** (*auto split*: *split_indicator*)

have *F_mono*: $a \leq x \Longrightarrow x \leq y \Longrightarrow y \leq b \Longrightarrow F\ x \leq F\ y$ **for** *x y*
using *f* **by** (*intro DERIV_nonneg_imp_nondecreasing*[*of x y F*]) (*auto intro*: *order_trans*)

have (*f has_integral F b − F a*) {*a..b*}
by (*intro fundamental_theorem_of_calculus*)
(*auto simp*: *has_real_derivative_iff_has_vector_derivative*[*symmetric*]
intro: *has_field_derivative_subset*[*OF f*(*1*)] ‹$a \leq b$›)
then have *i*: $((\lambda x.\ f\ x * indicator\ \{a\ ..\ b\}\ x)\ has\_integral\ F\ b - F\ a)\ UNIV$
unfolding *indicator_def of_bool_def if_distrib*[**where** *f*=$\lambda x.\ a * x$ **for** *a*]
by (*simp cong del*: *if_weak_cong del*: *atLeastAtMost_iff*)
then have *nn*: $(\int^+ x.\ f\ x * indicator\ \{a\ ..\ b\}\ x\ \partial lborel) = F\ b - F\ a$
by (*rule nn_integral_has_integral_lborel*[*OF* $*$])
then show *?has*
by (*rule has_bochner_integral_nn_integral*[*rotated 3*]) (*simp_all add*: $*$ *F_mono* ‹$a \leq b$›)
then show *?eq ?int*
unfolding *has_bochner_integral_iff* **by** *auto*
show *?nn*
by (*subst nn*[*symmetric*])
(*auto intro*!: *nn_integral_cong simp add*: *ennreal_mult f split*: *split_indicator*)
qed

lemma
fixes $f :: real \Rightarrow 'a :: euclidean\_space$
assumes $a \leq b$
assumes $\bigwedge x.\ a \leq x \Longrightarrow x \leq b \Longrightarrow (F\ has\_vector\_derivative\ f\ x)\ (at\ x\ within\ \{a\ ..\ b\})$
assumes *cont*: *continuous_on* $\{a\ ..\ b\}\ f$
shows *has_bochner_integral_FTC_Icc*:

has_bochner_integral lborel (λ*x*. *indicator* {*a* .. *b*} *x* ∗$_R$ *f x*) (*F b* − *F a*) (**is** *?has*)
and *integral_FTC_Icc*: (∫ *x*. *indicator* {*a* .. *b*} *x* ∗$_R$ *f x* ∂*lborel*) = *F b* − *F a* (**is** *?eq*)
proof −
let *?f* = λ*x*. *indicator* {*a* .. *b*} *x* ∗$_R$ *f x*
have *int*: *integrable lborel ?f*
using *borel_integrable_compact*[*OF _ cont*] **by** *auto*
have (*f has_integral F b* − *F a*) {*a..b*}
using *assms*(*1*,*2*) **by** (*intro fundamental_theorem_of_calculus*) *auto*
moreover
have (*f has_integral integral*^{L} *lborel ?f*) {*a..b*}
using *has_integral_integral_lborel*[*OF int*]
unfolding *indicator_def of_bool_def if_distrib*[**where** *f*=λ*x*. *x* ∗$_R$ *a* **for** *a*]
by (*simp cong del*: *if_weak_cong del*: *atLeastAtMost_iff*)
ultimately show *?eq*
by (*auto dest*: *has_integral_unique*)
then show *?has*
using *int* **by** (*auto simp*: *has_bochner_integral_iff*)
qed

lemma
fixes *f* :: *real* ⇒ *real*
assumes *a* ≤ *b*
assumes *deriv*: ⋀*x*. *a* ≤ *x* ⟹ *x* ≤ *b* ⟹ *DERIV F x* :> *f x*
assumes *cont*: ⋀*x*. *a* ≤ *x* ⟹ *x* ≤ *b* ⟹ *isCont f x*
shows *has_bochner_integral_FTC_Icc_real*:
has_bochner_integral lborel (λ*x*. *f x* ∗ *indicator* {*a* .. *b*} *x*) (*F b* − *F a*) (**is** *?has*)
and *integral_FTC_Icc_real*: (∫ *x*. *f x* ∗ *indicator* {*a* .. *b*} *x* ∂*lborel*) = *F b* − *F a* (**is** *?eq*)
proof −
have *1*: ⋀*x*. *a* ≤ *x* ⟹ *x* ≤ *b* ⟹ (*F has_vector_derivative f x*) (*at x within* {*a* .. *b*})
unfolding *has_real_derivative_iff_has_vector_derivative*[*symmetric*]
using *deriv* **by** (*auto intro*: *DERIV_subset*)
have *2*: *continuous_on* {*a* .. *b*} *f*
using *cont* **by** (*intro continuous_at_imp_continuous_on*) *auto*
show *?has ?eq*
using *has_bochner_integral_FTC_Icc*[*OF* ‹*a* ≤ *b*› *1 2*] *integral_FTC_Icc*[*OF* ‹*a* ≤ *b*› *1 2*]
by (*auto simp*: *mult.commute*)
qed

lemma *nn_integral_FTC_atLeast*:
fixes *f* :: *real* ⇒ *real*
assumes *f_borel*: *f* ∈ *borel_measurable borel*
assumes *f*: ⋀*x*. *a* ≤ *x* ⟹ *DERIV F x* :> *f x*
assumes *nonneg*: ⋀*x*. *a* ≤ *x* ⟹ *0* ≤ *f x*

```
  assumes lim: (F ⟶ T) at_top
  shows (∫⁺x. ennreal (f x) * indicator {a ..} x ∂lborel) = T - F a
proof -
  let ?f = λ(i::nat) (x::real). ennreal (f x) * indicator {a..a + real i} x
  let ?fR = λx. ennreal (f x) * indicator {a ..} x

  have F_mono: a ≤ x ⟹ x ≤ y ⟹ F x ≤ F y for x y
    using f nonneg by (intro DERIV_nonneg_imp_nondecreasing[of x y F]) (auto
intro: order_trans)
  then have F_le_T: a ≤ x ⟹ F x ≤ T for x
    by (intro tendsto_lowerbound[OF lim])
       (auto simp: eventually_at_top_linorder)

  have (SUP i. ?f i x) = ?fR x for x
  proof (rule LIMSEQ_unique[OF LIMSEQ_SUP])
    obtain n where x - a < real n
      using reals_Archimedean2[of x - a] ..
    then have eventually (λn. ?f n x = ?fR x) sequentially
      by (auto intro!: eventually_sequentiallyI[where c=n] split: split_indicator)
    then show (λn. ?f n x) ⟶ ?fR x
      by (rule tendsto_eventually)
  qed (auto simp: nonneg incseq_def le_fun_def split: split_indicator)
  then have integral^N lborel ?fR = (∫⁺ x. (SUP i. ?f i x) ∂lborel)
    by simp
  also have ... = (SUP i. (∫⁺ x. ?f i x ∂lborel))
  proof (rule nn_integral_monotone_convergence_SUP)
    show incseq ?f
      using nonneg by (auto simp: incseq_def le_fun_def split: split_indicator)
    show ⋀i. (?f i) ∈ borel_measurable lborel
      using f_borel by auto
  qed
  also have ... = (SUP i. ennreal (F (a + real i) - F a))
    by (subst nn_integral_FTC_Icc[OF f_borel f nonneg]) auto
  also have ... = T - F a
  proof (rule LIMSEQ_unique[OF LIMSEQ_SUP])
    have (λx. F (a + real x)) ⟶ T
      by (auto intro: filterlim_compose[OF lim filterlim_tendsto_add_at_top]
filterlim_real_sequentially)
    then show (λn. ennreal (F (a + real n) - F a)) ⟶ ennreal (T - F a)
      by (simp add: F_mono F_le_T tendsto_diff)
  qed (auto simp: incseq_def intro!: ennreal_le_iff[THEN iffD2] F_mono)
  finally show ?thesis .
qed

lemma integral_power:
  a ≤ b ⟹ (∫ x. x^k * indicator {a..b} x ∂lborel) = (b^Suc k - a^Suc k) / Suc k
proof (subst integral_FTC_Icc_real)
  fix x show DERIV (λx. x^Suc k / Suc k) x :> x^k
    by (intro derivative_eq_intros) auto
```

qed (*auto simp*: *field_simps simp del*: *of_nat_Suc*)

6.16.21 Integration by parts

lemma *integral_by_parts_integrable*:
fixes *f g F G*::*real* ⇒ *real*
assumes $a \leq b$
assumes *cont_f*[*intro*]: !!*x*. $a \leq x \Longrightarrow x \leq b \Longrightarrow$ *isCont f x*
assumes *cont_g*[*intro*]: !!*x*. $a \leq x \Longrightarrow x \leq b \Longrightarrow$ *isCont g x*
assumes [*intro*]: !!*x*. *DERIV F x* :> *f x*
assumes [*intro*]: !!*x*. *DERIV G x* :> *g x*
shows *integrable lborel* (λ*x*.((*F x*) ∗ (*g x*) + (*f x*) ∗ (*G x*)) ∗ *indicator* {*a* .. *b*} *x*)
by (*auto intro*!: *borel_integrable_atLeastAtMost continuous_intros*) (*auto intro*!: *DERIV_isCont*)

lemma *integral_by_parts*:
fixes *f g F G*::*real* ⇒ *real*
assumes [*arith*]: $a \leq b$
assumes *cont_f*[*intro*]: !!*x*. $a \leq x \Longrightarrow x \leq b \Longrightarrow$ *isCont f x*
assumes *cont_g*[*intro*]: !!*x*. $a \leq x \Longrightarrow x \leq b \Longrightarrow$ *isCont g x*
assumes [*intro*]: !!*x*. *DERIV F x* :> *f x*
assumes [*intro*]: !!*x*. *DERIV G x* :> *g x*
shows (∫ *x*. (*F x* ∗ *g x*) ∗ *indicator* {*a* .. *b*} *x* ∂*lborel*)
= *F b* ∗ *G b* − *F a* ∗ *G a* − ∫ *x*. (*f x* ∗ *G x*) ∗ *indicator* {*a* .. *b*} *x* ∂*lborel*
proof−
have (∫ *x*. (*F x* ∗ *g x* + *f x* ∗ *G x*) ∗ *indicator* {*a* .. *b*} *x* ∂*lborel*)
= *LBINT x*. *F x* ∗ *g x* ∗ *indicat_real* {*a*..*b*} *x* + *f x* ∗ *G x* ∗ *indicat_real* {*a*..*b*} *x*
by (*meson vector_space_over_itself*.*scale_left_distrib*)
also have ... = (∫ *x*. (*F x* ∗ *g x*) ∗ *indicator* {*a* .. *b*} *x* ∂*lborel*) + ∫ *x*. (*f x* ∗ *G x*) ∗ *indicator* {*a* .. *b*} *x* ∂*lborel*
proof (*intro Bochner_Integration.integral_add borel_integrable_atLeastAtMost cont_f cont_g continuous_intros*)
show ⋀*x*. ⟦*a* ≤ *x*; *x* ≤ *b*⟧ ⟹ *isCont F x* ⋀*x*. ⟦*a* ≤ *x*; *x* ≤ *b*⟧ ⟹ *isCont G x*
using *DERIV_isCont* **by** *blast*+
qed
finally have (∫ *x*. (*F x* ∗ *g x* + *f x* ∗ *G x*) ∗ *indicator* {*a* .. *b*} *x* ∂*lborel*) =
(∫ *x*. (*F x* ∗ *g x*) ∗ *indicator* {*a* .. *b*} *x* ∂*lborel*) + ∫ *x*. (*f x* ∗ *G x*) ∗ *indicator* {*a* .. *b*} *x* ∂*lborel* .
moreover have (∫ *x*. (*F x* ∗ *g x* + *f x* ∗ *G x*) ∗ *indicator* {*a* .. *b*} *x* ∂*lborel*) = *F b* ∗ *G b* − *F a* ∗ *G a*
proof (*intro integral_FTC_Icc_real derivative_eq_intros cont_f cont_g continuous_intros*)
show ⋀*x*. ⟦*a* ≤ *x*; *x* ≤ *b*⟧ ⟹ *isCont F x* ⋀*x*. ⟦*a* ≤ *x*; *x* ≤ *b*⟧ ⟹ *isCont G x*
using *DERIV_isCont* **by** *blast*+
qed *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *integral_by_parts'*:
fixes *f g F G*::*real* ⇒ *real*
assumes *a* ≤ *b*
assumes !!*x*. *a* ≤*x* ⟹ *x*≤*b* ⟹ *isCont f x*
assumes !!*x*. *a* ≤*x* ⟹ *x*≤*b* ⟹ *isCont g x*
assumes !!*x*. *DERIV F x* :> *f x*
assumes !!*x*. *DERIV G x* :> *g x*
shows (∫ *x*. *indicator* {*a* .. *b*} *x* ∗*R* (*F x* ∗ *g x*) ∂*lborel*)
= *F b* ∗ *G b* − *F a* ∗ *G a* − ∫ *x*. *indicator* {*a* .. *b*} *x* ∗*R* (*f x* ∗ *G x*) ∂*lborel*
using *integral_by_parts*[*OF assms*] **by** (*simp add*: *ac_simps*)

lemma *has_bochner_integral_even_function*:
fixes *f* :: *real* ⇒ *'a* :: {*banach*, *second_countable_topology*}
assumes *f*: *has_bochner_integral lborel* (λ*x*. *indicator* {*0*..} *x* ∗*R f x*) *x*
assumes *even*: ⋀*x*. *f* (− *x*) = *f x*
shows *has_bochner_integral lborel f* (*2* ∗*R x*)
proof −
have *indicator*: ⋀*x*::*real*. *indicator* {..*0*} (− *x*) = *indicator* {*0*..} *x*
by (*auto split*: *split_indicator*)
have *has_bochner_integral lborel* (λ*x*. *indicator* {.. *0*} *x* ∗*R f x*) *x*
by (*subst lborel_has_bochner_integral_real_affine_iff*[**where** *c*=−*1* **and** *t*=*0*])
(*auto simp*: *indicator even f*)
with *f* **have** *has_bochner_integral lborel* (λ*x*. *indicator* {*0*..} *x* ∗*R f x* + *indicator* {.. *0*} *x* ∗*R f x*) (*x* + *x*)
by (*rule has_bochner_integral_add*)
then have *has_bochner_integral lborel f* (*x* + *x*)
by (*rule has_bochner_integral_discrete_difference*[**where** *X*={*0*}, *THEN iffD1*, *rotated 4*])
(*auto split*: *split_indicator*)
then show *?thesis*
by (*simp add*: *scaleR_2*)
qed

lemma *has_bochner_integral_odd_function*:
fixes *f* :: *real* ⇒ *'a* :: {*banach*, *second_countable_topology*}
assumes *f*: *has_bochner_integral lborel* (λ*x*. *indicator* {*0*..} *x* ∗*R f x*) *x*
assumes *odd*: ⋀*x*. *f* (− *x*) = − *f x*
shows *has_bochner_integral lborel f 0*
proof −
have *indicator*: ⋀*x*::*real*. *indicator* {..*0*} (− *x*) = *indicator* {*0*..} *x*
by (*auto split*: *split_indicator*)
have *has_bochner_integral lborel* (λ*x*. − *indicator* {.. *0*} *x* ∗*R f x*) *x*
by (*subst lborel_has_bochner_integral_real_affine_iff*[**where** *c*=−*1* **and** *t*=*0*])
(*auto simp*: *indicator odd f*)
from *has_bochner_integral_minus*[*OF this*]
have *has_bochner_integral lborel* (λ*x*. *indicator* {.. *0*} *x* ∗*R f x*) (− *x*)
by *simp*

with *f* **have** *has_bochner_integral lborel* (λ*x*. *indicator* {*0*..} *x* ∗$_R$ *f x* + *indicator* {.. *0*} *x* ∗$_R$ *f x*) (*x* + − *x*)
by (*rule has_bochner_integral_add*)
then have *has_bochner_integral lborel f* (*x* + − *x*)
by (*rule has_bochner_integral_discrete_difference*[**where** *X*={*0*}, *THEN iffD1*, *rotated 4*])
(*auto split*: *split_indicator*)
then show *?thesis*
by *simp*
qed

6.16.22 A non-negative continuous function whose integral is zero must be zero

lemma *has_integral_0_closure_imp_0*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *real*
assumes *f*: *continuous_on* (*closure S*) *f*
and *nonneg_interior*: ⋀*x*. *x* ∈ *S* ⟹ *0* ≤ *f x*
and *pos*: *0* < *emeasure lborel S*
and *finite*: *emeasure lborel S* < ∞
and *regular*: *emeasure lborel* (*closure S*) = *emeasure lborel S*
and *opn*: *open S*
assumes *int*: (*f has_integral 0*) (*closure S*)
assumes *x*: *x* ∈ *closure S*
shows *f x* = *0*
proof −
have *zero*: *emeasure lborel* (*frontier S*) = *0*
using *finite closure_subset regular*
unfolding *frontier_def*
by (*subst emeasure_Diff*) (*auto simp*: *frontier_def interior_open* ‹*open S*›)
have *nonneg*: *0* ≤ *f x* **if** *x* ∈ *closure S* **for** *x*
using *continuous_ge_on_closure*[*OF f that nonneg_interior*] **by** *simp*
have *0* = *integral* (*closure S*) *f*
by (*blast intro*: *int sym*)
also
note *intl* = *has_integral_integrable*[*OF int*]
have *af*: *f absolutely_integrable_on* (*closure S*)
using *nonneg*
by (*intro absolutely_integrable_onI intl integrable_eq*[*OF intl*]) *simp*
then have *integral* (*closure S*) *f* = *set_lebesgue_integral lebesgue* (*closure S*) *f*
by (*intro set_lebesgue_integral_eq_integral*(*2*)[*symmetric*])
also have ... = *0* ⟷ (*AE x in lebesgue*. *indicator* (*closure S*) *x* ∗$_R$ *f x* = *0*)
unfolding *set_lebesgue_integral_def*
proof (*rule integral_nonneg_eq_0_iff_AE*)
show *integrable lebesgue* (λ*x*. *indicat_real* (*closure S*) *x* ∗$_R$ *f x*)
by (*metis af set_integrable_def*)
qed (*use nonneg* **in** ‹*auto simp*: *indicator_def*›)
also have ... ⟷ (*AE x in lebesgue*. *x* ∈ {*x*. *x* ∈ *closure S* ⟶ *f x* = *0*})
by (*auto simp*: *indicator_def*)

```
  finally have (AE x in lebesgue. x ∈ {x. x ∈ closure S ⟶ f x = 0}) by simp
  moreover have (AE x in lebesgue. x ∈ - frontier S)
    using zero
   by (auto simp: eventually_ae_filter null_sets_def intro!: exI[where x=frontier
S])
  ultimately have ae: AE x ∈ S in lebesgue. x ∈ {x ∈ closure S. f x = 0} (is
?th)
    by eventually_elim (use closure_subset in ‹auto simp: ›)
  have closed {0::real} by simp
  with continuous_on_closed_vimage[OF closed_closure, of S f] f
  have closed (f -‘ {0} ∩ closure S) by blast
  then have closed {x ∈ closure S. f x = 0} by (auto simp: vimage_def Int_def
conj_commute)
  with ‹open S› have x ∈ {x ∈ closure S. f x = 0} if x ∈ S for x using ae that
    by (rule mem_closed_if_AE_lebesgue_open)
  then have f x = 0 if x ∈ S for x using that by auto
  from continuous_constant_on_closure[OF f this ‹x ∈ closure S›]
  show f x = 0 .
qed

lemma has_integral_0_cbox_imp_0:
  fixes f :: 'a::euclidean_space ⇒ real
  assumes continuous_on (cbox a b) f and ⋀x. x ∈ box a b ⟹ 0 ≤ f x
  assumes (f has_integral 0) (cbox a b)
  assumes ne: box a b ≠ {}
  assumes x: x ∈ cbox a b
  shows f x = 0
proof -
  have 0 < emeasure lborel (box a b)
    using ne x unfolding emeasure_lborel_box_eq
    by (force intro!: prod_pos simp: mem_box algebra_simps)
  then show ?thesis using assms
    by (intro has_integral_0_closure_imp_0[of box a b f x])
     (auto simp: emeasure_lborel_box_eq emeasure_lborel_cbox_eq algebra_simps
mem_box)
qed

corollary integral_cbox_eq_0_iff:
  fixes f :: 'a::euclidean_space ⇒ real
  assumes continuous_on (cbox a b) f and box a b ≠ {}
    and ⋀x. x ∈ cbox a b ⟹ f x ≥ 0
  shows integral (cbox a b) f = 0 ⟷ (∀ x ∈ cbox a b. f x = 0) (is ?lhs = ?rhs)
proof
  assume int0: ?lhs
  show ?rhs
    using has_integral_0_cbox_imp_0 [of a b f] assms
    by (metis box_subset_cbox eq_integralD int0 integrable_continuous subsetD)
next
  assume ?rhs then show ?lhs
```

by (*meson has_integral_is_0_cbox integral_unique*)
qed

lemma *integral_eq_0_iff*:
fixes $f :: real \Rightarrow real$
assumes *continuous_on* $\{a..b\}$ f **and** $a < b$
and $\bigwedge x.\ x \in \{a..b\} \Longrightarrow f\ x \geq 0$
shows *integral* $\{a..b\}$ $f = 0 \longleftrightarrow (\forall\, x \in \{a..b\}.\ f\ x = 0)$
using *integral_cbox_eq_0_iff* [*of a b f*] *assms* **by** *simp*

lemma *integralL_eq_0_iff*:
fixes $f :: real \Rightarrow real$
assumes *contf*: *continuous_on* $\{a..b\}$ f **and** $a < b$
and $\bigwedge x.\ x \in \{a..b\} \Longrightarrow f\ x \geq 0$
shows $integral^L$ (*lebesgue_on* $\{a..b\}$) $f = 0 \longleftrightarrow (\forall\, x \in \{a..b\}.\ f\ x = 0)$
using *integral_eq_0_iff* [*OF assms*]
by (*simp add*: *contf continuous_imp_integrable_real lebesgue_integral_eq_integral*)

In fact, strict inequality is required only at a single point within the box.

lemma *integral_less*:
fixes $f :: 'n::euclidean\_space \Rightarrow real$
assumes *cont*: *continuous_on* (*cbox a b*) f *continuous_on* (*cbox a b*) g **and** *box a b* $\neq \{\}$
and *fg*: $\bigwedge x.\ x \in box\ a\ b \Longrightarrow f\ x < g\ x$
shows *integral* (*cbox a b*) $f <$ *integral* (*cbox a b*) g
proof −
obtain *int*: f *integrable_on* (*cbox a b*) g *integrable_on* (*cbox a b*)
using *cont integrable_continuous* **by** *blast*
then have *integral* (*cbox a b*) $f \leq$ *integral* (*cbox a b*) g
by (*metis fg integrable_on_open_interval integral_le integral_open_interval less_eq_real_def*)
moreover have *integral* (*cbox a b*) $f \neq$ *integral* (*cbox a b*) g
proof (*rule ccontr*)
assume ¬ *integral* (*cbox a b*) $f \neq$ *integral* (*cbox a b*) g
then have *0*: $((\lambda x.\ g\ x - f\ x)$ *has_integral 0*$)$ (*cbox a b*)
by (*metis* (*full_types*) *cancel_comm_monoid_add_class.diff_cancel has_integral_diff int integrable_integral*)
have *cgf*: *continuous_on* (*cbox a b*) $(\lambda x.\ g\ x - f\ x)$
using *cont continuous_on_diff* **by** *blast*
show *False*
using *has_integral_0_cbox_imp_0* [*OF cgf _ 0*] *assms*(*3*) *box_subset_cbox fg less_eq_real_def* **by** *fastforce*
qed
ultimately show *?thesis*
by *linarith*
qed

lemma *integral_less_real*:
fixes $f :: real \Rightarrow real$

assumes *continuous_on* {*a*..*b*} *f continuous_on* {*a*..*b*} *g* **and** {*a*<..<*b*} ≠ {}
and ⋀*x*. *x* ∈ {*a*<..<*b*} ⟹ *f x* < *g x*
shows *integral* {*a*..*b*} *f* < *integral* {*a*..*b*} *g*
by (*metis assms box_real integral_less*)

6.16.23 Various common equivalent forms of function measurability

lemma *indicator_sum_eq*:
fixes *m*::*real* **and** *f* :: *'a* ⇒ *real*
assumes |*m*| ≤ 2 ^ (2∗*n*) *m*/2^*n* ≤ *f x f x* < (*m*+1)/2^*n m* ∈ ℤ
shows (∑ *k*::*real* | *k* ∈ ℤ ∧ |*k*| ≤ 2 ^ (2∗*n*).
k/2^*n* ∗ *indicator* {*y*. *k*/2^*n* ≤ *f y* ∧ *f y* < (*k*+1)/2^*n*} *x*) = *m*/2^*n*
(**is** *sum ?f ?S* = _)
proof −
have *sum ?f ?S* = *sum* (λ*k*. *k*/2^*n* ∗ *indicator* {*y*. *k*/2^*n* ≤ *f y* ∧ *f y* < (*k*+1)/2^*n*} *x*) {*m*}
proof (*rule comm_monoid_add_class.sum.mono_neutral_right*)
show *finite ?S*
by (*rule finite_abs_int_segment*)
show {*m*} ⊆ {*k* ∈ ℤ. |*k*| ≤ 2 ^ (2∗*n*)}
using *assms* **by** *auto*
show ∀ *i*∈{*k* ∈ ℤ. |*k*| ≤ 2 ^ (2∗*n*)} − {*m*}. *?f i* = *0*
using *assms* **by** (*auto simp*: *indicator_def Ints_def abs_le_iff field_split_simps*)
qed
also have ... = *m*/2^*n*
using *assms* **by** (*auto simp*: *indicator_def not_less*)
finally show *?thesis* **.**
qed

lemma *measurable_on_sf_limit_lemma1*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *real*
assumes ⋀*a b*. {*x* ∈ *S*. *a* ≤ *f x* ∧ *f x* < *b*} ∈ *sets* (*lebesgue_on S*)
obtains *g* **where** ⋀*n*. *g n* ∈ *borel_measurable* (*lebesgue_on S*)
⋀*n*. *finite*(*range* (*g n*))
⋀*x*. (λ*n*. *g n x*) ⟶ *f x*
proof
show (λ*x*. *sum* (λ*k*::*real*. *k*/2^*n* ∗ *indicator* {*y*. *k*/2^*n* ≤ *f y* ∧ *f y* < (*k*+1)/2^*n*} *x*)
{*k* ∈ ℤ. |*k*| ≤ 2 ^ (2∗*n*)}) ∈ *borel_measurable* (*lebesgue_on S*)
(**is** *?g* ∈ _) **for** *n*
proof −
have ⋀*k*. ⟦*k* ∈ ℤ; |*k*| ≤ 2 ^ (2∗*n*)⟧
⟹ *Measurable.pred* (*lebesgue_on S*) (λ*x*. *k* / (2^*n*) ≤ *f x* ∧ *f x* < (*k*+1) / (2^*n*))
using *assms* **by** (*force simp*: *pred_def space_restrict_space*)
then show *?thesis*
by (*simp add*: *field_class.field_divide_inverse*)
qed

```
  show finite (range (?g n)) for n
  proof −
    have range (?g n) ⊆ (λk. k/2^n) ‘ {k ∈ ℤ. |k| ≤ 2 ^ (2∗n)}
    proof clarify
      fix x
      show ?g n x ∈ (λk. k/2^n) ‘ {k ∈ ℤ. |k| ≤ 2 ^ (2∗n)}
       proof (cases ∃ k::real. k ∈ ℤ ∧ |k| ≤ 2 ^ (2∗n) ∧ k/2^n ≤ (f x) ∧ (f x) <
(k+1)/2^n)
        case True
        then show ?thesis
          apply clarify
          by (subst indicator_sum_eq) auto
      next
        case False
        then have ?g n x = 0 by auto
        then show ?thesis by force
      qed
    qed
    moreover have finite ((λk::real. (k/2^n)) ‘ {k ∈ ℤ. |k| ≤ 2 ^ (2∗n)})
      by (simp add: finite_abs_int_segment)
    ultimately show ?thesis
      using finite_subset by blast
  qed
  show (λn. ?g n x) ⟶ f x for x
  proof (rule LIMSEQ_I)
    fix e::real
    assume e > 0
    obtain N1 where N1: |f x| < 2 ^ N1
      using real_arch_pow by fastforce
    obtain N2 where N2: (1/2) ^ N2 < e
      using real_arch_pow_inv ‹e > 0› by force
    have norm (?g n x − f x) < e if n: n ≥ max N1 N2 for n
    proof −
      define m where m ≡ floor(2^n ∗ (f x))
      have 1 ≤ |2^n| ∗ e
       using n N2 ‹e > 0› less_eq_real_def less_le_trans by (fastforce simp add:
field_split_simps)
      then have ∗: ⟦x ≤ y; y < x + 1⟧ ⟹ abs(x − y) < |2^n| ∗ e for x y::real
        by linarith
      have |2^n| ∗ |m/2^n − f x| = |2^n ∗ (m/2^n − f x)|
        by (simp add: abs_mult)
      also have ... = |real_of_int ⌊2^n ∗ f x⌋ − f x ∗ 2^n|
        by (simp add: algebra_simps m_def)
      also have ... < |2^n| ∗ e
        by (rule ∗; simp add: mult.commute)
      finally have |2^n| ∗ |m/2^n − f x| < |2^n| ∗ e .
      then have me: |m/2^n − f x| < e
        by simp
      have |real_of_int m| ≤ 2 ^ (2∗n)
```

```
    proof (cases f x < 0)
      case True
      then have −⌊f x⌋ ≤ ⌊(2::real) ^ N1⌋
        using N1 le_floor_iff minus_le_iff by fastforce
      with n True have |real_of_int⌊f x⌋| ≤ 2 ^ N1
        by linarith
      also have ... ≤ 2^n
        using n by (simp add: m_def)
      finally have |real_of_int ⌊f x⌋| * 2^n ≤ 2^n * 2^n
        by simp
      moreover
      have |real_of_int ⌊2^n * f x⌋| ≤ |real_of_int ⌊f x⌋| * 2^n
      proof −
        have |real_of_int ⌊2^n * f x⌋| = − (real_of_int ⌊2^n * f x⌋)
          using True by (simp add: abs_if mult_less_0_iff)
        also have ... ≤ − (real_of_int (⌊(2::real) ^ n⌋ * ⌊f x⌋))
          using le_mult_floor_Ints [of (2::real)^n] by simp
        also have ... ≤ |real_of_int ⌊f x⌋| * 2^n
          using True
          by simp
        finally show ?thesis .
      qed
      ultimately show ?thesis
      by (metis (no_types, opaque_lifting) m_def order_trans power2_eq_square
power_even_eq)
    next
      case False
      with n N1 have f x ≤ 2^n
         by (simp add: not_less) (meson less_eq_real_def one_le_numeral or-
der_trans power_increasing)
      moreover have 0 ≤ m
        using False m_def by force
      ultimately show ?thesis
       by (metis abs_of_nonneg floor_mono le_floor_iff m_def of_int_0_le_iff
power2_eq_square power_mult mult_le_cancel_iff1 zero_less_numeral mult.commute
zero_less_power)
    qed
    then have ?g n x = m/2^n
        by (rule indicator_sum_eq) (auto simp add: m_def field_split_simps,
linarith)
    then have norm (?g n x − f x) = norm (m/2^n − f x)
      by simp
    also have ... < e
      by (simp add: me)
    finally show ?thesis .
  qed
  then show ∃no. ∀n≥no. norm (?g n x − f x) < e
    by blast
qed
```

qed

lemma *borel_measurable_simple_function_limit*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
shows $f \in borel\_measurable\ (lebesgue\_on\ S) \longleftrightarrow$
$(\exists g.\ (\forall n.\ (g\ n) \in borel\_measurable\ (lebesgue\_on\ S)) \wedge$
$(\forall n.\ finite\ (range\ (g\ n))) \wedge (\forall x.\ (\lambda n.\ g\ n\ x) \longrightarrow f\ x))$
proof −
have $\exists g.\ (\forall n.\ (g\ n) \in borel\_measurable\ (lebesgue\_on\ S)) \wedge$
$(\forall n.\ finite\ (range\ (g\ n))) \wedge (\forall x.\ (\lambda n.\ g\ n\ x) \longrightarrow f\ x)$
if f: $\bigwedge a\ i.\ i \in Basis \Longrightarrow \{x \in S.\ f\ x \cdot i < a\} \in sets\ (lebesgue\_on\ S)$
proof −
have $\exists g.\ (\forall n.\ (g\ n) \in borel\_measurable\ (lebesgue\_on\ S)) \wedge$
$(\forall n.\ finite(image\ (g\ n)\ UNIV)) \wedge$
$(\forall x.\ ((\lambda n.\ g\ n\ x) \longrightarrow f\ x \cdot i))$ **if** $i \in Basis$ **for** i
proof (*rule measurable_on_sf_limit_lemma1* [*of S* $\lambda x.\ f\ x \cdot i$])
show $\{x \in S.\ a \le f\ x \cdot i \wedge f\ x \cdot i < b\} \in sets\ (lebesgue\_on\ S)$ **for** $a\ b$
proof −
have $\{x \in S.\ a \le f\ x \cdot i \wedge f\ x \cdot i < b\} = \{x \in S.\ f\ x \cdot i < b\} - \{x \in S.\ a > f\ x \cdot i\}$
by *auto*
also have $\ldots \in sets\ (lebesgue\_on\ S)$
using *f that* **by** *blast*
finally show *?thesis* **.**
qed
qed *blast*
then obtain g **where** g:
$\bigwedge i\ n.\ i \in Basis \Longrightarrow g\ i\ n \in borel\_measurable\ (lebesgue\_on\ S)$
$\bigwedge i\ n.\ i \in Basis \Longrightarrow finite(range\ (g\ i\ n))$
$\bigwedge i\ x.\ i \in Basis \Longrightarrow ((\lambda n.\ g\ i\ n\ x) \longrightarrow f\ x \cdot i)$
by *metis*
show *?thesis*
proof (*intro conjI allI exI*)
show $(\lambda x.\ \sum i{\in}Basis.\ g\ i\ n\ x *_R i) \in borel\_measurable\ (lebesgue\_on\ S)$ **for** n
by (*intro borel_measurable_sum borel_measurable_scaleR*) (*auto intro*: *g*)
show $finite\ (range\ (\lambda x.\ \sum i{\in}Basis.\ g\ i\ n\ x *_R i))$ **for** n
proof −
have $range\ (\lambda x.\ \sum i{\in}Basis.\ g\ i\ n\ x *_R i) \subseteq (\lambda h.\ \sum i{\in}Basis.\ h\ i *_R i)\ `\ PiE\ Basis\ (\lambda i.\ range\ (g\ i\ n))$
proof *clarify*
fix x
show $(\sum i{\in}Basis.\ g\ i\ n\ x *_R i) \in (\lambda h.\ \sum i{\in}Basis.\ h\ i *_R i)\ `\ (\Pi_E\ i{\in}Basis.\ range\ (g\ i\ n))$
by (*rule_tac* $x{=}\lambda i{\in}Basis.\ g\ i\ n\ x$ **in** *image_eqI*) *auto*
qed
moreover have $finite(PiE\ Basis\ (\lambda i.\ range\ (g\ i\ n)))$
by (*simp add*: *g finite_PiE*)

```
      ultimately show ?thesis
        by (metis (mono_tags, lifting) finite_surj)
    qed
    show (λn. ∑ i∈Basis. g i n x *R i) ⟶ f x for x
    proof −
      have (λn. ∑ i∈Basis. g i n x *R i) ⟶ (∑ i∈Basis. (f x • i) *R i)
        by (auto intro!: tendsto_sum tendsto_scaleR g)
      moreover have (∑ i∈Basis. (f x • i) *R i) = f x
        using euclidean_representation by blast
      ultimately show ?thesis
        by metis
    qed
  qed
 qed
 moreover have f ∈ borel_measurable (lebesgue_on S)
          if meas_g: ⋀n. g n ∈ borel_measurable (lebesgue_on S)
            and fin: ⋀n. finite (range (g n))
            and to_f: ⋀x. (λn. g n x) ⟶ f x for g
   by (rule borel_measurable_LIMSEQ_metric [OF meas_g to_f])
 ultimately show ?thesis
   using borel_measurable_vimage_halfspace_component_lt by blast
qed
```

6.16.24 Lebesgue sets and continuous images

```
proposition lebesgue_regular_inner:
 assumes S ∈ sets lebesgue
 obtains K C where negligible K ⋀n::nat. compact(C n) S = (⋃ n. C n) ∪ K
proof −
  have ∃ T. closed T ∧ T ⊆ S ∧ (S − T) ∈ lmeasurable ∧ emeasure lebesgue (S
− T) < ennreal ((1/2) ^ n) for n
    using sets_lebesgue_inner_closed assms
   by (metis sets_lebesgue_inner_closed zero_less_divide_1_iff zero_less_numeral
zero_less_power)
  then obtain C where clo: ⋀n. closed (C n) and subS: ⋀n. C n ⊆ S
    and mea: ⋀n. (S − C n) ∈ lmeasurable
    and less: ⋀n. emeasure lebesgue (S − C n) < ennreal ((1/2) ^ n)
    by metis
  have ∃ F. (∀ n::nat. compact(F n)) ∧ (⋃ n. F n) = C m for m::nat
    by (metis clo closed_Union_compact_subsets)
  then obtain D :: [nat,nat] ⇒ 'a set where D: ⋀m n. compact(D m n) ⋀m.
(⋃ n. D m n) = C m
    by metis
  let ?C = from_nat_into (⋃ m. range (D m))
  have countable (⋃ m. range (D m))
    by blast
  have range (from_nat_into (⋃ m. range (D m))) = (⋃ m. range (D m))
    using range_from_nat_into by simp
  then have CD: ∃ m n. ?C k = D m n  for k
```

```
    by (metis (mono_tags, lifting) UN_iff rangeE range_eqI)
  show thesis
  proof
    show negligible (S - (⋃n. C n))
    proof (clarsimp simp: negligible_outer_le)
      fix e :: real
      assume e > 0
      then obtain n where n: (1/2)^n < e
        using real_arch_pow_inv [of e 1/2] by auto
     show ∃T. S - (⋃n. C n) ⊆ T ∧ T ∈ lmeasurable ∧ measure lebesgue T ≤ e
      proof (intro exI conjI)
        show S - (⋃n. C n) ⊆ S - C n
          by blast
        show S - C n ∈ lmeasurable
          by (simp add: mea)
        show measure lebesgue (S - C n) ≤ e
          using less [of n] n
          by (simp add: emeasure_eq_measure2 less_le mea)
      qed
    qed
    show compact (?C n) for n
      using CD D by metis
    show S = (⋃n. ?C n) ∪ (S - (⋃n. C n)) (is _ = ?rhs)
    proof
      show S ⊆ ?rhs
        using D by fastforce
      show ?rhs ⊆ S
        using subS D CD by auto (metis Sup_upper range_eqI subsetCE)
    qed
  qed
qed

lemma sets_lebesgue_continuous_image:
  assumes T: T ∈ sets lebesgue and contf: continuous_on S f
    and negim: ⋀T. ⟦negligible T; T ⊆ S⟧ ⟹ negligible(f ` T) and T ⊆ S
 shows f ` T ∈ sets lebesgue
proof -
  obtain K C where negligible K and com: ⋀n::nat. compact(C n) and Teq: T
= (⋃n. C n) ∪ K
    using lebesgue_regular_inner [OF T] by metis
  then have comf: ⋀n::nat. compact(f ` C n)
    by (metis Un_subset_iff Union_upper ‹T ⊆ S› compact_continuous_image
contf continuous_on_subset rangeI)
  have ((⋃n. f ` C n) ∪ f ` K) ∈ sets lebesgue
  proof (rule sets.Un)
    have K ⊆ S
      using Teq ‹T ⊆ S› by blast
    show (⋃n. f ` C n) ∈ sets lebesgue
    proof (rule sets.countable_Union)
```

```
    show range (λn. f ‘ C n) ⊆ sets lebesgue
      using borel_compact comf by (auto simp: borel_compact)
  qed auto
  show f ‘ K ∈ sets lebesgue
    by (simp add: ‹K ⊆ S› ‹negligible K› negim negligible_imp_sets)
 qed
 then show ?thesis
  by (simp add: Teq image_Un image_Union)
qed

lemma differentiable_image_in_sets_lebesgue:
 fixes f :: 'm::euclidean_space ⇒ 'n::euclidean_space
 assumes S: S ∈ sets lebesgue and dim: DIM('m) ≤ DIM('n) and f: f differentiable_on S
 shows f‘S ∈ sets lebesgue
proof (rule sets_lebesgue_continuous_image [OF S])
 show continuous_on S f
  by (meson differentiable_imp_continuous_on f)
 show ⋀T. ⟦negligible T; T ⊆ S⟧ ⟹ negligible (f ‘ T)
  using differentiable_on_subset f
  by (auto simp: intro!: negligible_differentiable_image_negligible [OF dim])
qed auto

lemma sets_lebesgue_on_continuous_image:
  assumes S: S ∈ sets lebesgue and X: X ∈ sets (lebesgue_on S) and contf: continuous_on S f
   and negim: ⋀T. ⟦negligible T; T ⊆ S⟧ ⟹ negligible(f ‘ T)
 shows f ‘ X ∈ sets (lebesgue_on (f ‘ S))
proof −
 have X ⊆ S
  by (metis S X sets.Int_space_eq2 sets_restrict_space_iff)
 moreover have f ‘ S ∈ sets lebesgue
  using S contf negim sets_lebesgue_continuous_image by blast
 moreover have f ‘ X ∈ sets lebesgue
  by (metis S X contf negim sets_lebesgue_continuous_image sets_restrict_space_iff space_restrict_space space_restrict_space2)
 ultimately show ?thesis
  by (auto simp: sets_restrict_space_iff)
qed

lemma differentiable_image_in_sets_lebesgue_on:
 fixes f :: 'm::euclidean_space ⇒ 'n::euclidean_space
  assumes S: S ∈ sets lebesgue and X: X ∈ sets (lebesgue_on S) and dim: DIM('m) ≤ DIM('n)
      and f: f differentiable_on S
    shows f ‘ X ∈ sets (lebesgue_on (f‘S))
proof (rule sets_lebesgue_on_continuous_image [OF S X])
 show continuous_on S f
  by (meson differentiable_imp_continuous_on f)
```

show $\bigwedge T.\ [\![negligible\ T;\ T \subseteq S]\!] \Longrightarrow negligible\ (f\ `\ T)$
using *differentiable_on_subset f*
by (*auto simp*: *intro*!: *negligible_differentiable_image_negligible* [*OF dim*])
qed

6.16.25 Affine lemmas

lemma *borel_measurable_affine*:
fixes $f :: 'a :: euclidean\_space \Rightarrow 'b :: euclidean\_space$
assumes *f*: $f \in borel\_measurable\ lebesgue$ **and** $c \neq 0$
shows $(\lambda x.\ f(t + c *_R x)) \in borel\_measurable\ lebesgue$
proof −
{ fix *a b*
have $\{x.\ f\ x \in cbox\ a\ b\} \in sets\ lebesgue$
using *f cbox_borel lebesgue_measurable_vimage_borel* **by** *blast*
then have $(\lambda x.\ (x - t) /_R c)\ `\ \{x.\ f\ x \in cbox\ a\ b\} \in sets\ lebesgue$
proof (*rule differentiable_image_in_sets_lebesgue*)
show $(\lambda x.\ (x - t) /_R c)\ differentiable\_on\ \{x.\ f\ x \in cbox\ a\ b\}$
unfolding *differentiable_on_def differentiable_def*
by (*rule* ‹$c \neq 0$› *derivative_eq_intros strip exI* | *simp*)+
qed *auto*
moreover
have $\{x.\ f(t + c *_R x) \in cbox\ a\ b\} = (\lambda x.\ (x-t) /_R c)\ `\ \{x.\ f\ x \in cbox\ a\ b\}$
using ‹$c \neq 0$› **by** (*auto simp*: *image_def*)
ultimately have $\{x.\ f(t + c *_R x) \in cbox\ a\ b\} \in sets\ lebesgue$
by (*auto simp*: *borel_measurable_vimage_closed_interval*) **}**
then show *?thesis*
by (*subst lebesgue_on_UNIV_eq* [*symmetric*]; *auto simp*: *borel_measurable_vimage_closed_interval*)
qed

lemma *lebesgue_integrable_real_affine*:
fixes $f :: real \Rightarrow 'a :: euclidean\_space$
assumes *f*: *integrable lebesgue f* **and** $c \neq 0$
shows *integrable lebesgue* $(\lambda x.\ f(t + c * x))$
proof −
have $(\lambda x.\ norm\ (f\ x)) \in borel\_measurable\ lebesgue$
by (*simp add*: *borel_measurable_integrable f*)
then show *?thesis*
using *assms borel_measurable_affine* [*of f c*]
unfolding *integrable_iff_bounded*
by (*subst* (*asm*) *nn_integral_real_affine_lebesgue*[**where** $c=c$ **and** $t=t$]) (*auto simp*: *ennreal_mult_less_top*)
qed

lemma *lebesgue_integrable_real_affine_iff*:
fixes $f :: real \Rightarrow 'a :: euclidean\_space$
shows $c \neq 0 \Longrightarrow integrable\ lebesgue\ (\lambda x.\ f(t + c * x)) \longleftrightarrow integrable\ lebesgue\ f$
using *lebesgue_integrable_real_affine*[*of f c t*]
lebesgue_integrable_real_affine[*of* $\lambda x.\ f(t + c * x)\ 1/c\ -t/c$]

by (*auto simp*: *field_simps*)

lemma *lebesgue_integral_real_affine*:
fixes $f :: real \Rightarrow 'a :: euclidean\_space$ **and** $c :: real$
assumes c: $c \neq 0$ **shows** $(\int x.\ f\ x\ \partial\ lebesgue) = |c| *_R (\int x.\ f(t + c * x)\ \partial lebesgue)$
proof *cases*
have $(\lambda x.\ t + c * x) \in lebesgue \rightarrow_M lebesgue$
using *lebesgue_affine_measurable*[**where** $c= \lambda x::real.\ c$] ‹$c \neq 0$› **by** *simp*
moreover
assume *integrable lebesgue f*
ultimately show *?thesis*
by (*subst lebesgue_real_affine*[*OF c, of t*]) (*auto simp*: *integral_density integral_distr*)
next
assume ¬ *integrable lebesgue f* **with** *c* **show** *?thesis*
by (*simp add*: *lebesgue_integrable_real_affine_iff not_integrable_integral_eq*)
qed

lemma *has_bochner_integral_lebesgue_real_affine_iff*:
fixes $i :: 'a :: euclidean\_space$
shows $c \neq 0 \Longrightarrow$
has_bochner_integral lebesgue f i ⟷
has_bochner_integral lebesgue $(\lambda x.\ f(t + c * x))\ (i /_R |c|)$
unfolding *has_bochner_integral_iff lebesgue_integrable_real_affine_iff*
by (*simp_all add*: *lebesgue_integral_real_affine*[*symmetric*] *divideR_right cong*: *conj_cong*)

lemma *has_bochner_integral_reflect_real_lemma*[*intro*]:
fixes $f :: real \Rightarrow 'a::euclidean\_space$
assumes *has_bochner_integral* (*lebesgue_on* $\{a..b\}$) *f i*
shows *has_bochner_integral* (*lebesgue_on* $\{-b..-a\}$) $(\lambda x.\ f(-x))$ *i*
proof −
have *eq*: *indicat_real* $\{a..b\}\ (-\ x) *_R f(-\ x) =$ *indicat_real* $\{-\ b..-\ a\}\ x *_R f(-\ x)$ **for** *x*
by (*auto simp*: *indicator_def*)
have *i*: *has_bochner_integral lebesgue* $(\lambda x.$ *indicator* $\{a..b\}\ x *_R f\ x)$ *i*
using *assms* **by** (*auto simp*: *has_bochner_integral_restrict_space*)
then have *has_bochner_integral lebesgue* $(\lambda x.$ *indicator* $\{-b..-a\}\ x *_R f(-x))$ *i*
using *has_bochner_integral_lebesgue_real_affine_iff* [*of* -1 $(\lambda x.$ *indicator* $\{a..b\}\ x *_R f\ x)$ *i* 0]
by (*auto simp*: *eq*)
then show *?thesis*
by (*auto simp*: *has_bochner_integral_restrict_space*)
qed

lemma *has_bochner_integral_reflect_real*[*simp*]:
fixes $f :: real \Rightarrow 'a::euclidean\_space$

shows *has_bochner_integral* (*lebesgue_on* {−*b*..−*a*}) (λ*x*. *f*(−*x*)) *i* ⟷ *has_bochner_integral* (*lebesgue_on* {*a*..*b*}) *f i*
by (*auto simp*: *dest*: *has_bochner_integral_reflect_real_lemma*)

lemma *integrable_reflect_real*[*simp*]:
fixes *f* :: *real* ⇒ *'a*::*euclidean_space*
shows *integrable* (*lebesgue_on* {−*b*..−*a*}) (λ*x*. *f*(−*x*)) ⟷ *integrable* (*lebesgue_on* {*a*..*b*}) *f*
by (*metis has_bochner_integral_iff has_bochner_integral_reflect_real*)

lemma *integral_reflect_real*[*simp*]:
fixes *f* :: *real* ⇒ *'a*::*euclidean_space*
shows $\mathit{integral}^L$ (*lebesgue_on* {−*b* .. −*a*}) (λ*x*. *f*(−*x*)) = $\mathit{integral}^L$ (*lebesgue_on* {*a*..*b*::*real*}) *f*
using *has_bochner_integral_reflect_real* [*of b a f*]
by (*metis has_bochner_integral_iff not_integrable_integral_eq*)

6.16.26 More results on integrability

lemma *integrable_on_all_intervals_UNIV*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*banach*
assumes *intf*: ⋀*a b*. *f integrable_on cbox a b*
and *normf*: ⋀*x*. *norm*(*f x*) ≤ *g x* **and** *g*: *g integrable_on UNIV*
shows *f integrable_on UNIV*
proof −
have *intg*: (∀ *a b*. *g integrable_on cbox a b*)
and *gle_e*: ∀ *e*>*0*. ∃ *B*>*0*. ∀ *a b c d*.
ball 0 B ⊆ *cbox a b* ∧ *cbox a b* ⊆ *cbox c d* ⟶
|*integral* (*cbox a b*) *g* − *integral* (*cbox c d*) *g*|
< *e*
using *g*
by (*auto simp*: *integrable_alt_subset* [*of _ UNIV*] *intf*)
have *le*: *norm* (*integral* (*cbox a b*) *f* − *integral* (*cbox c d*) *f*) ≤ |*integral* (*cbox a b*) *g* − *integral* (*cbox c d*) *g*|
if *cbox a b* ⊆ *cbox c d* **for** *a b c d*
proof −
have *norm* (*integral* (*cbox a b*) *f* − *integral* (*cbox c d*) *f*) = *norm* (*integral* (*cbox c d* − *cbox a b*) *f*)
using *intf that* **by** (*simp add*: *norm_minus_commute integral_setdiff*)
also have ... ≤ *integral* (*cbox c d* − *cbox a b*) *g*
proof (*rule integral_norm_bound_integral* [*OF _ _ normf*])
show *f integrable_on cbox c d* − *cbox a b g integrable_on cbox c d* − *cbox a b*
by (*meson integrable_integral integrable_setdiff intf intg negligible_setdiff that*)+
qed
also have ... = *integral* (*cbox c d*) *g* − *integral* (*cbox a b*) *g*
using *intg that* **by** (*simp add*: *integral_setdiff*)
also have ... ≤ |*integral* (*cbox a b*) *g* − *integral* (*cbox c d*) *g*|
by *simp*

```
    finally show ?thesis .
  qed
  show ?thesis
    using gle_e
    apply (simp add: integrable_alt_subset [of _ UNIV] intf)
    apply (erule imp_forward all_forward ex_forward asm_rl)+
    by (meson not_less order_trans le)
qed

lemma integrable_on_all_intervals_integrable_bound:
  fixes f :: 'a::euclidean_space ⇒ 'b::banach
  assumes intf: ⋀a b. (λx. if x ∈ S then f x else 0) integrable_on cbox a b
    and normf: ⋀x. x ∈ S ⟹ norm(f x) ≤ g x and g: g integrable_on S
  shows f integrable_on S
  using integrable_on_all_intervals_UNIV [OF intf, of (λx. if x ∈ S then g x
else 0)]
  by (simp add: g integrable_restrict_UNIV normf)

lemma measurable_bounded_lemma:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes f: f ∈ borel_measurable lebesgue and g: g integrable_on cbox a b
    and normf: ⋀x. x ∈ cbox a b ⟹ norm(f x) ≤ g x
  shows f integrable_on cbox a b
proof −
  have g absolutely_integrable_on cbox a b
   by (metis (full_types) add_increasing g le_add_same_cancel1 nonnegative_absolutely_integrable_1
norm_ge_zero normf)
  then have integrable (lebesgue_on (cbox a b)) g
    by (simp add: integrable_restrict_space set_integrable_def)
  then have integrable (lebesgue_on (cbox a b)) f
  proof (rule Bochner_Integration.integrable_bound)
    show AE x in lebesgue_on (cbox a b). norm (f x) ≤ norm (g x)
      by (rule AE_I2) (auto intro: normf order_trans)
  qed (simp add: f measurable_restrict_space1)
  then show ?thesis
    by (simp add: integrable_on_lebesgue_on)
qed

proposition measurable_bounded_by_integrable_imp_integrable:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes f: f ∈ borel_measurable (lebesgue_on S) and g: g integrable_on S
    and normf: ⋀x. x ∈ S ⟹ norm(f x) ≤ g x and S: S ∈ sets lebesgue
  shows f integrable_on S
proof (rule integrable_on_all_intervals_integrable_bound [OF _ normf g])
  show (λx. if x ∈ S then f x else 0) integrable_on cbox a b for a b
  proof (rule measurable_bounded_lemma)
    show (λx. if x ∈ S then f x else 0) ∈ borel_measurable lebesgue
      by (simp add: S borel_measurable_if f)
    show (λx. if x ∈ S then g x else 0) integrable_on cbox a b
```

by (*simp add*: *g integrable_altD*(*1*))
show *norm* (*if x* ∈ *S then f x else 0*) ≤ (*if x* ∈ *S then g x else 0*) **for** *x*
using *normf* **by** *simp*
qed
qed

lemma *measurable_bounded_by_integrable_imp_lebesgue_integrable*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *f*: *f* ∈ *borel_measurable* (*lebesgue_on S*) **and** *g*: *integrable* (*lebesgue_on S*) *g*
and *normf*: ⋀*x*. *x* ∈ *S* ⟹ *norm*(*f x*) ≤ *g x* **and** *S*: *S* ∈ *sets lebesgue*
shows *integrable* (*lebesgue_on S*) *f*
proof −
have *f absolutely_integrable_on S*
by (*metis* (*no_types*) *S absolutely_integrable_integrable_bound f g integrable_on_lebesgue_on measurable_bounded_by_integrable_imp_integrable normf*)
then show *?thesis*
by (*simp add*: *S integrable_restrict_space set_integrable_def*)
qed

lemma *measurable_bounded_by_integrable_imp_integrable_real*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *real*
assumes *f* ∈ *borel_measurable* (*lebesgue_on S*) *g integrable_on S* ⋀*x*. *x* ∈ *S* ⟹ *abs*(*f x*) ≤ *g x S* ∈ *sets lebesgue*
shows *f integrable_on S*
using *measurable_bounded_by_integrable_imp_integrable* [*of f S g*] *assms* **by** *simp*

6.16.27 Relation between Borel measurability and integrability.

lemma *integrable_imp_measurable_weak*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *S* ∈ *sets lebesgue f integrable_on S*
shows *f* ∈ *borel_measurable* (*lebesgue_on S*)
by (*metis* (*mono_tags*, *lifting*) *assms has_integral_implies_lebesgue_measurable borel_measurable_restrict_space_iff integrable_on_def sets.Int_space_eq2*)

lemma *integrable_imp_measurable*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *f integrable_on S*
shows *f* ∈ *borel_measurable* (*lebesgue_on S*)
proof −
have (*UNIV*::*'a set*) ∈ *sets lborel*
by *simp*
then show *?thesis*
by (*metis* (*mono_tags*, *lifting*) *assms borel_measurable_if_D integrable_imp_measurable_weak integrable_restrict_UNIV lebesgue_on_UNIV_eq sets_lebesgue_on_refl*)
qed

lemma *integrable_iff_integrable_on*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes $S \in sets\ lebesgue\ (\int^+ x.\ ennreal\ (norm\ (f\ x))\ \partial lebesgue\_on\ S) < \infty$
shows $integrable\ (lebesgue\_on\ S)\ f \longleftrightarrow f\ integrable\_on\ S$
using *assms integrable_iff_bounded integrable_imp_measurable integrable_on_lebesgue_on*
by *blast*

lemma *absolutely_integrable_measurable*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes $S \in sets\ lebesgue$
shows $f\ absolutely\_integrable\_on\ S \longleftrightarrow f \in borel\_measurable\ (lebesgue\_on\ S) \wedge integrable\ (lebesgue\_on\ S)\ (norm \circ f)$
(**is** *?lhs* = *?rhs*)
proof
assume *L*: *?lhs*
then have $f \in borel\_measurable\ (lebesgue\_on\ S)$
by (*simp add*: *absolutely_integrable_on_def integrable_imp_measurable*)
then show *?rhs*
using *assms set_integrable_norm* [*of lebesgue S f*] *L*
by (*simp add*: *integrable_restrict_space set_integrable_def*)
next
assume *?rhs* **then show** *?lhs*
using *assms integrable_on_lebesgue_on*
by (*metis absolutely_integrable_integrable_bound comp_def eq_iff measurable_bounded_by_integrable_imp_integrable*)
qed

lemma *absolutely_integrable_measurable_real*:
fixes $f :: 'a::euclidean\_space \Rightarrow real$
assumes $S \in sets\ lebesgue$
shows $f\ absolutely\_integrable\_on\ S \longleftrightarrow f \in borel\_measurable\ (lebesgue\_on\ S) \wedge integrable\ (lebesgue\_on\ S)\ (\lambda x.\ |f\ x|)$
by (*simp add*: *absolutely_integrable_measurable assms o_def*)

lemma *absolutely_integrable_measurable_real'*:
fixes $f :: 'a::euclidean\_space \Rightarrow real$
assumes $S \in sets\ lebesgue$
shows $f\ absolutely\_integrable\_on\ S \longleftrightarrow f \in borel\_measurable\ (lebesgue\_on\ S) \wedge (\lambda x.\ |f\ x|)\ integrable\_on\ S$
by (*metis abs_absolutely_integrableI_1 absolutely_integrable_measurable_real assms measurable_bounded_by_integrable_imp_integrable order_refl real_norm_def set_integrable_abs set_lebesgue_integral_eq_integral*(*1*))

lemma *absolutely_integrable_imp_borel_measurable*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes $f\ absolutely\_integrable\_on\ S\ S \in sets\ lebesgue$

shows $f \in$ *borel_measurable* (*lebesgue_on* S)
using *absolutely_integrable_measurable assms* **by** *blast*

lemma *measurable_bounded_by_integrable_imp_absolutely_integrable*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes $f \in$ *borel_measurable* (*lebesgue_on* S) $S \in$ *sets lebesgue*
and g *integrable_on* S **and** $\bigwedge x.\ x \in S \Longrightarrow norm(f\ x) \leq (g\ x)$
shows f *absolutely_integrable_on* S
using *assms absolutely_integrable_integrable_bound measurable_bounded_by_integrable_imp_integrable*
by *blast*

proposition *negligible_differentiable_vimage*:
fixes $f :: 'a \Rightarrow 'a::euclidean\_space$
assumes *negligible* T
and f': $\bigwedge x.\ x \in S \Longrightarrow inj(f'\ x)$
and *derf*: $\bigwedge x.\ x \in S \Longrightarrow (f$ *has_derivative* $f'\ x)$ (*at* x *within* S)
shows *negligible* $\{x \in S.\ f\ x \in T\}$
proof −
define U **where**
$U \equiv \lambda n::nat.\ \{x \in S.\ \forall\, y.\ y \in S \wedge norm(y - x) < 1/n$
$\longrightarrow norm(y - x) \leq n * norm(f\ y - f\ x)\}$
have *negligible* $\{x \in U\ n.\ f\ x \in T\}$ **if** $n > 0$ **for** n
proof (*subst locally_negligible_alt*, *clarify*)
fix a
assume a: $a \in U\ n$ **and** *fa*: $f\ a \in T$
define V **where** $V \equiv \{x.\ x \in U\ n \wedge f\ x \in T\} \cap ball\ a\ (1\ /\ n\ /\ 2)$
show $\exists\, V.$ *openin* (*top_of_set* $\{x \in U\ n.\ f\ x \in T\}$) $V \wedge a \in V \wedge$ *negligible* V
proof (*intro exI conjI*)
have *noxy*: $norm(x - y) \leq n * norm(f\ x - f\ y)$ **if** $x \in V\ y \in V$ **for** $x\ y$
using *that* **unfolding** *U_def V_def mem_Collect_eq Int_iff mem_ball dist_norm*
by (*meson norm_triangle_half_r*)
then have *inj_on f V*
by (*force simp*: *inj_on_def*)
then obtain g **where** g: $\bigwedge x.\ x \in V \Longrightarrow g(f\ x) = x$
by (*metis inv_into_f_f*)
have $\exists\, T'\ B.$ *open* $T' \wedge f\ x \in T' \wedge$
$(\forall\, y \in f\ `\ V \cap T \cap T'.\ norm\ (g\ y - g\ (f\ x)) \leq B * norm\ (y - f\ x))$
if $f\ x \in T\ x \in V$ **for** x
using *that noxy*
by (*rule_tac* $x = ball\ (f\ x)\ 1$ **in** *exI*) (*force simp*: g)
then have *negligible* $(g\ `\ (f\ `\ V \cap T))$
by (*force simp*: ‹*negligible* T› *negligible_Int intro*!: *negligible_locally_Lipschitz_image*)
moreover have $V \subseteq g\ `\ (f\ `\ V \cap T)$
by (*force simp*: g *image_iff V_def*)
ultimately show *negligible* V
by (*rule negligible_subset*)
qed (*use a fa V_def that* **in** *auto*)
qed

with *negligible_countable_Union* **have** *negligible* ($\bigcup n \in \{0<..\}.\ \{x.\ x \in U\ n \wedge f\ x \in T\}$)
by *auto*
moreover have $\{x \in S.\ f\ x \in T\} \subseteq (\bigcup n \in \{0<..\}.\ \{x.\ x \in U\ n \wedge f\ x \in T\})$
proof *clarsimp*
fix x
assume $x \in S$ **and** $f\ x \in T$
then obtain *inj*: $inj(f'\ x)$ **and** *der*: $(f\ has\_derivative\ f'\ x)\ (at\ x\ within\ S)$
using *assms* **by** *metis*
moreover have $linear(f'\ x)$
and *eps*: $\bigwedge\varepsilon.\ \varepsilon > 0 \Longrightarrow \exists\delta{>}0.\ \forall y{\in}S.\ norm\ (y - x) < \delta \longrightarrow$
$norm\ (f\ y - f\ x - f'\ x\ (y - x)) \leq \varepsilon * norm\ (y - x)$
using *der* **by** (*auto simp*: *has_derivative_within_alt linear_linear*)
ultimately obtain g **where** *linear* g **and** g: $g \circ f'\ x = id$
using *linear_injective_left_inverse* **by** *metis*
then obtain B **where** $B > 0$ **and** B: $\bigwedge z.\ B * norm\ z \leq norm(f'\ x\ z)$
using *linear_invertible_bounded_below_pos* ‹*linear* $(f'\ x)$› **by** *blast*
then obtain i **where** $i \neq 0$ **and** i: $1\ /\ real\ i < B$
by (*metis* (*full_types*) *inverse_eq_divide real_arch_invD*)
then obtain δ **where** $\delta > 0$
and δ: $\bigwedge y.\ [\![y{\in}S;\ norm\ (y - x) < \delta]\!] \Longrightarrow$
$norm\ (f\ y - f\ x - f'\ x\ (y - x)) \leq (B - 1\ /\ real\ i) * norm\ (y - x)$
using *eps* [*of* $B - 1/i$] **by** *auto*
then obtain j **where** $j \neq 0$ **and** j: $inverse\ (real\ j) < \delta$
using *real_arch_inverse* **by** *blast*
have $norm\ (y - x)/(max\ i\ j) \leq norm\ (f\ y - f\ x)$
if $y \in S$ **and** *less*: $norm\ (y - x) < 1\ /\ (max\ i\ j)$ **for** y
proof −
have $1\ /\ real\ (max\ i\ j) < \delta$
using j ‹$j \neq 0$› ‹$0 < \delta$›
by (*auto simp*: *field_split_simps max_mult_distrib_left of_nat_max*)
then have $norm\ (y - x) < \delta$
using *less* **by** *linarith*
with δ ‹$y \in S$› **have** *le*: $norm\ (f\ y - f\ x - f'\ x\ (y - x)) \leq B * norm\ (y - x)$ − $norm\ (y - x)/i$
by (*auto simp*: *algebra_simps*)
have $norm\ (y - x)\ /\ real\ (max\ i\ j) \leq norm\ (y - x)\ /\ real\ i$
using ‹$i \neq 0$› ‹$j \neq 0$› **by** (*simp add*: *field_split_simps max_mult_distrib_left of_nat_max less_max_iff_disj*)
also have $... \leq norm\ (f\ y - f\ x)$
using B [*of* $y{-}x$] *le norm_triangle_ineq3* [*of* $f\ y - f\ x\ f'\ x\ (y - x)$]
by *linarith*
finally show *?thesis* **.**
qed
with ‹$x \in S$› ‹$i \neq 0$› ‹$j \neq 0$› **show** $\exists n{\in}\{0<..\}.\ x \in U\ n$
by (*rule_tac* $x{=}max\ i\ j$ **in** *bexI*) (*auto simp*: *field_simps U_def less_max_iff_disj*)
qed
ultimately show *?thesis*
by (*rule negligible_subset*)

qed

lemma *absolutely_integrable_Un*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes S: f *absolutely_integrable_on* S **and** T: f *absolutely_integrable_on* T
shows f *absolutely_integrable_on* $(S \cup T)$
proof −
have [*simp*]: $\{x.\ (\textit{if } x \in A \textit{ then } f\ x \textit{ else } 0) \neq 0\} = \{x \in A.\ f\ x \neq 0\}$ **for** A
by *auto*
let $?ST = \{x \in S.\ f\ x \neq 0\} \cap \{x \in T.\ f\ x \neq 0\}$
have $?ST \in sets\ lebesgue$
proof (*rule Sigma_Algebra.sets.Int*)
have f *integrable_on* S
using S *absolutely_integrable_on_def* **by** *blast*
then have $(\lambda x.\ \textit{if } x \in S \textit{ then } f\ x \textit{ else } 0)$ *integrable_on UNIV*
by (*simp add*: *integrable_restrict_UNIV*)
then have *borel*: $(\lambda x.\ \textit{if } x \in S \textit{ then } f\ x \textit{ else } 0) \in borel\_measurable\ (lebesgue\_on\ UNIV)$
using *integrable_imp_measurable lebesgue_on_UNIV_eq* **by** *blast*
then show $\{x \in S.\ f\ x \neq 0\} \in sets\ lebesgue$
unfolding *borel_measurable_vimage_open*
by (*rule allE* [**where** $x = -\{0\}$]) *auto*
next
have f *integrable_on* T
using T *absolutely_integrable_on_def* **by** *blast*
then have $(\lambda x.\ \textit{if } x \in T \textit{ then } f\ x \textit{ else } 0)$ *integrable_on UNIV*
by (*simp add*: *integrable_restrict_UNIV*)
then have *borel*: $(\lambda x.\ \textit{if } x \in T \textit{ then } f\ x \textit{ else } 0) \in borel\_measurable\ (lebesgue\_on\ UNIV)$
using *integrable_imp_measurable lebesgue_on_UNIV_eq* **by** *blast*
then show $\{x \in T.\ f\ x \neq 0\} \in sets\ lebesgue$
unfolding *borel_measurable_vimage_open*
by (*rule allE* [**where** $x = -\{0\}$]) *auto*
qed
then have f *absolutely_integrable_on* $?ST$
by (*rule set_integrable_subset* [*OF S*]) *auto*
then have *Int*: $(\lambda x.\ \textit{if } x \in ?ST \textit{ then } f\ x \textit{ else } 0)$ *absolutely_integrable_on UNIV*
using *absolutely_integrable_restrict_UNIV* **by** *blast*
have $(\lambda x.\ \textit{if } x \in S \textit{ then } f\ x \textit{ else } 0)$ *absolutely_integrable_on UNIV*
$(\lambda x.\ \textit{if } x \in T \textit{ then } f\ x \textit{ else } 0)$ *absolutely_integrable_on UNIV*
using S T *absolutely_integrable_restrict_UNIV* **by** *blast+*
then have $(\lambda x.\ (\textit{if } x \in S \textit{ then } f\ x \textit{ else } 0) + (\textit{if } x \in T \textit{ then } f\ x \textit{ else } 0))$ *absolutely_integrable_on UNIV*
by (*rule set_integral_add*)
then have $(\lambda x.\ ((\textit{if } x \in S \textit{ then } f\ x \textit{ else } 0) + (\textit{if } x \in T \textit{ then } f\ x \textit{ else } 0)) - (\textit{if } x \in ?ST \textit{ then } f\ x \textit{ else } 0))$ *absolutely_integrable_on UNIV*
using *Int* **by** (*rule set_integral_diff*)
then have $(\lambda x.\ \textit{if } x \in S \cup T \textit{ then } f\ x \textit{ else } 0)$ *absolutely_integrable_on UNIV*
by (*rule absolutely_integrable_spike*) (*auto intro*: *empty_imp_negligible*)

then show *?thesis*
unfolding *absolutely_integrable_restrict_UNIV* **.**
qed

lemma *absolutely_integrable_on_combine*:
fixes f :: *real* ⇒ *'a*::*euclidean_space*
assumes f *absolutely_integrable_on* {a..c}
and f *absolutely_integrable_on* {c..b}
and $a \leq c$
and $c \leq b$
shows f *absolutely_integrable_on* {a..b}
by (*metis absolutely_integrable_Un assms ivl_disj_un_two_touch*(*4*))

lemma *uniform_limit_set_lebesgue_integral_at_top*:
fixes f :: *'a* ⇒ *real* ⇒ *'b*::{*banach*, *second_countable_topology*}
and g :: *real* ⇒ *real*
assumes *bound*: ⋀x y. $x \in A \Longrightarrow y \geq a \Longrightarrow$ *norm* (f x y) ≤ g y
assumes *integrable*: *set_integrable M* {a..} g
assumes *measurable*: ⋀x. $x \in A \Longrightarrow$ *set_borel_measurable M* {a..} (f x)
assumes *sets borel* ⊆ *sets M*
shows *uniform_limit A* (λb x. *LINT* y:{a..b}|*M*. f x y) (λx. *LINT* y:{a..}|*M*. f x y) *at_top*
proof (*cases* $A = \{\}$)
case *False*
then obtain x **where** x: $x \in A$ **by** *auto*
have *g_nonneg*: g $y \geq 0$ **if** $y \geq a$ **for** y
proof −
have $0 \leq$ *norm* (f x y) **by** *simp*
also have ... ≤ g y **using** *bound*[*OF x that*] **by** *simp*
finally show *?thesis* **.**
qed

have *integrable'*: *set_integrable M* {a..} (λy. f x y) **if** $x \in A$ **for** x
unfolding *set_integrable_def*
proof (*rule Bochner_Integration.integrable_bound*)
show *integrable M* (λx. *indicator* {a..} x * g x)
using *integrable* **by** (*simp add*: *set_integrable_def*)
show (λy. *indicat_real* {a..} y *$_R$ f x y) ∈ *borel_measurable M* **using** *measurable*[*OF that*]
by (*simp add*: *set_borel_measurable_def*)
show *AE y in M*. *norm* (*indicat_real* {a..} y *$_R$ f x y) ≤ *norm* (*indicat_real* {a..} y * g y)
using *bound*[*OF that*] **by** (*intro AE_I2*) (*auto simp*: *indicator_def g_nonneg*)
qed

show *?thesis*
proof (*rule uniform_limitI*)
fix e :: *real* **assume** e: $e > 0$
have *sets* [*intro*]: $A \in$ *sets M* **if** $A \in$ *sets borel* **for** A

using *that assms* **by** *blast*

have $((\lambda b.\ LINT\ y{:}\{a..b\}|M.\ g\ y) \longrightarrow (LINT\ y{:}\{a..\}|M.\ g\ y))$ *at_top*
by (*intro tendsto_set_lebesgue_integral_at_top assms sets*) *auto*
with *e* **obtain** *b0* :: *real* **where** *b0*: $\forall\, b{\geq}b0.\ |(LINT\ y{:}\{a..\}|M.\ g\ y) - (LINT\ y{:}\{a..b\}|M.\ g\ y)| < e$
by (*auto simp*: *tendsto_iff eventually_at_top_linorder dist_real_def abs_minus_commute*)
define *b* **where** $b = max\ a\ b0$
have $a \leq b$ **by** (*simp add*: *b_def*)
from *b0* **have** $|(LINT\ y{:}\{a..\}|M.\ g\ y) - (LINT\ y{:}\{a..b\}|M.\ g\ y)| < e$
by (*auto simp*: *b_def*)
also have $\{a..\} = \{a..b\} \cup \{b{<}..\}$ **by** (*auto simp*: *b_def*)
also have $|(LINT\ y{:}\ldots|M.\ g\ y) - (LINT\ y{:}\{a..b\}|M.\ g\ y)| = |(LINT\ y{:}\{b{<}..\}|M.\ g\ y)|$
using ‹$a \leq b$› **by** (*subst set_integral_Un*) (*auto intro*!: *set_integrable_subset*[*OF integrable*])
also have $(LINT\ y{:}\{b{<}..\}|M.\ g\ y) \geq 0$
using *g_nonneg* ‹$a \leq b$› **unfolding** *set_lebesgue_integral_def*
by (*intro Bochner_Integration.integral_nonneg*) (*auto simp*: *indicator_def*)
hence $|(LINT\ y{:}\{b{<}..\}|M.\ g\ y)| = (LINT\ y{:}\{b{<}..\}|M.\ g\ y)$ **by** *simp*
finally have *less*: $(LINT\ y{:}\{b{<}..\}|M.\ g\ y) < e$ **.**

have *eventually* $(\lambda b.\ b \geq b0)$ *at_top* **by** (*rule eventually_ge_at_top*)
moreover have *eventually* $(\lambda b.\ b \geq a)$ *at_top* **by** (*rule eventually_ge_at_top*)
ultimately show *eventually* $(\lambda b.\ \forall x{\in}A.$
$dist\ (LINT\ y{:}\{a..b\}|M.\ f\ x\ y)\ (LINT\ y{:}\{a..\}|M.\ f\ x\ y) < e)$ *at_top*
proof *eventually_elim*
case (*elim b*)
show *?case*
proof
fix *x* **assume** *x*: $x \in A$
have $dist\ (LINT\ y{:}\{a..b\}|M.\ f\ x\ y)\ (LINT\ y{:}\{a..\}|M.\ f\ x\ y) =$
$norm\ ((LINT\ y{:}\{a..\}|M.\ f\ x\ y) - (LINT\ y{:}\{a..b\}|M.\ f\ x\ y))$
by (*simp add*: *dist_norm norm_minus_commute*)
also have $\{a..\} = \{a..b\} \cup \{b{<}..\}$ **using** *elim* **by** *auto*
also have $(LINT\ y{:}\ldots|M.\ f\ x\ y) - (LINT\ y{:}\{a..b\}|M.\ f\ x\ y) = (LINT\ y{:}\{b{<}..\}|M.\ f\ x\ y)$
using *elim x*
by (*subst set_integral_Un*) (*auto intro*!: *set_integrable_subset*[*OF integrable'*])
also have $norm\ \ldots \leq (LINT\ y{:}\{b{<}..\}|M.\ norm\ (f\ x\ y))$ **using** *elim x*
by (*intro set_integral_norm_bound set_integrable_subset*[*OF integrable'*]) *auto*
also have $\ldots \leq (LINT\ y{:}\{b{<}..\}|M.\ g\ y)$ **using** *elim x bound g_nonneg*
by (*intro set_integral_mono set_integrable_norm set_integrable_subset*[*OF integrable'*]
set_integrable_subset[*OF integrable*]) *auto*
also have $(LINT\ y{:}\{b{<}..\}|M.\ g\ y) \geq 0$

```
        using g_nonneg ‹a ≤ b› unfolding set_lebesgue_integral_def
       by (intro Bochner_Integration.integral_nonneg) (auto simp: indicator_def)
       hence (LINT y:{b<..}|M. g y) = |(LINT y:{b<..}|M. g y)| by simp
       also have ... = |(LINT y:{a..b} ∪ {b<..}|M. g y) - (LINT y:{a..b}|M. g
y)|
       using elim by (subst set_integral_Un) (auto intro!: set_integrable_subset[OF
integrable])
       also have {a..b} ∪ {b<..} = {a..} using elim by auto
       also have |(LINT y:{a..}|M. g y) - (LINT y:{a..b}|M. g y)| < e
         using b0 elim by blast
       finally show dist (LINT y:{a..b}|M. f x y) (LINT y:{a..}|M. f x y) < e .
     qed
   qed
 qed
qed auto
```

Differentiability of inverse function (most basic form)

```
proposition has_derivative_inverse_within:
 fixes f :: 'a::real_normed_vector ⇒ 'b::euclidean_space
 assumes der_f: (f has_derivative f') (at a within S)
   and cont_g: continuous (at (f a) within f ‘ S) g
   and a ∈ S linear g' and id: g' ∘ f' = id
   and gf: ⋀x. x ∈ S ⟹ g(f x) = x
 shows (g has_derivative g') (at (f a) within f ‘ S)
proof -
 have [simp]: g' (f' x) = x for x
   by (simp add: local.id pointfree_idE)
 have bounded_linear f'
   and f': ⋀e. e>0 ⟹ ∃ d>0. ∀ y∈S. norm (y - a) < d ⟶
                    norm (f y - f a - f' (y - a)) ≤ e * norm (y - a)
   using der_f by (auto simp: has_derivative_within_alt)
 obtain C where C > 0 and C: ⋀x. norm (g' x) ≤ C * norm x
   using linear_bounded_pos [OF ‹linear g'›] by metis
 obtain B k where B > 0 k > 0
   and Bk: ⋀x. ⟦x ∈ S; norm(f x - f a) < k⟧ ⟹ norm(x - a) ≤ B * norm(f x
- f a)
 proof -
   obtain B where B > 0 and B: ⋀x. B * norm x ≤ norm (f' x)
   using linear_inj_bounded_below_pos [of f'] ‹linear g'› id der_f has_derivative_linear
       linear_invertible_bounded_below_pos by blast
   then obtain d where d>0
     and d: ⋀y. ⟦y ∈ S; norm (y - a) < d⟧ ⟹
                  norm (f y - f a - f' (y - a)) ≤ B / 2 * norm (y - a)
     using f' [of B/2] by auto
   then obtain e where e > 0
     and e: ⋀x. ⟦x ∈ S; norm (f x - f a) < e⟧ ⟹ norm (g (f x) - g (f a)) < d
     using cont_g by (auto simp: continuous_within_eps_delta dist_norm)
   show thesis
```

```
    proof
      show 2/B > 0
        using ‹B > 0› by simp
      show norm (x − a) ≤ 2 / B ∗ norm (f x − f a)
        if x ∈ S norm (f x − f a) < e for x
      proof −
        have xa: norm (x − a) < d
          using e [OF that] gf by (simp add: ‹a ∈ S› that)
        have ∗: ⟦norm(y − f′) ≤ B / 2 ∗ norm x; B ∗ norm x ≤ norm f′⟧
              ⟹ norm y ≥ B / 2 ∗ norm x for y f′::′b and x::′a
          using norm_triangle_ineq3 [of y f′] by linarith
        show ?thesis
          using ∗ [OF d [OF ‹x ∈ S› xa] B] ‹B > 0› by (simp add: field_simps)
      qed
    qed (use ‹e > 0› in auto)
  qed
  show ?thesis
    unfolding has_derivative_within_alt
  proof (intro conjI impI allI)
    show bounded_linear g′
      using ‹linear g′› by (simp add: linear_linear)
  next
    fix e :: real
    assume e > 0
    then obtain d where d>0
      and d: ⋀y. ⟦y ∈ S; norm (y − a) < d⟧ ⟹
                 norm (f y − f a − f′ (y − a)) ≤ e / (B ∗ C) ∗ norm (y − a)
      using f′ [of e / (B ∗ C)] ‹B > 0› ‹C > 0› by auto
    have norm (x − a − g′ (f x − f a)) ≤ e ∗ norm (f x − f a)
     if x ∈ S and lt_k: norm (f x − f a) < k and lt_dB: norm (f x − f a) < d/B
for x
    proof −
      have norm (x − a) ≤ B ∗ norm(f x − f a)
        using Bk lt_k ‹x ∈ S› by blast
      also have ... < d
        by (metis ‹0 < B› lt_dB mult.commute pos_less_divide_eq)
      finally have lt_d: norm (x − a) < d .
      have norm (x − a − g′ (f x − f a)) ≤ norm(g′(f x − f a − (f′ (x − a))))
        by (simp add: linear_diff [OF ‹linear g′›] norm_minus_commute)
      also have ... ≤ C ∗ norm (f x − f a − f′ (x − a))
        using C by blast
      also have ... ≤ e ∗ norm (f x − f a)
      proof −
        have norm (f x − f a − f′ (x − a)) ≤ e / (B ∗ C) ∗ norm (x − a)
          using d [OF ‹x ∈ S› lt_d] .
        also have ... ≤ (norm (f x − f a) ∗ e) / C
          using ‹B > 0› ‹C > 0› ‹e > 0› by (simp add: field_simps Bk lt_k ‹x ∈
S›)
        finally show ?thesis
```

```
        using ‹C > 0› by (simp add: field_simps)
    qed
  finally show ?thesis .
  qed
  with ‹k > 0› ‹B > 0› ‹d > 0› ‹a ∈ S›
  show ∃ d>0. ∀ y∈f ‘ S.
            norm (y − f a) < d ⟶
            norm (g y − g (f a) − g′ (y − f a)) ≤ e ∗ norm (y − f a)
    by (rule_tac x=min k (d / B) in exI) (auto simp: gf)
 qed
qed

end
```

6.17 Complex Analysis Basics

Definitions of analytic and holomorphic functions, limit theorems, complex differentiation

```
theory Complex_Analysis_Basics
 imports Derivative HOL−Library.Nonpos_Ints Uncountable_Sets
begin
```

6.17.1 General lemmas

```
lemma nonneg_Reals_cmod_eq_Re: z ∈ ℝ≥0 ⟹ norm z = Re z
 by (simp add: complex_nonneg_Reals_iff cmod_eq_Re)

lemma fact_cancel:
 fixes c :: ′a::real_field
 shows of_nat (Suc n) ∗ c / (fact (Suc n)) = c / (fact n)
 using of_nat_neq_0 by force

lemma vector_derivative_cnj_within:
 assumes at x within A ≠ bot and f differentiable at x within A
 shows   vector_derivative (λz. cnj (f z)) (at x within A) =
           cnj (vector_derivative f (at x within A)) (is _ = cnj ?D)
proof −
 let ?D = vector_derivative f (at x within A)
 from assms have (f has_vector_derivative ?D) (at x within A)
   by (subst (asm) vector_derivative_works)
 hence ((λx. cnj (f x)) has_vector_derivative cnj ?D) (at x within A)
   by (rule has_vector_derivative_cnj)
 thus ?thesis using assms by (auto dest: vector_derivative_within)
qed

lemma vector_derivative_cnj:
 assumes f differentiable at x
```

shows *vector_derivative* ($\lambda z.$ *cnj* (*f z*)) (*at x*) = *cnj* (*vector_derivative f* (*at x*))
using *assms* **by** (*intro vector_derivative_cnj_within*) *auto*

lemma
shows *open_halfspace_Re_lt*: *open* $\{z.\ Re(z) < b\}$
and *open_halfspace_Re_gt*: *open* $\{z.\ Re(z) > b\}$
and *closed_halfspace_Re_ge*: *closed* $\{z.\ Re(z) \geq b\}$
and *closed_halfspace_Re_le*: *closed* $\{z.\ Re(z) \leq b\}$
and *closed_halfspace_Re_eq*: *closed* $\{z.\ Re(z) = b\}$
and *open_halfspace_Im_lt*: *open* $\{z.\ Im(z) < b\}$
and *open_halfspace_Im_gt*: *open* $\{z.\ Im(z) > b\}$
and *closed_halfspace_Im_ge*: *closed* $\{z.\ Im(z) \geq b\}$
and *closed_halfspace_Im_le*: *closed* $\{z.\ Im(z) \leq b\}$
and *closed_halfspace_Im_eq*: *closed* $\{z.\ Im(z) = b\}$
by (*intro open_Collect_less closed_Collect_le closed_Collect_eq continuous_on_Re continuous_on_Im continuous_on_id continuous_on_const*)+

lemma *uncountable_halfspace_Im_gt*: *uncountable* $\{z.\ Im\ z > c\}$
proof −
obtain *r* **where** *r*: $r > 0$ *ball* $((c + 1) *_R \mathrm{i})\ r \subseteq \{z.\ Im\ z > c\}$
using *open_halfspace_Im_gt*[*of c*] **unfolding** *open_contains_ball* **by** *force*
then show *?thesis*
using *countable_subset uncountable_ball* **by** *blast*
qed

lemma *uncountable_halfspace_Im_lt*: *uncountable* $\{z.\ Im\ z < c\}$
proof −
obtain *r* **where** *r*: $r > 0$ *ball* $((c - 1) *_R \mathrm{i})\ r \subseteq \{z.\ Im\ z < c\}$
using *open_halfspace_Im_lt*[*of c*] **unfolding** *open_contains_ball* **by** *force*
then show *?thesis*
using *countable_subset uncountable_ball* **by** *blast*
qed

lemma *uncountable_halfspace_Re_gt*: *uncountable* $\{z.\ Re\ z > c\}$
proof −
obtain *r* **where** *r*: $r > 0$ *ball* (*of_real*$(c + 1))\ r \subseteq \{z.\ Re\ z > c\}$
using *open_halfspace_Re_gt*[*of c*] **unfolding** *open_contains_ball* **by** *force*
then show *?thesis*
using *countable_subset uncountable_ball* **by** *blast*
qed

lemma *uncountable_halfspace_Re_lt*: *uncountable* $\{z.\ Re\ z < c\}$
proof −
obtain *r* **where** *r*: $r > 0$ *ball* (*of_real*$(c - 1))\ r \subseteq \{z.\ Re\ z < c\}$
using *open_halfspace_Re_lt*[*of c*] **unfolding** *open_contains_ball* **by** *force*
then show *?thesis*
using *countable_subset uncountable_ball* **by** *blast*
qed

lemma *connected_halfspace_Im_gt* [*intro*]: *connected* {*z*. *c* < *Im z*}
by (*intro convex_connected convex_halfspace_Im_gt*)

lemma *connected_halfspace_Im_lt* [*intro*]: *connected* {*z*. *c* > *Im z*}
by (*intro convex_connected convex_halfspace_Im_lt*)

lemma *connected_halfspace_Re_gt* [*intro*]: *connected* {*z*. *c* < *Re z*}
by (*intro convex_connected convex_halfspace_Re_gt*)

lemma *connected_halfspace_Re_lt* [*intro*]: *connected* {*z*. *c* > *Re z*}
by (*intro convex_connected convex_halfspace_Re_lt*)

lemma *closed_complex_Reals*: *closed* (ℝ :: *complex set*)
proof −
have (ℝ :: *complex set*) = {*z*. *Im z* = *0*}
by (*auto simp*: *complex_is_Real_iff*)
then show *?thesis*
by (*metis closed_halfspace_Im_eq*)
qed

lemma *closed_Real_halfspace_Re_le*: *closed* (ℝ ∩ {*w*. *Re w* ≤ *x*})
by (*simp add*: *closed_Int closed_complex_Reals closed_halfspace_Re_le*)

lemma *closed_nonpos_Reals_complex* [*simp*]: *closed* ($\mathbb{R}_{\leq 0}$ :: *complex set*)
proof −
have $\mathbb{R}_{\leq 0}$ = ℝ ∩ {*z*. *Re*(*z*) ≤ *0*}
using *complex_nonpos_Reals_iff complex_is_Real_iff* **by** *auto*
then show *?thesis*
by (*metis closed_Real_halfspace_Re_le*)
qed

lemma *closed_Real_halfspace_Re_ge*: *closed* (ℝ ∩ {*w*. *x* ≤ *Re*(*w*)})
using *closed_halfspace_Re_ge*
by (*simp add*: *closed_Int closed_complex_Reals*)

lemma *closed_nonneg_Reals_complex* [*simp*]: *closed* ($\mathbb{R}_{\geq 0}$ :: *complex set*)
proof −
have $\mathbb{R}_{\geq 0}$ = ℝ ∩ {*z*. *Re*(*z*) ≥ *0*}
using *complex_nonneg_Reals_iff complex_is_Real_iff* **by** *auto*
then show *?thesis*
by (*metis closed_Real_halfspace_Re_ge*)
qed

lemma *closed_real_abs_le*: *closed* {*w* ∈ ℝ. |*Re w*| ≤ *r*}
proof −
have {*w* ∈ ℝ. |*Re w*| ≤ *r*} = (ℝ ∩ {*w*. *Re w* ≤ *r*}) ∩ (ℝ ∩ {*w*. *Re w* ≥ −*r*})
by *auto*
then show *closed* {*w* ∈ ℝ. |*Re w*| ≤ *r*}

by (*simp add*: *closed_Int closed_Real_halfspace_Re_ge closed_Real_halfspace_Re_le*)
qed

lemma *real_lim*:
fixes *l*::*complex*
assumes (*f* ⟶ *l*) *F* **and** ¬ *trivial_limit F* **and** *eventually P F* **and** ⋀*a*. *P a* ⟹ *f a* ∈ ℝ
shows *l* ∈ ℝ
proof (*rule Lim_in_closed_set*[*OF closed_complex_Reals _ assms*(*2*,*1*)])
show *eventually* (λ*x*. *f x* ∈ ℝ) *F*
using *assms*(*3*, *4*) **by** (*auto intro*: *eventually_mono*)
qed

lemma *real_lim_sequentially*:
fixes *l*::*complex*
shows (*f* ⟶ *l*) *sequentially* ⟹ (∃ *N*. ∀ *n*≥*N*. *f n* ∈ ℝ) ⟹ *l* ∈ ℝ
by (*rule real_lim* [**where** *F*=*sequentially*]) (*auto simp*: *eventually_sequentially*)

lemma *real_series*:
fixes *l*::*complex*
shows *f sums l* ⟹ (⋀*n*. *f n* ∈ ℝ) ⟹ *l* ∈ ℝ
unfolding *sums_def*
by (*metis real_lim_sequentially sum_in_Reals*)

lemma *Lim_null_comparison_Re*:
assumes *eventually* (λ*x*. *norm*(*f x*) ≤ *Re*(*g x*)) *F* (*g* ⟶ *0*) *F* **shows** (*f* ⟶ *0*) *F*
by (*rule Lim_null_comparison*[*OF assms*(*1*)] *tendsto_eq_intros assms*(*2*))+ *simp*

6.17.2 Holomorphic functions

definition *holomorphic_on* :: [*complex* ⇒ *complex*, *complex set*] ⇒ *bool*
(**infixl** (*holomorphic'_on*) *50*)
where *f holomorphic_on s* ≡ ∀ *x*∈*s*. *f field_differentiable* (*at x within s*)

named_theorems *holomorphic_intros structural introduction rules for holomorphic_on*

lemma *holomorphic_onI* [*intro?*]: (⋀*x*. *x* ∈ *s* ⟹ *f field_differentiable* (*at x within s*)) ⟹ *f holomorphic_on s*
by (*simp add*: *holomorphic_on_def*)

lemma *holomorphic_onD* [*dest?*]: ⟦*f holomorphic_on s*; *x* ∈ *s*⟧ ⟹ *f field_differentiable* (*at x within s*)
by (*simp add*: *holomorphic_on_def*)

lemma *holomorphic_on_imp_differentiable_on*:
f holomorphic_on s ⟹ *f differentiable_on s*

unfolding *holomorphic_on_def differentiable_on_def*
by (*simp add*: *field_differentiable_imp_differentiable*)

lemma *holomorphic_on_imp_differentiable_at*:
⟦*f holomorphic_on s*; *open s*; *x* ∈ *s*⟧ ⟹ *f field_differentiable* (*at x*)
using *at_within_open holomorphic_on_def* **by** *fastforce*

lemma *holomorphic_on_empty* [*holomorphic_intros*]: *f holomorphic_on* {}
by (*simp add*: *holomorphic_on_def*)

lemma *holomorphic_on_open*:
open s ⟹ *f holomorphic_on s* ⟷ (∀ *x* ∈ *s*. ∃ *f'*. *DERIV f x* :> *f'*)
by (*auto simp*: *holomorphic_on_def field_differentiable_def has_field_derivative_def at_within_open* [*of* _ *s*])

lemma *holomorphic_on_UN_open*:
assumes ⋀*n*. *n* ∈ *I* ⟹ *f holomorphic_on A n* ⋀*n*. *n* ∈ *I* ⟹ *open* (*A n*)
shows *f holomorphic_on* (⋃ *n*∈*I*. *A n*)
proof −
have *f field_differentiable at z within* (⋃ *n*∈*I*. *A n*) **if** *z* ∈ (⋃ *n*∈*I*. *A n*) **for** *z*
proof −
from *that* **obtain** *n* **where** *n* ∈ *I z* ∈ *A n*
by *blast*
hence *f holomorphic_on A n open* (*A n*)
by (*simp add*: *assms*)+
with ‹*z* ∈ *A n*› **have** *f field_differentiable at z*
by (*auto simp*: *holomorphic_on_open field_differentiable_def*)
thus *?thesis*
by (*meson field_differentiable_at_within*)
qed
thus *?thesis*
by (*auto simp*: *holomorphic_on_def*)
qed

lemma *holomorphic_on_imp_continuous_on*:
f holomorphic_on s ⟹ *continuous_on s f*
by (*metis field_differentiable_imp_continuous_at continuous_on_eq_continuous_within holomorphic_on_def*)

lemma *holomorphic_closedin_preimage_constant*:
assumes *f holomorphic_on D*
shows *closedin* (*top_of_set D*) {*z*∈*D*. *f z* = *a*}
by (*simp add*: *assms continuous_closedin_preimage_constant holomorphic_on_imp_continuous_on*)

lemma *holomorphic_closed_preimage_constant*:
assumes *f holomorphic_on UNIV*
shows *closed* {*z*. *f z* = *a*}
using *holomorphic_closedin_preimage_constant* [*OF assms*] **by** *simp*

lemma *holomorphic_on_subset* [*elim*]:
  f holomorphic_on s ⟹ *t* ⊆ *s* ⟹ *f holomorphic_on t*
 unfolding *holomorphic_on_def*
 by (*metis field_differentiable_within_subset subsetD*)

lemma *holomorphic_transform*: ⟦*f holomorphic_on s*; ⋀*x*. *x* ∈ *s* ⟹ *f x* = *g x*⟧
⟹ *g holomorphic_on s*
 by (*metis field_differentiable_transform_within linordered_field_no_ub holomorphic_on_def*)

lemma *holomorphic_cong*: *s* = *t* ==> (⋀*x*. *x* ∈ *s* ⟹ *f x* = *g x*) ⟹ *f holomorphic_on s* ⟷ *g holomorphic_on t*
 by (*metis holomorphic_transform*)

lemma *holomorphic_on_linear* [*simp*, *holomorphic_intros*]: ((∗) *c*) *holomorphic_on s*
 unfolding *holomorphic_on_def* **by** (*metis field_differentiable_linear*)

lemma *holomorphic_on_const* [*simp*, *holomorphic_intros*]: (λ*z*. *c*) *holomorphic_on s*
 unfolding *holomorphic_on_def* **by** (*metis field_differentiable_const*)

lemma *holomorphic_on_ident* [*simp*, *holomorphic_intros*]: (λ*x*. *x*) *holomorphic_on s*
 unfolding *holomorphic_on_def* **by** (*metis field_differentiable_ident*)

lemma *holomorphic_on_id* [*simp*, *holomorphic_intros*]: *id holomorphic_on s*
 unfolding *id_def* **by** (*rule holomorphic_on_ident*)

lemma *constant_on_imp_holomorphic_on*:
 assumes *f constant_on A*
 shows *f holomorphic_on A*
proof −
 from *assms* **obtain** *c* **where** *c*: ∀ *x*∈*A*. *f x* = *c*
  unfolding *constant_on_def* **by** *blast*
 have *f holomorphic_on A* ⟷ (λ_. *c*) *holomorphic_on A*
  by (*intro holomorphic_cong*) (*use c* **in** *auto*)
 thus *?thesis*
  by *simp*
qed

lemma *holomorphic_on_compose*:
 f holomorphic_on s ⟹ *g holomorphic_on* (*f* ‘ *s*) ⟹ (*g o f*) *holomorphic_on s*
 using *field_differentiable_compose_within*[*of f _ s g*]
 by (*auto simp*: *holomorphic_on_def*)

lemma *holomorphic_on_compose_gen*:
 f holomorphic_on s ⟹ *g holomorphic_on t* ⟹ *f* ‘ *s* ⊆ *t* ⟹ (*g o f*) *holomorphic_on s*

by (*metis holomorphic_on_compose holomorphic_on_subset*)

lemma *holomorphic_on_balls_imp_entire*:
assumes ¬*bdd_above A* ⋀*r*. *r* ∈ *A* ⟹ *f holomorphic_on ball c r*
shows *f holomorphic_on B*
proof (*rule holomorphic_on_subset*)
show *f holomorphic_on UNIV* **unfolding** *holomorphic_on_def*
proof
fix *z* :: *complex*
from ‹¬*bdd_above A*› **obtain** *r* **where** *r*: *r* ∈ *A r* > *norm* (*z* − *c*)
by (*meson bdd_aboveI not_le*)
with *assms*(*2*) **have** *f holomorphic_on ball c r* **by** *blast*
moreover from *r* **have** *z* ∈ *ball c r* **by** (*auto simp*: *dist_norm norm_minus_commute*)
ultimately show *f field_differentiable at z*
by (*auto simp*: *holomorphic_on_def at_within_open*[*of* _ *ball c r*])
qed
qed *auto*

lemma *holomorphic_on_balls_imp_entire'*:
assumes ⋀*r*. *r* > *0* ⟹ *f holomorphic_on ball c r*
shows *f holomorphic_on B*
proof (*rule holomorphic_on_balls_imp_entire*)
{
fix *M* :: *real*
have ∃ *x*. *x* > *max M 0* **by** (*intro gt_ex*)
hence ∃ *x*>*0*. *x* > *M* **by** *auto*
}
thus ¬*bdd_above* {(*0*::*real*)<..} **unfolding** *bdd_above_def*
by (*auto simp*: *not_le*)
qed (*insert assms*, *auto*)

lemma *holomorphic_on_minus* [*holomorphic_intros*]: *f holomorphic_on A* ⟹
(λ*z*. −(*f z*)) *holomorphic_on A*
by (*metis field_differentiable_minus holomorphic_on_def*)

lemma *holomorphic_on_add* [*holomorphic_intros*]:
⟦*f holomorphic_on A*; *g holomorphic_on A*⟧ ⟹ (λ*z*. *f z* + *g z*) *holomorphic_on*
A
unfolding *holomorphic_on_def* **by** (*metis field_differentiable_add*)

lemma *holomorphic_on_diff* [*holomorphic_intros*]:
⟦*f holomorphic_on A*; *g holomorphic_on A*⟧ ⟹ (λ*z*. *f z* − *g z*) *holomorphic_on*
A
unfolding *holomorphic_on_def* **by** (*metis field_differentiable_diff*)

lemma *holomorphic_on_mult* [*holomorphic_intros*]:
⟦*f holomorphic_on A*; *g holomorphic_on A*⟧ ⟹ (λ*z*. *f z* ∗ *g z*) *holomorphic_on*
A
unfolding *holomorphic_on_def* **by** (*metis field_differentiable_mult*)

lemma *holomorphic_on_inverse* [*holomorphic_intros*]:
  ⟦*f holomorphic_on A*; ⋀*z*. *z* ∈ *A* ⟹ *f z* ≠ *0*⟧ ⟹ (λ*z*. *inverse* (*f z*)) *holomorphic_on A*
  unfolding *holomorphic_on_def* **by** (*metis field_differentiable_inverse*)

lemma *holomorphic_on_divide* [*holomorphic_intros*]:
  ⟦*f holomorphic_on A*; *g holomorphic_on A*; ⋀*z*. *z* ∈ *A* ⟹ *g z* ≠ *0*⟧ ⟹ (λ*z*. *f z* / *g z*) *holomorphic_on A*
  unfolding *holomorphic_on_def* **by** (*metis field_differentiable_divide*)

lemma *holomorphic_on_power* [*holomorphic_intros*]:
  f holomorphic_on A ⟹ (λ*z*. (*f z*) ^ *n*) *holomorphic_on A*
  unfolding *holomorphic_on_def* **by** (*metis field_differentiable_power*)

lemma *holomorphic_on_power_int* [*holomorphic_intros*]:
  assumes *nz*: *n* ≥ *0* ∨ (∀*x*∈*A*. *f x* ≠ *0*) **and** *f*: *f holomorphic_on A*
  shows (λ*x*. *f x powi n*) *holomorphic_on A*
proof (*cases n* ≥ *0*)
  case *True*
  have (λ*x*. *f x* ^ *nat n*) *holomorphic_on A*
    by (*simp add*: *f holomorphic_on_power*)
  with *True* **show** *?thesis*
    by (*simp add*: *power_int_def*)
next
  case *False*
  hence (λ*x*. *inverse* (*f x* ^ *nat* (−*n*))) *holomorphic_on A*
    using *nz* **by** (*auto intro*!: *holomorphic_intros f*)
  with *False* **show** *?thesis*
    by (*simp add*: *power_int_def power_inverse*)
qed

lemma *holomorphic_on_sum* [*holomorphic_intros*]:
  (⋀*i*. *i* ∈ *I* ⟹ (*f i*) *holomorphic_on A*) ⟹ (λ*x*. *sum* (λ*i*. *f i x*) *I*) *holomorphic_on A*
  unfolding *holomorphic_on_def* **by** (*metis field_differentiable_sum*)

lemma *holomorphic_on_prod* [*holomorphic_intros*]:
  (⋀*i*. *i* ∈ *I* ⟹ (*f i*) *holomorphic_on A*) ⟹ (λ*x*. *prod* (λ*i*. *f i x*) *I*) *holomorphic_on A*
  by (*induction I rule*: *infinite_finite_induct*) (*auto intro*: *holomorphic_intros*)

lemma *holomorphic_pochhammer* [*holomorphic_intros*]:
  f holomorphic_on A ⟹ (λ*s*. *pochhammer* (*f s*) *n*) *holomorphic_on A*
  by (*induction n*) (*auto intro*!: *holomorphic_intros simp*: *pochhammer_Suc*)

lemma *holomorphic_on_scaleR* [*holomorphic_intros*]:
  f holomorphic_on A ⟹ (λ*x*. *c* *$_R$ *f x*) *holomorphic_on A*
  by (*auto simp*: *scaleR_conv_of_real intro*!: *holomorphic_intros*)

lemma *holomorphic_on_Un* [*holomorphic_intros*]:
assumes *f holomorphic_on A f holomorphic_on B open A open B*
shows *f holomorphic_on* $(A \cup B)$
using *assms* **by** (*auto simp*: *holomorphic_on_def at_within_open*[*of _ A*] *at_within_open*[*of _ B*] *at_within_open*[*of _* $A \cup B$] *open_Un*)

lemma *holomorphic_on_If_Un* [*holomorphic_intros*]:
assumes *f holomorphic_on A g holomorphic_on B open A open B*
assumes $\bigwedge z.\ z \in A \Longrightarrow z \in B \Longrightarrow f\ z = g\ z$
shows $(\lambda z.\ \textit{if}\ z \in A\ \textit{then}\ f\ z\ \textit{else}\ g\ z)$ *holomorphic_on* $(A \cup B)$ (**is** *?h holomorphic_on _*)
proof (*intro holomorphic_on_Un*)
note ‹*f holomorphic_on A*›
also have *f holomorphic_on A* $\longleftrightarrow$ *?h holomorphic_on A*
by (*intro holomorphic_cong*) *auto*
finally show ... **.**
next
note ‹*g holomorphic_on B*›
also have *g holomorphic_on B* $\longleftrightarrow$ *?h holomorphic_on B*
using *assms* **by** (*intro holomorphic_cong*) *auto*
finally show ... **.**
qed (*insert assms*, *auto*)

lemma *holomorphic_derivI*:
$[\![f\ \textit{holomorphic\_on}\ S;\ \textit{open}\ S;\ x \in S]\!]$
$\Longrightarrow$ (*f has_field_derivative deriv f x*) (*at x within T*)
by (*metis DERIV_deriv_iff_field_differentiable at_within_open holomorphic_on_def has_field_derivative_at_within*)

lemma *complex_derivative_transform_within_open*:
$[\![f\ \textit{holomorphic\_on}\ s;\ g\ \textit{holomorphic\_on}\ s;\ \textit{open}\ s;\ z \in s;\ \bigwedge w.\ w \in s \Longrightarrow f\ w = g\ w]\!]$
$\Longrightarrow$ *deriv f z* = *deriv g z*
unfolding *holomorphic_on_def*
by (*rule DERIV_imp_deriv*)
(*metis DERIV_deriv_iff_field_differentiable has_field_derivative_transform_within_open at_within_open*)

lemma *holomorphic_on_compose_cnj_cnj*:
assumes *f holomorphic_on cnj ‘ A open A*
shows *cnj* $\circ$ *f* $\circ$ *cnj holomorphic_on A*
proof −
have [*simp*]: *open* (*cnj ‘ A*)
unfolding *image_cnj_conv_vimage_cnj* **using** *assms* **by** (*intro open_vimage*) *auto*
show *?thesis*
using *assms* **unfolding** *holomorphic_on_def*

by (*auto intro*!: *field_differentiable_cnj_cnj simp*: *at_within_open_NO_MATCH*)
qed

lemma *holomorphic_nonconstant*:
assumes *holf*: *f holomorphic_on S* **and** *open S ξ ∈ S deriv f ξ ≠ 0*
shows ¬ *f constant_on S*
by (*rule nonzero_deriv_nonconstant* [*of f deriv f ξ ξ S*])
(*use assms* **in** ‹*auto simp*: *holomorphic_derivI*›)

6.17.3 Analyticity on a set

definition *analytic_on* (**infixl** (*analytic'_on*) *50*)
where *f analytic_on S* ≡ ∀ *x* ∈ *S*. ∃ *e*. *0* < *e* ∧ *f holomorphic_on* (*ball x e*)

named_theorems *analytic_intros introduction rules for proving analyticity*

lemma *analytic_imp_holomorphic*: *f analytic_on S* ⟹ *f holomorphic_on S*
by (*simp add*: *at_within_open* [*OF _ open_ball*] *analytic_on_def holomorphic_on_def*)
(*metis centre_in_ball field_differentiable_at_within*)

lemma *analytic_on_open*: *open S* ⟹ *f analytic_on S* ⟷ *f holomorphic_on S*
by (*meson analytic_imp_holomorphic analytic_on_def holomorphic_on_subset openE*)

lemma *analytic_on_imp_differentiable_at*:
f analytic_on S ⟹ *x* ∈ *S* ⟹ *f field_differentiable* (*at x*)
using *analytic_on_def holomorphic_on_imp_differentiable_at* **by** *auto*

lemma *analytic_at_imp_isCont*:
assumes *f analytic_on* {*z*}
shows *isCont f z*
using *assms* **by** (*meson analytic_on_imp_differentiable_at field_differentiable_imp_continuous_at insertI1*)

lemma *analytic_at_neq_imp_eventually_neq*:
assumes *f analytic_on* {*x*} *f x* ≠ *c*
shows *eventually* (λ*y*. *f y* ≠ *c*) (*at x*)
proof (*intro tendsto_imp_eventually_ne*)
show *f* −*x*→ *f x*
using *assms* **by** (*simp add*: *analytic_at_imp_isCont isContD*)
qed (*use assms* **in** *auto*)

lemma *analytic_on_subset*: *f analytic_on S* ⟹ *T* ⊆ *S* ⟹ *f analytic_on T*
by (*auto simp*: *analytic_on_def*)

lemma *analytic_on_Un*: *f analytic_on* (*S* ∪ *T*) ⟷ *f analytic_on S* ∧ *f analytic_on T*
by (*auto simp*: *analytic_on_def*)

lemma *analytic_on_Union*: *f analytic_on* $(\bigcup \mathcal{T}) \longleftrightarrow (\forall\, T \in \mathcal{T}.\ f$ *analytic_on* $T)$
by (*auto simp*: *analytic_on_def*)

lemma *analytic_on_UN*: *f analytic_on* $(\bigcup i{\in}I.\ S\ i) \longleftrightarrow (\forall\, i{\in}I.\ f$ *analytic_on* $(S\ i))$
by (*auto simp*: *analytic_on_def*)

lemma *analytic_on_holomorphic*:
f analytic_on S $\longleftrightarrow$ $(\exists\, T.$ *open* $T \wedge S \subseteq T \wedge f$ *holomorphic_on* $T)$
(**is** *?lhs* = *?rhs*)
proof −
have *?lhs* $\longleftrightarrow$ $(\exists\, T.$ *open* $T \wedge S \subseteq T \wedge f$ *analytic_on* $T)$
proof *safe*
assume *f analytic_on S*
then show $\exists\, T.$ *open* $T \wedge S \subseteq T \wedge f$ *analytic_on* T
apply (*simp add*: *analytic_on_def*)
apply (*rule exI* [**where** $x{=}\bigcup\{U.$ *open* $U \wedge f$ *analytic_on* $U\}$], *auto*)
apply (*metis open_ball analytic_on_open centre_in_ball*)
by (*metis analytic_on_def*)
next
fix T
assume *open* T $S \subseteq T$ *f analytic_on* T
then show *f analytic_on S*
by (*metis analytic_on_subset*)
qed
also have ... $\longleftrightarrow$ *?rhs*
by (*auto simp*: *analytic_on_open*)
finally show *?thesis* **.**
qed

lemma *analytic_on_linear* [*analytic_intros*,*simp*]: $((*)\ c)$ *analytic_on S*
by (*auto simp add*: *analytic_on_holomorphic*)

lemma *analytic_on_const* [*analytic_intros*,*simp*]: $(\lambda z.\ c)$ *analytic_on S*
by (*metis analytic_on_def holomorphic_on_const zero_less_one*)

lemma *analytic_on_ident* [*analytic_intros*,*simp*]: $(\lambda x.\ x)$ *analytic_on S*
by (*simp add*: *analytic_on_def gt_ex*)

lemma *analytic_on_id* [*analytic_intros*]: *id analytic_on S*
unfolding *id_def* **by** (*rule analytic_on_ident*)

lemma *analytic_on_compose*:
assumes *f*: *f analytic_on S*
and *g*: *g analytic_on* $(f\ `\ S)$
shows $(g\ o\ f)$ *analytic_on S*
unfolding *analytic_on_def*

proof (*intro ballI*)
fix x
assume x: $x \in S$
then obtain e **where** e: $0 < e$ **and** fh: f *holomorphic_on ball* x e **using** f
by (*metis analytic_on_def*)
obtain e' **where** e': $0 < e'$ **and** gh: g *holomorphic_on ball* $(f\ x)$ e' **using** g
by (*metis analytic_on_def g image_eqI x*)
have *isCont f x*
by (*metis analytic_on_imp_differentiable_at field_differentiable_imp_continuous_at f x*)
with e' **obtain** d **where** d: $0 < d$ **and** fd: f ‘ *ball* x $d \subseteq$ *ball* $(f\ x)$ e'
by (*auto simp*: *continuous_at_ball*)
have $g \circ f$ *holomorphic_on ball* x (*min* d e)
apply (*rule holomorphic_on_compose*)
apply (*metis fh holomorphic_on_subset min.bounded_iff order_refl subset_ball*)
by (*metis fd gh holomorphic_on_subset image_mono min.cobounded1 subset_ball*)
then show $\exists e{>}0.$ $g \circ f$ *holomorphic_on ball* x e
by (*metis d e min_less_iff_conj*)
qed

lemma *analytic_on_compose_gen*:
f *analytic_on* $S \Longrightarrow$ g *analytic_on* $T \Longrightarrow (\bigwedge z.\ z \in S \Longrightarrow f\ z \in T)$
$\Longrightarrow$ g *o* f *analytic_on* S
by (*metis analytic_on_compose analytic_on_subset image_subset_iff*)

lemma *analytic_on_neg* [*analytic_intros*]:
f *analytic_on* $S \Longrightarrow (\lambda z.\ -(f\ z))$ *analytic_on* S
by (*metis analytic_on_holomorphic holomorphic_on_minus*)

lemma *analytic_on_add* [*analytic_intros*]:
assumes f: f *analytic_on* S
and g: g *analytic_on* S
shows $(\lambda z.\ f\ z + g\ z)$ *analytic_on* S
unfolding *analytic_on_def*
proof (*intro ballI*)
fix z
assume z: $z \in S$
then obtain e **where** e: $0 < e$ **and** fh: f *holomorphic_on ball* z e **using** f
by (*metis analytic_on_def*)
obtain e' **where** e': $0 < e'$ **and** gh: g *holomorphic_on ball* z e' **using** g
by (*metis analytic_on_def g z*)
have $(\lambda z.\ f\ z + g\ z)$ *holomorphic_on ball* z (*min* e e')
apply (*rule holomorphic_on_add*)
apply (*metis fh holomorphic_on_subset min.bounded_iff order_refl subset_ball*)
by (*metis gh holomorphic_on_subset min.bounded_iff order_refl subset_ball*)
then show $\exists e{>}0.$ $(\lambda z.\ f\ z + g\ z)$ *holomorphic_on ball* z e
by (*metis e e' min_less_iff_conj*)
qed

lemma *analytic_on_diff* [*analytic_intros*]:
  assumes *f*: *f analytic_on S*
    and *g*: *g analytic_on S*
  shows (λ*z*. *f z* − *g z*) *analytic_on S*
unfolding *analytic_on_def*
proof (*intro ballI*)
  fix *z*
  assume *z*: *z* ∈ *S*
  then obtain *e* **where** *e*: *0* < *e* **and** *fh*: *f holomorphic_on ball z e* **using** *f*
    by (*metis analytic_on_def*)
  obtain *e'* **where** *e'*: *0* < *e'* **and** *gh*: *g holomorphic_on ball z e'* **using** *g*
    by (*metis analytic_on_def g z*)
  have (λ*z*. *f z* − *g z*) *holomorphic_on ball z* (*min e e'*)
    apply (*rule holomorphic_on_diff*)
   apply (*metis fh holomorphic_on_subset min.bounded_iff order_refl subset_ball*)
    by (*metis gh holomorphic_on_subset min.bounded_iff order_refl subset_ball*)
  then show ∃ *e*>*0*. (λ*z*. *f z* − *g z*) *holomorphic_on ball z e*
    by (*metis e e' min_less_iff_conj*)
qed

lemma *analytic_on_mult* [*analytic_intros*]:
  assumes *f*: *f analytic_on S*
    and *g*: *g analytic_on S*
  shows (λ*z*. *f z* ∗ *g z*) *analytic_on S*
unfolding *analytic_on_def*
proof (*intro ballI*)
  fix *z*
  assume *z*: *z* ∈ *S*
  then obtain *e* **where** *e*: *0* < *e* **and** *fh*: *f holomorphic_on ball z e* **using** *f*
    by (*metis analytic_on_def*)
  obtain *e'* **where** *e'*: *0* < *e'* **and** *gh*: *g holomorphic_on ball z e'* **using** *g*
    by (*metis analytic_on_def g z*)
  have (λ*z*. *f z* ∗ *g z*) *holomorphic_on ball z* (*min e e'*)
    apply (*rule holomorphic_on_mult*)
   apply (*metis fh holomorphic_on_subset min.bounded_iff order_refl subset_ball*)
    by (*metis gh holomorphic_on_subset min.bounded_iff order_refl subset_ball*)
  then show ∃ *e*>*0*. (λ*z*. *f z* ∗ *g z*) *holomorphic_on ball z e*
    by (*metis e e' min_less_iff_conj*)
qed

lemma *analytic_on_inverse* [*analytic_intros*]:
  assumes *f*: *f analytic_on S*
    and *nz*: (⋀*z*. *z* ∈ *S* ⟹ *f z* ≠ *0*)
  shows (λ*z*. *inverse* (*f z*)) *analytic_on S*
unfolding *analytic_on_def*
proof (*intro ballI*)
  fix *z*
  assume *z*: *z* ∈ *S*

```
  then obtain e where e: 0 < e and fh: f holomorphic_on ball z e using f
    by (metis analytic_on_def)
  have continuous_on (ball z e) f
    by (metis fh holomorphic_on_imp_continuous_on)
  then obtain e' where e': 0 < e' and nz': ⋀y. dist z y < e' ⟹ f y ≠ 0
    by (metis open_ball centre_in_ball continuous_on_open_avoid e z nz)
  have (λz. inverse (f z)) holomorphic_on ball z (min e e')
    apply (rule holomorphic_on_inverse)
     apply (metis fh holomorphic_on_subset min.cobounded2 min.commute subset_ball)
    by (metis nz' mem_ball min_less_iff_conj)
  then show ∃e>0. (λz. inverse (f z)) holomorphic_on ball z e
    by (metis e e' min_less_iff_conj)
qed

lemma analytic_on_divide [analytic_intros]:
  assumes f: f analytic_on S
      and g: g analytic_on S
      and nz: (⋀z. z ∈ S ⟹ g z ≠ 0)
    shows (λz. f z / g z) analytic_on S
unfolding divide_inverse
by (metis analytic_on_inverse analytic_on_mult f g nz)

lemma analytic_on_power [analytic_intros]:
  f analytic_on S ⟹ (λz. (f z) ^ n) analytic_on S
by (induct n) (auto simp: analytic_on_mult)

lemma analytic_on_power_int [analytic_intros]:
  assumes nz: n ≥ 0 ∨ (∀x∈A. f x ≠ 0) and f: f analytic_on A
  shows   (λx. f x powi n) analytic_on A
proof (cases n ≥ 0)
  case True
  have (λx. f x ^ nat n) analytic_on A
    using analytic_on_power f by blast
  with True show ?thesis
    by (simp add: power_int_def)
next
  case False
  hence (λx. inverse (f x ^ nat (-n))) analytic_on A
    using nz by (auto intro!: analytic_intros f)
  with False show ?thesis
    by (simp add: power_int_def power_inverse)
qed

lemma analytic_on_sum [analytic_intros]:
  (⋀i. i ∈ I ⟹ (f i) analytic_on S) ⟹ (λx. sum (λi. f i x) I) analytic_on S
  by (induct I rule: infinite_finite_induct) (auto simp: analytic_on_add)

lemma analytic_on_prod [analytic_intros]:
```

(⋀*i*. *i* ∈ *I* ⟹ (*f i*) *analytic_on S*) ⟹ (λ*x*. *prod* (λ*i*. *f i x*) *I*) *analytic_on S*
by (*induct I rule*: *infinite_finite_induct*) (*auto simp*: *analytic_on_mult*)

lemma *deriv_left_inverse*:
assumes *f holomorphic_on S* **and** *g holomorphic_on T*
and *open S* **and** *open T*
and *f ‘ S* ⊆ *T*
and [*simp*]: ⋀*z*. *z* ∈ *S* ⟹ *g* (*f z*) = *z*
and *w* ∈ *S*
shows *deriv f w* ∗ *deriv g* (*f w*) = *1*
proof −
have *deriv f w* ∗ *deriv g* (*f w*) = *deriv g* (*f w*) ∗ *deriv f w*
by (*simp add*: *algebra_simps*)
also have ... = *deriv* (*g o f*) *w*
using *assms*
by (*metis analytic_on_imp_differentiable_at analytic_on_open deriv_chain image_subset_iff*)
also have ... = *deriv id w*
proof (*rule complex_derivative_transform_within_open* [**where** *s*=*S*])
show *g* ∘ *f holomorphic_on S*
by (*rule assms holomorphic_on_compose_gen holomorphic_intros*)+
qed (*use assms* **in** *auto*)
also have ... = *1*
by *simp*
finally show *?thesis* **.**
qed

6.17.4 Analyticity at a point

lemma *analytic_at_ball*:
f analytic_on {*z*} ⟷ (∃ *e*. *0*<*e* ∧ *f holomorphic_on ball z e*)
by (*metis analytic_on_def singleton_iff*)

lemma *analytic_at*:
f analytic_on {*z*} ⟷ (∃ *s*. *open s* ∧ *z* ∈ *s* ∧ *f holomorphic_on s*)
by (*metis analytic_on_holomorphic empty_subsetI insert_subset*)

lemma *holomorphic_on_imp_analytic_at*:
assumes *f holomorphic_on A open A z* ∈ *A*
shows *f analytic_on* {*z*}
using *assms* **by** (*meson analytic_at*)

lemma *analytic_on_analytic_at*:
f analytic_on s ⟷ (∀ *z* ∈ *s*. *f analytic_on* {*z*})
by (*metis analytic_at_ball analytic_on_def*)

lemma *analytic_at_two*:
f analytic_on {*z*} ∧ *g analytic_on* {*z*} ⟷
(∃ *s*. *open s* ∧ *z* ∈ *s* ∧ *f holomorphic_on s* ∧ *g holomorphic_on s*)

```
  (is ?lhs = ?rhs)
proof
  assume ?lhs
  then obtain s t
    where st: open s z ∈ s f holomorphic_on s
              open t z ∈ t g holomorphic_on t
    by (auto simp: analytic_at)
  show ?rhs
    apply (rule_tac x=s ∩ t in exI)
    using st
    apply (auto simp: holomorphic_on_subset)
    done
next
  assume ?rhs
  then show ?lhs
    by (force simp add: analytic_at)
qed
```

6.17.5 Combining theorems for derivative with "analytic at" hypotheses

```
lemma
  assumes f analytic_on {z} g analytic_on {z}
  shows complex_derivative_add_at: deriv (λw. f w + g w) z = deriv f z + deriv
g z
    and complex_derivative_diff_at: deriv (λw. f w − g w) z = deriv f z − deriv
g z
    and complex_derivative_mult_at: deriv (λw. f w ∗ g w) z =
          f z ∗ deriv g z + deriv f z ∗ g z
proof −
  obtain s where s: open s z ∈ s f holomorphic_on s g holomorphic_on s
    using assms by (metis analytic_at_two)
  show deriv (λw. f w + g w) z = deriv f z + deriv g z
    apply (rule DERIV_imp_deriv [OF DERIV_add])
    using s
   apply (auto simp: holomorphic_on_open field_differentiable_def DERIV_deriv_iff_field_differentiable)
    done
  show deriv (λw. f w − g w) z = deriv f z − deriv g z
    apply (rule DERIV_imp_deriv [OF DERIV_diff])
    using s
   apply (auto simp: holomorphic_on_open field_differentiable_def DERIV_deriv_iff_field_differentiable)
    done
  show deriv (λw. f w ∗ g w) z = f z ∗ deriv g z + deriv f z ∗ g z
    apply (rule DERIV_imp_deriv [OF DERIV_mult'])
    using s
   apply (auto simp: holomorphic_on_open field_differentiable_def DERIV_deriv_iff_field_differentiable)
    done
qed
```

lemma *deriv_cmult_at*:
f analytic_on $\{z\} \Longrightarrow$ *deriv* $(\lambda w.\ c * f\ w)\ z = c * deriv\ f\ z$
by (*auto simp*: *complex_derivative_mult_at*)

lemma *deriv_cmult_right_at*:
f analytic_on $\{z\} \Longrightarrow$ *deriv* $(\lambda w.\ f\ w * c)\ z = deriv\ f\ z * c$
by (*auto simp*: *complex_derivative_mult_at*)

6.17.6 Complex differentiation of sequences and series

lemma *has_complex_derivative_sequence*:
fixes S :: *complex set*
assumes *cvs*: *convex S*
and *df*: $\bigwedge n\ x.\ x \in S \Longrightarrow (f\ n\ has\_field\_derivative\ f'\ n\ x)\ (at\ x\ within\ S)$
and *conv*: $\bigwedge e.\ 0 < e \Longrightarrow \exists N.\ \forall n\ x.\ n \geq N \longrightarrow x \in S \longrightarrow norm\ (f'\ n\ x - g'\ x) \leq e$
and $\exists x\ l.\ x \in S \wedge ((\lambda n.\ f\ n\ x) \longrightarrow l)\ sequentially$
shows $\exists g.\ \forall x \in S.\ ((\lambda n.\ f\ n\ x) \longrightarrow g\ x)\ sequentially \wedge$
$(g\ has\_field\_derivative\ (g'\ x))\ (at\ x\ within\ S)$
proof −
from *assms* **obtain** $x\ l$ **where** x: $x \in S$ **and** *tf*: $((\lambda n.\ f\ n\ x) \longrightarrow l)\ sequentially$
by *blast*
{ fix *e*::*real* **assume** *e*: $e > 0$
then obtain N **where** N: $\forall n{\geq}N.\ \forall x.\ x \in S \longrightarrow cmod\ (f'\ n\ x - g'\ x) \leq e$
by (*metis conv*)
have $\exists N.\ \forall n{\geq}N.\ \forall x{\in}S.\ \forall h.\ cmod\ (f'\ n\ x * h - g'\ x * h) \leq e * cmod\ h$
proof (*rule exI* [*of* _ *N*], *clarify*)
fix *n y h*
assume $N \leq n\ y \in S$
then have $cmod\ (f'\ n\ y - g'\ y) \leq e$
by (*metis N*)
then have $cmod\ h * cmod\ (f'\ n\ y - g'\ y) \leq cmod\ h * e$
by (*auto simp*: *antisym_conv2 mult_le_cancel_left norm_triangle_ineq2*)
then show $cmod\ (f'\ n\ y * h - g'\ y * h) \leq e * cmod\ h$
by (*simp add*: *norm_mult* [*symmetric*] *field_simps*)
qed
} note $** = this$
show *?thesis*
unfolding *has_field_derivative_def*
proof (*rule has_derivative_sequence* [*OF cvs* _ _ *x*])
show $(\lambda n.\ f\ n\ x) \longrightarrow l$
by (*rule tf*)
next show $\bigwedge e.\ e > 0 \Longrightarrow \forall_F\ n\ in\ sequentially.\ \forall x{\in}S.\ \forall h.\ cmod\ (f'\ n\ x * h - g'\ x * h) \leq e * cmod\ h$
unfolding *eventually_sequentially* **by** (*blast intro*: $**$)
qed (*metis has_field_derivative_def df*)
qed

lemma *has_complex_derivative_series*:

fixes S :: *complex set*
assumes *cvs*: *convex* S
and *df*: $\bigwedge n\ x.\ x \in S \Longrightarrow (f\ n\ has\_field\_derivative\ f'\ n\ x)\ (at\ x\ within\ S)$
and *conv*: $\bigwedge e.\ 0 < e \Longrightarrow \exists N.\ \forall n\ x.\ n \geq N \longrightarrow x \in S \longrightarrow cmod\ ((\sum i{<}n.\ f'\ i\ x) - g'\ x) \leq e$
and $\exists x\ l.\ x \in S \wedge ((\lambda n.\ f\ n\ x)\ sums\ l)$
shows $\exists g.\ \forall x \in S.\ ((\lambda n.\ f\ n\ x)\ sums\ g\ x) \wedge ((g\ has\_field\_derivative\ g'\ x)\ (at\ x\ within\ S))$
proof −
from *assms* **obtain** $x\ l$ **where** x: $x \in S$ **and** *sf*: $((\lambda n.\ f\ n\ x)\ sums\ l)$
by *blast*
{ fix *e*::*real* **assume** *e*: $e > 0$
then obtain N **where** N: $\forall n\ x.\ n \geq N \longrightarrow x \in S \longrightarrow cmod\ ((\sum i{<}n.\ f'\ i\ x) - g'\ x) \leq e$
by (*metis conv*)
have $\exists N.\ \forall n{\geq}N.\ \forall x{\in}S.\ \forall h.\ cmod\ ((\sum i{<}n.\ h * f'\ i\ x) - g'\ x * h) \leq e * cmod\ h$
proof (*rule exI* [*of* _ N], *clarify*)
fix $n\ y\ h$
assume $N \leq n\ y \in S$
then have $cmod\ ((\sum i{<}n.\ f'\ i\ y) - g'\ y) \leq e$
by (*metis* N)
then have $cmod\ h * cmod\ ((\sum i{<}n.\ f'\ i\ y) - g'\ y) \leq cmod\ h * e$
by (*auto simp*: *antisym_conv2 mult_le_cancel_left norm_triangle_ineq2*)
then show $cmod\ ((\sum i{<}n.\ h * f'\ i\ y) - g'\ y * h) \leq e * cmod\ h$
by (*simp add*: *norm_mult* [*symmetric*] *field_simps sum_distrib_left*)
qed
} note $** = this$
show *?thesis*
unfolding *has_field_derivative_def*
proof (*rule has_derivative_series* [*OF cvs* _ _ x])
fix $n\ x$
assume $x \in S$
then show $((f\ n)\ has\_derivative\ (\lambda z.\ z * f'\ n\ x))\ (at\ x\ within\ S)$
by (*metis df has_field_derivative_def mult_commute_abs*)
next show $((\lambda n.\ f\ n\ x)\ sums\ l)$
by (*rule sf*)
next show $\bigwedge e.\ e{>}0 \Longrightarrow \forall_F\ n\ in\ sequentially.\ \forall x{\in}S.\ \forall h.\ cmod\ ((\sum i{<}n.\ h * f'\ i\ x) - g'\ x * h) \leq e * cmod\ h$
unfolding *eventually_sequentially* **by** (*blast intro*: $**$)
qed
qed

6.17.7 Taylor on Complex Numbers

lemma *sum_Suc_reindex*:
fixes f :: $nat \Rightarrow {'a}$::*ab_group_add*
shows $sum\ f\ \{0..n\} = f\ 0 - f\ (Suc\ n) + sum\ (\lambda i.\ f\ (Suc\ i))\ \{0..n\}$
by (*induct* n) *auto*

```
lemma field_Taylor:
  assumes S: convex S
      and f: ⋀i x. x ∈ S ⟹ i ≤ n ⟹ (f i has_field_derivative f (Suc i) x) (at
x within S)
      and B: ⋀x. x ∈ S ⟹ norm (f (Suc n) x) ≤ B
      and w: w ∈ S
      and z: z ∈ S
    shows norm(f 0 z − (∑ i≤n. f i w * (z−w) ^ i / (fact i)))
          ≤ B * norm(z − w)^(Suc n) / fact n
proof −
  have wzs: closed_segment w z ⊆ S using assms
    by (metis convex_contains_segment)
  { fix u
    assume u ∈ closed_segment w z
    then have u ∈ S
      by (metis wzs subsetD)
    have (∑ i≤n. f i u * (− of_nat i * (z−u)^(i − 1)) / (fact i) +
                    f (Suc i) u * (z−u)^i / (fact i)) =
          f (Suc n) u * (z−u) ^ n / (fact n)
    proof (induction n)
      case 0 show ?case by simp
    next
      case (Suc n)
      have (∑ i≤Suc n. f i u * (− of_nat i * (z−u) ^ (i − 1)) / (fact i) +
                          f (Suc i) u * (z−u) ^ i / (fact i)) =
          f (Suc n) u * (z−u) ^ n / (fact n) +
          f (Suc (Suc n)) u * ((z−u) * (z−u) ^ n) / (fact (Suc n)) −
          f (Suc n) u * ((1 + of_nat n) * (z−u) ^ n) / (fact (Suc n))
        using Suc by simp
      also have ... = f (Suc (Suc n)) u * (z−u) ^ Suc n / (fact (Suc n))
      proof −
        have (fact(Suc n)) *
             (f(Suc n) u *(z−u) ^ n / (fact n) +
               f(Suc(Suc n)) u *((z−u) *(z−u) ^ n) / (fact(Suc n)) −
               f(Suc n) u *((1 + of_nat n) *(z−u) ^ n) / (fact(Suc n))) =
            ((fact(Suc n)) *(f(Suc n) u *(z−u) ^ n)) / (fact n) +
            ((fact(Suc n)) *(f(Suc(Suc n)) u *((z−u) *(z−u) ^ n)) / (fact(Suc n)))
−
            ((fact(Suc n)) *(f(Suc n) u *(of_nat(Suc n) *(z−u) ^ n))) / (fact(Suc
n))
          by (simp add: algebra_simps del: fact_Suc)
        also have ... = ((fact (Suc n)) * (f (Suc n) u * (z−u) ^ n)) / (fact n) +
                        (f (Suc (Suc n)) u * ((z−u) * (z−u) ^ n)) −
                        (f (Suc n) u * ((1 + of_nat n) * (z−u) ^ n))
          by (simp del: fact_Suc)
        also have ... = (of_nat (Suc n) * (f (Suc n) u * (z−u) ^ n)) +
                        (f (Suc (Suc n)) u * ((z−u) * (z−u) ^ n)) −
                        (f (Suc n) u * ((1 + of_nat n) * (z−u) ^ n))
```

```
        by (simp only: fact_Suc of_nat_mult ac_simps) simp
      also have ... = f (Suc (Suc n)) u * ((z-u) * (z-u) ^ n)
        by (simp add: algebra_simps)
      finally show ?thesis
       by (simp add: mult_left_cancel [where c = (fact (Suc n)), THEN iffD1]
del: fact_Suc)
    qed
    finally show ?case .
  qed
  then have ((λv. (∑ i≤n. f i v * (z - v)^i / (fact i)))
           has_field_derivative f (Suc n) u * (z-u) ^ n / (fact n))
          (at u within S)
    apply (intro derivative_eq_intros)
    apply (blast intro: assms ‹u ∈ S›)
    apply (rule refl)+
    apply (auto simp: field_simps)
    done
 } note sum_deriv = this
 { fix u
  assume u: u ∈ closed_segment w z
  then have us: u ∈ S
    by (metis wzs subsetD)
   have norm (f (Suc n) u) * norm (z - u) ^ n ≤ norm (f (Suc n) u) * norm
(u - z) ^ n
    by (metis norm_minus_commute order_refl)
  also have ... ≤ norm (f (Suc n) u) * norm (z - w) ^ n
    by (metis mult_left_mono norm_ge_zero power_mono segment_bound [OF
u])
  also have ... ≤ B * norm (z - w) ^ n
    by (metis norm_ge_zero zero_le_power mult_right_mono  B [OF us])
  finally have norm (f (Suc n) u) * norm (z - u) ^ n ≤ B * norm (z - w) ^
n .
 } note cmod_bound = this
 have (∑ i≤n. f i z * (z - z) ^ i / (fact i)) = (∑ i≤n. (f i z / (fact i)) * 0 ^ i)
   by simp
 also have ... = f 0 z / (fact 0)
   by (subst sum_zero_power) simp
 finally have norm (f 0 z - (∑ i≤n. f i w * (z - w) ^ i / (fact i)))
           ≤ norm ((∑ i≤n. f i w * (z - w) ^ i / (fact i)) -
                 (∑ i≤n. f i z * (z - z) ^ i / (fact i)))
   by (simp add: norm_minus_commute)
 also have ... ≤ B * norm (z - w) ^ n / (fact n) * norm (w - z)
   apply (rule field_differentiable_bound
     [where f' = λw. f (Suc n) w * (z - w)^n / (fact n)
        and S = closed_segment w z, OF convex_closed_segment])
   apply (auto simp: DERIV_subset [OF sum_deriv wzs]
           norm_divide norm_mult norm_power divide_le_cancel cmod_bound)
   done
 also have ... ≤ B * norm (z - w) ^ Suc n / (fact n)
```

```
    by (simp add: algebra_simps norm_minus_commute)
  finally show ?thesis .
qed

lemma complex_Taylor:
  assumes S: convex S
      and f: ⋀i x. x ∈ S ⟹ i ≤ n ⟹ (f i has_field_derivative f (Suc i) x) (at
x within S)
      and B: ⋀x. x ∈ S ⟹ cmod (f (Suc n) x) ≤ B
      and w: w ∈ S
      and z: z ∈ S
    shows cmod(f 0 z − (∑ i≤n. f i w * (z−w) ^ i / (fact i)))
        ≤ B * cmod(z − w)^(Suc n) / fact n
  using assms by (rule field_Taylor)
```

Something more like the traditional MVT for real components

```
lemma complex_mvt_line:
  assumes ⋀u. u ∈ closed_segment w z ⟹ (f has_field_derivative f'(u)) (at u)
    shows ∃ u. u ∈ closed_segment w z ∧ Re(f z) − Re(f w) = Re(f'(u) * (z − w))
proof −
  have twz: ⋀t. (1 − t) *R w + t *R z = w + t *R (z − w)
   by (simp add: real_vector.scale_left_diff_distrib real_vector.scale_right_diff_distrib)
  note assms[unfolded has_field_derivative_def, derivative_intros]
  show ?thesis
    apply (cut_tac mvt_simple
                  [of 0 1 Re o f o (λt. (1 − t) *R w +  t *R z)
                   λu. Re o (λh. f'((1 − u) *R w + u *R z) * h) o (λt. t *R (z −
w))])
    apply auto
    apply (rule_tac x=(1 − x) *R w + x *R z in exI)
    apply (auto simp: closed_segment_def twz) []
    apply (intro derivative_eq_intros has_derivative_at_withinI, simp_all)
    apply (simp add: fun_eq_iff real_vector.scale_right_diff_distrib)
    apply (force simp: twz closed_segment_def)
    done
qed

lemma complex_Taylor_mvt:
  assumes ⋀i x. ⟦x ∈ closed_segment w z; i ≤ n⟧ ⟹ ((f i) has_field_derivative
f (Suc i) x) (at x)
    shows ∃ u. u ∈ closed_segment w z ∧
          Re (f 0 z) =
          Re ((∑ i = 0..n. f i w * (z − w) ^ i / (fact i)) +
              (f (Suc n) u * (z−u)^n / (fact n)) * (z − w))
proof −
  { fix u
    assume u: u ∈ closed_segment w z
    have (∑ i = 0..n.
            (f (Suc i) u * (z−u) ^ i − of_nat i * (f i u * (z−u) ^ (i − Suc 0))) /
```

```
            (fact i)) =
        f (Suc 0) u -
          (f (Suc (Suc n)) u * ((z-u) ^ Suc n) - (of_nat (Suc n)) * (z-u) ^ n
* f (Suc n) u) /
          (fact (Suc n)) +
          (∑ i = 0..n.
            (f (Suc (Suc i)) u * ((z-u) ^ Suc i) - of_nat (Suc i) * (f (Suc i)
u * (z-u) ^ i)) /
            (fact (Suc i)))
      by (subst sum_Suc_reindex) simp
    also have ... = f (Suc 0) u -
          (f (Suc (Suc n)) u * ((z-u) ^ Suc n) - (of_nat (Suc n)) * (z-u) ^ n
* f (Suc n) u) /
          (fact (Suc n)) +
          (∑ i = 0..n.
            f (Suc (Suc i)) u * ((z-u) ^ Suc i) / (fact (Suc i)) -
            f (Suc i) u * (z-u) ^ i / (fact i))
      by (simp only: diff_divide_distrib fact_cancel ac_simps)
    also have ... = f (Suc 0) u -
          (f (Suc (Suc n)) u * (z-u) ^ Suc n - of_nat (Suc n) * (z-u) ^ n * f
(Suc n) u) /
          (fact (Suc n)) +
          f (Suc (Suc n)) u * (z-u) ^ Suc n / (fact (Suc n)) - f (Suc 0) u
      by (subst sum_Suc_diff) auto
    also have ... = f (Suc n) u * (z-u) ^ n / (fact n)
      by (simp only: algebra_simps diff_divide_distrib fact_cancel)
    finally have (∑ i = 0..n. (f (Suc i) u * (z - u) ^ i
                        - of_nat i * (f i u * (z-u) ^ (i - Suc 0))) / (fact i)) =
              f (Suc n) u * (z - u) ^ n / (fact n) .
   then have ((λu. ∑ i = 0..n. f i u * (z - u) ^ i / (fact i)) has_field_derivative
            f (Suc n) u * (z - u) ^ n / (fact n)) (at u)
      apply (intro derivative_eq_intros)+
      apply (force intro: u assms)
      apply (rule refl)+
      apply (auto simp: ac_simps)
      done
  }
  then show ?thesis
    apply (cut_tac complex_mvt_line [of w z λu. ∑ i = 0..n. f i u * (z-u) ^ i /
(fact i)
            λu. (f (Suc n) u * (z-u)^n / (fact n))])
    apply (auto simp add: intro: open_closed_segment)
    done
qed


end
```

6.18 Complex Transcendental Functions

By John Harrison et al. Ported from HOL Light by L C Paulson (2015)

theory *Complex_Transcendental*
imports
Complex_Analysis_Basics Summation_Tests HOL−Library.Periodic_Fun
begin

6.18.1 Möbius transformations

definition *moebius a b c d* ≡ (λ*z*. (*a*∗*z*+*b*) / (*c*∗*z*+*d* :: *'a* :: *field*))

theorem *moebius_inverse*:
assumes *a* ∗ *d* ≠ *b* ∗ *c c* ∗ *z* + *d* ≠ *0*
shows *moebius d* (−*b*) (−*c*) *a* (*moebius a b c d z*) = *z*
proof −
from *assms* **have** (−*c*) ∗ *moebius a b c d z* + *a* ≠ *0* **unfolding** *moebius_def*
by (*simp add*: *field_simps*)
with *assms* **show** *?thesis*
unfolding *moebius_def* **by** (*simp add*: *moebius_def divide_simps*) (*simp add*: *algebra_simps*)*?*
qed

lemma *moebius_inverse'*:
assumes *a* ∗ *d* ≠ *b* ∗ *c c* ∗ *z* − *a* ≠ *0*
shows *moebius a b c d* (*moebius d* (−*b*) (−*c*) *a z*) = *z*
using *assms moebius_inverse*[*of d a* −*b* −*c z*]
by (*auto simp*: *algebra_simps*)

lemma *cmod_add_real_less*:
assumes *Im z* ≠ *0 r*≠*0*
shows *cmod* (*z* + *r*) < *cmod z* + |*r*|
proof (*cases z*)
case (*Complex x y*)
then have *0* < *y* ∗ *y*
using *assms mult_neg_neg* **by** *force*
with *assms* **have** *r* ∗ *x* / |*r*| < *sqrt* (*x*∗*x* + *y*∗*y*)
by (*simp add*: *real_less_rsqrt power2_eq_square*)
then show *?thesis* **using** *assms Complex*
apply (*simp add*: *cmod_def*)
apply (*rule power2_less_imp_less*, *auto*)
apply (*simp add*: *power2_eq_square field_simps*)
done
qed

lemma *cmod_diff_real_less*: *Im z* ≠ *0* ⟹ *x*≠*0* ⟹ *cmod* (*z* − *x*) < *cmod z* + |*x*|
using *cmod_add_real_less* [*of z* −*x*]
by *simp*

lemma *cmod_square_less_1_plus*:
assumes $Im\ z = 0 \Longrightarrow |Re\ z| < 1$
shows $(cmod\ z)^2 < 1 + cmod\ (1 - z^2)$
proof (*cases* $Im\ z = 0 \vee Re\ z = 0$)
case *True*
with *assms abs_square_less_1* **show** *?thesis*
by (*force simp add*: *Re_power2 Im_power2 cmod_def*)
next
case *False*
with *cmod_diff_real_less* [*of* $1 - z^2$ *1*] **show** *?thesis*
by (*simp add*: *norm_power Im_power2*)
qed

6.18.2 The Exponential Function

lemma *norm_exp_i_times* [*simp*]: $norm\ (exp(\mathrm{i} * of\_real\ y)) = 1$
by *simp*

lemma *norm_exp_imaginary*: $norm(exp\ z) = 1 \Longrightarrow Re\ z = 0$
by *simp*

lemma *field_differentiable_within_exp*: *exp field_differentiable* (*at z within s*)
using *DERIV_exp field_differentiable_at_within field_differentiable_def* **by** *blast*

lemma *continuous_within_exp*:
fixes *z*::'*a*::{*real_normed_field*,*banach*}
shows *continuous* (*at z within s*) *exp*
by (*simp add*: *continuous_at_imp_continuous_within*)

lemma *holomorphic_on_exp* [*holomorphic_intros*]: *exp holomorphic_on s*
by (*simp add*: *field_differentiable_within_exp holomorphic_on_def*)

lemma *holomorphic_on_exp'* [*holomorphic_intros*]:
f holomorphic_on s $\Longrightarrow$ $(\lambda x.\ exp\ (f\ x))$ *holomorphic_on s*
using *holomorphic_on_compose*[*OF* _ *holomorphic_on_exp*] **by** (*simp add*: *o_def*)

lemma *exp_analytic_on* [*analytic_intros*]:
assumes *f analytic_on A*
shows $(\lambda z.\ exp\ (f\ z))$ *analytic_on A*
by (*metis analytic_on_holomorphic assms holomorphic_on_exp'*)

lemma
assumes $\bigwedge w.\ w \in A \Longrightarrow exp\ (f\ w) = w$
assumes *f holomorphic_on A z* $\in$ *A open A*
shows *deriv_complex_logarithm*: *deriv f z = 1 / z*
and *has_field_derivative_complex_logarithm*: (*f has_field_derivative 1 / z*)

(*at z*)
proof −
have [*simp*]: *z* ≠ *0*
using *assms*(*1*)[*of z*] *assms*(*3*) **by** *auto*
have *deriv* [*derivative_intros*]: (*f has_field_derivative deriv f z*) (*at z*)
using *assms holomorphic_derivI* **by** *blast*
have ((λ*w*. *w*) *has_field_derivative 1*) (*at z*)
by (*intro derivative_intros*)
also have *?this* ⟷ ((λ*w*. *exp* (*f w*)) *has_field_derivative 1*) (*at z*)
by (*smt* (*verit*, *best*) *assms has_field_derivative_transform_within_open*)
finally have ((λ*w*. *exp* (*f w*)) *has_field_derivative 1*) (*at z*) **.**
moreover have ((λ*w*. *exp* (*f w*)) *has_field_derivative exp* (*f z*) ∗ *deriv f z*) (*at z*)
by (*rule derivative_eq_intros refl*)+
ultimately have *exp* (*f z*) ∗ *deriv f z* = *1*
using *DERIV_unique* **by** *blast*
with *assms* **show** *deriv f z* = *1* / *z*
by (*simp add*: *field_simps*)
with *deriv* **show** (*f has_field_derivative 1* / *z*) (*at z*)
by *simp*
qed

6.18.3 Euler and de Moivre formulas

The sine series times *i*

lemma *sin_i_eq*: (λ*n*. (i ∗ *sin_coeff n*) ∗ *z*^*n*) *sums* (i ∗ *sin z*)
proof −
have (λ*n*. i ∗ *sin_coeff n* ∗$_R$ *z*^*n*) *sums* (i ∗ *sin z*)
using *sin_converges sums_mult* **by** *blast*
then show *?thesis*
by (*simp add*: *scaleR_conv_of_real field_simps*)
qed

theorem *exp_Euler*: *exp*(i ∗ *z*) = *cos*(*z*) + i ∗ *sin*(*z*)
proof −
have (λ*n*. (*cos_coeff n* + i ∗ *sin_coeff n*) ∗ *z*^*n*) = (λ*n*. (i ∗ *z*) ^ *n* /$_R$ (*fact n*))
by (*force simp*: *cos_coeff_def sin_coeff_def scaleR_conv_of_real field_simps elim*!: *evenE oddE*)
also have . . . *sums* (*exp* (i ∗ *z*))
by (*rule exp_converges*)
finally have (λ*n*. (*cos_coeff n* + i ∗ *sin_coeff n*) ∗ *z*^*n*) *sums* (*exp* (i ∗ *z*)) **.**
moreover have (λ*n*. (*cos_coeff n* + i ∗ *sin_coeff n*) ∗ *z*^*n*) *sums* (*cos z* + i ∗ *sin z*)
using *sums_add* [*OF cos_converges* [*of z*] *sin_i_eq* [*of z*]]
by (*simp add*: *field_simps scaleR_conv_of_real*)
ultimately show *?thesis*
using *sums_unique2* **by** *blast*
qed

corollary *exp_minus_Euler*: $exp(-(\mathrm{i} * z)) = cos(z) - \mathrm{i} * sin(z)$
using *exp_Euler* [*of* $-z$] **by** *simp*

lemma *sin_exp_eq*: $sin\ z = (exp(\mathrm{i} * z) - exp(-(\mathrm{i} * z))) / (2{*}\mathrm{i})$
by (*simp add*: *exp_Euler exp_minus_Euler*)

lemma *sin_exp_eq'*: $sin\ z = \mathrm{i} * (exp(-(\mathrm{i} * z)) - exp(\mathrm{i} * z)) / 2$
by (*simp add*: *exp_Euler exp_minus_Euler*)

lemma *cos_exp_eq*: $cos\ z = (exp(\mathrm{i} * z) + exp(-(\mathrm{i} * z))) / 2$
by (*simp add*: *exp_Euler exp_minus_Euler*)

theorem *Euler*: $exp(z) = of\_real(exp(Re\ z)) *$
$(of\_real(cos(Im\ z)) + \mathrm{i} * of\_real(sin(Im\ z)))$
by (*simp add*: *Complex_eq cis.code exp_eq_polar*)

lemma *Re_sin*: $Re(sin\ z) = sin(Re\ z) * (exp(Im\ z) + exp(-(Im\ z))) / 2$
by (*simp add*: *sin_exp_eq field_simps Re_divide Im_exp*)

lemma *Im_sin*: $Im(sin\ z) = cos(Re\ z) * (exp(Im\ z) - exp(-(Im\ z))) / 2$
by (*simp add*: *sin_exp_eq field_simps Im_divide Re_exp*)

lemma *Re_cos*: $Re(cos\ z) = cos(Re\ z) * (exp(Im\ z) + exp(-(Im\ z))) / 2$
by (*simp add*: *cos_exp_eq field_simps Re_divide Re_exp*)

lemma *Im_cos*: $Im(cos\ z) = sin(Re\ z) * (exp(-(Im\ z)) - exp(Im\ z)) / 2$
by (*simp add*: *cos_exp_eq field_simps Im_divide Im_exp*)

lemma *Re_sin_pos*: $0 < Re\ z \Longrightarrow Re\ z < pi \Longrightarrow Re\ (sin\ z) > 0$
by (*auto simp*: *Re_sin Im_sin add_pos_pos sin_gt_zero*)

lemma *Im_sin_nonneg*: $Re\ z = 0 \Longrightarrow 0 \leq Im\ z \Longrightarrow 0 \leq Im\ (sin\ z)$
by (*simp add*: *Re_sin Im_sin algebra_simps*)

lemma *Im_sin_nonneg2*: $Re\ z = pi \Longrightarrow Im\ z \leq 0 \Longrightarrow 0 \leq Im\ (sin\ z)$
by (*simp add*: *Re_sin Im_sin algebra_simps*)

6.18.4 Relationships between real and complex trigonometric and hyperbolic functions

lemma *real_sin_eq* [*simp*]: $Re(sin(of\_real\ x)) = sin\ x$
by (*simp add*: *sin_of_real*)

lemma *real_cos_eq* [*simp*]: $Re(cos(of\_real\ x)) = cos\ x$
by (*simp add*: *cos_of_real*)

lemma *DeMoivre*: $(cos\ z + \mathrm{i} * sin\ z)\ \hat{}\ n = cos(n * z) + \mathrm{i} * sin(n * z)$
by (*metis exp_Euler* [*symmetric*] *exp_of_nat_mult mult.left_commute*)

lemma *exp_cnj*: *cnj* (*exp z*) = *exp* (*cnj z*)
by (*simp add*: *cis_cnj exp_eq_polar*)

lemma *cnj_sin*: *cnj*(*sin z*) = *sin*(*cnj z*)
by (*simp add*: *sin_exp_eq exp_cnj field_simps*)

lemma *cnj_cos*: *cnj*(*cos z*) = *cos*(*cnj z*)
by (*simp add*: *cos_exp_eq exp_cnj field_simps*)

lemma *field_differentiable_at_sin*: *sin field_differentiable at z*
using *DERIV_sin field_differentiable_def* **by** *blast*

lemma *field_differentiable_within_sin*: *sin field_differentiable* (*at z within S*)
by (*simp add*: *field_differentiable_at_sin field_differentiable_at_within*)

lemma *field_differentiable_at_cos*: *cos field_differentiable at z*
using *DERIV_cos field_differentiable_def* **by** *blast*

lemma *field_differentiable_within_cos*: *cos field_differentiable* (*at z within S*)
by (*simp add*: *field_differentiable_at_cos field_differentiable_at_within*)

lemma *holomorphic_on_sin*: *sin holomorphic_on S*
by (*simp add*: *field_differentiable_within_sin holomorphic_on_def*)

lemma *holomorphic_on_cos*: *cos holomorphic_on S*
by (*simp add*: *field_differentiable_within_cos holomorphic_on_def*)

lemma *holomorphic_on_sin'* [*holomorphic_intros*]:
assumes *f holomorphic_on A*
shows (*λx. sin* (*f x*)) *holomorphic_on A*
using *holomorphic_on_compose*[*OF assms holomorphic_on_sin*] **by** (*simp add*: *o_def*)

lemma *holomorphic_on_cos'* [*holomorphic_intros*]:
assumes *f holomorphic_on A*
shows (*λx. cos* (*f x*)) *holomorphic_on A*
using *holomorphic_on_compose*[*OF assms holomorphic_on_cos*] **by** (*simp add*: *o_def*)

lemma *analytic_on_sin* [*analytic_intros*]: *sin analytic_on A*
using *analytic_on_holomorphic holomorphic_on_sin* **by** *blast*

lemma *analytic_on_sin'* [*analytic_intros*]:
f analytic_on A ⟹ (⋀*z*. *z* ∈ *A* ⟹ *f z* ∉ *range* (*λn. complex_of_real pi* ∗ *of_int n*)) ⟹
(*λz. sin* (*f z*)) *analytic_on A*
using *analytic_on_compose_gen*[*OF _ analytic_on_sin*[*of UNIV*], *of f A*] **by** (*simp add*: *o_def*)

lemma *analytic_on_cos* [*analytic_intros*]: *cos analytic_on A*
using *analytic_on_holomorphic holomorphic_on_cos* **by** *blast*

lemma *analytic_on_cos'* [*analytic_intros*]:
f analytic_on A $\Longrightarrow$ ($\bigwedge$*z. z* $\in$ *A* $\Longrightarrow$ *f z* $\notin$ *range* (λ*n. complex_of_real pi* $*$ *of_int n*)) $\Longrightarrow$
(λ*z. cos* (*f z*)) *analytic_on A*
using *analytic_on_compose_gen*[*OF _ analytic_on_cos*[*of UNIV*], *of f A*] **by** (*simp add*: *o_def*)

6.18.5 More on the Polar Representation of Complex Numbers

lemma *exp_Complex*: *exp*(*Complex r t*) = *of_real*(*exp r*) $*$ *Complex* (*cos t*) (*sin t*)
using *Complex_eq Euler complex.sel* **by** *presburger*

lemma *exp_eq_1*: *exp z* = *1* $\longleftrightarrow$ *Re*(*z*) = *0* $\wedge$ ($\exists$ *n*::*int*. *Im*(*z*) = *of_int* (*2* $*$ *n*) $*$ *pi*)
(**is** *?lhs* = *?rhs*)
proof
assume *exp z* = *1*
then have *Re z* = *0*
by (*metis exp_eq_one_iff norm_exp_eq_Re norm_one*)
with ‹*?lhs*› **show** *?rhs*
by (*metis Re_exp cos_one_2pi_int exp_zero mult.commute mult_1 of_int_mult of_int_numeral one_complex.simps*(*1*))
next
assume *?rhs* **then show** *?lhs*
using *Im_exp Re_exp complex_eq_iff*
by (*simp add*: *cos_one_2pi_int cos_one_sin_zero mult.commute*)
qed

lemma *exp_eq*: *exp w* = *exp z* $\longleftrightarrow$ ($\exists$ *n*::*int*. *w* = *z* + (*of_int* (*2* $*$ *n*) $*$ *pi*) $*$ i)
(**is** *?lhs* = *?rhs*)
proof −
have *exp w* = *exp z* $\longleftrightarrow$ *exp* (*w*−*z*) = *1*
by (*simp add*: *exp_diff*)
also have ... $\longleftrightarrow$ (*Re w* = *Re z* $\wedge$ ($\exists$ *n*::*int*. *Im w* − *Im z* = *of_int* (*2* $*$ *n*) $*$ *pi*))
by (*simp add*: *exp_eq_1*)
also have ... $\longleftrightarrow$ *?rhs*
by (*auto simp*: *algebra_simps intro*!: *complex_eqI*)
finally show *?thesis* **.**
qed

lemma *exp_complex_eqI*: |*Im w* − *Im z*| < *2*$*$*pi* $\Longrightarrow$ *exp w* = *exp z* $\Longrightarrow$ *w* = *z*
by (*auto simp*: *exp_eq abs_mult*)

lemma *exp_integer_2pi*:
assumes $n \in \mathbb{Z}$
shows $exp((2 * n * pi) * \mathrm{i}) = 1$
by (*metis assms cis_conv_exp cis_multiple_2pi mult.assoc mult.commute*)

lemma *exp_plus_2pin* [*simp*]: $exp\ (z + \mathrm{i} * (of\_int\ n * (of\_real\ pi * 2))) = exp\ z$
by (*simp add*: *exp_eq*)

lemma *exp_integer_2pi_plus1*:
assumes $n \in \mathbb{Z}$
shows $exp(((2 * n + 1) * pi) * \mathrm{i}) = - 1$
using *exp_integer_2pi* [*OF assms*]
by (*metis cis_conv_exp cis_mult cis_pi distrib_left mult.commute mult.right_neutral*)

lemma *inj_on_exp_pi*:
fixes *z*::*complex* **shows** *inj_on exp* (*ball z pi*)
proof (*clarsimp simp*: *inj_on_def exp_eq*)
fix *y n*
assume *dist z* $(y + 2 * of\_int\ n * of\_real\ pi * \mathrm{i}) < pi$
dist z y $< pi$
then have *dist y* $(y + 2 * of\_int\ n * of\_real\ pi * \mathrm{i}) < pi+pi$
using *dist_commute_lessI dist_triangle_less_add* **by** *blast*
then have *norm* $(2 * of\_int\ n * of\_real\ pi * \mathrm{i}) < 2*pi$
by (*simp add*: *dist_norm*)
then show $n = 0$
by (*auto simp*: *norm_mult*)
qed

lemma *cmod_add_squared*:
fixes *r1 r2*::*real*
shows $(cmod\ (r1 * exp\ (\mathrm{i} * \vartheta 1) + r2 * exp\ (\mathrm{i} * \vartheta 2)))^2 = r1^2 + r2^2 + 2 * r1 * r2 * cos\ (\vartheta 1 - \vartheta 2)$ (**is** $(cmod\ (?z1 + ?z2))^2 = ?rhs$)
proof −
have $(cmod\ (?z1 + ?z2))^2 = (?z1 + ?z2) * cnj\ (?z1 + ?z2)$
by (*rule complex_norm_square*)
also have $\ldots = (?z1 * cnj\ ?z1 + ?z2 * cnj\ ?z2) + (?z1 * cnj\ ?z2 + cnj\ ?z1 * ?z2)$
by (*simp add*: *algebra_simps*)
also have $\ldots = (norm\ ?z1)^2 + (norm\ ?z2)^2 + 2 * Re\ (?z1 * cnj\ ?z2)$
unfolding *complex_norm_square* [*symmetric*] *cnj_add_mult_eq_Re* **by** *simp*
also have $\ldots = ?rhs$
by (*simp add*: *norm_mult*) (*simp add*: *exp_Euler complex_is_Real_iff* [*THEN iffD1*] *cos_diff algebra_simps*)
finally show *?thesis*
using *of_real_eq_iff* **by** *blast*
qed

lemma *cmod_diff_squared*:
fixes *r1 r2*::*real*

shows $(cmod\ (r1 * exp\ (\mathrm{i} * \vartheta 1) - r2 * exp\ (\mathrm{i} * \vartheta 2)))^2 = r1^2 + r2^2 - 2*r1*r2*cos\ (\vartheta 1 - \vartheta 2)$
using *cmod_add_squared* [*of r1 _ −r2*] **by** *simp*

lemma *polar_convergence*:
fixes *R*::*real*
assumes $\bigwedge j.\ r\ j > 0\ R > 0$
shows $((\lambda j.\ r\ j * exp\ (\mathrm{i} * \vartheta\ j)) \longrightarrow (R * exp\ (\mathrm{i} * \Theta))) \longleftrightarrow$
$(r \longrightarrow R) \wedge (\exists k.\ (\lambda j.\ \vartheta\ j - of\_int\ (k\ j) * (2 * pi)) \longrightarrow \Theta)$ (**is** $(?z \longrightarrow ?Z) = ?rhs$)
proof
assume *L*: $?z \longrightarrow ?Z$
have *rR*: $r \longrightarrow R$
using *tendsto_norm* [*OF L*] *assms* **by** (*auto simp*: *norm_mult abs_of_pos*)
moreover obtain *k* **where** $(\lambda j.\ \vartheta\ j - of\_int\ (k\ j) * (2 * pi)) \longrightarrow \Theta$
proof −
have $cos\ (\vartheta\ j - \Theta) = ((r\ j)^2 + R^2 - (norm(?z\ j - ?Z))^2) / (2 * R * r\ j)$ **for** *j*
using *assms* **by** (*auto simp*: *cmod_diff_squared less_le*)
moreover have $(\lambda j.\ ((r\ j)^2 + R^2 - (norm(?z\ j - ?Z))^2) / (2 * R * r\ j)) \longrightarrow ((R^2 + R^2 - (norm(?Z - ?Z))^2) / (2 * R * R))$
by (*intro L rR tendsto_intros*) (*use* ‹$R > 0$› **in** *force*)
moreover have $((R^2 + R^2 - (norm(?Z - ?Z))^2) / (2 * R * R)) = 1$
using ‹$R > 0$› **by** (*simp add*: *power2_eq_square field_split_simps*)
ultimately have $(\lambda j.\ cos\ (\vartheta\ j - \Theta)) \longrightarrow 1$
by *auto*
then show *?thesis*
using *that cos_diff_limit_1* **by** *blast*
qed
ultimately show *?rhs*
by *metis*
next
assume *R*: *?rhs*
show $?z \longrightarrow ?Z$
proof (*rule tendsto_mult*)
show $(\lambda x.\ complex\_of\_real\ (r\ x)) \longrightarrow of\_real\ R$
using *R* **by** (*auto simp*: *tendsto_of_real_iff*)
obtain *k* **where** $(\lambda j.\ \vartheta\ j - of\_int\ (k\ j) * (2 * pi)) \longrightarrow \Theta$
using *R* **by** *metis*
then have $(\lambda j.\ complex\_of\_real\ (\vartheta\ j - of\_int\ (k\ j) * (2 * pi))) \longrightarrow of\_real\ \Theta$
using *tendsto_of_real_iff* **by** *force*
then have $(\lambda j.\ exp\ (\mathrm{i} * of\_real\ (\vartheta\ j - of\_int\ (k\ j) * (2 * pi)))) \longrightarrow exp\ (\mathrm{i} * \Theta)$
using *tendsto_mult* [*OF tendsto_const*] *isCont_exp isCont_tendsto_compose* **by** *blast*
moreover have $exp\ (\mathrm{i} * of\_real\ (\vartheta\ j - of\_int\ (k\ j) * (2 * pi))) = exp\ (\mathrm{i} * \vartheta\ j)$ **for** *j*
unfolding *exp_eq*

```
    by (rule_tac x=− k j in exI) (auto simp: algebra_simps)
   ultimately show (λj. exp (i ∗ ϑ j)) ⟶ exp (i ∗ Θ)
    by auto
 qed
qed

lemma sin_cos_eq_iff: sin y = sin x ∧ cos y = cos x ⟷ (∃ n::int. y = x + 2
∗ pi ∗ n) (is ?L=?R)
proof
 assume ?L
 then have cos (y−x) = 1
  using cos_add [of y −x] by simp
 then show ?R
  by (metis cos_one_2pi_int add.commute diff_add_cancel mult.assoc mult.commute)

next
 assume ?R
 then show ?L
  by (auto simp: sin_add cos_add)
qed

lemma exp_i_ne_1:
 assumes 0 < x x < 2∗pi
 shows exp(i ∗ of_real x) ≠ 1
 by (smt (verit) Im_i_times Re_complex_of_real assms exp_complex_eqI exp_zero
zero_complex.sel(2))

lemma sin_eq_0:
 fixes z::complex
 shows sin z = 0 ⟷ (∃ n::int. z = of_real(n ∗ pi))
 by (simp add: sin_exp_eq exp_eq)

lemma cos_eq_0:
 fixes z::complex
 shows cos z = 0 ⟷ (∃ n::int. z = complex_of_real(n ∗ pi) + of_real pi/2)
 using sin_eq_0 [of z − of_real pi/2]
 by (simp add: sin_diff algebra_simps)

lemma cos_eq_1:
 fixes z::complex
 shows cos z = 1 ⟷ (∃ n::int. z = complex_of_real(2 ∗ n ∗ pi))
 by (metis Re_complex_of_real cos_of_real cos_one_2pi_int cos_one_sin_zero
mult.commute of_real_1 sin_eq_0)

lemma csin_eq_1:
 fixes z::complex
 shows sin z = 1 ⟷ (∃ n::int. z = of_real(2 ∗ n ∗ pi) + of_real pi/2)
 using cos_eq_1 [of z − of_real pi/2]
 by (simp add: cos_diff algebra_simps)
```

lemma *csin_eq_minus1*:
fixes *z*::*complex*
shows *sin z* = −*1* ⟷ (∃ *n*::*int*. *z* = *complex_of_real*(*2* ∗ *n* ∗ *pi*) + *3*/*2*∗*pi*)
(**is** _ = *?rhs*)
proof −
have *sin z* = −*1* ⟷ *sin* (−*z*) = *1*
by (*simp add*: *equation_minus_iff*)
also have ... ⟷ (∃ *n*::*int*. *z* = − *of_real*(*2* ∗ *n* ∗ *pi*) − *of_real pi*/*2*)
by (*metis* (*mono_tags*, *lifting*) *add_uminus_conv_diff csin_eq_1 equation_minus_iff minus_add_distrib*)
also have ... = *?rhs*
apply *safe*
apply (*rule_tac* [*2*] *x*=−(*x*+*1*) **in** *exI*)
apply (*rule_tac x*=−(*x*+*1*) **in** *exI*)
apply (*simp_all add*: *algebra_simps*)
done
finally show *?thesis* **.**
qed

lemma *ccos_eq_minus1*:
fixes *z*::*complex*
shows *cos z* = −*1* ⟷ (∃ *n*::*int*. *z* = *complex_of_real*(*2* ∗ *n* ∗ *pi*) + *pi*)
using *csin_eq_1* [*of z* − *of_real pi*/*2*]
by (*simp add*: *sin_diff algebra_simps equation_minus_iff*)

lemma *sin_eq_1*: *sin x* = *1* ⟷ (∃ *n*::*int*. *x* = (*2* ∗ *n* + *1* / *2*) ∗ *pi*)
(**is** _ = *?rhs*)
proof −
have *sin x* = *1* ⟷ *sin* (*complex_of_real x*) = *1*
by (*metis of_real_1 one_complex.simps*(*1*) *real_sin_eq sin_of_real*)
also have ... ⟷ (∃ *n*::*int*. *x* = *of_real*(*2* ∗ *n* ∗ *pi*) + *of_real pi*/*2*)
by (*metis csin_eq_1 Re_complex_of_real id_apply of_real_add of_real_divide of_real_eq_id of_real_numeral*)
also have ... = *?rhs*
by (*auto simp*: *algebra_simps*)
finally show *?thesis* **.**
qed

lemma *sin_eq_minus1*: *sin x* = −*1* ⟷ (∃ *n*::*int*. *x* = (*2*∗*n* + *3*/*2*) ∗ *pi*) (**is** _ = *?rhs*)
proof −
have *sin x* = −*1* ⟷ *sin* (*complex_of_real x*) = −*1*
by (*metis Re_complex_of_real of_real_def scaleR_minus1_left sin_of_real*)
also have ... ⟷ (∃ *n*::*int*. *x* = *of_real*(*2* ∗ *n* ∗ *pi*) + *3*/*2*∗*pi*)
by (*metis Re_complex_of_real csin_eq_minus1 id_apply of_real_add of_real_eq_id*)
also have ... = *?rhs*
by (*auto simp*: *algebra_simps*)
finally show *?thesis* **.**

qed

lemma *cos_eq_minus1*: *cos x = −1 ⟷ (∃ n::int. x = (2∗n + 1) ∗ pi)*
(**is** *_ = ?rhs*)
proof −
have *cos x = −1 ⟷ cos (complex_of_real x) = −1*
by (*metis Re_complex_of_real of_real_def scaleR_minus1_left cos_of_real*)
also have *... ⟷ (∃ n::int. x = of_real(2 ∗ n ∗ pi) + pi)*
by (*metis ccos_eq_minus1 id_apply of_real_add of_real_eq_id of_real_eq_iff*)
also have *... = ?rhs*
by (*auto simp*: *algebra_simps*)
finally show *?thesis* **.**
qed

lemma *cos_gt_neg1*:
assumes (*t::real*) ∈ {*−pi<..<pi*}
shows *cos t > −1*
using *assms*
by *simp* (*metis cos_minus cos_monotone_0_pi cos_monotone_minus_pi_0 cos_pi linorder_le_cases*)

lemma *dist_exp_i_1*: *norm*(*exp*(i ∗ *of_real t*) − *1*) = *2* ∗ |*sin*(*t* / *2*)|
proof −
have *sqrt* (*2* − *cos t* ∗ *2*) = *2* ∗ |*sin* (*t* / *2*)|
using *cos_double_sin* [*of t/2*] **by** (*simp add*: *real_sqrt_mult*)
then show *?thesis*
by (*simp add*: *exp_Euler cmod_def power2_diff sin_of_real cos_of_real algebra_simps*)
qed

lemma *sin_cx_2pi* [*simp*]: ⟦*z = of_int m*; *even m*⟧ ⟹ *sin* (*z* ∗ *complex_of_real pi*) = *0*
by (*simp add*: *sin_eq_0*)

lemma *cos_cx_2pi* [*simp*]: ⟦*z = of_int m*; *even m*⟧ ⟹ *cos* (*z* ∗ *complex_of_real pi*) = *1*
using *cos_eq_1* **by** *auto*

lemma *complex_sin_eq*:
fixes *w* :: *complex*
shows *sin w = sin z ⟷ (∃ n ∈ ℤ. w = z + of_real(2∗n∗pi) ∨ w = −z + of_real((2∗n + 1)∗pi))*
(**is** *?lhs = ?rhs*)
proof
assume *?lhs*
then consider *sin* ((*w* − *z*) / *2*) = *0* | *cos* ((*w* + *z*) / *2*) = *0*
by (*metis divide_eq_0_iff nonzero_eq_divide_eq right_minus_eq sin_diff_sin zero_neq_numeral*)
then show *?rhs*

```
  proof cases
    case 1
    then show ?thesis
    by (simp add: sin_eq_0 algebra_simps) (metis Ints_of_int of_real_of_int_eq)
  next
    case 2
    then show ?thesis
    by (simp add: cos_eq_0 algebra_simps) (metis Ints_of_int of_real_of_int_eq)
  qed
next
  assume ?rhs
  then consider n::int where w = z + of_real (2 * of_int n * pi)
             | n::int where   w = -z + of_real ((2 * of_int n + 1) * pi)
    using Ints_cases by blast
  then show ?lhs
  proof cases
    case 1
    then show ?thesis
      using Periodic_Fun.sin.plus_of_int [of z n]
      by (auto simp: algebra_simps)
  next
    case 2
    then show ?thesis
      using Periodic_Fun.sin.plus_of_int [of -z n]
      apply (simp add: algebra_simps)
    by (metis add.commute add.inverse_inverse add_diff_cancel_left diff_add_cancel
sin_plus_pi)
  qed
qed

lemma complex_cos_eq:
  fixes w :: complex
  shows cos w = cos z ⟷ (∃n ∈ ℤ. w = z + of_real(2*n*pi) ∨ w = -z +
of_real(2*n*pi))
        (is ?lhs = ?rhs)
proof
  assume ?lhs
  then consider sin ((w + z) / 2) = 0 | sin ((z - w) / 2) = 0
    by (metis mult_eq_0_iff cos_diff_cos right_minus_eq zero_neq_numeral)
  then show ?rhs
  proof cases
    case 1
    then obtain n where w + z = of_int n * (complex_of_real pi * 2)
      by (auto simp: sin_eq_0 algebra_simps)
    then have w = -z + of_real(2 * of_int n * pi)
      by (auto simp: algebra_simps)
    then show ?thesis
      using Ints_of_int by blast
  next
```

```
    case 2
    then obtain n where z = w + of_int n * (complex_of_real pi * 2)
      by (auto simp: sin_eq_0 algebra_simps)
    then have w = z + complex_of_real (2 * of_int(−n) * pi)
      by (auto simp: algebra_simps)
    then show ?thesis
      using Ints_of_int by blast
  qed
next
  assume ?rhs
  then obtain n::int where w: w = z + of_real (2* of_int n*pi) ∨
                             w = −z + of_real(2*n*pi)
    using Ints_cases  by (metis of_int_mult of_int_numeral)
  then show ?lhs
    using Periodic_Fun.cos.plus_of_int [of z n]
    apply (simp add: algebra_simps)
    by (metis cos.plus_of_int cos_minus minus_add_cancel mult.commute)
qed

lemma sin_eq:
   sin x = sin y ⟷ (∃ n ∈ ℤ. x = y + 2*n*pi ∨ x = −y + (2*n + 1)*pi)
  using complex_sin_eq [of x y]
 by (simp only: sin_of_real Re_complex_of_real of_real_add [symmetric] of_real_minus
[symmetric] of_real_mult [symmetric] of_real_eq_iff)

lemma cos_eq:
   cos x = cos y ⟷ (∃ n ∈ ℤ. x = y + 2*n*pi ∨ x = −y + 2*n*pi)
  using complex_cos_eq [of x y] unfolding cos_of_real
  by (metis Re_complex_of_real of_real_add of_real_minus)

lemma sinh_complex:
  fixes z :: complex
  shows (exp z − inverse (exp z)) / 2 = −i * sin(i * z)
  by (simp add: sin_exp_eq field_split_simps exp_minus)

lemma sin_i_times:
  fixes z :: complex
  shows sin(i * z) = i * ((exp z − inverse (exp z)) / 2)
  using sinh_complex by auto

lemma sinh_real:
  fixes x :: real
  shows of_real((exp x − inverse (exp x)) / 2) = −i * sin(i * of_real x)
  by (simp add: exp_of_real sin_i_times)

lemma cosh_complex:
  fixes z :: complex
  shows (exp z + inverse (exp z)) / 2 = cos(i * z)
  by (simp add: cos_exp_eq field_split_simps exp_minus exp_of_real)
```

lemma *cosh_real*:
fixes *x* :: *real*
shows *of_real*((*exp x* + *inverse* (*exp x*)) / 2) = *cos*(i ∗ *of_real x*)
by (*simp add*: *cos_exp_eq field_split_simps exp_minus exp_of_real*)

lemmas *cos_i_times* = *cosh_complex* [*symmetric*]

lemma *norm_cos_squared*:
norm(*cos z*) ^ 2 = *cos*(*Re z*) ^ 2 + (*exp*(*Im z*) − *inverse*(*exp*(*Im z*))) ^ 2 / 4
proof (*cases z*)
case (*Complex x1 x2*)
then show *?thesis*
apply (*simp only*: *cos_add cmod_power2 cos_of_real sin_of_real Complex_eq*)
apply (*simp add*: *cos_exp_eq sin_exp_eq exp_minus exp_of_real Re_divide Im_divide power_divide*)
apply (*simp only*: *left_diff_distrib* [*symmetric*] *power_mult_distrib sin_squared_eq*)
apply (*simp add*: *power2_eq_square field_split_simps*)
done
qed

lemma *norm_sin_squared*:
norm(*sin z*) ^ 2 = (*exp*(2 ∗ *Im z*) + *inverse*(*exp*(2 ∗ *Im z*)) − 2 ∗ *cos*(2 ∗ *Re z*)) / 4
using *cos_double_sin* [*of Re z*]
apply (*simp add*: *sin_cos_eq norm_cos_squared exp_minus mult.commute* [*of _ 2*] *exp_double*)
apply (*simp add*: *algebra_simps power2_eq_square*)
done

lemma *exp_uminus_Im*: *exp* (− *Im z*) ≤ *exp* (*cmod z*)
using *abs_Im_le_cmod linear order_trans* **by** *fastforce*

lemma *norm_cos_le*:
fixes *z*::*complex*
shows *norm*(*cos z*) ≤ *exp*(*norm z*)
proof −
have *Im z* ≤ *cmod z*
using *abs_Im_le_cmod abs_le_D1* **by** *auto*
then have *exp* (− *Im z*) + *exp* (*Im z*) ≤ *exp* (*cmod z*) ∗ 2
by (*metis exp_uminus_Im add_mono exp_le_cancel_iff mult_2_right*)
then show *?thesis*
by (*force simp add*: *cos_exp_eq norm_divide intro*: *order_trans* [*OF norm_triangle_ineq*])
qed

lemma *norm_cos_plus1_le*:
fixes *z*::*complex*
shows *norm*(1 + *cos z*) ≤ 2 ∗ *exp*(*norm z*)
by (*metis mult_2 norm_cos_le norm_ge_zero norm_one norm_triangle_mono*

one_le_exp_iff)

lemma *sinh_conv_sin*: *sinh z* = −i ∗ *sin* (i∗*z*)
by (*simp add*: *sinh_field_def sin_i_times exp_minus*)

lemma *cosh_conv_cos*: *cosh z* = *cos* (i∗*z*)
by (*simp add*: *cosh_field_def cos_i_times exp_minus*)

lemma *tanh_conv_tan*: *tanh z* = −i ∗ *tan* (i∗*z*)
by (*simp add*: *tanh_def sinh_conv_sin cosh_conv_cos tan_def*)

lemma *sin_conv_sinh*: *sin z* = −i ∗ *sinh* (i∗*z*)
by (*simp add*: *sinh_conv_sin*)

lemma *cos_conv_cosh*: *cos z* = *cosh* (i∗*z*)
by (*simp add*: *cosh_conv_cos*)

lemma *tan_conv_tanh*: *tan z* = −i ∗ *tanh* (i∗*z*)
by (*simp add*: *tan_def sin_conv_sinh cos_conv_cosh tanh_def*)

lemma *sinh_complex_eq_iff*:
sinh (*z* :: *complex*) = *sinh w* ⟷
(∃ *n*∈ℤ. *z* = *w* − *2* ∗ i ∗ *of_real n* ∗ *of_real pi* ∨
z = −(*2* ∗ *complex_of_real n* + *1*) ∗ i ∗ *complex_of_real pi* − *w*) (**is** _ = *?rhs*)
proof −
have *sinh z* = *sinh w* ⟷ *sin* (i ∗ *z*) = *sin* (i ∗ *w*)
by (*simp add*: *sinh_conv_sin*)
also have ... ⟷ *?rhs*
by (*subst complex_sin_eq*) (*force simp*: *field_simps complex_eq_iff*)
finally show *?thesis* **.**
qed

6.18.6 Taylor series for complex exponential, sine and cosine

declare *power_Suc* [*simp del*]

lemma *Taylor_exp_field*:
fixes *z*::*'a*::{*banach*,*real_normed_field*}
shows *norm* (*exp z* − (∑ *i*≤*n*. *z* ^ *i* / *fact i*)) ≤ *exp* (*norm z*) ∗ (*norm z* ^ *Suc n*) / *fact n*
proof (*rule field_Taylor*[*of* _ *n* λ*k*. *exp exp* (*norm z*) *0 z*, *simplified*])
show *convex* (*closed_segment 0 z*)
by (*rule convex_closed_segment* [*of 0 z*])
next
fix *k x*
assume *x* ∈ *closed_segment 0 z k* ≤ *n*
show (*exp has_field_derivative exp x*) (*at x within closed_segment 0 z*)
using *DERIV_exp DERIV_subset* **by** *blast*

next
fix *x*
assume *x*: *x* ∈ *closed_segment 0 z*
have *norm* (*exp x*) ≤ *exp* (*norm x*)
by (*rule norm_exp*)
also have *norm x* ≤ *norm z*
using *x* **by** (*auto simp*: *closed_segment_def intro*!: *mult_left_le_one_le*)
finally show *norm* (*exp x*) ≤ *exp* (*norm z*)
by *simp*
qed *auto*

For complex *z*, a tighter bound than in the previous result

lemma *Taylor_exp*:
norm(*exp z* − (∑ *k*≤*n*. *z* ^ *k* / (*fact k*))) ≤ *exp*|*Re z*| ∗ (*norm z*) ^ (*Suc n*) / (*fact n*)
proof (*rule complex_Taylor* [*of* _ *n* λ*k*. *exp exp*|*Re z*| *0 z*, *simplified*])
show *convex* (*closed_segment 0 z*)
by (*rule convex_closed_segment* [*of 0 z*])
next
fix *k x*
assume *x* ∈ *closed_segment 0 z k* ≤ *n*
show (*exp has_field_derivative exp x*) (*at x within closed_segment 0 z*)
using *DERIV_exp DERIV_subset* **by** *blast*
next
fix *x*
assume *x* ∈ *closed_segment 0 z*
then obtain *u* **where** *u*: *x* = *complex_of_real u* ∗ *z 0* ≤ *u u* ≤ *1*
by (*auto simp*: *closed_segment_def scaleR_conv_of_real*)
then have *u* ∗ *Re z* ≤ |*Re z*|
by (*metis abs_ge_self abs_ge_zero mult.commute mult.right_neutral mult_mono*)
then show *Re x* ≤ |*Re z*|
by (*simp add*: *u*)
qed *auto*

lemma
assumes *0* ≤ *u u* ≤ *1*
shows *cmod_sin_le_exp*: *cmod* (*sin* (*u* ∗*R z*)) ≤ *exp* |*Im z*|
and *cmod_cos_le_exp*: *cmod* (*cos* (*u* ∗*R z*)) ≤ *exp* |*Im z*|
proof −
have *mono*: ⋀*u w z*::*real*. *w* ≤ *u* ⟹ *z* ≤ *u* ⟹ (*w* + *z*)/*2* ≤ *u*
by *simp*
have ∗: (*cmod* (*exp* (i ∗ (*u* ∗ *z*))) + *cmod* (*exp* (− (i ∗ (*u* ∗ *z*))))) / *2* ≤ *exp* |*Im z*|
proof (*rule mono*)
show *cmod* (*exp* (i ∗ (*u* ∗ *z*))) ≤ *exp* |*Im z*|
using *assms*
by (*auto simp*: *abs_if mult_left_le_one_le not_less intro*: *order_trans* [*of* _ *0*])
show *cmod* (*exp* (− (i ∗ (*u* ∗ *z*)))) ≤ *exp* |*Im z*|

```
    using assms
    by (auto simp: abs_if mult_left_le_one_le mult_nonneg_nonpos intro: order_trans [of _ 0])
  qed
  have cmod (sin (u *R z)) = cmod (exp (i * (u * z)) - exp (- (i * (u * z)))) / 2
    by (auto simp: scaleR_conv_of_real norm_mult norm_power sin_exp_eq norm_divide)
  also have ... ≤ (cmod (exp (i * (u * z))) + cmod (exp (- (i * (u * z))))) / 2
    by (intro divide_right_mono norm_triangle_ineq4) simp
  also have ... ≤ exp |Im z|
    by (rule *)
  finally show cmod (sin (u *R z)) ≤ exp |Im z| .
  have cmod (cos (u *R z)) = cmod (exp (i * (u * z)) + exp (- (i * (u * z)))) / 2
    by (auto simp: scaleR_conv_of_real norm_mult norm_power cos_exp_eq norm_divide)
  also have ... ≤ (cmod (exp (i * (u * z))) + cmod (exp (- (i * (u * z))))) / 2
    by (intro divide_right_mono norm_triangle_ineq) simp
  also have ... ≤ exp |Im z|
    by (rule *)
  finally show cmod (cos (u *R z)) ≤ exp |Im z| .
qed

lemma Taylor_sin:
  norm(sin z - (∑k≤n. complex_of_real (sin_coeff k) * z ^ k))
  ≤ exp|Im z| * (norm z) ^ (Suc n) / (fact n)
proof -
  have mono: ⋀u w z::real. w ≤ u ⟹ z ≤ u ⟹ w + z ≤ u*2
    by arith
  have *: cmod (sin z -
                (∑i≤n. (-1) ^ (i div 2) * (if even i then sin 0 else cos 0) * z ^ i /
(fact i)))
          ≤ exp |Im z| * cmod z ^ Suc n / (fact n)
  proof (rule complex_Taylor [of closed_segment 0 z n
                                λk x. (-1)^(k div 2) * (if even k then sin x else cos x)
                                exp|Im z| 0 z, simplified])
    fix k x
   show ((λx. (- 1) ^ (k div 2) * (if even k then sin x else cos x)) has_field_derivative
          (- 1) ^ (Suc k div 2) * (if odd k then sin x else cos x))
          (at x within closed_segment 0 z)
     by (cases even k) (intro derivative_eq_intros | simp add: power_Suc)+
  next
    fix x
    assume x ∈ closed_segment 0 z
    then show cmod ((- 1) ^ (Suc n div 2) * (if odd n then sin x else cos x)) ≤
exp |Im z|
    by (auto simp: closed_segment_def norm_mult norm_power cmod_sin_le_exp
cmod_cos_le_exp)
  qed
  have **: ⋀k. complex_of_real (sin_coeff k) * z ^ k
```

```
        = (−1)^(k div 2) * (if even k then sin 0 else cos 0) * z^k / of_nat (fact
k)
    by (auto simp: sin_coeff_def elim!: oddE)
  show ?thesis
    by (simp add: ** order_trans [OF _ *])
qed

lemma Taylor_cos:
  norm(cos z − (∑k≤n. complex_of_real (cos_coeff k) * z ^ k))
   ≤ exp|Im z| * (norm z) ^ Suc n / (fact n)
proof −
  have mono: ⋀u w z::real. w ≤ u ⟹ z ≤ u ⟹ w + z ≤ u*2
     by arith
  have *: cmod (cos z −
                (∑ i≤n. (−1) ^ (Suc i div 2) * (if even i then cos 0 else sin 0) * z
^ i / (fact i)))
         ≤ exp |Im z| * cmod z ^ Suc n / (fact n)
  proof (rule complex_Taylor [of closed_segment 0 z n
                                λk x. (−1)^(Suc k div 2) * (if even k then cos x else sin
x)
                                exp|Im z| 0 z, simplified])
    fix k x
    assume x ∈ closed_segment 0 z k ≤ n
      show ((λx. (− 1) ^ (Suc k div 2) * (if even k then cos x else sin x))
has_field_derivative
           (− 1) ^ Suc (k div 2) * (if odd k then cos x else sin x))
            (at x within closed_segment 0 z)
      by (cases even k) (intro derivative_eq_intros | simp add: power_Suc)+
  next
    fix x
    assume x ∈ closed_segment 0 z
    then show cmod ((− 1) ^ Suc (n div 2) * (if odd n then cos x else sin x)) ≤
exp |Im z|
    by (auto simp: closed_segment_def norm_mult norm_power cmod_sin_le_exp
cmod_cos_le_exp)
  qed
  have **: ⋀k. complex_of_real (cos_coeff k) * z ^ k
          = (−1)^(Suc k div 2) * (if even k then cos 0 else sin 0) * z^k / of_nat
(fact k)
    by (auto simp: cos_coeff_def elim!: evenE)
  show ?thesis
    by (simp add: ** order_trans [OF _ *])
qed

declare power_Suc [simp]
```

32-bit Approximation to e

```
lemma e_approx_32: |exp(1) − 5837465777 / 2147483648| ≤ (inverse(2 ^ 32)::real)
  using Taylor_exp [of 1 14] exp_le
```

apply (*simp add*: *sum_distrib_right in_Reals_norm Re_exp atMost_nat_numeral fact_numeral*)
apply (*simp only*: *pos_le_divide_eq* [*symmetric*])
done

lemma *e_less_272*: *exp 1* < (*272/100*::*real*)
using *e_approx_32*
by (*simp add*: *abs_if split*: *if_split_asm*)

lemma *ln_272_gt_1*: *ln* (*272/100*) > (*1*::*real*)
by (*metis e_less_272 exp_less_cancel_iff exp_ln_iff less_trans ln_exp*)

Apparently redundant. But many arguments involve integers.

lemma *ln3_gt_1*: *ln 3* > (*1*::*real*)
by (*simp add*: *less_trans* [*OF ln_272_gt_1*])

6.18.7 The argument of a complex number (HOL Light version)

definition *is_Arg* :: [*complex*,*real*] ⇒ *bool*
where *is_Arg z r* ≡ *z* = *of_real*(*norm z*) ∗ *exp*(i ∗ *of_real r*)

definition *Arg2pi* :: *complex* ⇒ *real*
where *Arg2pi z* ≡ *if z* = *0 then 0 else THE t*. *0* ≤ *t* ∧ *t* < *2*∗*pi* ∧ *is_Arg z t*

lemma *is_Arg_2pi_iff*: *is_Arg z* (*r* + *of_int k* ∗ (*2* ∗ *pi*)) ⟷ *is_Arg z r*
by (*simp add*: *algebra_simps is_Arg_def*)

lemma *is_Arg_eqI*:
assumes *is_Arg z r* **and** *is_Arg z s* **and** *abs* (*r*−*s*) < *2*∗*pi* **and** *z* ≠ *0*
shows *r* = *s*
using *assms* **unfolding** *is_Arg_def*
by (*metis Im_i_times Re_complex_of_real exp_complex_eqI mult_cancel_left mult_eq_0_iff*)

This function returns the angle of a complex number from its representation in polar coordinates. Due to periodicity, its range is arbitrary. *Arg2pi* follows HOL Light in adopting the interval $[0,2\pi)$. But we have the same periodicity issue with logarithms, and it is usual to adopt the same interval for the complex logarithm and argument functions. Further on down, we shall define both functions for the interval $(-\pi,\pi]$. The present version is provided for compatibility.

lemma *Arg2pi_0* [*simp*]: *Arg2pi*(*0*) = *0*
by (*simp add*: *Arg2pi_def*)

lemma *Arg2pi_unique_lemma*:
assumes *is_Arg z t*
and *is_Arg z t′*

```
    and 0 ≤ t  t < 2*pi
    and 0 ≤ t' t' < 2*pi
    and z ≠ 0
  shows t' = t
  using is_Arg_eqI assms by force

lemma Arg2pi: 0 ≤ Arg2pi z ∧ Arg2pi z < 2*pi ∧ is_Arg z (Arg2pi z)
proof (cases z=0)
  case True then show ?thesis
    by (simp add: Arg2pi_def is_Arg_def)
next
  case False
  obtain t where t: 0 ≤ t t < 2*pi
            and ReIm: Re z / cmod z = cos t Im z / cmod z = sin t
    using sincos_total_2pi [OF complex_unit_circle [OF False]]
    by blast
  then have z: is_Arg z t
    unfolding is_Arg_def
    using t False ReIm
   by (intro complex_eqI) (auto simp: exp_Euler sin_of_real cos_of_real field_split_simps)
  show ?thesis
    apply (simp add: Arg2pi_def False)
    apply (rule theI [where a=t])
    using t z False
    apply (auto intro: Arg2pi_unique_lemma)
    done
qed

corollary
  shows Arg2pi_ge_0: 0 ≤ Arg2pi z
    and Arg2pi_lt_2pi: Arg2pi z < 2*pi
    and Arg2pi_eq: z = of_real(norm z) * exp(i * of_real(Arg2pi z))
  using Arg2pi is_Arg_def by auto

lemma complex_norm_eq_1_exp: norm z = 1 ⟷ exp(i * of_real (Arg2pi z))
= z
  by (metis Arg2pi_eq cis_conv_exp mult.left_neutral norm_cis of_real_1)

lemma Arg2pi_unique: ⟦of_real r * exp(i * of_real a) = z; 0 < r; 0 ≤ a; a <
2*pi⟧ ⟹ Arg2pi z = a
  by (rule Arg2pi_unique_lemma [unfolded is_Arg_def, OF _ Arg2pi_eq]) (use
Arg2pi [of z] in ‹auto simp: norm_mult›)

lemma cos_Arg2pi: cmod z * cos (Arg2pi z) = Re z and sin_Arg2pi: cmod z *
sin (Arg2pi z) = Im z
  using Arg2pi_eq [of z] cis_conv_exp Re_rcis Im_rcis unfolding rcis_def by
metis+

lemma Arg2pi_minus:
```

assumes $z \neq 0$ **shows** *Arg2pi* $(-z) =$ (*if Arg2pi* $z <$ *pi then Arg2pi* $z +$ *pi else Arg2pi* $z -$ *pi*)
apply (*rule Arg2pi_unique* [*of norm z*, *OF complex_eqI*])
using *cos_Arg2pi sin_Arg2pi Arg2pi_ge_0 Arg2pi_lt_2pi* [*of z*] *assms*
by (*auto simp*: *Re_exp Im_exp*)

lemma *Arg2pi_times_of_real* [*simp*]:
assumes $0 < r$ **shows** *Arg2pi* (*of_real* $r *$ z) = *Arg2pi* z
by (*metis* (*no_types*, *lifting*) *Arg2pi Arg2pi_eq Arg2pi_unique assms mult.assoc*

mult_eq_0_iff mult_pos_pos of_real_mult zero_less_norm_iff)

lemma *Arg2pi_times_of_real2* [*simp*]: $0 < r \Longrightarrow$ *Arg2pi* ($z *$ *of_real* r) = *Arg2pi* z
by (*metis Arg2pi_times_of_real mult.commute*)

lemma *Arg2pi_divide_of_real* [*simp*]: $0 < r \Longrightarrow$ *Arg2pi* (z / *of_real* r) = *Arg2pi* z
by (*metis Arg2pi_times_of_real2 less_irrefl nonzero_eq_divide_eq of_real_eq_0_iff*)

lemma *Arg2pi_le_pi*: *Arg2pi* $z \leq$ *pi* $\longleftrightarrow$ $0 \leq$ *Im* z
proof (*cases z=0*)
case *False*
have $0 \leq$ *Im* $z \longleftrightarrow 0 \leq$ *Im* (*of_real* (*cmod z*) $*$ *exp* (i $*$ *complex_of_real* (*Arg2pi z*)))
by (*metis Arg2pi_eq*)
also have ... = ($0 \leq$ *Im* (*exp* (i $*$ *complex_of_real* (*Arg2pi z*))))
using *False* **by** (*simp add*: *zero_le_mult_iff*)
also have ... $\longleftrightarrow$ *Arg2pi* $z \leq$ *pi*
by (*simp add*: *Im_exp*) (*metis Arg2pi_ge_0 Arg2pi_lt_2pi sin_lt_zero sin_ge_zero not_le*)
finally show *?thesis*
by *blast*
qed *auto*

lemma *Arg2pi_lt_pi*: $0 <$ *Arg2pi* $z \wedge$ *Arg2pi* $z <$ *pi* $\longleftrightarrow 0 <$ *Im* z
using *Arg2pi_le_pi* [*of z*]
by (*smt* (*verit*, *del_insts*) *Arg2pi_0 Arg2pi_le_pi Arg2pi_minus uminus_complex.simps*(2) *zero_complex.simps*(2))

lemma *Arg2pi_eq_0*: *Arg2pi* $z = 0 \longleftrightarrow z \in \mathbb{R} \wedge 0 \leq$ *Re* z
proof (*cases z=0*)
case *False*
then have $z \in \mathbb{R} \wedge 0 \leq$ *Re* $z \longleftrightarrow z \in \mathbb{R} \wedge 0 \leq$ *Re* (*exp* (i $*$ *complex_of_real* (*Arg2pi z*)))
by (*metis cis.sel*(1) *cis_conv_exp cos_Arg2pi norm_ge_zero norm_le_zero_iff zero_le_mult_iff*)
also have ... $\longleftrightarrow$ *Arg2pi* $z = 0$
proof −

```
    have [simp]: Arg2pi z = 0 ⟹ z ∈ ℝ
      using Arg2pi_eq [of z] by (auto simp: Reals_def)
    moreover have ⟦z ∈ ℝ; 0 ≤ cos (Arg2pi z)⟧ ⟹ Arg2pi z = 0
      by (smt (verit, ccfv_SIG) Arg2pi_ge_0 Arg2pi_le_pi Arg2pi_lt_pi com-
plex_is_Real_iff cos_pi)
    ultimately show ?thesis
      by (auto simp: Re_exp)
  qed
  finally show ?thesis
    by blast
qed auto

corollary Arg2pi_gt_0:
  assumes z ∉ ℝ≥0
    shows Arg2pi z > 0
  using Arg2pi_eq_0 Arg2pi_ge_0 assms dual_order.strict_iff_order
  unfolding nonneg_Reals_def by fastforce

lemma Arg2pi_eq_pi: Arg2pi z = pi ⟷ z ∈ ℝ ∧ Re z < 0
   using Arg2pi_le_pi [of z] Arg2pi_lt_pi [of z] Arg2pi_eq_0 [of z] Arg2pi_ge_0
[of z]
    by (fastforce simp: complex_is_Real_iff)

lemma Arg2pi_eq_0_pi: Arg2pi z = 0 ∨ Arg2pi z = pi ⟷ z ∈ ℝ
  using Arg2pi_eq_0 Arg2pi_eq_pi not_le by auto

lemma Arg2pi_of_real: Arg2pi (of_real r) = (if r<0 then pi else 0)
  using Arg2pi_eq_0_pi Arg2pi_eq_pi by fastforce

lemma Arg2pi_real: z ∈ ℝ ⟹ Arg2pi z = (if 0 ≤ Re z then 0 else pi)
  using Arg2pi_eq_0 Arg2pi_eq_0_pi by auto

lemma Arg2pi_inverse: Arg2pi(inverse z) = (if z ∈ ℝ then Arg2pi z else 2*pi −
Arg2pi z)
proof (cases z=0)
  case False
  show ?thesis
    apply (rule Arg2pi_unique [of inverse (norm z)])
    using Arg2pi_eq False Arg2pi_ge_0 [of z] Arg2pi_lt_2pi [of z] Arg2pi_eq_0
[of z]
    by (auto simp: Arg2pi_real in_Reals_norm exp_diff field_simps)
qed auto

lemma Arg2pi_eq_iff:
  assumes w ≠ 0 z ≠ 0
  shows Arg2pi w = Arg2pi z ⟷ (∃x. 0 < x ∧ w = of_real x * z) (is ?lhs =
?rhs)
proof
  assume ?lhs
```

then have (*cmod w*) ∗ (*z* / *cmod z*) = *w*
by (*metis Arg2pi_eq assms*(2) *mult_eq_0_iff nonzero_mult_div_cancel_left*)
then show *?rhs*
by (*metis assms divide_pos_pos of_real_divide times_divide_eq_left times_divide_eq_right zero_less_norm_iff*)
qed *auto*

lemma *Arg2pi_inverse_eq_0*: *Arg2pi*(*inverse z*) = 0 ⟷ *Arg2pi z* = 0
by (*metis Arg2pi_eq_0 Arg2pi_inverse inverse_inverse_eq*)

lemma *Arg2pi_divide*:
assumes *w* ≠ 0 *z* ≠ 0 *Arg2pi w* ≤ *Arg2pi z*
shows *Arg2pi*(*z* / *w*) = *Arg2pi z* − *Arg2pi w*
apply (*rule Arg2pi_unique* [*of norm*(*z* / *w*)])
using *assms Arg2pi_eq Arg2pi_ge_0* [*of w*] *Arg2pi_lt_2pi* [*of z*]
apply (*auto simp*: *exp_diff norm_divide field_simps*)
done

lemma *Arg2pi_le_div_sum*:
assumes *w* ≠ 0 *z* ≠ 0 *Arg2pi w* ≤ *Arg2pi z*
shows *Arg2pi z* = *Arg2pi w* + *Arg2pi*(*z* / *w*)
by (*simp add*: *Arg2pi_divide assms*)

lemma *Arg2pi_le_div_sum_eq*:
assumes *w* ≠ 0 *z* ≠ 0
shows *Arg2pi w* ≤ *Arg2pi z* ⟷ *Arg2pi z* = *Arg2pi w* + *Arg2pi*(*z* / *w*)
using *assms* **by** (*auto simp*: *Arg2pi_ge_0 intro*: *Arg2pi_le_div_sum*)

lemma *Arg2pi_diff*:
assumes *w* ≠ 0 *z* ≠ 0
shows *Arg2pi w* − *Arg2pi z* = (*if Arg2pi z* ≤ *Arg2pi w then Arg2pi*(*w* / *z*) *else Arg2pi*(*w*/*z*) − 2∗*pi*)
using *assms Arg2pi_divide Arg2pi_inverse* [*of w*/*z*] *Arg2pi_eq_0_pi*
by (*force simp add*: *Arg2pi_ge_0 Arg2pi_divide not_le split*: *if_split_asm*)

lemma *Arg2pi_add*:
assumes *w* ≠ 0 *z* ≠ 0
shows *Arg2pi w* + *Arg2pi z* = (*if Arg2pi w* + *Arg2pi z* < 2∗*pi then Arg2pi*(*w* ∗ *z*) *else Arg2pi*(*w* ∗ *z*) + 2∗*pi*)
using *assms Arg2pi_diff* [*of w*∗*z z*] *Arg2pi_le_div_sum_eq* [*of z w*∗*z*] *Arg2pi* [*of w* ∗ *z*]
by *auto*

lemma *Arg2pi_times*:
assumes *w* ≠ 0 *z* ≠ 0
shows *Arg2pi* (*w* ∗ *z*) = (*if Arg2pi w* + *Arg2pi z* < 2∗*pi then Arg2pi w* + *Arg2pi z*
else (*Arg2pi w* + *Arg2pi z*) − 2∗*pi*)
using *Arg2pi_add* [*OF assms*] **by** *auto*

lemma *Arg2pi_cnj_eq_inverse*:
assumes $z \neq 0$ **shows** *Arg2pi* (*cnj z*) = *Arg2pi* (*inverse z*)
proof −
have $\exists r{>}0$. *of_real r* / *z* = *cnj z*
by (*metis assms complex_norm_square nonzero_mult_div_cancel_left zero_less_norm_iff zero_less_power*)
then show *?thesis*
by (*metis Arg2pi_times_of_real2 divide_inverse_commute*)
qed

lemma *Arg2pi_cnj*: *Arg2pi*(*cnj z*) = (*if z* ∈ ℝ *then Arg2pi z else* 2∗*pi* − *Arg2pi z*)
by (*metis Arg2pi_cnj_eq_inverse Arg2pi_inverse Reals_cnj_iff complex_cnj_zero*)

lemma *Arg2pi_exp*: 0 ≤ *Im z* ⟹ *Im z* < 2∗*pi* ⟹ *Arg2pi*(*exp z*) = *Im z*
by (*simp add*: *Arg2pi_unique exp_eq_polar*)

lemma *complex_split_polar*:
obtains *r a*::*real* **where** *z* = *complex_of_real r* ∗ (*cos a* + i ∗ *sin a*) 0 ≤ *r* 0 ≤ *a a* < 2∗*pi*
using *Arg2pi cis.ctr cis_conv_exp* **unfolding** *Complex_eq is_Arg_def* **by** *fastforce*

lemma *Re_Im_le_cmod*: *Im w* ∗ *sin φ* + *Re w* ∗ *cos φ* ≤ *cmod w*
proof (*cases w rule*: *complex_split_polar*)
case (*1 r a*) **with** *sin_cos_le1* [*of a φ*] **show** *?thesis*
apply (*simp add*: *norm_mult cmod_unit_one*)
by (*metis* (*no_types*, *opaque_lifting*) *abs_le_D1 distrib_left mult.commute mult.left_commute mult_left_le*)
qed

6.18.8 Analytic properties of tangent function

lemma *cnj_tan*: *cnj*(*tan z*) = *tan*(*cnj z*)
by (*simp add*: *cnj_cos cnj_sin tan_def*)

lemma *field_differentiable_at_tan*: *cos z* ≠ 0 ⟹ *tan field_differentiable at z*
unfolding *field_differentiable_def*
using *DERIV_tan* **by** *blast*

lemma *field_differentiable_within_tan*: *cos z* ≠ 0
⟹ *tan field_differentiable* (*at z within s*)
using *field_differentiable_at_tan field_differentiable_at_within* **by** *blast*

lemma *continuous_within_tan*: *cos z* ≠ 0 ⟹ *continuous* (*at z within s*) *tan*
using *continuous_at_imp_continuous_within isCont_tan* **by** *blast*

lemma *continuous_on_tan* [*continuous_intros*]: (⋀*z*. *z* ∈ *s* ⟹ *cos z* ≠ 0) ⟹

continuous_on s tan
by (*simp add*: *continuous_at_imp_continuous_on*)

lemma *holomorphic_on_tan*: $(\bigwedge z.\ z \in s \Longrightarrow \cos z \neq 0) \Longrightarrow \tan$ *holomorphic_on s*
by (*simp add*: *field_differentiable_within_tan holomorphic_on_def*)

6.18.9 The principal branch of the Complex logarithm

instantiation *complex* :: *ln*
begin

definition *ln_complex* :: *complex* ⇒ *complex*
where *ln_complex* ≡ λ*z*. *THE w*. *exp w* = *z* & −*pi* < *Im*(*w*) & *Im*(*w*) ≤ *pi*

NOTE: within this scope, the constant Ln is not yet available!

lemma
assumes *z* ≠ *0*
shows *exp_Ln* [*simp*]: *exp*(*ln z*) = *z*
and *mpi_less_Im_Ln*: −*pi* < *Im*(*ln z*)
and *Im_Ln_le_pi*: *Im*(*ln z*) ≤ *pi*
proof −
obtain ψ **where** *z*: *z* / (*cmod z*) = *Complex* (*cos* ψ) (*sin* ψ)
using *complex_unimodular_polar* [*of z* / (*norm z*)] *assms*
by (*auto simp*: *norm_divide field_split_simps*)
obtain φ **where** φ: − *pi* < φ φ ≤ *pi sin* φ = *sin* ψ *cos* φ = *cos* ψ
using *sincos_principal_value* [*of* ψ] *assms*
by (*auto simp*: *norm_divide field_split_simps*)
have *exp*(*ln z*) = *z* & −*pi* < *Im*(*ln z*) & *Im*(*ln z*) ≤ *pi* **unfolding** *ln_complex_def*
apply (*rule theI* [**where** *a* = *Complex* (*ln*(*norm z*)) φ])
using *z assms* φ
apply (*auto simp*: *field_simps exp_complex_eqI exp_eq_polar cis.code*)
done
then show *exp*(*ln z*) = *z* −*pi* < *Im*(*ln z*) *Im*(*ln z*) ≤ *pi*
by *auto*
qed

lemma *Ln_exp* [*simp*]:
assumes −*pi* < *Im*(*z*) *Im*(*z*) ≤ *pi*
shows *ln*(*exp z*) = *z*
proof (*rule exp_complex_eqI*)
show |*Im* (*ln* (*exp z*)) − *Im z*| < *2* ∗ *pi*
using *assms mpi_less_Im_Ln* [*of exp z*] *Im_Ln_le_pi* [*of exp z*] **by** *auto*
qed *auto*

6.18.10 Relation to Real Logarithm

lemma *Ln_of_real*:
assumes *0* < *z*

shows *ln*(*of_real z*::*complex*) = *of_real*(*ln z*)
by (*smt* (*verit*) *Im_complex_of_real Ln_exp assms exp_ln of_real_exp pi_ge_two*)

corollary *Ln_in_Reals* [*simp*]: $z \in \mathbb{R} \Longrightarrow Re\ z > 0 \Longrightarrow ln\ z \in \mathbb{R}$
by (*auto simp*: *Ln_of_real elim*: *Reals_cases*)

corollary *Im_Ln_of_real* [*simp*]: $r > 0 \Longrightarrow Im\ (ln\ (of\_real\ r)) = 0$
by (*simp add*: *Ln_of_real*)

lemma *cmod_Ln_Reals* [*simp*]: $z \in \mathbb{R} \Longrightarrow 0 < Re\ z \Longrightarrow cmod\ (ln\ z) = norm\ (ln\ (Re\ z))$
using *Ln_of_real* **by** *force*

lemma *Ln_Reals_eq*: $[\![x \in \mathbb{R}; Re\ x > 0]\!] \Longrightarrow ln\ x = of\_real\ (ln\ (Re\ x))$
using *Ln_of_real* **by** *force*

lemma *Ln_1* [*simp*]: *ln 1* = (*0*::*complex*)
by (*smt* (*verit*, *best*) *Ln_of_real ln_one of_real_0 of_real_1*)

lemma *Ln_eq_zero_iff* [*simp*]: $x \notin \mathbb{R}_{\leq 0} \Longrightarrow ln\ x = 0 \longleftrightarrow x = 1$ **for** *x*::*complex*
by (*metis* (*mono_tags*, *lifting*) *Ln_1 exp_Ln exp_zero nonpos_Reals_zero_I*)

instance
by *intro_classes* (*rule ln_complex_def Ln_1*)

end

abbreviation *Ln* :: *complex* ⇒ *complex*
where *Ln* ≡ *ln*

lemma *Ln_eq_iff*: $w \neq 0 \Longrightarrow z \neq 0 \Longrightarrow (Ln\ w = Ln\ z \longleftrightarrow w = z)$
by (*metis exp_Ln*)

lemma *Ln_unique*: $exp(z) = w \Longrightarrow -pi < Im(z) \Longrightarrow Im(z) \leq pi \Longrightarrow Ln\ w = z$
using *Ln_exp* **by** *blast*

lemma *Re_Ln* [*simp*]: $z \neq 0 \Longrightarrow Re(Ln\ z) = ln(norm\ z)$
by (*metis exp_Ln ln_exp norm_exp_eq_Re*)

corollary *ln_cmod_le*:
assumes *z*: $z \neq 0$
shows *ln* (*cmod z*) ≤ *cmod* (*Ln z*)
by (*metis Re_Ln complex_Re_le_cmod z*)

proposition *exists_complex_root*:
fixes *z* :: *complex*
assumes $n \neq 0$ **obtains** *w* **where** $z = w \mathbin{\hat{}} n$
by (*metis assms exp_Ln exp_of_nat_mult nonzero_mult_div_cancel_left of_nat_eq_0_iff power_0_left times_divide_eq_right*)

corollary *exists_complex_root_nonzero*:
fixes *z*::*complex*
assumes $z \neq 0$ $n \neq 0$
obtains *w* **where** $w \neq 0$ $z = w \mathbin{\hat{}} n$
by (*metis exists_complex_root* [*of n z*] *assms power_0_left*)

6.18.11 Derivative of Ln away from the branch cut

lemma *Im_Ln_less_pi*:
assumes $z \notin \mathbb{R}_{\leq 0}$**shows** *Im* (*Ln z*) < *pi*
proof −
have *znz* [*simp*]: $z \neq 0$
using *assms* **by** *auto*
with *Im_Ln_le_pi* [*of z*] **show** *?thesis*
by (*smt* (*verit*, *best*) *Arg2pi_eq_0_pi Arg2pi_exp Ln_in_Reals assms complex_is_Real_iff complex_nonpos_Reals_iff exp_Ln pi_ge_two*)
qed

lemma *has_field_derivative_Ln*:
assumes $z \notin \mathbb{R}_{\leq 0}$
shows (*Ln has_field_derivative inverse*(*z*)) (*at z*)
proof −
have *znz* [*simp*]: $z \neq 0$
using *assms* **by** *auto*
then have *Im* (*Ln z*) ≠ *pi*
by (*smt* (*verit*, *best*) *Arg2pi_eq_0_pi Arg2pi_exp Ln_in_Reals assms complex_is_Real_iff complex_nonpos_Reals_iff exp_Ln pi_ge_two*)
let *?U* = {*w*. −*pi* < *Im*(*w*) ∧ *Im*(*w*) < *pi*}
have *1*: *open ?U*
by (*simp add*: *open_Collect_conj open_halfspace_Im_gt open_halfspace_Im_lt*)
have *2*: ⋀*x*. *x* ∈ *?U* ⟹ (*exp has_derivative blinfun_apply*(*Blinfun* ((∗) (*exp x*)))) (*at x*)
by (*simp add*: *bounded_linear_Blinfun_apply bounded_linear_mult_right has_field_derivative_imp_*
have *3*: *continuous_on ?U* (λ*x*. *Blinfun* ((∗) (*exp x*)))
unfolding *blinfun_mult_right.abs_eq* [*symmetric*] **by** (*intro continuous_intros*)
have *4*: *Ln z* ∈ *?U*
by (*simp add*: *Im_Ln_less_pi assms mpi_less_Im_Ln*)
have *5*: *Blinfun* ((∗) (*inverse z*)) o_L *Blinfun* ((∗) (*exp* (*Ln z*))) = *id_blinfun*
by (*rule blinfun_eqI*) (*simp add*: *bounded_linear_mult_right bounded_linear_Blinfun_apply*)
obtain *U′ V g g′* **where** *open U′* **and** *sub*: *U′* ⊆ *?U*
and *Ln z* ∈ *U′ open V z* ∈ *V*
and *hom*: *homeomorphism U′ V exp g*
and *g*: ⋀*y*. *y* ∈ *V* ⟹ (*g has_derivative* (*g′ y*)) (*at y*)
and *g′*: ⋀*y*. *y* ∈ *V* ⟹ *g′ y* = *inv* ((∗) (*exp* (*g y*)))
and *bij*: ⋀*y*. *y* ∈ *V* ⟹ *bij* ((∗) (*exp* (*g y*)))
using *inverse_function_theorem* [*OF 1 2 3 4 5*]
by (*simp add*: *bounded_linear_Blinfun_apply bounded_linear_mult_right*)
blast

show (*Ln has_field_derivative inverse*(*z*)) (*at z*)
unfolding *has_field_derivative_def*
proof (*rule has_derivative_transform_within_open*)
show *g_eq_Ln*: *g y* = *Ln y* **if** *y* ∈ *V* **for** *y*
by (*smt* (*verit*, *ccfv_threshold*) *Ln_exp hom homeomorphism_def imageI mem_Collect_eq sub subset_iff that*)
have *0* ∉ *V*
by (*meson exp_not_eq_zero hom homeomorphism_def*)
then have ⋀*y*. *y* ∈ *V* ⟹ *g*′ *y* = *inv* ((∗) *y*)
by (*metis exp_Ln g*′ *g_eq_Ln*)
then have *g*′: *g*′ *z* = (λ*x*. *x*/*z*)
by (*metis* ‹*z* ∈ *V*› *bij bij_inv_eq_iff exp_Ln g_eq_Ln nonzero_mult_div_cancel_left znz*)
show (*g has_derivative* (∗) (*inverse z*)) (*at z*)
using *g* [*OF* ‹*z* ∈ *V*›] *g*′ **by** (*simp add*: *divide_inverse_commute*)
qed (*auto simp*: ‹*z* ∈ *V*› ‹*open V*›)
qed

declare *has_field_derivative_Ln* [*derivative_intros*]
declare *has_field_derivative_Ln* [*THEN DERIV_chain2*, *derivative_intros*]

lemma *field_differentiable_at_Ln*: *z* ∉ ℝ≤0 ⟹ *Ln field_differentiable at z*
using *field_differentiable_def has_field_derivative_Ln* **by** *blast*

lemma *field_differentiable_within_Ln*: *z* ∉ ℝ≤0
⟹ *Ln field_differentiable* (*at z within S*)
using *field_differentiable_at_Ln field_differentiable_within_subset* **by** *blast*

lemma *continuous_at_Ln*: *z* ∉ ℝ≤0 ⟹ *continuous* (*at z*) *Ln*
by (*simp add*: *field_differentiable_imp_continuous_at field_differentiable_within_Ln*)

lemma *isCont_Ln*′ [*simp*,*continuous_intros*]:
⟦*isCont f z*; *f z* ∉ ℝ≤0⟧ ⟹ *isCont* (λ*x*. *Ln* (*f x*)) *z*
by (*blast intro*: *isCont_o2* [*OF* _ *continuous_at_Ln*])

lemma *continuous_within_Ln* [*continuous_intros*]: *z* ∉ ℝ≤0 ⟹ *continuous* (*at z within S*) *Ln*
using *continuous_at_Ln continuous_at_imp_continuous_within* **by** *blast*

lemma *continuous_on_Ln* [*continuous_intros*]: (⋀*z*. *z* ∈ *S* ⟹ *z* ∉ ℝ≤0) ⟹ *continuous_on S Ln*
by (*simp add*: *continuous_at_imp_continuous_on continuous_within_Ln*)

lemma *continuous_on_Ln*′ [*continuous_intros*]:
continuous_on S f ⟹ (⋀*z*. *z* ∈ *S* ⟹ *f z* ∉ ℝ≤0) ⟹ *continuous_on S* (λ*x*. *Ln* (*f x*))
by (*rule continuous_on_compose2*[*OF continuous_on_Ln*, *of UNIV* − *nonpos_Reals S f*]) *auto*

lemma *holomorphic_on_Ln* [*holomorphic_intros*]: $S \cap \mathbb{R}_{\leq 0} = \{\} \Longrightarrow$ *Ln holomorphic_on S*
by (*simp add*: *disjoint_iff field_differentiable_within_Ln holomorphic_on_def*)

lemma *holomorphic_on_Ln'* [*holomorphic_intros*]:
$(\bigwedge z.\ z \in A \Longrightarrow f\ z \notin \mathbb{R}_{\leq 0}) \Longrightarrow f$ *holomorphic_on* $A \Longrightarrow (\lambda z.\ Ln\ (f\ z))$ *holomorphic_on A*
using *holomorphic_on_compose_gen*[*OF _ holomorphic_on_Ln*, *of f A* $- \mathbb{R}_{\leq 0}$]
by (*auto simp*: *o_def*)

lemma *tendsto_Ln* [*tendsto_intros*]:
assumes $(f \longrightarrow L)\ F\ L \notin \mathbb{R}_{\leq 0}$
shows $((\lambda x.\ Ln\ (f\ x)) \longrightarrow Ln\ L)\ F$
by (*simp add*: *assms isCont_tendsto_compose*)

lemma *divide_ln_mono*:
fixes *x y*::*real*
assumes $3 \leq x\ x \leq y$
shows $x\ /\ ln\ x \leq y\ /\ ln\ y$
proof −
have $\bigwedge u.\ [\![x \leq u;\ u \leq y]\!] \Longrightarrow ((\lambda z.\ z\ /\ Ln\ z)$ *has_field_derivative* $1\ /\ Ln\ u - 1\ /\ (Ln\ u)^2)$ *(at u)*
using ‹$3 \leq x$› **by** (*force intro*!: *derivative_eq_intros simp*: *field_simps power_eq_if*)
moreover
have $x\ /\ ln\ x \leq y\ /\ ln\ y$
if $Re\ (y\ /\ Ln\ y) - Re\ (x\ /\ Ln\ x) = (Re\ (1\ /\ Ln\ u) - Re\ (1\ /\ (Ln\ u)^2)) * (y - x)$
and *x*: $x \leq u\ u \leq y$ **for** *u*
proof −
have *eq*: $y\ /\ ln\ y = (1\ /\ ln\ u - 1\ /\ (ln\ u)^2) * (y - x) + x\ /\ ln\ x$
using *that* ‹$3 \leq x$› **by** (*auto simp*: *Ln_Reals_eq in_Reals_norm group_add_class.diff_eq_eq*)
show *?thesis*
using *exp_le* ‹$3 \leq x$› *x* **by** (*simp add*: *eq*) (*simp add*: *power_eq_if divide_simps ln_ge_iff*)
qed
ultimately show *?thesis*
using *complex_mvt_line* [*of x y* $\lambda z.\ z\ /\ Ln\ z\ \lambda z.\ 1/(Ln\ z) - 1/(Ln\ z)\hat{}2$] *assms*
by (*force simp add*: *closed_segment_Reals closed_segment_eq_real_ivl*)
qed

theorem *Ln_series*:
fixes *z* :: *complex*
assumes *norm z* < *1*
shows $(\lambda n.\ (-1)\hat{}Suc\ n\ /\ of\_nat\ n * z\hat{}n)$ *sums ln* $(1 + z)$ (**is** $(\lambda n.\ ?f\ n * z\hat{}n)$ *sums* _)
proof −
let *?F* = $\lambda z.\ \sum n.\ ?f\ n * z\hat{}n$ **and** *?F'* = $\lambda z.\ \sum n.\ diffs\ ?f\ n * z\hat{}n$
have *r*: *conv_radius ?f* = *1*

```
    by (intro conv_radius_ratio_limit_nonzero[of _ 1])
      (simp_all add: norm_divide LIMSEQ_Suc_n_over_n del: of_nat_Suc)

  have ∃c. ∀z∈ball 0 1. ln (1 + z) - ?F z = c
  proof (rule has_field_derivative_zero_constant)
    fix z :: complex assume z': z ∈ ball 0 1
    hence z: norm z < 1 by simp
    define t :: complex where t = of_real (1 + norm z) / 2
    from z have t: norm z < norm t norm t < 1 unfolding t_def
      by (simp_all add: field_simps norm_divide del: of_real_add)

    have Re (-z) ≤ norm (-z) by (rule complex_Re_le_cmod)
    also from z have ... < 1 by simp
    finally have ((λz. ln (1 + z)) has_field_derivative inverse (1+z)) (at z)
      by (auto intro!: derivative_eq_intros simp: complex_nonpos_Reals_iff)
    moreover have (?F has_field_derivative ?F' z) (at z) using t r
      by (intro termdiffs_strong[of _ t] summable_in_conv_radius) simp_all
    ultimately have ((λz. ln (1 + z) - ?F z) has_field_derivative (inverse (1 +
z) - ?F' z))
                    (at z within ball 0 1)
      by (intro derivative_intros) (simp_all add: at_within_open[OF z'])
    also have (λn. of_nat n * ?f n * z ^ (n - Suc 0)) sums ?F' z using t r
     by (intro diffs_equiv termdiff_converges[OF t(1)] summable_in_conv_radius)
simp_all
    from sums_split_initial_segment[OF this, of 1]
       have (λi. (-z) ^ i) sums ?F' z by (simp add: power_minus[of z] del:
of_nat_Suc)
      hence ?F' z = inverse (1 + z) using z by (simp add: sums_iff sum-
inf_geometric divide_inverse)
    also have inverse (1 + z) - inverse (1 + z) = 0 by simp
    finally show ((λz. ln (1 + z) - ?F z) has_field_derivative 0) (at z within ball
0 1) .
  qed simp_all
  then obtain c where c: ⋀z. z ∈ ball 0 1 ⟹ ln (1 + z) - ?F z = c by blast
  from c[of 0] have c = 0 by (simp only: powser_zero) simp
  with c[of z] assms have ln (1 + z) = ?F z by simp
  moreover have summable (λn. ?f n * z^n) using assms r
    by (intro summable_in_conv_radius) simp_all
  ultimately show ?thesis by (simp add: sums_iff)
qed

lemma Ln_series': cmod z < 1 ⟹ (λn. - ((-z)^n) / of_nat n) sums ln (1 +
z)
  by (drule Ln_series) (simp add: power_minus')

lemma ln_series':
  fixes x::real
  assumes |x| < 1
  shows   (λn. - ((-x)^n) / of_nat n) sums ln (1 + x)
```

proof −
from *assms* **have** (λ*n*. − ((−*of_real x*)^*n*) / *of_nat n*) *sums ln* (1 + *complex_of_real x*)
by (*intro Ln_series'*) *simp_all*
also have (λ*n*. − ((−*of_real x*)^*n*) / *of_nat n*) = (λ*n*. *complex_of_real* (− ((−*x*)^*n*) / *of_nat n*))
by (*rule ext*) *simp*
also from *assms* **have** *ln* (1 + *complex_of_real x*) = *of_real* (*ln* (1 + *x*))
by (*smt* (*verit*) *Ln_of_real of_real_1 of_real_add*)
finally show *?thesis* **by** (*subst* (*asm*) *sums_of_real_iff*)
qed

lemma *Ln_approx_linear*:
fixes *z* :: *complex*
assumes *norm z* < *1*
shows *norm* (*ln* (1 + *z*) − *z*) ≤ *norm z*^*2* / (1 − *norm z*)
proof −
let *?f* = λ*n*. (−*1*)^*Suc n* / *of_nat n*
from *assms* **have** (λ*n*. *?f n* ∗ *z*^*n*) *sums ln* (1 + *z*) **using** *Ln_series* **by** *simp*
moreover have (λ*n*. (*if n* = *1 then 1 else 0*) ∗ *z*^*n*) *sums z* **using** *powser_sums_if*[*of 1*] **by** *simp*
ultimately have (λ*n*. (*?f n* − (*if n* = *1 then 1 else 0*)) ∗ *z*^*n*) *sums* (*ln* (1 + *z*) − *z*)
by (*subst left_diff_distrib*, *intro sums_diff*) *simp_all*
from *sums_split_initial_segment*[*OF this*, *of Suc 1*]
have (λ*i*. (−(*z*^*2*)) ∗ *inverse* (2 + *of_nat i*) ∗ (− *z*)^*i*) *sums* (*Ln* (1 + *z*) − *z*)
by (*simp add*: *power2_eq_square mult_ac power_minus*[*of z*] *divide_inverse*)
hence (*Ln* (1 + *z*) − *z*) = (∑ *i*. (−(*z*^*2*)) ∗ *inverse* (*of_nat* (*i*+*2*)) ∗ (−*z*)^*i*)
by (*simp add*: *sums_iff*)
also have *A*: *summable* (λ*n*. *norm z*^*2* ∗ (*inverse* (*real_of_nat* (*Suc* (*Suc n*))) ∗ *cmod z* ^ *n*))
by (*rule summable_mult*, *rule summable_comparison_test_ev*[*OF _ summable_geometric*[*of norm z*]])
(*auto simp*: *assms field_simps intro*!: *always_eventually*)
hence *norm* (∑ *i*. (−(*z*^*2*)) ∗ *inverse* (*of_nat* (*i*+*2*)) ∗ (−*z*)^*i*)
≤ (∑ *i*. *norm* (−(*z*^*2*) ∗ *inverse* (*of_nat* (*i*+*2*)) ∗ (−*z*)^*i*))
by (*intro summable_norm*)
(*auto simp*: *norm_power norm_inverse norm_mult mult_ac simp del*: *of_nat_add of_nat_Suc*)
also have *norm* ((−*z*)^*2* ∗ (−*z*)^*i*) ∗ *inverse* (*of_nat* (*i*+*2*)) ≤ *norm* ((−*z*)^*2* ∗ (−*z*)^*i*) ∗ *1* **for** *i*
by (*intro mult_left_mono*) (*simp_all add*: *field_split_simps*)
hence (∑ *i*. *norm* (−(*z*^*2*) ∗ *inverse* (*of_nat* (*i*+*2*)) ∗ (−*z*)^*i*))
≤ (∑ *i*. *norm* (−(*z*^*2*) ∗ (−*z*)^*i*))
using *A assms*
unfolding *norm_power norm_inverse norm_divide norm_mult*
apply (*intro suminf_le summable_mult summable_geometric*)
apply (*auto simp*: *norm_power field_simps simp del*: *of_nat_add of_nat_Suc*)

```
    done
  also have ... = norm z^2 * (∑ i. norm z^i) using assms
    by (subst suminf_mult [symmetric]) (auto intro!: summable_geometric simp: norm_mult norm_power)
  also have (∑ i. norm z^i) = inverse (1 - norm z) using assms
    by (subst suminf_geometric) (simp_all add: divide_inverse)
  also have norm z^2 * ... = norm z^2 / (1 - norm z) by (simp add: divide_inverse)
  finally show ?thesis .
qed


lemma norm_Ln_le:
  fixes z :: complex
  assumes norm z < 1/2
  shows   norm (Ln(1+z)) ≤ 2 * norm z
proof -
  have sums: (λn. - ((- z) ^ n) / of_nat n) sums ln (1 + z)
    by (intro Ln_series') (use assms in auto)
  have summable: summable (λn. norm (- ((- z) ^ n / of_nat n)))
    using ln_series'[of -norm z] assms
    by (simp add: sums_iff summable_minus_iff norm_power norm_divide)

  have norm (ln (1 + z)) = norm (∑ n. -((-z) ^ n / of_nat n))
    using sums by (simp add: sums_iff)
  also have ... ≤ (∑ n. norm (-((-z) ^ n / of_nat n)))
    using summable by (rule summable_norm)
  also have ... = (∑ n. norm (-((-z) ^ Suc n / of_nat (Suc n))))
    using summable by (subst suminf_split_head) auto
  also have ... ≤ (∑ n. norm z * (1 / 2) ^ n)
  proof (rule suminf_le)
    show summable (λn. norm z * (1 / 2) ^ n)
      by (intro summable_mult summable_geometric) auto
  next
    show summable (λn. norm (- ((- z) ^ Suc n / of_nat (Suc n))))
      using summable by (subst summable_Suc_iff)
  next
    fix n
    have norm (-((-z) ^ Suc n / of_nat (Suc n))) = norm z * (norm z ^ n / real (Suc n))
      by (simp add: norm_power norm_divide norm_mult del: of_nat_Suc)
    also have ... ≤ norm z * ((1 / 2) ^ n / 1)
      using assms by (intro mult_left_mono frac_le power_mono) auto
    finally show norm (- ((- z) ^ Suc n / of_nat (Suc n))) ≤ norm z * (1 / 2) ^ n
      by simp
  qed
  also have ... = norm z * (∑ n. (1 / 2) ^ n)
    by (subst suminf_mult) (auto intro: summable_geometric)
```

also have $(\sum n.\ (1\ /\ 2 :: real)\ \hat{}\ n) = 2$
using *geometric_sums*[*of* $1\ /\ 2 :: real$] **by** (*simp add*: *sums_iff*)
finally show *?thesis*
by (*simp add*: *mult_ac*)
qed

6.18.12 Quadrant-type results for Ln

lemma *cos_lt_zero_pi*: $pi/2 < x \Longrightarrow x < 3{*}pi/2 \Longrightarrow cos\ x < 0$
using *cos_minus_pi cos_gt_zero_pi* [*of* $x{-}pi$]
by *simp*

lemma *Re_Ln_pos_le*:
assumes $z \neq 0$
shows $|Im(Ln\ z)| \leq pi/2 \longleftrightarrow 0 \leq Re(z)$
proof −
{ fix w
assume $w = Ln\ z$
then have w: $Im\ w \leq pi - pi < Im\ w$
using *Im_Ln_le_pi* [*of z*] *mpi_less_Im_Ln* [*of z*] *assms*
by *auto*
then have $|Im\ w| \leq pi/2 \longleftrightarrow 0 \leq Re(exp\ w)$
using *cos_lt_zero_pi* [*of* $-\ (Im\ w)$] *cos_lt_zero_pi* [*of* $(Im\ w)$] *not_le*
by (*auto simp*: *Re_exp zero_le_mult_iff abs_if intro*: *cos_ge_zero*)
}
then show *?thesis* **using** *assms*
by *auto*
qed

lemma *Re_Ln_pos_lt*:
assumes $z \neq 0$
shows $|Im(Ln\ z)| < pi/2 \longleftrightarrow 0 < Re(z)$
using *Re_Ln_pos_le assms*
by (*smt* (*verit*) *Re_exp arccos_cos cos_minus cos_pi_half exp_Ln exp_gt_zero field_sum_of_halves mult_eq_0_iff*)

lemma *Im_Ln_pos_le*:
assumes $z \neq 0$
shows $0 \leq Im(Ln\ z) \wedge Im(Ln\ z) \leq pi \longleftrightarrow 0 \leq Im(z)$
proof −
{ fix w
assume $w = Ln\ z$
then have w: $Im\ w \leq pi - pi < Im\ w$
using *Im_Ln_le_pi* [*of z*] *mpi_less_Im_Ln* [*of z*] *assms*
by *auto*
then have $0 \leq Im\ w \wedge Im\ w \leq pi \longleftrightarrow 0 \leq Im(exp\ w)$
using *sin_ge_zero* [*of* $-\ (Im\ w)$] *sin_ge_zero* [*of* $abs(Im\ w)$] *sin_zero_pi_iff* [*of Im w*]
by (*force simp*: *Im_exp zero_le_mult_iff sin_ge_zero*) **}**

then show *?thesis* **using** *assms*
by *auto*
qed

lemma *Im_Ln_pos_lt*:
assumes $z \neq 0$
shows $0 < Im(Ln\ z) \wedge Im(Ln\ z) < pi \longleftrightarrow 0 < Im(z)$
using *Im_Ln_pos_le* [*OF assms*] *assms*
by (*smt* (*verit*, *best*) *Arg2pi_exp Arg2pi_lt_pi exp_Ln*)

lemma *Re_Ln_pos_lt_imp*: $0 < Re(z) \Longrightarrow |Im(Ln\ z)| < pi/2$
by (*metis Re_Ln_pos_lt less_irrefl zero_complex.simps*(1))

lemma *Im_Ln_pos_lt_imp*: $0 < Im(z) \Longrightarrow 0 < Im(Ln\ z) \wedge Im(Ln\ z) < pi$
by (*metis Im_Ln_pos_lt not_le order_refl zero_complex.simps*(2))

A reference to the set of positive real numbers

lemma *Im_Ln_eq_0*: $z \neq 0 \Longrightarrow (Im(Ln\ z) = 0 \longleftrightarrow 0 < Re(z) \wedge Im(z) = 0)$
using *Im_Ln_pos_le Im_Ln_pos_lt Re_Ln_pos_lt* **by** *fastforce*

lemma *Im_Ln_eq_pi*: $z \neq 0 \Longrightarrow (Im(Ln\ z) = pi \longleftrightarrow Re(z) < 0 \wedge Im(z) = 0)$
using *Im_Ln_eq_0 Im_Ln_pos_le Im_Ln_pos_lt complex.expand* **by** *fastforce*

6.18.13 More Properties of Ln

lemma *cnj_Ln*: **assumes** $z \notin \mathbb{R}_{\leq 0}$ **shows** $cnj(Ln\ z) = Ln(cnj\ z)$
proof (*cases z=0*)
case *False*
show *?thesis*
by (*smt* (*verit*) *False Im_Ln_less_pi Ln_exp assms cnj.sel*(2) *exp_Ln exp_cnj mpi_less_Im_Ln*)
qed (*use assms* **in** *auto*)

lemma *Ln_inverse*: **assumes** $z \notin \mathbb{R}_{\leq 0}$ **shows** $Ln(inverse\ z) = -(Ln\ z)$
proof (*cases z=0*)
case *False*
show *?thesis*
by (*smt* (*verit*) *False Im_Ln_less_pi Ln_exp assms exp_Ln exp_minus mpi_less_Im_Ln uminus_complex.sel*(2))
qed (*use assms* **in** *auto*)

lemma *Ln_minus1* [*simp*]: $Ln(-1) = \mathrm{i} * pi$
proof (*rule exp_complex_eqI*)
show $|Im\ (Ln\ (-\ 1)) - Im\ (\mathrm{i} * pi)| < 2 * pi$
using *Im_Ln_le_pi* [*of* -1] *mpi_less_Im_Ln* [*of* -1] **by** *auto*
qed *auto*

lemma *Ln_ii* [*simp*]: $Ln\ \mathrm{i} = \mathrm{i} * of\_real\ pi/2$

using *Ln_exp* [*of* i ∗ (*of_real pi*/*2*)]
unfolding *exp_Euler*
by *simp*

lemma *Ln_minus_ii* [*simp*]: *Ln*(−i) = − (i ∗ *pi*/*2*)
using *Ln_inverse* **by** *fastforce*

lemma *Ln_times*:
assumes *w* ≠ *0* *z* ≠ *0*
shows *Ln*(*w* ∗ *z*) =
(*if Im*(*Ln w* + *Ln z*) ≤ −*pi then* (*Ln*(*w*) + *Ln*(*z*)) + i ∗ *of_real*(*2*∗*pi*)
else if Im(*Ln w* + *Ln z*) > *pi then* (*Ln*(*w*) + *Ln*(*z*)) − i ∗ *of_real*(*2*∗*pi*)
else Ln(*w*) + *Ln*(*z*))
using *pi_ge_zero Im_Ln_le_pi* [*of w*] *Im_Ln_le_pi* [*of z*]
using *assms mpi_less_Im_Ln* [*of w*] *mpi_less_Im_Ln* [*of z*]
by (*auto simp*: *exp_add exp_diff sin_double cos_double exp_Euler intro*!: *Ln_unique*)

corollary *Ln_times_simple*:
⟦*w* ≠ *0*; *z* ≠ *0*; −*pi* < *Im*(*Ln w*) + *Im*(*Ln z*); *Im*(*Ln w*) + *Im*(*Ln z*) ≤ *pi*⟧
⟹ *Ln*(*w* ∗ *z*) = *Ln*(*w*) + *Ln*(*z*)
by (*simp add*: *Ln_times*)

corollary *Ln_times_of_real*:
⟦*r* > *0*; *z* ≠ *0*⟧ ⟹ *Ln*(*of_real r* ∗ *z*) = *ln r* + *Ln*(*z*)
using *mpi_less_Im_Ln Im_Ln_le_pi*
by (*force simp*: *Ln_times*)

corollary *Ln_times_of_nat*:
⟦*r* > *0*; *z* ≠ *0*⟧ ⟹ *Ln*(*of_nat r* ∗ *z* :: *complex*) = *ln* (*of_nat r*) + *Ln*(*z*)
using *Ln_times_of_real*[*of of_nat r z*] **by** *simp*

corollary *Ln_times_Reals*:
⟦*r* ∈ *Reals*; *Re r* > *0*; *z* ≠ *0*⟧ ⟹ *Ln*(*r* ∗ *z*) = *ln* (*Re r*) + *Ln*(*z*)
using *Ln_Reals_eq Ln_times_of_real* **by** *fastforce*

corollary *Ln_divide_of_real*:
⟦*r* > *0*; *z* ≠ *0*⟧ ⟹ *Ln*(*z* / *of_real r*) = *Ln*(*z*) − *ln r*
using *Ln_times_of_real* [*of inverse r z*]
by (*simp add*: *ln_inverse Ln_of_real mult.commute divide_inverse flip*: *of_real_inverse*)

corollary *Ln_prod*:
fixes *f* :: *'a* ⇒ *complex*
assumes *finite A* ⋀*x*. *x* ∈ *A* ⟹ *f x* ≠ *0*
shows ∃ *n*. *Ln* (*prod f A*) = (∑ *x* ∈ *A*. *Ln* (*f x*) + (*of_int* (*n x*) ∗ (*2* ∗ *pi*)) ∗ i)
using *assms*
proof (*induction A*)
case (*insert x A*)
then obtain *n* **where** *n*: *Ln* (*prod f A*) = (∑ *x*∈*A*. *Ln* (*f x*) + *of_real* (*of_int* (*n x*) ∗ (*2* ∗ *pi*)) ∗ i)

by *auto*
define D **where** $D \equiv Im\ (Ln\ (f\ x)) + Im\ (Ln\ (prod\ f\ A))$
define q::*int* **where** $q \equiv (if\ D \leq -pi\ then\ 1\ else\ if\ D > pi\ then\ -1\ else\ 0)$
have $prod\ f\ A \neq 0\ f\ x \neq 0$
by (*auto simp*: *insert.hyps insert.prems*)
with *insert.hyps pi_ge_zero* **show** *?case*
by (*rule_tac x*=n(*x*:=q) **in** *exI*) (*force simp*: *Ln_times q_def D_def n intro*!: *sum.cong*)
qed *auto*

lemma *Ln_minus*:
assumes $z \neq 0$
shows $Ln(-z) = (if\ Im(z) \leq 0 \wedge \neg(Re(z) < 0 \wedge Im(z) = 0)$
$then\ Ln(z) + \mathrm{i} * pi$
$else\ Ln(z) - \mathrm{i} * pi)$
using *Im_Ln_le_pi* [*of z*] *mpi_less_Im_Ln* [*of z*] *assms*
Im_Ln_eq_pi [*of z*] *Im_Ln_pos_lt* [*of z*]
by (*intro Ln_unique*) (*auto simp*: *exp_add exp_diff*)

lemma *Ln_inverse_if*:
assumes $z \neq 0$
shows $Ln\ (inverse\ z) = (if\ z \in \mathbb{R}_{\leq 0}\ then\ -(Ln\ z) + \mathrm{i} * 2 * complex\_of\_real\ pi\ else\ -(Ln\ z))$
proof (*cases* $z \in \mathbb{R}_{\leq 0}$)
case *False* **then show** *?thesis*
by (*simp add*: *Ln_inverse*)
next
case *True*
then have z: $Im\ z = 0\ Re\ z < 0 - z \notin \mathbb{R}_{\leq 0}$
using *assms complex_eq_iff complex_nonpos_Reals_iff* **by** *auto*
have $Ln(inverse\ z) = Ln(-\ (inverse\ (-z)))$
by *simp*
also have $\ldots = Ln\ (inverse\ (-z)) + \mathrm{i} * complex\_of\_real\ pi$
using *assms z* **by** (*simp add*: *Ln_minus divide_less_0_iff*)
also have $\ldots = -\ Ln\ (-\ z) + \mathrm{i} * complex\_of\_real\ pi$
using *z Ln_inverse* **by** *presburger*
also have $\ldots = -\ (Ln\ z) + \mathrm{i} * 2 * complex\_of\_real\ pi$
using *Ln_minus assms z* **by** *auto*
finally show *?thesis* **by** (*simp add*: *True*)
qed

lemma *Ln_times_ii*:
assumes $z \neq 0$
shows $Ln(\mathrm{i} * z) = (if\ 0 \leq Re(z) \mid Im(z) < 0$
$then\ Ln(z) + \mathrm{i} * of\_real\ pi/2$
$else\ Ln(z) - \mathrm{i} * of\_real(3 * pi/2))$
using *Im_Ln_le_pi* [*of z*] *mpi_less_Im_Ln* [*of z*] *assms*
Im_Ln_eq_pi [*of z*] *Im_Ln_pos_lt* [*of z*] *Re_Ln_pos_le* [*of z*]
by (*simp add*: *Ln_times*) *auto*

lemma *Ln_of_nat* [*simp*]: $0 < n \Longrightarrow Ln\ (of\_nat\ n) = of\_real\ (ln\ (of\_nat\ n))$
by (*metis Ln_of_real of_nat_0_less_iff of_real_of_nat_eq*)

lemma *Ln_of_nat_over_of_nat*:
assumes $m > 0\ n > 0$
shows $Ln\ (of\_nat\ m\ /\ of\_nat\ n) = of\_real\ (ln\ (of\_nat\ m) - ln\ (of\_nat\ n))$
proof −
have $of\_nat\ m\ /\ of\_nat\ n = (of\_real\ (of\_nat\ m\ /\ of\_nat\ n) :: complex)$ **by** *simp*
also from *assms* **have** $Ln\ \ldots = of\_real\ (ln\ (of\_nat\ m\ /\ of\_nat\ n))$
by (*simp add*: *Ln_of_real*[*symmetric*])
also from *assms* **have** $\ldots = of\_real\ (ln\ (of\_nat\ m) - ln\ (of\_nat\ n))$
by (*simp add*: *ln_div*)
finally show *?thesis* **.**
qed

lemma *norm_Ln_times_le*:
assumes $w \neq 0\ z \neq 0$
shows $cmod\ (Ln(w * z)) \leq cmod\ (Ln(w) + Ln(z))$
proof (*cases* $- pi < Im(Ln\ w + Ln\ z) \wedge Im(Ln\ w + Ln\ z) \leq pi$)
case *True*
then show *?thesis*
by (*simp add*: *Ln_times_simple assms*)
next
case *False*
then show *?thesis*
by (*smt* (*verit*) *Im_Ln_le_pi assms cmod_Im_le_iff exp_Ln exp_add ln_unique mpi_less_Im_Ln mult_eq_0_iff norm_exp_eq_Re*)
qed

corollary *norm_Ln_prod_le*:
fixes $f :: 'a \Rightarrow complex$
assumes $\bigwedge x.\ x \in A \Longrightarrow f\ x \neq 0$
shows $cmod\ (Ln\ (prod\ f\ A)) \leq (\sum x \in A.\ cmod\ (Ln\ (f\ x)))$
using *assms*
proof (*induction A rule*: *infinite_finite_induct*)
case (*insert x A*)
then show *?case*
by *simp* (*smt* (*verit*) *norm_Ln_times_le norm_triangle_ineq prod_zero_iff*)
qed *auto*

lemma *norm_Ln_exp_le*: $norm\ (Ln\ (exp\ z)) \leq norm\ z$
by (*smt* (*verit*) *Im_Ln_le_pi Ln_exp Re_Ln cmod_Im_le_iff exp_not_eq_zero ln_exp mpi_less_Im_Ln norm_exp_eq_Re*)

6.18.14 Uniform convergence and products

lemma *uniformly_convergent_on_prod_aux*:

```
  fixes f :: nat ⇒ complex ⇒ complex
  assumes norm_f: ⋀n x. x ∈ A ⟹ norm (f n x) < 1
  assumes cont: ⋀n. continuous_on A (f n)
  assumes conv: uniformly_convergent_on A (λN x. ∑ n<N. ln (1 + f n x))
  assumes A: compact A
  shows uniformly_convergent_on A (λN x. ∏ n<N. 1 + f n x)
proof −
  from conv obtain S where S: uniform_limit A (λN x. ∑ n<N. ln (1 + f n
x)) S sequentially
    by (auto simp: uniformly_convergent_on_def)
  have cont': continuous_on A S
  proof (rule uniform_limit_theorem[OF _ S])
    have f n x + 1 ∉ ℝ≤0 if x ∈ A for n x
    proof
      assume f n x + 1 ∈ ℝ≤0
      then obtain t where t: t ≤ 0 f n x = of_real (t − 1)
        by (metis add_diff_cancel nonpos_Reals_cases of_real_1 of_real_diff)
      moreover have norm ... ≥ 1
        using t by (subst norm_of_real) auto
      ultimately show False
        using norm_f[of x n] that by auto
    qed
    thus ∀F n in sequentially. continuous_on A (λx. ∑ n<n. Ln (1 + f n x))
      by (auto intro!: always_eventually continuous_intros cont simp: add_ac)
  qed auto

  define B where B = {x + y |x y. x ∈ S ‘ A ∧ y ∈ cball 0 1}
  have compact B
    unfolding B_def by (intro compact_sums compact_continuous_image cont'
A) auto

  have uniformly_convergent_on A (λN x. exp ((∑ n<N. ln (1 + f n x))))
    using conv
  proof (rule uniformly_convergent_on_compose_uniformly_continuous_on)
    show closed B
      using ‹compact B› by (auto dest: compact_imp_closed)
    show uniformly_continuous_on B exp
      by (intro compact_uniformly_continuous continuous_intros ‹compact B›)

    have eventually (λN. ∀x∈A. dist (∑ n<N. Ln (1 + f n x)) (S x) < 1)
sequentially
      using S unfolding uniform_limit_iff by simp
    thus eventually (λN. ∀x∈A. (∑ n<N. Ln (1 + f n x)) ∈ B) sequentially
    proof eventually_elim
      case (elim N)
      show ∀x∈A. (∑ n<N. Ln (1 + f n x)) ∈ B
      proof safe
        fix x assume x: x ∈ A
        have (∑ n<N. Ln (1 + f n x)) = S x + ((∑ n<N. Ln (1 + f n x)) − S x)
```

```
      by auto
    moreover have ((∑ n<N. Ln (1 + f n x)) − S x) ∈ ball 0 1
      using elim x by (auto simp: dist_norm norm_minus_commute)
    ultimately show (∑ n<N. Ln (1 + f n x)) ∈ B
      unfolding B_def using x by fastforce
  qed
 qed
 qed
 also have ?this ⟷ uniformly_convergent_on A (λN x. ∏ n<N. 1 + f n x)
 proof (intro uniformly_convergent_cong refl always_eventually allI ballI)
  fix N :: nat and x assume x: x ∈ A
  have exp (∑ n<N. ln (1 + f n x)) = (∏ n<N. exp (ln (1 + f n x)))
   by (simp add: exp_sum)
  also have ... = (∏ n<N. 1 + f n x)
   using norm_f[of x] x
  by (smt (verit, best) add.right_neutral add_diff_cancel exp_Ln norm_minus_commute
norm_one prod.cong)
  finally show exp (∑ n<N. ln (1 + f n x)) = (∏ n<N. 1 + f n x) .
 qed
 finally show ?thesis .
qed
```

Theorem 17.6 by Bak and Newman, Complex Analysis [roughly]

```
lemma uniformly_convergent_on_prod:
 fixes f :: nat ⇒ complex ⇒ complex
 assumes cont: ⋀n. continuous_on A (f n)
 assumes A: compact A
 assumes conv_sum: uniformly_convergent_on A (λN x. ∑ n<N. norm (f n x))
 shows   uniformly_convergent_on A (λN x. ∏ n<N. 1 + f n x)
proof −
 obtain M where M: ⋀n x. n ≥ M ⟹ x ∈ A ⟹ norm (f n x) < 1 / 2
 proof −
  from conv_sum have uniformly_Cauchy_on A (λN x. ∑ n<N. norm (f n x))
   using uniformly_convergent_Cauchy by blast
  moreover have (1 / 2 :: real) > 0
   by simp
  ultimately obtain M where M:
    ⋀x m n. x ∈ A ⟹ m ≥ M ⟹ n ≥ M ⟹ dist (∑ k<m. norm (f k x))
(∑ k<n. norm (f k x)) < 1 / 2
   unfolding uniformly_Cauchy_on_def by fast
  show ?thesis
  proof (rule that[of M])
   fix n x assume nx: n ≥ M x ∈ A
   have dist (∑ k<Suc n. norm (f k x)) (∑ k<n. norm (f k x)) < 1 / 2
    by (rule M) (use nx in auto)
   also have dist (∑ k<Suc n. norm (f k x)) (∑ k<n. norm (f k x)) = norm (f
n x)
    by (simp add: dist_norm)
   finally show norm (f n x) < 1 / 2 .
```

```
    qed
  qed

  have conv: uniformly_convergent_on A (λN x. ∑n<N. ln (1 + f (n + M) x))
  proof (rule uniformly_summable_comparison_test)
    show norm (ln (1 + f (n + M) x)) ≤ 2 * norm (f (n + M) x) if x ∈ A for
n x
      by (rule norm_Ln_le) (use M[of n + M x] that in auto)
    have ∗: filterlim (λn. n + M) at_top at_top
      by (rule filterlim_add_const_nat_at_top)
    have uniformly_convergent_on A (λN x. 2 * ((∑n<N+M. norm (f n x)) −
(∑n<M. norm (f n x))))
      by (intro uniformly_convergent_mult uniformly_convergent_minus
          uniformly_convergent_on_compose[OF conv_sum ∗] uniformly_convergent_on_const)
    also have (λN x. 2 * ((∑n<N+M. norm (f n x)) − (∑n<M. norm (f n
x)))) =
              (λN x. ∑n<N. 2 * norm (f (n + M) x)) (is ?lhs = ?rhs)
    proof (intro ext)
      fix N x
    have (∑n<N+M. norm (f n x)) − (∑n<M. norm (f n x)) = (∑n∈{..<N+M}−{..<M}.
norm (f n x))
        by (subst sum_diff) auto
      also have ... = (∑n<N. norm (f (n + M) x))
        by (intro sum.reindex_bij_witness[of _ λn. n + M λn. n − M]) auto
      finally show ?lhs N x = ?rhs N x
        by (simp add: sum_distrib_left)
    qed
    finally show uniformly_convergent_on A (λN x. ∑n<N. 2 * cmod (f (n +
M) x)) .
  qed

  have conv': uniformly_convergent_on A (λN x. ∏n<N. 1 + f (n + M) x)
  proof (rule uniformly_convergent_on_prod_aux)
    show norm (f (n + M) x) < 1 if x ∈ A for n x
      using M[of n + M x] that by simp
  qed (use M assms conv in auto)

  then obtain S where S: uniform_limit A (λN x. ∏n<N. 1 + f (n + M) x)
S sequentially
    by (auto simp: uniformly_convergent_on_def)
  have cont': continuous_on A S
    by (intro uniform_limit_theorem[OF _ S] always_eventually ballI allI contin-
uous_intros cont) auto

  have uniform_limit A (λN x. (∏n<M. 1 + f n x) * (∏n<N. 1 + f (n + M)
x)) (λx. (∏n<M. 1 + f n x) * S x) sequentially
  proof (rule uniform_lim_mult[OF uniform_limit_const S])
    show bounded (S ‘ A)
      by (intro compact_imp_bounded compact_continuous_image A cont')
```

```
    show bounded ((λx. ∏n<M. 1 + f n x) ‘ A)
      by (intro compact_imp_bounded compact_continuous_image A continuous_intros cont)
  qed
  hence uniformly_convergent_on A (λN x. (∏n<M. 1 + f n x) * (∏n<N. 1 + f (n + M) x))
    by (auto simp: uniformly_convergent_on_def)
  also have (λN x. (∏n<M. 1 + f n x) * (∏n<N. 1 + f (n + M) x)) = (λN x. (∏n<M+N. 1 + f n x))
  proof (intro ext)
    fix N :: nat and x :: complex
    have (∏n<N. 1 + f (n + M) x) = (∏n∈{M..<M+N}. 1 + f n x)
      by (intro prod.reindex_bij_witness[of _ λn. n - M λn. n + M]) auto
    also have (∏n<M. 1 + f n x) * ... = (∏n∈{..<M}∪{M..<M+N}. 1 + f n x)
      by (subst prod.union_disjoint) auto
    also have {..<M} ∪ {M..<M+N} = {..<M+N}
      by auto
   finally show (∏n<M. 1 + f n x) * (∏n<N. 1 + f (n + M) x) = (∏n<M+N. 1 + f n x) .
  qed
  finally have uniformly_convergent_on A (λN x. ∏n<M + N. 1 + f n x)
    by simp
  hence uniformly_convergent_on A (λN x. ∏n<M + (N - M). 1 + f n x)
   by (rule uniformly_convergent_on_compose) (rule filterlim_minus_const_nat_at_top)
  also have ?this ⟷ ?thesis
  proof (rule uniformly_convergent_cong)
    show eventually (λx. ∀y∈A. (∏n<M + (x - M). 1 + f n y) = (∏n<x. 1 + f n y)) at_top
      using eventually_ge_at_top[of M] by eventually_elim auto
  qed auto
  finally show ?thesis .
qed

lemma uniformly_convergent_on_prod':
  fixes f :: nat ⇒ complex ⇒ complex
  assumes cont: ⋀n. continuous_on A (f n)
  assumes A: compact A
  assumes conv_sum: uniformly_convergent_on A (λN x. ∑n<N. norm (f n x - 1))
  shows uniformly_convergent_on A (λN x. ∏n<N. f n x)
proof -
  have uniformly_convergent_on A (λN x. ∏n<N. 1 + (f n x - 1))
    by (rule uniformly_convergent_on_prod) (use assms in ‹auto intro!: continuous_intros›)
  thus ?thesis
    by simp
qed
```

Prop 17.6 of Bak and Newman, Complex Analysis, p. 243. Only this version

is for the reals. Can the two proofs be consolidated?

lemma *uniformly_convergent_on_prod_real*:
fixes $u :: nat \Rightarrow real \Rightarrow real$
assumes *contu*: $\bigwedge k.$ *continuous_on* K $(u\ k)$
and *uconv*: *uniformly_convergent_on* K $(\lambda n\ x.\ \sum k{<}n.\ |u\ k\ x|)$
and K: *compact* K
shows *uniformly_convergent_on* K $(\lambda n\ x.\ \prod k{<}n.\ 1 + u\ k\ x)$
proof −
define f **where** $f \equiv \lambda k.$ *complex_of_real* $\circ$ $u\ k \circ$ *Re*
define L **where** $L \equiv$ *complex_of_real* ‘ K
have *compact* L
by (*simp add*: ‹*compact* K› L_*def compact_continuous_image*)
have *Re* ‘ *complex_of_real* ‘ $X = X$ **for** X
by (*auto simp*: *image_iff*)
with *contu* **have** *contf*: $\bigwedge k.$ *continuous_on* L $(f\ k)$
unfolding f_*def* L_*def* **by** (*intro continuous_intros*) *auto*
obtain S **where** S: $\bigwedge \varepsilon.\ \varepsilon{>}0 \Longrightarrow \forall_F\ n$ *in sequentially*. $\forall x{\in}K.$ *dist* $(\sum k{<}n.\ |u\ k\ x|)\ (S\ x) < \varepsilon$
using *uconv* **unfolding** *uniformly_convergent_on_def uniform_limit_iff* **by** *presburger*
have $\forall_F\ n$ *in sequentially*. $\forall z{\in}L.$ *dist* $(\sum k{<}n.\ cmod\ (f\ k\ z))\ ((of\_real \circ S \circ Re)\ z) < \varepsilon$
if $\varepsilon{>}0$ **for** ε
using S [*OF that*] **by** *eventually_elim* (*simp add*: L_*def* f_*def*)
then have *uconvf*: *uniformly_convergent_on* L $(\lambda n\ z.\ \sum k{<}n.\ norm\ (f\ k\ z))$
unfolding *uniformly_convergent_on_def uniform_limit_iff* **by** *blast*
obtain P **where** P: $\bigwedge \varepsilon.\ \varepsilon{>}0 \Longrightarrow \forall_F\ n$ *in sequentially*. $\forall z{\in}L.$ *dist* $(\prod k{<}n.\ 1 + f\ k\ z)\ (P\ z) < \varepsilon$
using *uniformly_convergent_on_prod* [*OF contf* ‹*compact* L› *uconvf*]
unfolding *uniformly_convergent_on_def uniform_limit_iff* **by** *blast*
have §: $|(\prod k{<}n.\ 1 + u\ k\ x) - Re\ (P\ x)| \le cmod\ ((\prod k{<}n.\ 1 + of\_real\ (u\ k\ x)) - P\ x)$ **for** $n\ x$
proof −
have $(\prod k{\in}N.\ of\_real\ (1 + u\ k\ x)) = (\prod k{\in}N.\ 1 + of\_real\ (u\ k\ x))$ **for** N
by *force*
then show *?thesis*
by (*metis Re_complex_of_real abs_Re_le_cmod minus_complex.sel*(1) *of_real_prod*)
qed
have $\forall_F\ n$ *in sequentially*. $\forall x{\in}K.$ *dist* $(\prod k{<}n.\ 1 + u\ k\ x)\ ((Re \circ P \circ of\_real)\ x) < \varepsilon$
if $\varepsilon{>}0$ **for** ε
using P [*OF that*] **by** *eventually_elim* (*simp add*: L_*def* f_*def dist_norm le_less_trans* [*OF* §])
then show *?thesis*
unfolding *uniformly_convergent_on_def uniform_limit_iff* **by** *blast*
qed

6.18.15 The Argument of a Complex Number

Unlike in HOL Light, it's defined for the same interval as the complex logarithm: $(-\pi,\pi]$.

lemma *Arg_eq_Im_Ln*:
  assumes $z \neq 0$ **shows** $Arg\ z = Im\ (Ln\ z)$
proof (*rule cis_Arg_unique*)
  show $sgn\ z = cis\ (Im\ (Ln\ z))$
   by (*metis assms exp_Ln exp_eq_polar nonzero_mult_div_cancel_left norm_eq_zero*
        norm_exp_eq_Re of_real_eq_0_iff sgn_eq)
  show $-\ pi < Im\ (Ln\ z)$
    by (*simp add*: *assms mpi_less_Im_Ln*)
  show $Im\ (Ln\ z) \leq pi$
    by (*simp add*: *Im_Ln_le_pi assms*)
qed

The 1990s definition of argument coincides with the more recent one

lemma *Arg_def*:
  shows $Arg\ z = (if\ z = 0\ then\ 0\ else\ Im\ (Ln\ z))$
  by (*simp add*: *Arg_eq_Im_Ln Arg_zero*)

lemma *Arg_of_real* [*simp*]: $Arg\ (of\_real\ r) = (if\ r<0\ then\ pi\ else\ 0)$
  by (*simp add*: *Im_Ln_eq_pi Arg_def*)

lemma *mpi_less_Arg*: $-pi < Arg\ z$ **and** *Arg_le_pi*: $Arg\ z \leq pi$
  by (*auto simp*: *Arg_def mpi_less_Im_Ln Im_Ln_le_pi*)

lemma *Arg_eq*:
  assumes $z \neq 0$
  shows $z = of\_real(norm\ z) * exp(\mathrm{i} * Arg\ z)$
  using *cis_conv_exp rcis_cmod_Arg rcis_def* **by** *force*

lemma *is_Arg_Arg*: $z \neq 0 \Longrightarrow is\_Arg\ z\ (Arg\ z)$
  by (*simp add*: *Arg_eq is_Arg_def*)

lemma *Argument_exists*:
  assumes $z \neq 0$ **and** *R*: $R = \{r-pi<..r+pi\}$
  obtains *s* **where** *is_Arg z s* $s \in R$
proof −
  let *?rp* $= r - Arg\ z + pi$
  define *k* **where** $k \equiv \lfloor ?rp\ /\ (2 * pi)\rfloor$
  have $(Arg\ z + of\_int\ k * (2 * pi)) \in R$
    using *floor_divide_lower* [*of 2*pi ?rp*] *floor_divide_upper* [*of 2*pi ?rp*]
    by (*auto simp*: *k_def algebra_simps R*)
  then show *?thesis*
    using *Arg_eq* ‹$z \neq 0$› *is_Arg_2pi_iff is_Arg_def that* **by** *blast*
qed

lemma *Argument_exists_unique*:

assumes $z \neq 0$ **and** R: $R = \{r-pi<..r+pi\}$
obtains s **where** *is_Arg z s* $s\in R$ $\bigwedge t.$ $[\![$*is_Arg z t*; $t\in R]\!] \Longrightarrow s=t$
proof −
obtain s **where** s: *is_Arg z s* $s\in R$
using *Argument_exists* [*OF assms*] **.**
moreover have $\bigwedge t.$ $[\![$*is_Arg z t*; $t\in R]\!] \Longrightarrow s=t$
using *assms s* **by** (*auto simp*: *is_Arg_eqI*)
ultimately show *thesis*
using *that* **by** *blast*
qed

lemma *Argument_Ex1*:
assumes $z \neq 0$ **and** R: $R = \{r-pi<..r+pi\}$
shows $\exists!s.$ *is_Arg z s* $\wedge$ $s \in R$
using *Argument_exists_unique* [*OF assms*] **by** *metis*

lemma *Arg_divide*:
assumes $w \neq 0$ $z \neq 0$
shows *is_Arg* (*z* / *w*) (*Arg z* − *Arg w*)
using *Arg_eq* [*of z*] *Arg_eq* [*of w*] *Arg_eq* [*of norm*(*z* / *w*)] *assms*
by (*auto simp*: *is_Arg_def norm_divide field_simps exp_diff Arg_of_real*)

lemma *Arg_unique_lemma*:
assumes *is_Arg z t is_Arg z t′*
and − *pi* < *t* *t* ≤ *pi*
and − *pi* < *t′* *t′* ≤ *pi*
and $z \neq 0$
shows $t' = t$
using *is_Arg_eqI assms* **by** *force*

lemma *complex_norm_eq_1_exp_eq*: *norm z* = 1 ⟷ *exp*(i ∗ (*Arg z*)) = *z*
by (*metis Arg2pi_eq Arg_eq complex_norm_eq_1_exp norm_eq_zero norm_exp_i_times*)

lemma *Arg_unique*: $[\![$*of_real r* ∗ *exp*(i ∗ *a*) = *z*; 0 < *r*; −*pi* < *a*; *a* ≤ *pi*$]\!] \Longrightarrow$
Arg z = *a*
by (*rule Arg_unique_lemma* [*unfolded is_Arg_def*, *OF* _ *Arg_eq*])
(*use mpi_less_Arg Arg_le_pi* **in** ‹*auto simp*: *norm_mult*›)

lemma *Arg_minus*:
assumes $z \neq 0$
shows *Arg* (−*z*) = (*if Arg z* ≤ 0 *then Arg z* + *pi else Arg z* − *pi*)
proof −
have [*simp*]: *cmod z* ∗ *cos* (*Arg z*) = *Re z*
using *assms Arg_eq* [*of z*] **by** (*metis Re_exp exp_Ln norm_exp_eq_Re Arg_def*)
have [*simp*]: *cmod z* ∗ *sin* (*Arg z*) = *Im z*
using *assms Arg_eq* [*of z*] **by** (*metis Im_exp exp_Ln norm_exp_eq_Re Arg_def*)
show *?thesis*

using *mpi_less_Arg* [*of z*] *Arg_le_pi* [*of z*] *assms*
by (*intro Arg_unique* [*of norm z*, *OF complex_eqI*]) (*auto simp*: *Re_exp Im_exp*)
qed

lemma *Arg_1* [*simp*]: *Arg 1 = 0*
by (*rule Arg_unique*[*of 1*]) *auto*

lemma *Arg_numeral* [*simp*]: *Arg* (*numeral n*) = *0*
by (*rule Arg_unique*[*of numeral n*]) *auto*

lemma *Arg_times_of_real* [*simp*]:
assumes *0 < r* **shows** *Arg* (*of_real r* ∗ *z*) = *Arg z*
using *Arg_def Ln_times_of_real assms* **by** *auto*

lemma *Arg_times_of_real2* [*simp*]: *0 < r* ⟹ *Arg* (*z* ∗ *of_real r*) = *Arg z*
by (*metis Arg_times_of_real mult.commute*)

lemma *Arg_divide_of_real* [*simp*]: *0 < r* ⟹ *Arg* (*z* / *of_real r*) = *Arg z*
by (*metis Arg_times_of_real2 less_irrefl nonzero_eq_divide_eq of_real_eq_0_iff*)

lemma *Arg_less_0*: *0* ≤ *Arg z* ⟷ *0* ≤ *Im z*
using *Im_Ln_le_pi Im_Ln_pos_le*
by (*simp add*: *Arg_def*)

converse fails because the argument can equal π.

lemma *Arg_uminus*: *Arg z < 0* ⟹ *Arg* (−*z*) > *0*
by (*smt* (*verit*) *Arg_bounded Arg_minus Complex.Arg_def*)

lemma *Arg_eq_pi*: *Arg z* = *pi* ⟷ *Re z < 0* ∧ *Im z* = *0*
by (*auto simp*: *Arg_def Im_Ln_eq_pi*)

lemma *Arg_lt_pi*: *0 < Arg z* ∧ *Arg z < pi* ⟷ *0 < Im z*
using *Arg_less_0* [*of z*] *Im_Ln_pos_lt*
by (*auto simp*: *order.order_iff_strict Arg_def*)

lemma *Arg_eq_0*: *Arg z* = *0* ⟷ *z* ∈ ℝ ∧ *0* ≤ *Re z*
using *Arg_def Im_Ln_eq_0 complex_eq_iff complex_is_Real_iff* **by** *auto*

corollary *Arg_ne_0*: **assumes** *z* ∉ $\mathbb{R}_{\geq 0}$ **shows** *Arg z* ≠ *0*
using *assms* **by** (*auto simp*: *nonneg_Reals_def Arg_eq_0*)

lemma *Arg_eq_pi_iff*: *Arg z* = *pi* ⟷ *z* ∈ ℝ ∧ *Re z < 0*
using *Arg_eq_pi complex_is_Real_iff* **by** *blast*

lemma *Arg_eq_0_pi*: *Arg z* = *0* ∨ *Arg z* = *pi* ⟷ *z* ∈ ℝ
using *Arg_eq_pi_iff Arg_eq_0* **by** *force*

lemma *Arg_real*: *z* ∈ ℝ ⟹ *Arg z* = (*if 0* ≤ *Re z then 0 else pi*)

using *Arg_eq_0 Arg_eq_0_pi* **by** *auto*

lemma *Arg_inverse*: *Arg*(*inverse z*) = (*if z* ∈ ℝ *then Arg z else* − *Arg z*)
proof (*cases z* ∈ ℝ)
case *False*
then show *?thesis*
by (*simp add*: *Arg_def Ln_inverse complex_is_Real_iff complex_nonpos_Reals_iff*)
qed (*use Arg_real Re_inverse* **in** *auto*)

lemma *Arg_eq_iff*:
assumes *w* ≠ *0 z* ≠ *0*
shows *Arg w* = *Arg z* ⟷ (∃ *x*. *0* < *x* ∧ *w* = *of_real x* ∗ *z*) (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then have *w* = (*cmod w* / *cmod z*) ∗ *z*
by (*metis Arg_eq assms divide_divide_eq_right eq_divide_eq exp_not_eq_zero of_real_divide*)
then show *?rhs*
using *assms divide_pos_pos zero_less_norm_iff* **by** *blast*
qed *auto*

lemma *Arg_inverse_eq_0*: *Arg*(*inverse z*) = *0* ⟷ *Arg z* = *0*
by (*metis Arg_eq_0 Arg_inverse inverse_inverse_eq*)

lemma *Arg_cnj_eq_inverse*: *z*≠*0* ⟹ *Arg* (*cnj z*) = *Arg* (*inverse z*)
using *Arg2pi_cnj_eq_inverse Arg2pi_eq_iff Arg_eq_iff* **by** *auto*

lemma *Arg_cnj*: *Arg*(*cnj z*) = (*if z* ∈ ℝ *then Arg z else* − *Arg z*)
by (*metis Arg_cnj_eq_inverse Arg_inverse Reals_0 complex_cnj_zero*)

lemma *Arg_exp*: −*pi* < *Im z* ⟹ *Im z* ≤ *pi* ⟹ *Arg*(*exp z*) = *Im z*
by (*simp add*: *Arg_eq_Im_Ln*)

lemma *Arg_cis*: *x* ∈ {−*pi*<..*pi*} ⟹ *Arg* (*cis x*) = *x*
unfolding *cis_conv_exp* **by** (*subst Arg_exp*) *auto*

lemma *Arg_rcis*: *x* ∈ {−*pi*<..*pi*} ⟹ *r* > *0* ⟹ *Arg* (*rcis r x*) = *x*
unfolding *rcis_def* **by** (*subst Arg_times_of_real*) (*auto simp*: *Arg_cis*)

lemma *Ln_Arg*: *z*≠*0* ⟹ *Ln*(*z*) = *ln*(*norm z*) + i ∗ *Arg*(*z*)
by (*metis Arg_def Re_Ln complex_eq*)

lemma *continuous_at_Arg*:
assumes *z* ∉ ℝ≤0
shows *continuous* (*at z*) *Arg*
proof −
have (λ*z*. *Im* (*Ln z*)) −*z*→ *Arg z*
using *Arg_def assms continuous_at* **by** *fastforce*
then show *?thesis*

unfolding *continuous_at*
by (*smt* (*verit*, *del_insts*) *Arg_eq_Im_Ln Lim_transform_away_at assms nonpos_Reals_zero_I*)
qed

lemma *continuous_within_Arg*: $z \notin \mathbb{R}_{\leq 0} \Longrightarrow$ *continuous* (*at z within S*) *Arg*
using *continuous_at_Arg continuous_at_imp_continuous_within* **by** *blast*

lemma *Arg_Re_pos*: $|Arg\ z| < pi\ /\ 2 \longleftrightarrow Re\ z > 0 \vee z = 0$
using *Arg_def Re_Ln_pos_lt* **by** *auto*

lemma *Arg_Re_nonneg*: $|Arg\ z| \leq pi\ /\ 2 \longleftrightarrow Re\ z \geq 0$
using *Re_Ln_pos_le*[*of z*] **by** (*cases z = 0*) (*auto simp*: *Arg_eq_Im_Ln Arg_zero*)

lemma *Arg_times*:
assumes $Arg\ z + Arg\ w \in \{-pi<..pi\}$ $z \neq 0$ $w \neq 0$
shows $Arg\ (z * w) = Arg\ z + Arg\ w$
using *Arg_eq_Im_Ln Ln_times_simple assms* **by** *auto*

6.18.16 The Unwinding Number and the Ln product Formula

Note that in this special case the unwinding number is -1, 0 or 1. But it's always an integer.

lemma *is_Arg_exp_Im*: *is_Arg* (*exp z*) (*Im z*)
using *exp_eq_polar is_Arg_def norm_exp_eq_Re* **by** *auto*

lemma *is_Arg_exp_diff_2pi*:
assumes *is_Arg* (*exp z*) ϑ
shows $\exists k.\ Im\ z - of\_int\ k * (2 * pi) = \vartheta$
proof (*intro exI is_Arg_eqI*)
let *?k* = $\lfloor (Im\ z - \vartheta)\ /\ (2 * pi) \rfloor$
show *is_Arg* (*exp z*) (*Im z* − *real_of_int ?k* ∗ (2 ∗ *pi*))
by (*metis diff_add_cancel is_Arg_2pi_iff is_Arg_exp_Im*)
show $|Im\ z - real\_of\_int\ ?k * (2 * pi) - \vartheta| < 2 * pi$
using *floor_divide_upper* [*of 2∗pi Im z* − ϑ] *floor_divide_lower* [*of 2∗pi Im z* − ϑ]
by (*auto simp*: *algebra_simps abs_if*)
qed (*auto simp*: *is_Arg_exp_Im assms*)

lemma *Arg_exp_diff_2pi*: $\exists k.\ Im\ z - of\_int\ k * (2 * pi) = Arg\ (exp\ z)$
using *is_Arg_exp_diff_2pi* [*OF is_Arg_Arg*] **by** *auto*

lemma *unwinding_in_Ints*: $(z - Ln(exp\ z))\ /\ (of\_real(2{*}pi) * \mathrm{i}) \in \mathbb{Z}$
using *Arg_exp_diff_2pi* [*of z*]
by (*force simp*: *Ints_def image_def field_simps Arg_def intro*!: *complex_eqI*)

definition *unwinding* :: *complex* ⇒ *int* **where**

unwinding z ≡ *THE k. of_int k* = (*z* − *Ln*(*exp z*)) / (*of_real*(2∗*pi*) ∗ i)

lemma *unwinding*: *of_int* (*unwinding z*) = (*z* − *Ln*(*exp z*)) / (*of_real*(2∗*pi*) ∗ i)
using *unwinding_in_Ints* [*of z*]
unfolding *unwinding_def Ints_def* **by** *force*

lemma *unwinding_2pi*: (2∗*pi*) ∗ i ∗ *unwinding*(*z*) = *z* − *Ln*(*exp z*)
by (*simp add*: *unwinding*)

lemma *Ln_times_unwinding*:
w ≠ *0* ⟹ *z* ≠ *0* ⟹ *Ln*(*w* ∗ *z*) = *Ln*(*w*) + *Ln*(*z*) − (2∗*pi*) ∗ i ∗ *unwinding*(*Ln w* + *Ln z*)
using *unwinding_2pi* **by** (*simp add*: *exp_add*)

lemma *arg_conv_arctan*:
assumes *Re z* > *0*
shows *Arg z* = *arctan* (*Im z* / *Re z*)
proof (*rule cis_Arg_unique*)
show *sgn z* = *cis* (*arctan* (*Im z* / *Re z*))
proof (*rule complex_eqI*)
have *Re* (*cis* (*arctan* (*Im z* / *Re z*))) = *1* / *sqrt* (*1* + (*Im z*)^2 / (*Re z*)^2)
by (*simp add*: *cos_arctan power_divide*)
also have *1* + *Im z* ^ *2* / *Re z* ^ *2* = *norm z* ^ *2* / *Re z* ^ *2*
using *assms* **by** (*simp add*: *cmod_def field_simps*)
also have *1* / *sqrt* ... = *Re z* / *norm z*
using *assms* **by** (*simp add*: *real_sqrt_divide*)
finally show *Re* (*sgn z*) = *Re* (*cis* (*arctan* (*Im z* / *Re z*)))
by *simp*
next
have *Im* (*cis* (*arctan* (*Im z* / *Re z*))) = *Im z* / (*Re z* ∗ *sqrt* (*1* + (*Im z*)^2 / (*Re z*)^2))
by (*simp add*: *sin_arctan field_simps*)
also have *1* + *Im z* ^ *2* / *Re z* ^ *2* = *norm z* ^ *2* / *Re z* ^ *2*
using *assms* **by** (*simp add*: *cmod_def field_simps*)
also have *Im z* / (*Re z* ∗ *sqrt* ...) = *Im z* / *norm z*
using *assms* **by** (*simp add*: *real_sqrt_divide*)
finally show *Im* (*sgn z*) = *Im* (*cis* (*arctan* (*Im z* / *Re z*)))
by *simp*
qed
next
show *arctan* (*Im z* / *Re z*) > −*pi*
by (*smt* (*verit*, *ccfv_SIG*) *arctan field_sum_of_halves*)
next
show *arctan* (*Im z* / *Re z*) ≤ *pi*
by (*smt* (*verit*, *best*) *arctan field_sum_of_halves*)
qed

6.18.17 Characterisation of *Im (Ln z)* (Wenda Li)

lemma *Im_Ln_eq_pi_half*:
  $z \neq 0 \Longrightarrow (Im(Ln\ z) = pi/2 \longleftrightarrow 0 < Im(z) \wedge Re(z) = 0)$
  $z \neq 0 \Longrightarrow (Im(Ln\ z) = -pi/2 \longleftrightarrow Im(z) < 0 \wedge Re(z) = 0)$
 using *Im_Ln_pos_lt Im_Ln_pos_le Re_Ln_pos_le Re_Ln_pos_lt pi_ge_two*
by *fastforce+*

lemma *Im_Ln_eq*:
 assumes *z≠0*
 shows *Im (Ln z) = (if Re z≠0 then*
 if Re z>0 then
 arctan (Im z/Re z)
 else if Im z≥0 then
 arctan (Im z/Re z) + pi
 else
 arctan (Im z/Re z) − pi
 else
 if Im z>0 then pi/2 else −pi/2)
proof −
 have *eq_arctan_pos*: *Im (Ln z) = arctan (Im z/Re z)* **when** *Re z>0* **for** *z*
  by (*metis Arg_eq_Im_Ln arg_conv_arctan order_less_irrefl that zero_complex.simps(1)*)
 have *?thesis* **when** *Re z=0*
  using *Im_Ln_eq_pi_half*[*OF ‹z≠0›*] *that*
  using *assms complex_eq_iff* **by** *auto*
 moreover have *?thesis* **when** *Re z>0*
  using *eq_arctan_pos*[*OF that*] *that* **by** *auto*
 moreover have *?thesis* **when** *Re z<0 Im z≥0*
 proof −
  have *Im (Ln (− z)) = arctan (Im (− z) / Re (− z))*
   by (*simp add*: *eq_arctan_pos that(1)*)
  moreover have *Ln (− z) = Ln z − i ∗ complex_of_real pi*
   using *Ln_minus assms that* **by** *fastforce*
  ultimately show *?thesis* **using** *that* **by** *auto*
 qed
 moreover have *?thesis* **when** *Re z<0 Im z<0*
 proof −
  have *Im (Ln (− z)) = arctan (Im (− z) / Re (− z))*
   by (*simp add*: *eq_arctan_pos that(1)*)
  moreover have *Ln (− z) = Ln z + i ∗ complex_of_real pi*
   using *Ln_minus assms that* **by** *auto*
  ultimately show *?thesis* **using** *that* **by** *auto*
 qed
 ultimately show *?thesis* **by** *linarith*
qed

6.18.18 Relation between Ln and Arg2pi, and hence continuity of Arg2pi

lemma *Arg2pi_Ln*: $0 < Arg2pi\ z \Longrightarrow Arg2pi\ z = Im(Ln(-z)) + pi$

by (*smt* (*verit*, *best*) *Arg2pi_0 Arg2pi_exp Arg2pi_minus Arg_exp Arg_minus Im_Ln_le_pi*
exp_Ln mpi_less_Im_Ln neg_equal_0_iff_equal)

lemma *continuous_at_Arg2pi*:
assumes $z \notin \mathbb{R}_{\geq 0}$
shows *continuous* (*at z*) *Arg2pi*
proof −
have *isCont* (λz. *Im* (*Ln* (− *z*)) + *pi*) *z*
by (*rule Complex.isCont_Im isCont_Ln′ continuous_intros* | *simp add*: *assms complex_is_Real_iff*)+
moreover consider *Re z* < *0* | *Im z* ≠ *0* **using** *assms*
using *complex_nonneg_Reals_iff not_le* **by** *blast*
ultimately have (λz. *Im* (*Ln* (− *z*)) + *pi*) −*z*→ *Arg2pi z*
by (*simp add*: *Arg2pi_Ln Arg2pi_gt_0 assms continuous_within*)
then show *?thesis*
unfolding *continuous_at*
by (*metis* (*mono_tags*, *lifting*) *Arg2pi_Ln Arg2pi_gt_0 Compl_iff Lim_transform_within_open assms*
closed_nonneg_Reals_complex open_Compl)
qed

Relation between Arg2pi and arctangent in upper halfplane

lemma *Arg2pi_arctan_upperhalf*:
assumes *0* < *Im z*
shows *Arg2pi z* = *pi*/*2* − *arctan*(*Re z* / *Im z*)
proof (*cases z* = *0*)
case *False*
show *?thesis*
proof (*rule Arg2pi_unique* [*of norm z*])
show (*cmod z*) ∗ *exp* (i ∗ (*pi* / *2* − *arctan* (*Re z* / *Im z*))) = *z*
apply (*rule complex_eqI*)
using *assms norm_complex_def* [*of z*, *symmetric*]
unfolding *exp_Euler cos_diff sin_diff sin_of_real cos_of_real*
by (*simp_all add*: *field_simps real_sqrt_divide sin_arctan cos_arctan*)
qed (*use False arctan* [*of Re z* / *Im z*] **in** *auto*)
qed (*use assms* **in** *auto*)

lemma *Arg2pi_eq_Im_Ln*:
assumes *0* ≤ *Im z 0* < *Re z*
shows *Arg2pi z* = *Im* (*Ln z*)
by (*smt* (*verit*, *ccfv_SIG*) *Arg2pi_exp Im_Ln_pos_le assms exp_Ln pi_neq_zero zero_complex.simps*(*1*))

lemma *continuous_within_upperhalf_Arg2pi*:
assumes *z* ≠ *0*
shows *continuous* (*at z within* {*z*. *0* ≤ *Im z*}) *Arg2pi*
proof (*cases z* ∈ $\mathbb{R}_{\geq 0}$)
case *False* **then show** *?thesis*

```
    using continuous_at_Arg2pi continuous_at_imp_continuous_within by auto
next
  case True
  then have z: z ∈ ℝ 0 < Re z
    using assms  by (auto simp: complex_nonneg_Reals_iff complex_is_Real_iff
complex_neq_0)
  then have [simp]: Arg2pi z = 0 Im (Ln z) = 0
    by (auto simp: Arg2pi_eq_0 Im_Ln_eq_0 assms complex_is_Real_iff)
  show ?thesis
  proof (clarsimp simp add: continuous_within Lim_within dist_norm)
    fix e::real
    assume 0 < e
    moreover have continuous (at z) (λx. Im (Ln x))
      using z by (simp add: continuous_at_Ln complex_nonpos_Reals_iff)
    ultimately
    obtain d where d: d>0 ⋀x. x ≠ z ⟹ cmod (x − z) < d ⟹ |Im (Ln x)| <
e
      by (auto simp: continuous_within Lim_within dist_norm)
    { fix x
      assume cmod (x − z) < Re z / 2
      then have |Re x − Re z| < Re z / 2
        by (metis le_less_trans abs_Re_le_cmod minus_complex.simps(1))
      then have 0 < Re x
        using z by linarith
    }
    then show ∃ d>0. ∀ x. 0 ≤ Im x ⟶ x ≠ z ∧ cmod (x − z) < d ⟶ |Arg2pi
x| < e
      apply (rule_tac x=min d (Re z / 2) in exI)
      using z d by (auto simp: Arg2pi_eq_Im_Ln)
  qed
qed

lemma continuous_on_upperhalf_Arg2pi: continuous_on ({z. 0 ≤ Im z} − {0})
Arg2pi
  unfolding continuous_on_eq_continuous_within
 by (metis DiffE Diff_subset continuous_within_subset continuous_within_upperhalf_Arg2pi
insertCI)

lemma open_Arg2pi2pi_less_Int:
  assumes 0 ≤ s t ≤ 2∗pi
    shows open ({y. s < Arg2pi y} ∩ {y. Arg2pi y < t})
proof −
  have 1: continuous_on (UNIV − ℝ≥0) Arg2pi
    using continuous_at_Arg2pi continuous_at_imp_continuous_within
    by (auto simp: continuous_on_eq_continuous_within)
  have 2: open (UNIV − ℝ≥0 :: complex set)  by (simp add: open_Diff)
  have open ({z. s < z} ∩ {z. z < t})
    using open_lessThan [of t] open_greaterThan [of s]
    by (metis greaterThan_def lessThan_def open_Int)
```

moreover have $\{y.\ s < Arg2pi\ y\} \cap \{y.\ Arg2pi\ y < t\} \subseteq - \mathbb{R}_{\geq 0}$
using *assms* **by** (*auto simp*: *Arg2pi_real complex_nonneg_Reals_iff complex_is_Real_iff*)
ultimately show *?thesis*
using *continuous_imp_open_vimage* [*OF 1 2*, *of* $\{z.\ Re\ z > s\} \cap \{z.\ Re\ z < t\}$]
by *auto*
qed

lemma *open_Arg2pi2pi_gt*: *open* $\{z.\ t < Arg2pi\ z\}$
proof (*cases* $t < 0$)
case *True* **then have** $\{z.\ t < Arg2pi\ z\} = UNIV$
using *Arg2pi_ge_0 less_le_trans* **by** *auto*
then show *?thesis*
by *simp*
next
case *False* **then show** *?thesis*
using *open_Arg2pi2pi_less_Int* [*of t 2∗pi*] *Arg2pi_lt_2pi*
by *auto*
qed

lemma *closed_Arg2pi2pi_le*: *closed* $\{z.\ Arg2pi\ z \leq t\}$
using *open_Arg2pi2pi_gt* [*of t*]
by (*simp add*: *closed_def Set.Collect_neg_eq* [*symmetric*] *not_le*)

6.18.19 Complex Powers

lemma *powr_to_1* [*simp*]: *z powr 1* = (*z*::*complex*)
by (*simp add*: *powr_def*)

lemma *powr_nat*:
fixes *n*::*nat* **and** *z*::*complex* **shows** *z powr n* = (*if z* = *0 then 0 else z^n*)
by (*simp add*: *exp_of_nat_mult powr_def*)

lemma *powr_nat′*: $(z :: complex) \neq 0 \vee n \neq 0 \Longrightarrow z\ powr\ of\_nat\ n = z \hat{}\ n$
by (*cases z* = *0*) (*auto simp*: *powr_nat*)

lemma *norm_powr_real*: $w \in \mathbb{R} \Longrightarrow 0 < Re\ w \Longrightarrow norm(w\ powr\ z) = exp(Re\ z * ln(Re\ w))$
using *Ln_Reals_eq norm_exp_eq_Re* **by** (*auto simp*: *Im_Ln_eq_0 powr_def norm_complex_def*)

lemma *norm_powr_real_powr′*: $w \in \mathbb{R} \Longrightarrow norm\ (z\ powr\ w) = norm\ z\ powr\ Re\ w$
by (*auto simp*: *powr_def Reals_def*)

lemma *powr_complexpow* [*simp*]:
fixes *x*::*complex* **shows** $x \neq 0 \Longrightarrow x\ powr\ (of\_nat\ n) = x \hat{} n$
by (*simp add*: *powr_nat′*)

lemma *powr_complexnumeral* [*simp*]:
fixes *x*::*complex* **shows** *x powr* (*numeral n*) = *x* ^ (*numeral n*)
by (*metis of_nat_numeral power_zero_numeral powr_nat*)

lemma *cnj_powr*:
assumes *Im a* = *0* ⟹ *Re a* ≥ *0*
shows *cnj* (*a powr b*) = *cnj a powr cnj b*
proof (*cases a* = *0*)
case *False*
with *assms* **have** *a* ∉ $\mathbb{R}_{\leq 0}$ **by** (*auto simp*: *complex_eq_iff complex_nonpos_Reals_iff*)
with *False* **show** *?thesis* **by** (*simp add*: *powr_def exp_cnj cnj_Ln*)
qed *simp*

lemma *powr_real_real*:
assumes *w* ∈ ℝ *z* ∈ ℝ *0* < *Re w*
shows *w powr z* = *exp*(*Re z* ∗ *ln*(*Re w*))
proof −
have *w* ≠ *0*
using *assms* **by** *auto*
with *assms* **show** *?thesis*
by (*simp add*: *powr_def Ln_Reals_eq of_real_exp*)
qed

lemma *powr_of_real*:
fixes *x*::*real* **and** *y*::*real*
shows *0* ≤ *x* ⟹ *of_real x powr* (*of_real y*::*complex*) = *of_real* (*x powr y*)
by (*simp_all add*: *powr_def exp_eq_polar*)

lemma *powr_of_int*:
fixes *z*::*complex* **and** *n*::*int*
assumes *z*≠(*0*::*complex*)
shows *z powr of_int n* = (*if n*≥*0 then z*^*nat n else inverse* (*z*^*nat* (−*n*)))
by (*metis assms not_le of_int_of_nat powr_complexpow powr_minus*)

lemma *complex_powr_of_int*: *z* ≠ *0* ∨ *n* ≠ *0* ⟹ *z powr of_int n* = (*z* :: *complex*) *powi n*
by (*cases z* = *0* ∨ *n* = *0*)
(*auto simp*: *power_int_def powr_minus powr_nat powr_of_int power_0_left power_inverse*)

lemma *powr_Reals_eq*: ⟦*x* ∈ ℝ; *y* ∈ ℝ; *Re x* ≥ *0*⟧ ⟹ *x powr y* = *of_real* (*Re x powr Re y*)
by (*metis of_real_Re powr_of_real*)

lemma *norm_powr_real_mono*:
⟦*w* ∈ ℝ; *1* < *Re w*⟧ ⟹ *cmod*(*w powr z1*) ≤ *cmod*(*w powr z2*) ⟷ *Re z1* ≤ *Re z2*
by (*auto simp*: *powr_def algebra_simps Reals_def Ln_of_real*)

lemma *powr_times_real*:
  $[\![x \in \mathbb{R};\ y \in \mathbb{R};\ 0 \le Re\ x;\ 0 \le Re\ y]\!]$
    $\Longrightarrow (x * y)\ powr\ z = x\ powr\ z * y\ powr\ z$
 by (*auto simp*: *Reals_def powr_def Ln_times exp_add algebra_simps less_eq_real_def Ln_of_real*)

lemma *Re_powr_le*: $r \in \mathbb{R}_{\ge 0} \Longrightarrow Re\ (r\ powr\ z) \le Re\ r\ powr\ Re\ z$
 by (*auto simp*: *powr_def nonneg_Reals_def order_trans* [*OF complex_Re_le_cmod*])

lemma
 fixes *w*::*complex*
 assumes $w \in \mathbb{R}_{\ge 0}$ $z \in \mathbb{R}$
  shows *Reals_powr* [*simp*]: *w powr z* $\in \mathbb{R}$ **and** *nonneg_Reals_powr* [*simp*]: *w powr z* $\in \mathbb{R}_{\ge 0}$
 using *assms* **by** (*auto simp*: *nonneg_Reals_def Reals_def powr_of_real*)

lemma *powr_neg_real_complex*:
 fixes *w*::*complex*
 shows *(− of_real x) powr w = (−1) powr (of_real (sgn x) * w) * of_real x powr w*
proof (*cases x = 0*)
 assume *x*: *x* ≠ *0*
 hence *(−x) powr w = exp (w * ln (−of_real x))* **by** (*simp add*: *powr_def*)
 also from *x* **have** *ln (−of_real x) = Ln (of_real x) + of_real (sgn x) * pi * i*
   by (*simp add*: *Ln_minus Ln_of_real*)
 also from *x* **have** *exp (w * ...) = cis pi powr (of_real (sgn x) * w) * of_real x powr w*
   by (*simp add*: *powr_def exp_add algebra_simps Ln_of_real cis_conv_exp*)
 also note *cis_pi*
 finally show *?thesis* **by** *simp*
qed *simp_all*

lemma *has_field_derivative_powr*:
 fixes *z* :: *complex*
 assumes $z \notin \mathbb{R}_{\le 0}$
 shows *((λz. z powr s) has_field_derivative (s * z powr (s − 1))) (at z)*
proof (*cases z=0*)
 case *False*
 then have §: *exp (s * Ln z) * inverse z = exp ((s − 1) * Ln z)*
   by (*simp add*: *divide_complex_def exp_diff left_diff_distrib'*)
 show *?thesis*
   unfolding *powr_def*
 proof (*rule has_field_derivative_transform_within*)
   show *((λz. exp (s * Ln z)) has_field_derivative s * (if z = 0 then 0 else exp ((s − 1) * Ln z)))*
       (at z)
     by (*intro derivative_eq_intros* | *simp add*: *assms False* §)+
 qed (*use False* **in** *auto*)

qed (*use assms* **in** *auto*)

declare *has_field_derivative_powr*[*THEN DERIV_chain2*, *derivative_intros*]

lemma *has_field_derivative_powr_of_int*:
fixes *z* :: *complex*
assumes *gderiv*: (*g has_field_derivative gd*) (*at z within S*) **and** *g z* ≠ *0*
shows ((λ*z*. *g z powr of_int n*) *has_field_derivative* (*n* ∗ *g z powr* (*of_int n* − *1*) ∗ *gd*)) (*at z within S*)
proof −
obtain *e* **where** *e*>*0* **and** *e_dist*: ∀ *y*∈*S*. *dist z y* < *e* ⟶ *g y* ≠ *0*
using *DERIV_continuous assms continuous_within_avoid gderiv* **by** *blast*
define *D* **where** *D* = *of_int n* ∗ *g z powr* (*of_int* (*n* − *1*)) ∗ *gd*
define *E* **where** *E* = *of_int n* ∗ *g z powi* (*n* − *1*) ∗ *gd*
have ((λ*z*. *g z powr of_int n*) *has_field_derivative D*) (*at z within S*)
⟷ ((λ*z*. *g z powr of_int n*) *has_field_derivative E*) (*at z within S*)
using *assms complex_powr_of_int D_def E_def* **by** *presburger*
also have ... ⟷ ((λ*z*. *g z powi n*) *has_field_derivative E*) (*at z within S*)
proof (*rule has_field_derivative_cong_eventually*)
show ∀ *F x in at z within S*. *g x powr of_int n* = *g x powi n*
unfolding *eventually_at* **by** (*metis* ‹*0* < *e*› *complex_powr_of_int dist_commute e_dist*)
qed (*simp add*: *assms complex_powr_of_int*)
also have ((λ*z*. *g z powi n*) *has_field_derivative E*) (*at z within S*)
unfolding *E_def* **using** *gderiv assms* **by** (*auto intro*!: *derivative_eq_intros*)
finally show *?thesis*
by (*simp add*: *D_def*)
qed

lemma *field_differentiable_powr_of_int*:
fixes *z* :: *complex*
assumes *g field_differentiable* (*at z within S*) **and** *g z* ≠ *0*
shows (λ*z*. *g z powr of_int n*) *field_differentiable* (*at z within S*)
using *has_field_derivative_powr_of_int assms field_differentiable_def* **by** *blast*

lemma *holomorphic_on_powr_of_int* [*holomorphic_intros*]:
assumes *f holomorphic_on S* **and** ⋀*z*. *z*∈*S* ⟹ *f z* ≠ *0*
shows (λ*z*. (*f z*) *powr of_int n*) *holomorphic_on S*
using *assms field_differentiable_powr_of_int holomorphic_on_def* **by** *auto*

lemma *has_field_derivative_powr_right* [*derivative_intros*]:
w ≠ *0* ⟹ ((λ*z*. *w powr z*) *has_field_derivative Ln w* ∗ *w powr z*) (*at z*)
unfolding *powr_def* **by** (*intro derivative_eq_intros* | *simp*)+

lemma *field_differentiable_powr_right* [*derivative_intros*]:
fixes *w*::*complex*
shows *w* ≠ *0* ⟹ (λ*z*. *w powr z*) *field_differentiable* (*at z*)
using *field_differentiable_def has_field_derivative_powr_right* **by** *blast*

lemma *holomorphic_on_powr_right* [*holomorphic_intros*]:
assumes *f holomorphic_on S*
shows (λ*z*. *w powr* (*f z*)) *holomorphic_on S*
proof (*cases w* = *0*)
case *False*
with *assms* **show** *?thesis*
unfolding *holomorphic_on_def field_differentiable_def*
by (*metis* (*full_types*) *DERIV_chain'* *has_field_derivative_powr_right*)
qed *simp*

lemma *holomorphic_on_divide_gen* [*holomorphic_intros*]:
assumes *f holomorphic_on S g holomorphic_on S* **and** ⋀*z z'*. ⟦*z* ∈ *S*; *z'* ∈ *S*⟧
⟹ *g z* = *0* ⟷ *g z'* = *0*
shows (λ*z*. *f z* / *g z*) *holomorphic_on S*
by (*metis* (*no_types*, *lifting*) *assms division_ring_divide_zero holomorphic_on_divide*
holomorphic_transform)

lemma *norm_powr_real_powr*:
w ∈ ℝ ⟹ *0* ≤ *Re w* ⟹ *cmod* (*w powr z*) = *Re w powr Re z*
by (*metis dual_order.order_iff_strict norm_powr_real norm_zero of_real_0*
of_real_Re powr_def)

lemma *tendsto_powr_complex*:
fixes *f g* :: *_* ⇒ *complex*
assumes *a*: *a* ∉ ℝ≤0
assumes *f*: (*f* ⟶ *a*) *F* **and** *g*: (*g* ⟶ *b*) *F*
shows ((λ*z*. *f z powr g z*) ⟶ *a powr b*) *F*
proof −
from *a* **have** [*simp*]: *a* ≠ *0* **by** *auto*
from *f g a* **have** ((λ*z*. *exp* (*g z* ∗ *ln* (*f z*))) ⟶ *a powr b*) *F* (**is** *?P*)
by (*auto intro*!: *tendsto_intros simp*: *powr_def*)
also **{**
have *eventually* (λ*z*. *z* ≠ *0*) (*nhds a*)
by (*intro t1_space_nhds*) *simp_all*
with *f* **have** *eventually* (λ*z*. *f z* ≠ *0*) *F* **using** *filterlim_iff* **by** *blast*
}
hence *?P* ⟷ ((λ*z*. *f z powr g z*) ⟶ *a powr b*) *F*
by (*intro tendsto_cong refl*) (*simp_all add*: *powr_def mult_ac*)
finally show *?thesis* **.**
qed

lemma *tendsto_powr_complex_0*:
fixes *f g* :: *'a* ⇒ *complex*
assumes *f*: (*f* ⟶ *0*) *F* **and** *g*: (*g* ⟶ *b*) *F* **and** *b*: *Re b* > *0*
shows ((λ*z*. *f z powr g z*) ⟶ *0*) *F*
proof (*rule tendsto_norm_zero_cancel*)
define *h* **where**
h = (λ*z*. *if f z* = *0 then 0 else exp* (*Re* (*g z*) ∗ *ln* (*cmod* (*f z*)) + *abs* (*Im* (*g z*))

```
* pi))
  {
    fix z :: 'a assume z: f z ≠ 0
    define c where c = abs (Im (g z)) * pi
    from mpi_less_Im_Ln[OF z] Im_Ln_le_pi[OF z]
      have abs (Im (Ln (f z))) ≤ pi by simp
    from mult_left_mono[OF this, of abs (Im (g z))]
      have abs (Im (g z)) * Im (ln (f z))) ≤ c by (simp add: abs_mult c_def)
    hence -Im (g z) * Im (ln (f z)) ≤ c by simp
    hence norm (f z powr g z) ≤ h z by (simp add: powr_def field_simps h_def
c_def)
  }
  hence le: norm (f z powr g z) ≤ h z for z
    by (simp add: h_def)

  have g': (g ⟶ b) (inf F (principal {z. f z ≠ 0}))
    by (rule tendsto_mono[OF _ g]) simp_all
  have ((λx. norm (f x)) ⟶ 0) (inf F (principal {z. f z ≠ 0}))
    by (subst tendsto_norm_zero_iff, rule tendsto_mono[OF _ f]) simp_all
  moreover {
    have filterlim (λx. norm (f x)) (principal {0<..}) (principal {z. f z ≠ 0})
      by (auto simp: filterlim_def)
    hence filterlim (λx. norm (f x)) (principal {0<..}) (inf F (principal {z. f z ≠
0}))
      by (rule filterlim_mono) simp_all
  }
  ultimately have norm: filterlim (λx. norm (f x)) (at_right 0) (inf F (principal
{z. f z ≠ 0}))
    by (simp add: filterlim_inf at_within_def)

  have A: LIM x inf F (principal {z. f z ≠ 0}). Re (g x) * -ln (cmod (f x)) :>
at_top
    by (rule filterlim_tendsto_pos_mult_at_top tendsto_intros g' b
      filterlim_compose[OF filterlim_uminus_at_top_at_bot] filterlim_compose[OF
ln_at_0] norm)+
  have B: LIM x inf F (principal {z. f z ≠ 0}).
        -|Im (g x)| * pi + -(Re (g x) * ln (cmod (f x))) :> at_top
   by (rule filterlim_tendsto_add_at_top tendsto_intros g')+ (insert A, simp_all)
  have C: (h ⟶ 0) F unfolding h_def
    by (intro filterlim_If tendsto_const filterlim_compose[OF exp_at_bot])
      (insert B, auto simp: filterlim_uminus_at_bot algebra_simps)
  show ((λx. norm (f x powr g x)) ⟶ 0) F
    by (rule Lim_null_comparison[OF always_eventually C]) (insert le, auto)
qed

lemma tendsto_powr_complex' [tendsto_intros]:
  fixes f g :: _ ⇒ complex
  assumes a ∉ ℝ≤0 ∨ (a = 0 ∧ Re b > 0) and (f ⟶ a) F (g ⟶ b) F
  shows   ((λz. f z powr g z) ⟶ a powr b) F
```

using *assms tendsto_powr_complex tendsto_powr_complex_0* **by** *fastforce*

lemma *tendsto_neg_powr_complex_of_real*:
assumes *filterlim f at_top F* **and** *Re s < 0*
shows $((\lambda x.\ complex\_of\_real\ (f\ x)\ powr\ s) \longrightarrow 0)\ F$
proof −
have $((\lambda x.\ norm\ (complex\_of\_real\ (f\ x)\ powr\ s)) \longrightarrow 0)\ F$
proof (*rule Lim_transform_eventually*)
from *assms*(*1*) **have** *eventually* $(\lambda x.\ f\ x \geq 0)\ F$
by (*auto simp*: *filterlim_at_top*)
thus *eventually* $(\lambda x.\ f\ x\ powr\ Re\ s = norm\ (of\_real\ (f\ x)\ powr\ s))\ F$
by *eventually_elim* (*simp add*: *norm_powr_real_powr*)
from *assms* **show** $((\lambda x.\ f\ x\ powr\ Re\ s) \longrightarrow 0)\ F$
by (*intro tendsto_neg_powr*)
qed
thus *?thesis* **by** (*simp add*: *tendsto_norm_zero_iff*)
qed

lemma *tendsto_neg_powr_complex_of_nat*:
assumes *filterlim f at_top F* **and** *Re s < 0*
shows $((\lambda x.\ of\_nat\ (f\ x)\ powr\ s) \longrightarrow 0)\ F$
using *tendsto_neg_powr_complex_of_real* [*of real o f F s*]
proof −
have $((\lambda x.\ of\_real\ (real\ (f\ x))\ powr\ s) \longrightarrow 0)\ F$ **using** *assms*(*2*)
by (*intro filterlim_compose*[*OF _ tendsto_neg_powr_complex_of_real*]
filterlim_compose[*OF _ assms*(*1*)] *filterlim_real_sequentially filterlim_ident*) *auto*
thus *?thesis* **by** *simp*
qed

lemma *continuous_powr_complex*:
assumes *f* (*netlimit F*) $\notin \mathbb{R}_{\leq 0}$ *continuous F f continuous F g*
shows *continuous F* $(\lambda z.\ f\ z\ powr\ g\ z :: complex)$
using *assms* **unfolding** *continuous_def* **by** (*intro tendsto_powr_complex*) *simp_all*

lemma *isCont_powr_complex* [*continuous_intros*]:
assumes *f z* $\notin \mathbb{R}_{\leq 0}$ *isCont f z isCont g z*
shows *isCont* $(\lambda z.\ f\ z\ powr\ g\ z :: complex)\ z$
using *assms* **unfolding** *isCont_def* **by** (*intro tendsto_powr_complex*) *simp_all*

lemma *continuous_on_powr_complex* [*continuous_intros*]:
assumes $A \subseteq \{z.\ Re\ (f\ z) \geq 0 \vee Im\ (f\ z) \neq 0\}$
assumes $\bigwedge z.\ z \in A \Longrightarrow f\ z = 0 \Longrightarrow Re\ (g\ z) > 0$
assumes *continuous_on A f continuous_on A g*
shows *continuous_on A* $(\lambda z.\ f\ z\ powr\ g\ z)$
unfolding *continuous_on_def*
proof
fix *z* **assume** *z*: $z \in A$
show $((\lambda z.\ f\ z\ powr\ g\ z) \longrightarrow f\ z\ powr\ g\ z)\ (at\ z\ within\ A)$

```
  proof (cases f z = 0)
    case False
    from assms(1,2) z have Re (f z) ≥ 0 ∨ Im (f z) ≠ 0 f z = 0 ⟶ Re (g z) >
0 by auto
    with assms(3,4) z show ?thesis
      by (intro tendsto_powr_complex') 
        (auto elim!: nonpos_Reals_cases simp: complex_eq_iff continuous_on_def)
  next
    case True
    with assms z show ?thesis
      by (auto intro!: tendsto_powr_complex_0 simp: continuous_on_def)
  qed
qed
```

6.18.20 Some Limits involving Logarithms

```
lemma lim_Ln_over_power:
  fixes s::complex
  assumes 0 < Re s
    shows (λn. Ln (of_nat n) / of_nat n powr s) ⟶⟶ 0
proof (simp add: lim_sequentially dist_norm, clarify)
  fix e::real
  assume e: 0 < e
  have ∃xo>0. ∀x≥xo. 0 < e * 2 + (e * Re s * 2 - 2) * x + e * (Re s)^2 * x^2
  proof (rule_tac x=2/(e * (Re s)^2) in exI, safe)
    show 0 < 2 / (e * (Re s)^2)
      using e assms by (simp add: field_simps)
  next
    fix x::real
    assume x: 2 / (e * (Re s)^2) ≤ x
    have 2 / (e * (Re s)^2) > 0
      using e assms by simp
    with x have x > 0
      by linarith
    then have x * 2 ≤ e * (x^2 * (Re s)^2)
      using e assms x by (auto simp: power2_eq_square field_simps)
    also have ... < e * (2 + (x * (Re s * 2) + x^2 * (Re s)^2))
      using e assms ‹x > 0›
      by (auto simp: power2_eq_square field_simps add_pos_pos)
    finally show 0 < e * 2 + (e * Re s * 2 - 2) * x + e * (Re s)^2 * x^2
      by (auto simp: algebra_simps)
  qed
  then have ∃xo>0. ∀x≥xo. x / e < 1 + (Re s * x) + (1/2) * (Re s * x)^2
    using e  by (simp add: field_simps)
  then have ∃xo>0. ∀x≥xo. x / e < exp (Re s * x)
    using assms
    by (force intro: less_le_trans [OF _ exp_lower_Taylor_quadratic])
  then obtain xo where xo > 0 and xo: ⋀x. x ≥ xo ⟹ x < e * exp (Re s * x)
    using e  by (auto simp: field_simps)
```

```
  have norm (Ln (of_nat n) / of_nat n powr s) < e if n ≥ nat ⌈exp xo⌉ for n
  proof −
    have ln (real n) ≥ xo
      using that exp_gt_zero ln_ge_iff [of n] nat_ceiling_le_eq by fastforce
    then show ?thesis
    using e xo [of ln n] by (auto simp: norm_divide norm_powr_real field_split_simps)
  qed
  then show ∃no. ∀n≥no. norm (Ln (of_nat n) / of_nat n powr s) < e
    by blast
qed

lemma lim_Ln_over_n: ((λn. Ln(of_nat n) / of_nat n) ⟶ 0) sequentially
  using lim_Ln_over_power [of 1] by simp

lemma lim_ln_over_power:
  fixes s :: real
  assumes 0 < s
  shows (λn. ln (real n) / real n powr s) ⟶ 0
proof −
  have (λn. ln (Suc n) / (Suc n) powr s) ⟶ 0
     using lim_Ln_over_power [of of_real s, THEN filterlim_sequentially_Suc
[THEN iffD2]] assms
   by (simp add: lim_sequentially dist_norm Ln_Reals_eq norm_powr_real_powr
norm_divide)
  then show ?thesis
    using filterlim_sequentially_Suc[of λn::nat. ln n / n powr s] by auto
qed

lemma lim_ln_over_n [tendsto_intros]: ((λn. ln(real_of_nat n) / of_nat n)
⟶ 0) sequentially
  using lim_ln_over_power [of 1] by auto

lemma lim_log_over_n [tendsto_intros]:
  (λn. log k n/n) ⟶ 0
proof −
  have ∗: log k n/n = (1/ln k) ∗ (ln n / n) for n
    unfolding log_def by auto
  have (λn. (1/ln k) ∗ (ln n / n)) ⟶ (1/ln k) ∗ 0
    by (intro tendsto_intros)
  then show ?thesis
    unfolding ∗ by auto
qed

lemma lim_1_over_complex_power:
  assumes 0 < Re s
  shows (λn. 1 / of_nat n powr s) ⟶ 0
proof (rule Lim_null_comparison)
  have ∀n>0. 3 ≤ n ⟶ 1 ≤ ln (real_of_nat n)
    using ln_272_gt_1
```

```
    by (force intro: order_trans [of _ ln (272/100)])
  then show ∀ F x in sequentially. cmod (1 / of_nat x powr s) ≤ cmod (Ln (of_nat
x) / of_nat x powr s)
    by (auto simp: norm_divide field_split_simps eventually_sequentially)
  show (λn. cmod (Ln (of_nat n) / of_nat n powr s)) ⟶ 0
    using lim_Ln_over_power [OF assms] by (metis tendsto_norm_zero_iff)
qed

lemma lim_1_over_real_power:
  fixes s :: real
  assumes 0 < s
  shows ((λn. 1 / (of_nat n powr s)) ⟶ 0) sequentially
  using lim_1_over_complex_power [of of_real s, THEN filterlim_sequentially_Suc
[THEN iffD2]] assms
  apply (subst filterlim_sequentially_Suc [symmetric])
  by (simp add: lim_sequentially dist_norm Ln_Reals_eq norm_powr_real_powr
norm_divide)

lemma lim_1_over_Ln: (λn. 1 / Ln (complex_of_nat n)) ⟶ 0
proof (clarsimp simp add: lim_sequentially dist_norm norm_divide field_split_simps)
  fix r::real
  assume 0 < r
  have ir: inverse (exp (inverse r)) > 0
    by simp
  obtain n where n: 1 < of_nat n * inverse (exp (inverse r))
    using ex_less_of_nat_mult [of _ 1, OF ir]
    by auto
  then have exp (inverse r) < of_nat n
    by (simp add: field_split_simps)
  then have ln (exp (inverse r)) < ln (of_nat n)
    by (metis exp_gt_zero less_trans ln_exp ln_less_cancel_iff)
  with ‹0 < r› have 1 < r * ln (real_of_nat n)
    by (simp add: field_simps)
  moreover have n > 0 using n
    using neq0_conv by fastforce
  ultimately show ∃ no. ∀ k. Ln (of_nat k) ≠ 0 ⟶ no ≤ k ⟶ 1 < r * cmod
(Ln (of_nat k))
    using n ‹0 < r›
    by (rule_tac x=n in exI) (force simp: field_split_simps intro: less_le_trans)
qed

lemma lim_1_over_ln: (λn. 1 / ln (real n)) ⟶ 0
  using lim_1_over_Ln [THEN filterlim_sequentially_Suc [THEN iffD2]]
  apply (subst filterlim_sequentially_Suc [symmetric])
  by (simp add: lim_sequentially dist_norm Ln_Reals_eq norm_powr_real_powr
norm_divide)

lemma lim_ln1_over_ln: (λn. ln(Suc n) / ln n) ⟶ 1
proof (rule Lim_transform_eventually)
```

```
  have (λn. ln(1 + 1/n) / ln n) ⟶ 0
  proof (rule Lim_transform_bound)
    show (inverse o real) ⟶ 0
      by (metis comp_def lim_inverse_n lim_explicit)
    show ∀F n in sequentially. norm (ln (1 + 1 / n) / ln n) ≤ norm ((inverse ∘
real) n)
    proof
      fix n::nat
      assume n: 3 ≤ n
      then have ln 3 ≤ ln n and ln0: 0 ≤ ln n
        by auto
      with ln3_gt_1 have 1/ ln n ≤ 1
        by (simp add: field_split_simps)
      moreover have ln (1 + 1 / real n) ≤ 1/n
        by (simp add: ln_add_one_self_le_self)
      ultimately have ln (1 + 1 / real n) * (1 / ln n) ≤ (1/n) * 1
        by (intro mult_mono) (use n in auto)
      then show norm (ln (1 + 1 / n) / ln n) ≤ norm ((inverse ∘ real) n)
        by (simp add: field_simps ln0)
      qed
  qed
  then show (λn. 1 + ln(1 + 1/n) / ln n) ⟶ 1
    by (metis (full_types) add.right_neutral tendsto_add_const_iff)
  show ∀F k in sequentially. 1 + ln (1 + 1 / k) / ln k = ln(Suc k) / ln k
    by (simp add: field_split_simps ln_div eventually_sequentiallyI [of 2])
qed

lemma lim_ln_over_ln1: (λn. ln n / ln(Suc n)) ⟶ 1
  using tendsto_inverse [OF lim_ln1_over_ln] by force
```

6.18.21 Relation between Square Root and exp/ln, hence its derivative

```
lemma csqrt_exp_Ln:
  assumes z ≠ 0
    shows csqrt z = exp(Ln z / 2)
proof −
  have (exp (Ln z / 2))^2 = (exp (Ln z))
     by (metis exp_double nonzero_mult_div_cancel_left times_divide_eq_right
zero_neq_numeral)
  also have ... = z
    using assms exp_Ln by blast
  finally have csqrt z = csqrt ((exp (Ln z / 2))^2)
    by simp
  also have ... = exp (Ln z / 2)
    apply (rule csqrt_square)
     using cos_gt_zero_pi [of (Im (Ln z) / 2)] Im_Ln_le_pi mpi_less_Im_Ln
assms
    by (fastforce simp: Re_exp Im_exp)
```

```
  finally show ?thesis using assms csqrt_square
    by simp
qed

lemma csqrt_conv_powr: csqrt z = z powr (1/2)
  by (auto simp: csqrt_exp_Ln powr_def)

lemma csqrt_mult:
  assumes Arg z + Arg w ∈ {−pi<..pi}
  shows   csqrt (z * w) = csqrt z * csqrt w
proof (cases z = 0 ∨ w = 0)
  case False
  have csqrt (z * w) = exp ((ln (z * w)) / 2)
    using False by (intro csqrt_exp_Ln) auto
  also have ... = exp ((Ln z + Ln w) / 2)
    using False assms by (subst Ln_times_simple) (auto simp: Arg_eq_Im_Ln)
  also have (Ln z + Ln w) / 2 = Ln z / 2 + Ln w / 2
    by (simp add: add_divide_distrib)
  also have exp ... = csqrt z * csqrt w
    using False by (simp add: exp_add csqrt_exp_Ln)
  finally show ?thesis .
qed auto

lemma Arg_csqrt [simp]: Arg (csqrt z) = Arg z / 2
proof (cases z = 0)
  case False
  have Im (Ln z) ∈ {−pi<..pi}
    by (simp add: False Im_Ln_le_pi mpi_less_Im_Ln)
  also have ... ⊆ {−2*pi<..2*pi}
    by auto
  finally show ?thesis
    using False by (auto simp: csqrt_exp_Ln Arg_exp Arg_eq_Im_Ln)
qed (auto simp: Arg_zero)

lemma csqrt_inverse:
  z ∉ ℝ≤0 ⟹ csqrt (inverse z) = inverse (csqrt z)
  by (metis Ln_inverse csqrt_eq_0 csqrt_exp_Ln divide_minus_left exp_minus
      inverse_nonzero_iff_nonzero)

lemma cnj_csqrt: z ∉ ℝ≤0 ⟹ cnj(csqrt z) = csqrt(cnj z)
 by (metis cnj_Ln complex_cnj_divide complex_cnj_numeral complex_cnj_zero_iff
csqrt_eq_0 csqrt_exp_Ln exp_cnj)

lemma has_field_derivative_csqrt:
  assumes z ∉ ℝ≤0
    shows (csqrt has_field_derivative inverse(2 * csqrt z)) (at z)
proof −
  have z: z ≠ 0
    using assms by auto
```

then have $*$: *inverse z = inverse (2*z) * 2*
by (*simp add*: *field_split_simps*)
have [*simp*]: *exp (Ln z / 2) * inverse z = inverse (csqrt z)*
by (*simp add*: *z field_simps csqrt_exp_Ln* [*symmetric*]) (*metis power2_csqrt power2_eq_square*)
have *Im z = 0 ⟹ 0 < Re z*
using *assms complex_nonpos_Reals_iff not_less* **by** *blast*
with *z* **have** ((λ*z*. *exp (Ln z / 2)*) *has_field_derivative inverse (2 * csqrt z)*) (*at z*)
by (*force intro*: *derivative_eq_intros * simp add*: *assms*)
then show *?thesis*
proof (*rule has_field_derivative_transform_within*)
show ⋀*x*. *dist x z < cmod z ⟹ exp (Ln x / 2) = csqrt x*
by (*metis csqrt_exp_Ln dist_0_norm less_irrefl*)
qed (*use z* **in** *auto*)
qed

lemma *field_differentiable_at_csqrt*:
z ∉ $\mathbb{R}_{\leq 0}$ ⟹ *csqrt field_differentiable at z*
using *field_differentiable_def has_field_derivative_csqrt* **by** *blast*

lemma *field_differentiable_within_csqrt*:
z ∉ $\mathbb{R}_{\leq 0}$ ⟹ *csqrt field_differentiable (at z within s)*
using *field_differentiable_at_csqrt field_differentiable_within_subset* **by** *blast*

lemma *continuous_at_csqrt*:
z ∉ $\mathbb{R}_{\leq 0}$ ⟹ *continuous (at z) csqrt*
by (*simp add*: *field_differentiable_within_csqrt field_differentiable_imp_continuous_at*)

corollary *isCont_csqrt'* [*simp*]:
⟦*isCont f z*; *f z* ∉ $\mathbb{R}_{\leq 0}$⟧ ⟹ *isCont* (λ*x*. *csqrt (f x)*) *z*
by (*blast intro*: *isCont_o2* [*OF _ continuous_at_csqrt*])

lemma *continuous_within_csqrt*:
z ∉ $\mathbb{R}_{\leq 0}$ ⟹ *continuous (at z within s) csqrt*
by (*simp add*: *field_differentiable_imp_continuous_at field_differentiable_within_csqrt*)

lemma *continuous_on_csqrt* [*continuous_intros*]:
continuous_on (−$\mathbb{R}_{\leq 0}$) *csqrt*
by (*simp add*: *continuous_at_imp_continuous_on continuous_within_csqrt*)

lemma *holomorphic_on_csqrt* [*holomorphic_intros*]: *csqrt holomorphic_on* −$\mathbb{R}_{\leq 0}$
by (*simp add*: *field_differentiable_within_csqrt holomorphic_on_def*)

lemma *holomorphic_on_csqrt'* [*holomorphic_intros*]:
f holomorphic_on A ⟹ (⋀*z*. *z* ∈ *A* ⟹ *f z* ∉ $\mathbb{R}_{\leq 0}$) ⟹ (λ*z*. *csqrt (f z)*) *holomorphic_on A*
using *holomorphic_on_compose_gen*[*OF _ holomorphic_on_csqrt*, *of f A*] **by** (*auto simp*: *o_def*)

lemma *analytic_on_csqrt* [*analytic_intros*]: *csqrt analytic_on* $-\mathbb{R}_{\leq 0}$
using *holomorphic_on_csqrt* **by** (*subst analytic_on_open*) *auto*

lemma *analytic_on_csqrt'* [*analytic_intros*]:
f analytic_on A $\Longrightarrow$ $(\bigwedge z.\ z \in A \Longrightarrow f\ z \notin \mathbb{R}_{\leq 0}) \Longrightarrow$ $(\lambda z.\ csqrt\ (f\ z))$ *analytic_on A*
using *analytic_on_compose_gen*[*OF* _ *analytic_on_csqrt*, *of f A*] **by** (*auto simp*: *o_def*)

lemma *continuous_within_closed_nontrivial*:
closed s $\Longrightarrow$ *a* $\notin$ *s* ==> *continuous* (*at a within s*) *f*
using *Compl_iff continuous_within_topological open_Compl* **by** *fastforce*

lemma *continuous_within_csqrt_posreal*:
continuous (*at z within* ($\mathbb{R} \cap \{w.\ 0 \leq Re(w)\}$)) *csqrt*
proof (*cases z* $\in \mathbb{R}_{\leq 0}$)
case *True*
then have [*simp*]: *Im z* = *0* **and** *0*: *Re z* < *0* $\vee$ *z* = *0*
using *complex_nonpos_Reals_iff complex_eq_iff* **by** *force+*
show *?thesis*
using *0*
proof
assume *Re z* < *0*
then show *?thesis*
by (*auto simp*: *continuous_within_closed_nontrivial* [*OF closed_Real_halfspace_Re_ge*])
next
assume *z* = *0*
moreover
have $\bigwedge e.\ 0 < e$
$\Longrightarrow \forall x' \in \mathbb{R} \cap \{w.\ 0 \leq Re\ w\}.\ cmod\ x' < e\hat{}2 \longrightarrow cmod\ (csqrt\ x') < e$
by (*auto simp*: *Reals_def real_less_lsqrt*)
ultimately show *?thesis*
using *zero_less_power* **by** (*fastforce simp*: *continuous_within_eps_delta*)
qed
qed (*blast intro*: *continuous_within_csqrt*)

6.18.22 Complex arctangent

The branch cut gives standard bounds in the real case.

definition *Arctan* :: *complex* $\Rightarrow$ *complex* **where**
Arctan $\equiv \lambda z.\ (\mathrm{i}/2) * Ln((1 - \mathrm{i}*z) / (1 + \mathrm{i}*z))$

lemma *Arctan_def_moebius*: *Arctan z* = i/*2* $*$ *Ln* (*moebius* (−i) *1* i *1 z*)
by (*simp add*: *Arctan_def moebius_def add_ac*)

lemma *Ln_conv_Arctan*:
assumes *z* $\neq$ *−1*
shows *Ln z* = *−2*$*$i $*$ *Arctan* (*moebius 1* (*− 1*) (− i) (− i) *z*)

proof −
have *Arctan* (*moebius 1* (− *1*) (− i) (− i) *z*) =
i/*2* ∗ *Ln* (*moebius* (− i) *1* i *1* (*moebius 1* (− *1*) (− i) (− i) *z*))
by (*simp add*: *Arctan_def_moebius*)
also from *assms* **have** i ∗ *z* ≠ i ∗ (−*1*) **by** (*subst mult_left_cancel*) *simp*
hence i ∗ *z* − −i ≠ *0* **by** (*simp add*: *eq_neg_iff_add_eq_0*)
from *moebius_inverse'*[*OF* _ *this*, *of 1 1*]
have *moebius* (− i) *1* i *1* (*moebius 1* (− *1*) (− i) (− i) *z*) = *z* **by** *simp*
finally show *?thesis* **by** (*simp add*: *field_simps*)
qed

lemma *Arctan_0* [*simp*]: *Arctan 0* = *0*
by (*simp add*: *Arctan_def*)

lemma *Im_complex_div_lemma*: *Im*((*1* − i∗*z*) / (*1* + i∗*z*)) = *0* ⟷ *Re z* = *0*
by (*auto simp*: *Im_complex_div_eq_0 algebra_simps*)

lemma *Re_complex_div_lemma*: *0* < *Re*((*1* − i∗*z*) / (*1* + i∗*z*)) ⟷ *norm z* < *1*
by (*simp add*: *Re_complex_div_gt_0 algebra_simps cmod_def power2_eq_square*)

lemma *tan_Arctan*:
assumes *z*² ≠ −*1*
shows [*simp*]: *tan*(*Arctan z*) = *z*
proof −
obtain *1* + i∗*z* ≠ *0* *1* − i∗*z* ≠ *0*
by (*metis add_diff_cancel_left' assms diff_0 i_times_eq_iff mult_cancel_left2 power2_i power2_minus right_minus_eq*)
then show *?thesis*
by (*simp add*: *Arctan_def tan_def sin_exp_eq cos_exp_eq exp_minus divide_simps*
flip: *csqrt_exp_Ln power2_eq_square*)
qed

lemma *Arctan_tan* [*simp*]:
assumes |*Re z*| < *pi*/*2*
shows *Arctan*(*tan z*) = *z*
proof −
have *Ln* ((*1* − i ∗ *tan z*) / (*1* + i ∗ *tan z*)) = *2* ∗ *z* / i
proof (*rule Ln_unique*)
have *ge_pi2*: ⋀*n*::*int*. |*of_int* (*2*∗*n* + *1*) ∗ *pi*/*2*| ≥ *pi*/*2*
by (*case_tac n rule*: *int_cases*) (*auto simp*: *abs_mult*)
have *exp* (i∗*z*)∗*exp* (i∗*z*) = −*1* ⟷ *exp* (*2*∗i∗*z*) = −*1*
by (*metis distrib_right exp_add mult_2*)
also have ... ⟷ *exp* (*2*∗i∗*z*) = *exp* (i∗*pi*)
using *cis_conv_exp cis_pi* **by** *auto*
also have ... ⟷ *exp* (*2*∗i∗*z* − i∗*pi*) = *1*
by (*metis* (*no_types*) *diff_add_cancel diff_minus_eq_add exp_add exp_minus_inverse mult.commute*)

```
    also have ... ⟷ Re(i*2*z - i*pi) = 0 ∧ (∃n::int. Im(i*2*z - i*pi) =
of_int (2 * n) * pi)
     by (simp add: exp_eq_1)
   also have ... ⟷ Im z = 0 ∧ (∃n::int. 2 * Re z = of_int (2*n + 1) * pi)
     by (simp add: algebra_simps)
   also have ... ⟷ False
     using assms ge_pi2
     by (metis eq_divide_eq linorder_not_less mult.commute zero_neq_numeral)
   finally have exp (i*z)*exp (i*z) + 1 ≠ 0
     by (auto simp: add.commute minus_unique)
   then show exp (2 * z / i) = (1 - i * tan z) / (1 + i * tan z)
     apply (simp add: tan_def sin_exp_eq cos_exp_eq exp_minus divide_simps)
     by (simp add: algebra_simps flip: power2_eq_square exp_double)
 qed (use assms in auto)
 then show ?thesis
   by (auto simp: Arctan_def)
qed

lemma
 assumes Re z = 0 ⟹ |Im z| < 1
 shows Re_Arctan_bounds: |Re(Arctan z)| < pi/2
    and has_field_derivative_Arctan: (Arctan has_field_derivative inverse(1 +
z^2)) (at z)
proof −
 have nz0: 1 + i*z ≠ 0
   using assms
  by (metis abs_one add_diff_cancel_left' complex_i_mult_minus diff_0 i_squared
imaginary_unit.simps
             less_asym neg_equal_iff_equal)
 have z ≠ −i using assms
   by auto
 then have zz: 1 + z * z ≠ 0
    by (metis abs_one assms i_squared imaginary_unit.simps less_irrefl mi-
nus_unique square_eq_iff)
 have nz1: 1 − i*z ≠ 0
   using assms by (force simp add: i_times_eq_iff)
 have nz2: inverse (1 + i*z) ≠ 0
   using assms
   by (metis Im_complex_div_lemma Re_complex_div_lemma cmod_eq_Im di-
vide_complex_def
       less_irrefl mult_zero_right zero_complex.simps(1) zero_complex.simps(2))
 have nzi: ((1 − i*z) * inverse (1 + i*z)) ≠ 0
   using nz1 nz2 by auto
 have Im ((1 − i*z) / (1 + i*z)) = 0 ⟹ 0 < Re ((1 − i*z) / (1 + i*z))
  by (simp add: Im_complex_div_lemma Re_complex_div_lemma assms cmod_eq_Im)
 then have *: ((1 − i*z) / (1 + i*z)) ∉ ℝ≤0
   by (auto simp add: complex_nonpos_Reals_iff)
 show |Re(Arctan z)| < pi/2
   unfolding Arctan_def divide_complex_def
```

```
    using mpi_less_Im_Ln [OF nzi]
   by (auto simp: abs_if intro!: Im_Ln_less_pi * [unfolded divide_complex_def])
  show (Arctan has_field_derivative inverse(1 + z^2)) (at z)
    unfolding Arctan_def scaleR_conv_of_real
    apply (intro derivative_eq_intros | simp add: nz0 *)+
    using nz1 zz
    apply (simp add: field_split_simps power2_eq_square)
    apply algebra
    done
qed

lemma field_differentiable_at_Arctan: (Re z = 0 ⟹ |Im z| < 1) ⟹ Arctan
field_differentiable at z
  using has_field_derivative_Arctan
  by (auto simp: field_differentiable_def)

lemma field_differentiable_within_Arctan:
    (Re z = 0 ⟹ |Im z| < 1) ⟹ Arctan field_differentiable (at z within s)
  using field_differentiable_at_Arctan field_differentiable_at_within by blast

declare has_field_derivative_Arctan [derivative_intros]
declare has_field_derivative_Arctan [THEN DERIV_chain2, derivative_intros]

lemma continuous_at_Arctan:
    (Re z = 0 ⟹ |Im z| < 1) ⟹ continuous (at z) Arctan
 by (simp add: field_differentiable_imp_continuous_at field_differentiable_within_Arctan)

lemma continuous_within_Arctan:
    (Re z = 0 ⟹ |Im z| < 1) ⟹ continuous (at z within s) Arctan
  using continuous_at_Arctan continuous_at_imp_continuous_within by blast

lemma continuous_on_Arctan [continuous_intros]:
    (⋀z. z ∈ s ⟹ Re z = 0 ⟹ |Im z| < 1) ⟹ continuous_on s Arctan
  by (auto simp: continuous_at_imp_continuous_on continuous_within_Arctan)

lemma holomorphic_on_Arctan:
    (⋀z. z ∈ s ⟹ Re z = 0 ⟹ |Im z| < 1) ⟹ Arctan holomorphic_on s
  by (simp add: field_differentiable_within_Arctan holomorphic_on_def)

theorem Arctan_series:
  assumes z: norm (z :: complex) < 1
  defines g ≡ λn. if odd n then −i*i^n / n else 0
  defines h ≡ λz n. (−1)^n / of_nat (2*n+1) * (z::complex)^(2*n+1)
  shows   (λn. g n * z^n) sums Arctan z
  and     h z sums Arctan z
proof −
  define G where [abs_def]: G z = (∑ n. g n * z^n) for z
  have summable: summable (λn. g n * u^n) if norm u < 1 for u
  proof (cases u = 0)
```

case *False*
have (λ*n*. *ereal* (*norm* (*h u n*) / *norm* (*h u* (*Suc n*)))) = (λ*n*. *ereal* (*inverse* (*norm u*)^2) *
ereal ((2 + *inverse* (*real* (*Suc n*))) / (2 − *inverse* (*real* (*Suc n*)))))
proof
fix *n*
have *ereal* (*norm* (*h u n*) / *norm* (*h u* (*Suc n*))) =
ereal (*inverse* (*norm u*)^2) * *ereal* (((2*Suc *n*+1) / (*Suc n*)) /
((2*Suc *n*−1) / (*Suc n*)))
by (*simp add*: *h_def norm_mult norm_power norm_divide field_split_simps*
power2_eq_square eval_nat_numeral del: *of_nat_add of_nat_Suc*)
also have *of_nat* (2*Suc *n*+1) / *of_nat* (*Suc n*) = (2::*real*) + *inverse* (*real* (*Suc n*))
by (*auto simp*: *field_split_simps simp del*: *of_nat_Suc*) *simp_all?*
also have *of_nat* (2*Suc *n*−1) / *of_nat* (*Suc n*) = (2::*real*) − *inverse* (*real* (*Suc n*))
by (*auto simp*: *field_split_simps simp del*: *of_nat_Suc*) *simp_all?*
finally show *ereal* (*norm* (*h u n*) / *norm* (*h u* (*Suc n*))) = *ereal* (*inverse* (*norm u*)^2) *
ereal ((2 + *inverse* (*real* (*Suc n*))) / (2 − *inverse* (*real* (*Suc n*)))) .
qed
also have ... ⟶ *ereal* (*inverse* (*norm u*)^2) * *ereal* ((2 + 0) / (2 − 0))
by (*intro tendsto_intros LIMSEQ_inverse_real_of_nat*) *simp_all*
finally have *liminf* (λ*n*. *ereal* (*cmod* (*h u n*) / *cmod* (*h u* (*Suc n*)))) = *inverse* (*norm u*)^2
by (*intro lim_imp_Liminf*) *simp_all*
moreover from *power_strict_mono*[*OF that*, *of 2*] *False* **have** *inverse* (*norm u*)^2 > 1
by (*simp add*: *field_split_simps*)
ultimately have *A*: *liminf* (λ*n*. *ereal* (*cmod* (*h u n*) / *cmod* (*h u* (*Suc n*)))) > 1 **by** *simp*
from *False* **have** *summable* (*h u*)
by (*intro summable_norm_cancel*[*OF ratio_test_convergence*[*OF _ A*]])
(*auto simp*: *h_def norm_divide norm_mult norm_power simp del*: *of_nat_Suc*
intro!: *mult_pos_pos divide_pos_pos always_eventually*)
thus *summable* (λ*n*. *g n* * *u*^*n*)
by (*subst summable_mono_reindex*[*of* λ*n*. 2*n+1, *symmetric*])
(*auto simp*: *power_mult strict_mono_def g_def h_def elim*!: *oddE*)
qed (*simp add*: *h_def*)

have ∃ *c*. ∀ *u*∈*ball 0 1*. *Arctan u* − *G u* = *c*
proof (*rule has_field_derivative_zero_constant*)
fix *u* :: *complex* **assume** *u* ∈ *ball 0 1*
hence *u*: *norm u* < *1* **by** (*simp*)
define *K* **where** *K* = (*norm u* + *1*) / *2*
from *u* **and** *abs_Im_le_cmod*[*of u*] **have** *Im_u*: |*Im u*| < *1* **by** *linarith*
from *u* **have** *K*: *0* ≤ *K norm u* < *K K* < *1* **by** (*simp_all add*: *K_def*)
hence (*G has_field_derivative* (∑ *n*. *diffs g n* * *u* ^ *n*)) (*at u*) **unfolding** *G_def*

by (*intro termdiffs_strong*[*of* _ *of_real K*] *summable*) *simp_all*
also have (λ*n*. *diffs g n* ∗ *u*^*n*) = (λ*n*. *if even n then* (i∗*u*)^*n else 0*)
by (*intro ext*) (*simp_all del*: *of_nat_Suc add*: *g_def diffs_def power_mult_distrib*)
also have *suminf* ... = (∑ *n*. (−(*u*^2))^*n*)
by (*subst suminf_mono_reindex*[*of* λ*n*. 2∗*n*, *symmetric*])
(*auto elim*!: *evenE simp*: *strict_mono_def power_mult power_mult_distrib*)
also from *u* **have** *norm u*^2 < 1^2 **by** (*intro power_strict_mono*) *simp_all*
hence (∑ *n*. (−(*u*^2))^*n*) = *inverse* (1 + *u*^2)
by (*subst suminf_geometric*) (*simp_all add*: *norm_power inverse_eq_divide*)
finally have (*G has_field_derivative inverse* (1 + *u*²)) (*at u*) .
from *DERIV_diff*[*OF has_field_derivative_Arctan this*] *Im_u u*
show ((λ*u*. *Arctan u* − *G u*) *has_field_derivative 0*) (*at u within ball 0 1*)
by (*simp_all add*: *at_within_open*[*OF* _ *open_ball*])
qed *simp_all*
then obtain *c* **where** *c*: ⋀*u*. *norm u* < *1* ⟹ *Arctan u* − *G u* = *c* **by** *auto*
from *this*[*of 0*] **have** *c* = *0* **by** (*simp add*: *G_def g_def*)
with *c z* **have** *Arctan z* = *G z* **by** *simp*
with *summable*[*OF z*] **show** (λ*n*. *g n* ∗ *z*^*n*) *sums Arctan z* **unfolding** *G_def*
by (*simp add*: *sums_iff*)
thus *h z sums Arctan z* **by** (*subst* (*asm*) *sums_mono_reindex*[*of* λ*n*. 2∗*n*+*1*, *symmetric*])
(*auto elim*!: *oddE simp*: *strict_mono_def power_mult g_def h_def*)
qed

A quickly-converging series for the logarithm, based on the arctangent.

theorem *ln_series_quadratic*:
assumes *x*: *x* > (*0*::*real*)
shows (λ*n*. (2∗((*x* − *1*) / (*x* + *1*)) ^ (2∗*n*+1) / *of_nat* (2∗*n*+1))) *sums ln x*
proof −
define *y* :: *complex* **where** *y* = *of_real* ((*x*−1)/(*x*+1))
from *x* **have** *x'*: *complex_of_real x* ≠ *of_real* (−*1*) **by** (*subst of_real_eq_iff*) *auto*
from *x* **have** |*x* − *1*| < |*x* + *1*| **by** *linarith*
hence *norm* (*complex_of_real* (*x* − *1*) / *complex_of_real* (*x* + *1*)) < *1*
by (*simp add*: *norm_divide del*: *of_real_add of_real_diff*)
hence *norm* (i ∗ *y*) < *1* **unfolding** *y_def* **by** (*subst norm_mult*) *simp*
hence (λ*n*. (−*2*∗i) ∗ ((−*1*)^*n* / *of_nat* (2∗*n*+1) ∗ (i∗*y*)^(2∗*n*+1))) *sums* ((−*2*∗i) ∗ *Arctan* (i∗*y*))
by (*intro Arctan_series sums_mult*) *simp_all*
also have (λ*n*. (−*2*∗i) ∗ ((−*1*)^*n* / *of_nat* (2∗*n*+1) ∗ (i∗*y*)^(2∗*n*+1))) =
(λ*n*. (−*2*∗i) ∗ ((−*1*)^*n* ∗ (i∗*y*∗(−*y*²)^*n*)/*of_nat* (2∗*n*+1)))
by (*intro ext*) (*simp_all add*: *power_mult power_mult_distrib*)
also have ... = (λ*n*. 2∗*y*∗ ((−*1*) ∗ (−*y*²))^*n*/*of_nat* (2∗*n*+1))
by (*intro ext*, *subst power_mult_distrib*) (*simp add*: *algebra_simps power_mult*)
also have ... = (λ*n*. 2∗*y*^(2∗*n*+1) / *of_nat* (2∗*n*+1))
by (*subst power_add*, *subst power_mult*) (*simp add*: *mult_ac*)
also have ... = (λ*n*. *of_real* (2∗((*x*−1)/(*x*+1))^(2∗*n*+1) / *of_nat* (2∗*n*+1)))
by (*intro ext*) (*simp add*: *y_def*)

also have $\mathrm{i} * y = (\mathit{of\_real}\ x - 1)\ /\ (-\mathrm{i} * (\mathit{of\_real}\ x + 1))$
by (*subst divide_divide_eq_left* [*symmetric*]) (*simp add*: *y_def*)
also have $\ldots = \mathit{moebius}\ 1\ (-1)\ (-\mathrm{i})\ (-\mathrm{i})\ (\mathit{of\_real}\ x)$ **by** (*simp add*: *moebius_def algebra_simps*)
also from x' **have** $-2 * \mathrm{i} * \mathit{Arctan} \ldots = \mathit{Ln}\ (\mathit{of\_real}\ x)$ **by** (*intro Ln_conv_Arctan* [*symmetric*]) *simp_all*
also from x **have** $\ldots = \mathit{ln}\ x$ **by** (*rule Ln_of_real*)
finally show *?thesis* **by** (*subst* (*asm*) *sums_of_real_iff*)
qed

6.18.23 Real arctangent

lemma *Im_Arctan_of_real* [*simp*]: $\mathit{Im}\ (\mathit{Arctan}\ (\mathit{of\_real}\ x)) = 0$
proof −
have *ne*: $1 + x^2 \neq 0$
by (*metis power_one sum_power2_eq_zero_iff zero_neq_one*)
have *ne1*: $1 + \mathrm{i} * \mathit{complex\_of\_real}\ x \neq 0$
using *Complex_eq complex_eq_cancel_iff2* **by** *fastforce*
have $\mathit{Re}\ (\mathit{Ln}\ ((1 - \mathrm{i} * x) * \mathit{inverse}\ (1 + \mathrm{i} * x))) = 0$
apply (*rule norm_exp_imaginary*)
using *ne*
apply (*simp add*: *ne1 cmod_def*)
apply (*auto simp*: *field_split_simps*)
apply *algebra*
done
then show *?thesis*
unfolding *Arctan_def divide_complex_def* **by** (*simp add*: *complex_eq_iff*)
qed

lemma *arctan_eq_Re_Arctan*: $\mathit{arctan}\ x = \mathit{Re}\ (\mathit{Arctan}\ (\mathit{of\_real}\ x))$
proof (*rule arctan_unique*)
have $(1 - \mathrm{i} * x)\ /\ (1 + \mathrm{i} * x) \notin \mathbb{R}_{\leq 0}$
by (*auto simp*: *Im_complex_div_lemma complex_nonpos_Reals_iff*)
then show $-(\mathit{pi}\ /\ 2) < \mathit{Re}\ (\mathit{Arctan}\ (\mathit{complex\_of\_real}\ x))$
by (*simp add*: *Arctan_def Im_Ln_less_pi*)
next
have $*$: $(1 - \mathrm{i} * x)\ /\ (1 + \mathrm{i} * x) \neq 0$
by (*simp add*: *field_split_simps*) (*simp add*: *complex_eq_iff*)
show $\mathit{Re}\ (\mathit{Arctan}\ (\mathit{complex\_of\_real}\ x)) < \mathit{pi}\ /\ 2$
using *mpi_less_Im_Ln* [*OF* $*$]
by (*simp add*: *Arctan_def*)
next
have $\mathit{tan}\ (\mathit{Re}\ (\mathit{Arctan}\ (\mathit{of\_real}\ x))) = \mathit{Re}\ (\mathit{tan}\ (\mathit{Arctan}\ (\mathit{of\_real}\ x)))$
by (*metis Im_Arctan_of_real Re_complex_of_real complex_is_Real_iff of_real_Re tan_of_real*)
also have $\ldots = x$
proof −
have $(\mathit{complex\_of\_real}\ x)^2 \neq -1$
by (*smt* (*verit*, *best*) *Im_complex_of_real imaginary_unit.sel*(2) *of_real_minus*

```
power2_eq_iff power2_i)
    then show ?thesis
      by simp
  qed
  finally show tan (Re (Arctan (complex_of_real x))) = x .
qed

lemma Arctan_of_real: Arctan (of_real x) = of_real (arctan x)
  unfolding arctan_eq_Re_Arctan divide_complex_def
  by (simp add: complex_eq_iff)

lemma Arctan_in_Reals [simp]: z ∈ ℝ ⟹ Arctan z ∈ ℝ
  by (metis Reals_cases Reals_of_real Arctan_of_real)

declare arctan_one [simp]

lemma arctan_less_pi4_pos: x < 1 ⟹ arctan x < pi/4
  by (metis arctan_less_iff arctan_one)

lemma arctan_less_pi4_neg: -1 < x ⟹ -(pi/4) < arctan x
  by (metis arctan_less_iff arctan_minus arctan_one)

lemma arctan_less_pi4: |x| < 1 ⟹ |arctan x| < pi/4
  by (metis abs_less_iff arctan_less_pi4_pos arctan_minus)

lemma arctan_le_pi4: |x| ≤ 1 ⟹ |arctan x| ≤ pi/4
  by (metis abs_le_iff arctan_le_iff arctan_minus arctan_one)

lemma abs_arctan: |arctan x| = arctan |x|
  by (simp add: abs_if arctan_minus)

lemma arctan_add_raw:
  assumes |arctan x + arctan y| < pi/2
    shows arctan x + arctan y = arctan((x + y) / (1 - x * y))
proof (rule arctan_unique [symmetric])
  show 12: - (pi / 2) < arctan x + arctan y arctan x + arctan y < pi / 2
    using assms by linarith+
  show tan (arctan x + arctan y) = (x + y) / (1 - x * y)
    using cos_gt_zero_pi [OF 12] by (simp add: arctan tan_add)
qed

lemma arctan_inverse:
  0 < x ⟹arctan(inverse x) = pi/2 - arctan x
  by (smt (verit, del_insts) arctan arctan_unique tan_cot zero_less_arctan_iff)

lemma arctan_add_small:
  assumes |x * y| < 1
    shows (arctan x + arctan y = arctan((x + y) / (1 - x * y)))
proof (cases x = 0 ∨ y = 0)
```

```
  case False
  with assms have |x| < inverse |y|
    by (simp add: field_split_simps abs_mult)
  with False have |arctan x| < pi / 2 − |arctan y| using assms
    by (auto simp add: abs_arctan arctan_inverse [symmetric] arctan_less_iff)
  then show ?thesis
    by (intro arctan_add_raw) linarith
qed auto

lemma abs_arctan_le:
  fixes x::real shows |arctan x| ≤ |x|
proof −
  have 1: ⋀x. x ∈ ℝ ⟹ cmod (inverse (1 + x²)) ≤ 1
   by (simp add: norm_divide divide_simps in_Reals_norm complex_is_Real_iff
power2_eq_square)
  have cmod (Arctan w − Arctan z) ≤ 1 * cmod (w−z) if w ∈ ℝ z ∈ ℝ for w z
    apply (rule field_differentiable_bound [OF convex_Reals, of Arctan _ 1])
     apply (rule has_field_derivative_at_within [OF has_field_derivative_Arctan])
    using 1 that by (auto simp: Reals_def)
  then have cmod (Arctan (of_real x) − Arctan 0) ≤ 1 * cmod (of_real x − 0)
    using Reals_0 Reals_of_real by blast
  then show ?thesis
    by (simp add: Arctan_of_real)
qed

lemma arctan_le_self: 0 ≤ x ⟹ arctan x ≤ x
  by (metis abs_arctan_le abs_of_nonneg zero_le_arctan_iff)

lemma abs_tan_ge: |x| < pi/2 ⟹ |x| ≤ |tan x|
  by (metis abs_arctan_le abs_less_iff arctan_tan minus_less_iff)

lemma arctan_bounds:
  assumes 0 ≤ x x < 1
  shows arctan_lower_bound:
    (∑k<2 * n. (− 1) ^k * (1 / real (k * 2 + 1) * x ^(k * 2 + 1))) ≤ arctan
x (is (∑k<_. _ * ?a k) ≤ _)
    and arctan_upper_bound:
    arctan x ≤ (∑k<2 * n + 1. (− 1) ^k * (1 / real (k * 2 + 1) * x ^(k * 2
+ 1)))
proof −
  have tendsto_zero: ?a ⟶ 0
  proof (rule tendsto_eq_rhs)
    show (λk. 1 / real (k * 2 + 1) * x ^(k * 2 + 1)) ⟶ 0 * 0
      using assms
      by (intro tendsto_mult real_tendsto_divide_at_top)
        (auto simp: filterlim_sequentially_iff_filterlim_real
       intro!: real_tendsto_divide_at_top tendsto_power_zero filterlim_real_sequentially
       tendsto_eq_intros filterlim_at_top_mult_tendsto_pos filterlim_tendsto_add_at_top)
  qed simp
```

have *nonneg*: $0 \leq ?a\ n$ **for** n
by (*force intro*!: *divide_nonneg_nonneg mult_nonneg_nonneg zero_le_power assms*)
have *le*: $?a\ (Suc\ n) \leq ?a\ n$ **for** n
by (*rule mult_mono*[*OF _ power_decreasing*]) (*auto simp*: *field_split_simps assms less_imp_le*)
from *summable_Leibniz'*(4)[*of ?a*, *OF tendsto_zero nonneg le*, *of n*]
summable_Leibniz'(2)[*of ?a*, *OF tendsto_zero nonneg le*, *of n*]
assms
show $(\sum k<2*n.\ (-1)\hat{}k * ?a\ k) \leq arctan\ x\ arctan\ x \leq (\sum k<2 * n + 1.\ (-1)\hat{}k * ?a\ k)$
by (*auto simp*: *arctan_series*)
qed

6.18.24 Bounds on pi using real arctangent

lemma *pi_machin*: $pi = 16 * arctan\ (1 / 5) - 4 * arctan\ (1 / 239)$
using *machin* **by** *simp*

lemma *pi_approx*: $3.141592653588 \leq pi\ pi \leq 3.1415926535899$
unfolding *pi_machin*
using *arctan_bounds*[*of 1/5 4*]
arctan_bounds[*of 1/239 4*]
by (*simp_all add*: *eval_nat_numeral*)

lemma *pi_gt3*: $pi > 3$
using *pi_approx* **by** *simp*

6.18.25 Inverse Sine

definition *Arcsin* :: *complex* ⇒ *complex* **where**
$Arcsin \equiv \lambda z.\ -\mathrm{i} * Ln(\mathrm{i} * z + csqrt(1 - z^2))$

lemma *Arcsin_body_lemma*: $\mathrm{i} * z + csqrt(1 - z^2) \neq 0$
using *power2_csqrt* [*of* $1 - z^2$]
by (*metis add.inverse_unique diff_0 diff_add_cancel mult.left_commute mult_minus1_right power2_i power2_minus power_mult_distrib zero_neq_one*)

lemma *Arcsin_range_lemma*: $|Re\ z| < 1 \Longrightarrow 0 < Re(\mathrm{i} * z + csqrt(1 - z^2))$
using *Complex.cmod_power2* [*of z*, *symmetric*]
by (*simp add*: *real_less_rsqrt algebra_simps Re_power2 cmod_square_less_1_plus*)

lemma *Re_Arcsin*: $Re(Arcsin\ z) = Im\ (Ln\ (\mathrm{i} * z + csqrt(1 - z^2)))$
by (*simp add*: *Arcsin_def*)

lemma *Im_Arcsin*: $Im(Arcsin\ z) = -\ ln\ (cmod\ (\mathrm{i} * z + csqrt\ (1 - z^2)))$
by (*simp add*: *Arcsin_def Arcsin_body_lemma*)

lemma *one_minus_z2_notin_nonpos_Reals*:
assumes $Im\ z = 0 \Longrightarrow |Re\ z| < 1$

```
  shows 1 − z^2 ∉ ℝ≤0
proof (cases Im z = 0)
  case True
  with assms show ?thesis
    by (simp add: complex_nonpos_Reals_iff flip: abs_square_less_1)
next
  case False
  have ¬ (Im z)^2 ≤ − 1
    using False power2_less_eq_zero_iff by fastforce
  with False show ?thesis
    by (auto simp add: complex_nonpos_Reals_iff Re_power2 Im_power2)
qed

lemma isCont_Arcsin_lemma:
  assumes le0: Re (i ∗ z + csqrt (1 − z^2)) ≤ 0 and (Im z = 0 ⟹ |Re z| < 1)
    shows False
proof (cases Im z = 0)
  case True
  then show ?thesis
    using assms by (fastforce simp: cmod_def abs_square_less_1 [symmetric])
next
  case False
  have leim: (cmod (1 − z^2) + (1 − Re (z^2))) / 2 ≤ (Im z)^2
    using le0 sqrt_le_D by fastforce
  have neq: (cmod z)^2 ≠ 1 + cmod (1 − z^2)
  proof (clarsimp simp add: cmod_def)
    assume (Re z)^2 + (Im z)^2 = 1 + sqrt ((1 − Re (z^2))^2 + (Im (z^2))^2)
    then have ((Re z)^2 + (Im z)^2 − 1)^2 = ((1 − Re (z^2))^2 + (Im (z^2))^2)
      by simp
    then show False using False
      by (simp add: power2_eq_square algebra_simps)
  qed
  moreover have 2: (Im z)^2 = (1 + ((Im z)^2 + cmod (1 − z^2)) − (Re z)^2) / 2
    using leim cmod_power2 [of z] norm_triangle_ineq2 [of z^2 1]
    by (simp add: norm_power Re_power2 norm_minus_commute [of 1])
  ultimately show False
    by (simp add: Re_power2 Im_power2 cmod_power2)
qed

lemma isCont_Arcsin:
  assumes (Im z = 0 ⟹ |Re z| < 1)
    shows isCont Arcsin z
proof −
  have 1: i ∗ z + csqrt (1 − z^2) ∉ ℝ≤0
    by (metis isCont_Arcsin_lemma assms complex_nonpos_Reals_iff)
  have 2: 1 − z^2 ∉ ℝ≤0
    by (simp add: one_minus_z2_notin_nonpos_Reals assms)
  show ?thesis
    using assms unfolding Arcsin_def by (intro isCont_Ln' isCont_csqrt' con-
```

tinuous_intros 1 2)
qed

lemma *isCont_Arcsin'* [*simp*]:
shows *isCont f z* $\Longrightarrow$ (*Im* (*f z*) = *0* $\Longrightarrow$ |*Re* (*f z*)| < *1*) $\Longrightarrow$ *isCont* (λ*x*. *Arcsin* (*f x*)) *z*
by (*blast intro*: *isCont_o2* [*OF* _ *isCont_Arcsin*])

lemma *sin_Arcsin* [*simp*]: *sin*(*Arcsin z*) = *z*
proof −
have i∗*z*∗*2* + *csqrt* (*1* − *z*^{2})∗*2* = *0* ⟷ (i∗*z*)∗*2* + *csqrt* (*1* − *z*^{2})∗*2* = *0*
by (*simp add*: *algebra_simps*) — Cancelling a factor of 2
moreover have ... ⟷ (i∗*z*) + *csqrt* (*1* − *z*^{2}) = *0*
by (*metis Arcsin_body_lemma distrib_right no_zero_divisors zero_neq_numeral*)
ultimately show *?thesis*
apply (*simp add*: *sin_exp_eq Arcsin_def Arcsin_body_lemma exp_minus divide_simps*)
apply (*simp add*: *algebra_simps*)
apply (*simp add*: *right_diff_distrib flip*: *power2_eq_square*)
done
qed

lemma *Re_eq_pihalf_lemma*:
|*Re z*| = *pi*/*2* $\Longrightarrow$ *Im z* = *0* $\Longrightarrow$
Re ((*exp* (i∗*z*) + *inverse* (*exp* (i∗*z*))) / *2*) = *0* ∧ *0* ≤ *Im* ((*exp* (i∗*z*) + *inverse* (*exp* (i∗*z*))) / *2*)
apply (*simp add*: *cos_i_times* [*symmetric*] *Re_cos Im_cos abs_if del*: *eq_divide_eq_numeral1*)
by (*metis cos_minus cos_pi_half*)

lemma *Re_less_pihalf_lemma*:
assumes |*Re z*| < *pi* / *2*
shows *0* < *Re* ((*exp* (i∗*z*) + *inverse* (*exp* (i∗*z*))) / *2*)
proof −
have *0* < *cos* (*Re z*) **using** *assms*
using *cos_gt_zero_pi* **by** *auto*
then show *?thesis*
by (*simp add*: *cos_i_times* [*symmetric*] *Re_cos Im_cos add_pos_pos*)
qed

lemma *Arcsin_sin*:
assumes |*Re z*| < *pi*/*2* ∨ (|*Re z*| = *pi*/*2* ∧ *Im z* = *0*)
shows *Arcsin*(*sin z*) = *z*
proof −
have *Arcsin*(*sin z*) = − (i ∗ *Ln* (*csqrt* (*1* − (i ∗ (*exp* (i∗*z*) − *inverse* (*exp* (i∗*z*))))^2 / *4*) − (*inverse* (*exp* (i∗*z*)) − *exp* (i∗*z*)) / *2*))
by (*simp add*: *sin_exp_eq Arcsin_def exp_minus power_divide*)
also have ... = − (i ∗ *Ln* (*csqrt* (((*exp* (i∗*z*) + *inverse* (*exp* (i∗*z*)))/*2*)^2) − (*inverse* (*exp* (i∗*z*)) − *exp* (i∗*z*)) / *2*))
by (*simp add*: *field_simps power2_eq_square*)

```
  also have ... = − (i ∗ Ln (((exp (i∗z) + inverse (exp (i∗z)))/2) − (inverse (exp
(i∗z)) − exp (i∗z)) / 2))
    apply (subst csqrt_square)
    using assms Re_eq_pihalf_lemma Re_less_pihalf_lemma by auto
  also have ... =  − (i ∗ Ln (exp (i∗z)))
    by (simp add: field_simps power2_eq_square)
  also have ... = z
     using assms by (auto simp: abs_if simp del: eq_divide_eq_numeral1 split:
if_split_asm)
  finally show ?thesis .
qed

lemma Arcsin_unique:
    ⟦sin z = w; |Re z| < pi/2 ∨ (|Re z| = pi/2 ∧ Im z = 0)⟧ ⟹ Arcsin w = z
  by (metis Arcsin_sin)

lemma Arcsin_0 [simp]: Arcsin 0 = 0
  by (simp add: Arcsin_unique)

lemma Arcsin_1 [simp]: Arcsin 1 = pi/2
  using Arcsin_unique sin_of_real_pi_half by fastforce

lemma Arcsin_minus_1 [simp]: Arcsin(−1) = − (pi/2)
  by (simp add: Arcsin_unique)

lemma has_field_derivative_Arcsin:
  assumes Im z = 0 ⟹ |Re z| < 1
    shows (Arcsin has_field_derivative inverse(cos(Arcsin z))) (at z)
proof −
  have (sin (Arcsin z))^2 ≠ 1
    using assms one_minus_z2_notin_nonpos_Reals by force
  then have cos (Arcsin z) ≠ 0
    by (metis diff_0_right power_zero_numeral sin_squared_eq)
  then show ?thesis
     by (rule has_field_derivative_inverse_basic [OF DERIV_sin _ _ open_ball
[of z 1]]) (auto intro: isCont_Arcsin assms)
qed

declare has_field_derivative_Arcsin [derivative_intros]
declare has_field_derivative_Arcsin [THEN DERIV_chain2, derivative_intros]

lemma field_differentiable_at_Arcsin:
    (Im z = 0 ⟹ |Re z| < 1) ⟹ Arcsin field_differentiable at z
  using field_differentiable_def has_field_derivative_Arcsin by blast

lemma field_differentiable_within_Arcsin:
    (Im z = 0 ⟹ |Re z| < 1) ⟹ Arcsin field_differentiable (at z within s)
  using field_differentiable_at_Arcsin field_differentiable_within_subset by blast
```

lemma *continuous_within_Arcsin*:
  $(Im\ z = 0 \Longrightarrow |Re\ z| < 1) \Longrightarrow continuous\ (at\ z\ within\ s)\ Arcsin$
 using *continuous_at_imp_continuous_within isCont_Arcsin* **by** *blast*

lemma *continuous_on_Arcsin* [*continuous_intros*]:
  $(\bigwedge z.\ z \in s \Longrightarrow Im\ z = 0 \Longrightarrow |Re\ z| < 1) \Longrightarrow continuous\_on\ s\ Arcsin$
 by (*simp add*: *continuous_at_imp_continuous_on*)

lemma *holomorphic_on_Arcsin*: $(\bigwedge z.\ z \in s \Longrightarrow Im\ z = 0 \Longrightarrow |Re\ z| < 1) \Longrightarrow$
Arcsin holomorphic_on s
 by (*simp add*: *field_differentiable_within_Arcsin holomorphic_on_def*)

6.18.26 Inverse Cosine

definition *Arccos* :: *complex* ⇒ *complex* **where**
  $Arccos \equiv \lambda z.\ -\mathrm{i} * Ln(z + \mathrm{i} * csqrt(1 - z^2))$

lemma *Arccos_range_lemma*: $|Re\ z| < 1 \Longrightarrow 0 < Im(z + \mathrm{i} * csqrt(1 - z^2))$
 using *Arcsin_range_lemma* [*of* $-z$] **by** *simp*

lemma *Arccos_body_lemma*: $z + \mathrm{i} * csqrt(1 - z^2) \neq 0$
 by (*metis Arcsin_body_lemma complex_i_mult_minus diff_0 diff_eq_eq power2_minus*)

lemma *Re_Arccos*: $Re(Arccos\ z) = Im\ (Ln\ (z + \mathrm{i} * csqrt(1 - z^2)))$
 by (*simp add*: *Arccos_def*)

lemma *Im_Arccos*: $Im(Arccos\ z) = -\ ln\ (cmod\ (z + \mathrm{i} * csqrt\ (1 - z^2)))$
 by (*simp add*: *Arccos_def Arccos_body_lemma*)

A very tricky argument to find!

lemma *isCont_Arccos_lemma*:
 assumes *eq0*: $Im\ (z + \mathrm{i} * csqrt\ (1 - z^2)) = 0$ **and** $Im\ z = 0 \Longrightarrow |Re\ z| < 1$
  shows *False*
proof (*cases* $Im\ z = 0$)
 case *True*
 then show *?thesis*
  using *assms* **by** (*fastforce simp add*: *cmod_def abs_square_less_1* [*symmetric*])
next
 case *False*
 have *Imz*: $Im\ z = -\ sqrt\ ((1 + ((Im\ z)^2 + cmod\ (1 - z^2)) - (Re\ z)^2)\ /\ 2)$
  using *eq0 abs_Re_le_cmod* [*of* $1-z^2$]
  by (*simp add*: *Re_power2 algebra_simps*)
 have $(cmod\ z)^2 - 1 \neq cmod\ (1 - z^2)$
 proof (*clarsimp simp add*: *cmod_def*)
  assume $(Re\ z)^2 + (Im\ z)^2 - 1 = sqrt\ ((1 - Re\ (z^2))^2 + (Im\ (z^2))^2)$
  then have $((Re\ z)^2 + (Im\ z)^2 - 1)^2 = ((1 - Re\ (z^2))^2 + (Im\ (z^2))^2)$
   by *simp*
  then show *False* **using** *False*
   by (*simp add*: *power2_eq_square algebra_simps*)

```
  qed
  moreover have (Im z)^2 = (1 + ((Im z)^2 + cmod (1 - z^2)) - (Re z)^2) / 2
    using abs_Re_le_cmod [of 1-z^2] by (subst Imz) (simp add: Re_power2)
  ultimately show False
    by (simp add: cmod_power2)
qed

lemma isCont_Arccos:
  assumes (Im z = 0 ⟹ |Re z| < 1)
    shows isCont Arccos z
proof -
  have z + i * csqrt (1 - z^2) ∉ ℝ≤0
    by (metis complex_nonpos_Reals_iff isCont_Arccos_lemma assms)
  with assms show ?thesis
    unfolding Arccos_def
    by (simp_all add: one_minus_z2_notin_nonpos_Reals assms)
qed

lemma isCont_Arccos' [simp]:
  isCont f z ⟹ (Im (f z) = 0 ⟹ |Re (f z)| < 1) ⟹ isCont (λx. Arccos (f x)) z
  by (blast intro: isCont_o2 [OF _ isCont_Arccos])

lemma cos_Arccos [simp]: cos(Arccos z) = z
proof -
  have z*2 + i * (2 * csqrt (1 - z^2)) = 0 ⟷ z*2 + i * csqrt (1 - z^2)*2 = 0
    by (simp add: algebra_simps)  — Cancelling a factor of 2
  moreover have ... ⟷ z + i * csqrt (1 - z^2) = 0
    by (metis distrib_right mult_eq_0_iff zero_neq_numeral)
  ultimately show ?thesis
   by (simp add: cos_exp_eq Arccos_def Arccos_body_lemma exp_minus field_simps
flip: power2_eq_square)
qed

lemma Arccos_cos:
    assumes 0 < Re z ∧ Re z < pi ∨
            Re z = 0 ∧ 0 ≤ Im z ∨
            Re z = pi ∧ Im z ≤ 0
      shows Arccos(cos z) = z
proof -
  have *: ((i - (exp (i * z))^2 * i) / (2 * exp (i * z))) = sin z
    by (simp add: sin_exp_eq exp_minus field_simps power2_eq_square)
  have 1 - (exp (i * z) + inverse (exp (i * z)))^2 / 4 = ((i - (exp (i * z))^2 * i) /
(2 * exp (i * z)))^2
    by (simp add: field_simps power2_eq_square)
  then have Arccos(cos z) = - (i * Ln ((exp (i * z) + inverse (exp (i * z))) / 2
+
                        i * csqrt (((i - (exp (i * z))^2 * i) / (2 * exp (i * z)))^2)))
    by (simp add: cos_exp_eq Arccos_def exp_minus power_divide)
  also have ... = - (i * Ln ((exp (i * z) + inverse (exp (i * z))) / 2 +
```

```
                    i * ((i - (exp (i * z))^2 * i) / (2 * exp (i * z)))))
    apply (subst csqrt_square)
    using assms Re_sin_pos [of z] Im_sin_nonneg [of z] Im_sin_nonneg2 [of z]
    by (auto simp: * Re_sin Im_sin)
  also have ... = - (i * Ln (exp (i*z)))
    by (simp add: field_simps power2_eq_square)
  also have ... = z
    using assms
    by (subst Complex_Transcendental.Ln_exp, auto)
  finally show ?thesis .
qed

lemma Arccos_unique:
    ⟦cos z = w;
      0 < Re z ∧ Re z < pi ∨
      Re z = 0 ∧ 0 ≤ Im z ∨
      Re z = pi ∧ Im z ≤ 0⟧ ⟹ Arccos w = z
  using Arccos_cos by blast

lemma Arccos_0 [simp]: Arccos 0 = pi/2
  by (rule Arccos_unique) auto

lemma Arccos_1 [simp]: Arccos 1 = 0
  by (rule Arccos_unique) auto

lemma Arccos_minus1: Arccos(-1) = pi
  by (rule Arccos_unique) auto

lemma has_field_derivative_Arccos:
  assumes (Im z = 0 ⟹ |Re z| < 1)
    shows (Arccos has_field_derivative - inverse(sin(Arccos z))) (at z)
proof -
  have x^2 ≠ -1 for x::real
    by (sos ((R<1 + (([~1] * A=0) + (R<1 * (R<1 * [x___]^2))))))
  with assms have (cos (Arccos z))^2 ≠ 1
    by (auto simp: complex_eq_iff Re_power2 Im_power2 abs_square_eq_1)
  then have - sin (Arccos z) ≠ 0
    by (metis cos_squared_eq diff_0_right mult_zero_left neg_0_equal_iff_equal
power2_eq_square)
  then have (Arccos has_field_derivative inverse(- sin(Arccos z))) (at z)
    by (rule has_field_derivative_inverse_basic [OF DERIV_cos _ _ open_ball
[of z 1]])
       (auto intro: isCont_Arccos assms)
  then show ?thesis
    by simp
qed

declare has_field_derivative_Arcsin [derivative_intros]
declare has_field_derivative_Arcsin [THEN DERIV_chain2, derivative_intros]
```

lemma *field_differentiable_at_Arccos*:
  $(Im\ z = 0 \Longrightarrow |Re\ z| < 1) \Longrightarrow Arccos\ field\_differentiable\ at\ z$
 using *field_differentiable_def has_field_derivative_Arccos* **by** *blast*

lemma *field_differentiable_within_Arccos*:
  $(Im\ z = 0 \Longrightarrow |Re\ z| < 1) \Longrightarrow Arccos\ field\_differentiable\ (at\ z\ within\ s)$
 using *field_differentiable_at_Arccos field_differentiable_within_subset* **by** *blast*

lemma *continuous_within_Arccos*:
  $(Im\ z = 0 \Longrightarrow |Re\ z| < 1) \Longrightarrow continuous\ (at\ z\ within\ s)\ Arccos$
 using *continuous_at_imp_continuous_within isCont_Arccos* **by** *blast*

lemma *continuous_on_Arccos* [*continuous_intros*]:
  $(\bigwedge z.\ z \in s \Longrightarrow Im\ z = 0 \Longrightarrow |Re\ z| < 1) \Longrightarrow continuous\_on\ s\ Arccos$
 by (*simp add*: *continuous_at_imp_continuous_on*)

lemma *holomorphic_on_Arccos*: $(\bigwedge z.\ z \in s \Longrightarrow Im\ z = 0 \Longrightarrow |Re\ z| < 1) \Longrightarrow$ *Arccos holomorphic_on s*
 by (*simp add*: *field_differentiable_within_Arccos holomorphic_on_def*)

6.18.27 Upper and Lower Bounds for Inverse Sine and Cosine

lemma *Arcsin_bounds*: $|Re\ z| < 1 \Longrightarrow |Re(Arcsin\ z)| < pi/2$
 unfolding *Re_Arcsin*
 by (*blast intro*: *Re_Ln_pos_lt_imp Arcsin_range_lemma*)

lemma *Arccos_bounds*: $|Re\ z| < 1 \Longrightarrow 0 < Re(Arccos\ z) \wedge Re(Arccos\ z) < pi$
 unfolding *Re_Arccos*
 by (*blast intro*!: *Im_Ln_pos_lt_imp Arccos_range_lemma*)

lemma *Re_Arccos_bounds*: $-pi < Re(Arccos\ z) \wedge Re(Arccos\ z) \leq pi$
 unfolding *Re_Arccos*
 by (*blast intro*!: *mpi_less_Im_Ln Im_Ln_le_pi Arccos_body_lemma*)

lemma *Re_Arccos_bound*: $|Re(Arccos\ z)| \leq pi$
 by (*meson Re_Arccos_bounds abs_le_iff less_eq_real_def minus_less_iff*)

lemma *Im_Arccos_bound*: $|Im\ (Arccos\ w)| \leq cmod\ w$
proof −
 have $(Im\ (Arccos\ w))^2 \leq (cmod\ (cos\ (Arccos\ w)))^2 - (cos\ (Re\ (Arccos\ w)))^2$
  using *norm_cos_squared* [*of Arccos w*] *real_le_abs_sinh* [*of Im* (*Arccos w*)]
  by (*simp only*: *abs_le_square_iff*) (*simp add*: *field_split_simps*)
 also have $\ldots \leq (cmod\ w)^2$
  by (*auto simp*: *cmod_power2*)
 finally show *?thesis*
  using *abs_le_square_iff* **by** *force*
qed

lemma *Re_Arcsin_bounds*: $-pi < Re(Arcsin\ z)$ & $Re(Arcsin\ z) \leq pi$
unfolding *Re_Arcsin*
by (*blast intro*!: *mpi_less_Im_Ln Im_Ln_le_pi Arcsin_body_lemma*)

lemma *Re_Arcsin_bound*: $|Re(Arcsin\ z)| \leq pi$
by (*meson Re_Arcsin_bounds abs_le_iff less_eq_real_def minus_less_iff*)

lemma *norm_Arccos_bounded*:
fixes $w :: complex$
shows $norm\ (Arccos\ w) \leq pi + norm\ w$
proof −
have *Re*: $(Re\ (Arccos\ w))^2 \leq pi^2$ $(Im\ (Arccos\ w))^2 \leq (cmod\ w)^2$
using *Re_Arccos_bound* [*of w*] *Im_Arccos_bound* [*of w*] *abs_le_square_iff* **by** *force*+
have $Arccos\ w \cdot Arccos\ w \leq pi^2 + (cmod\ w)^2$
using *Re* **by** (*simp add*: *dot_square_norm cmod_power2* [*of Arccos w*])
then have $cmod\ (Arccos\ w) \leq pi + cmod\ (cos\ (Arccos\ w))$
by (*smt* (*verit*) *Im_Arccos_bound Re_Arccos_bound cmod_le cos_Arccos*)
then show $cmod\ (Arccos\ w) \leq pi + cmod\ w$
by *auto*
qed

6.18.28 Interrelations between Arcsin and Arccos

lemma *cos_Arcsin_nonzero*: $z^2 \neq 1 \Longrightarrow cos(Arcsin\ z) \neq 0$
by (*metis diff_0_right power_zero_numeral sin_Arcsin sin_squared_eq*)

lemma *sin_Arccos_nonzero*: $z^2 \neq 1 \Longrightarrow sin(Arccos\ z) \neq 0$
by (*metis add.right_neutral cos_Arccos power2_eq_square power_zero_numeral sin_cos_squared_add3*)

lemma *cos_sin_csqrt*:
assumes $0 < cos(Re\ z) \ \vee \ cos(Re\ z) = 0 \wedge Im\ z * sin(Re\ z) \leq 0$
shows $cos\ z = csqrt(1 - (sin\ z)^2)$
proof (*rule csqrt_unique* [*THEN sym*])
show $(cos\ z)^2 = 1 - (sin\ z)^2$
by (*simp add*: *cos_squared_eq*)
qed (*use assms* **in** ‹*auto simp*: *Re_cos Im_cos add_pos_pos mult_le_0_iff zero_le_mult_iff*›)

lemma *sin_cos_csqrt*:
assumes $0 < sin(Re\ z) \ \vee \ sin(Re\ z) = 0 \wedge 0 \leq Im\ z * cos(Re\ z)$
shows $sin\ z = csqrt(1 - (cos\ z)^2)$
proof (*rule csqrt_unique* [*THEN sym*])
show $(sin\ z)^2 = 1 - (cos\ z)^2$
by (*simp add*: *sin_squared_eq*)
qed (*use assms* **in** ‹*auto simp*: *Re_sin Im_sin add_pos_pos mult_le_0_iff zero_le_mult_iff*›)

lemma *Arcsin_Arccos_csqrt_pos*:

$(0 < Re\ z \vee Re\ z = 0 \wedge 0 \leq Im\ z) \Longrightarrow Arcsin\ z = Arccos(csqrt(1 - z^2))$
by (*simp add*: *Arcsin_def Arccos_def Complex.csqrt_square add.commute*)

lemma *Arccos_Arcsin_csqrt_pos*:
$(0 < Re\ z \vee Re\ z = 0 \wedge 0 \leq Im\ z) \Longrightarrow Arccos\ z = Arcsin(csqrt(1 - z^2))$
by (*simp add*: *Arcsin_def Arccos_def Complex.csqrt_square add.commute*)

lemma *sin_Arccos*:
$0 < Re\ z \vee Re\ z = 0 \wedge 0 \leq Im\ z \Longrightarrow sin(Arccos\ z) = csqrt(1 - z^2)$
by (*simp add*: *Arccos_Arcsin_csqrt_pos*)

lemma *cos_Arcsin*:
$0 < Re\ z \vee Re\ z = 0 \wedge 0 \leq Im\ z \Longrightarrow cos(Arcsin\ z) = csqrt(1 - z^2)$
by (*simp add*: *Arcsin_Arccos_csqrt_pos*)

6.18.29 Relationship with Arcsin on the Real Numbers

lemma *of_real_arcsin*: $|x| \leq 1 \Longrightarrow of\_real(arcsin\ x) = Arcsin(of\_real\ x)$
by (*smt* (*verit*, *best*) *Arcsin_sin Im_complex_of_real Re_complex_of_real arcsin sin_of_real*)

lemma *Im_Arcsin_of_real*: $|x| \leq 1 \Longrightarrow Im\ (Arcsin\ (of\_real\ x)) = 0$
by (*metis Im_complex_of_real of_real_arcsin*)

corollary *Arcsin_in_Reals* [*simp*]: $z \in \mathbb{R} \Longrightarrow |Re\ z| \leq 1 \Longrightarrow Arcsin\ z \in \mathbb{R}$
by (*metis Im_Arcsin_of_real Re_complex_of_real Reals_cases complex_is_Real_iff*)

lemma *arcsin_eq_Re_Arcsin*: $|x| \leq 1 \Longrightarrow arcsin\ x = Re\ (Arcsin\ (of\_real\ x))$
by (*metis Re_complex_of_real of_real_arcsin*)

6.18.30 Relationship with Arccos on the Real Numbers

lemma *of_real_arccos*: $|x| \leq 1 \Longrightarrow of\_real(arccos\ x) = Arccos(of\_real\ x)$
by (*smt* (*verit*, *del_insts*) *Arccos_unique Im_complex_of_real Re_complex_of_real arccos_lbound*
arccos_ubound cos_arccos_abs cos_of_real)

lemma *Im_Arccos_of_real*: $|x| \leq 1 \Longrightarrow Im\ (Arccos\ (of\_real\ x)) = 0$
by (*metis Im_complex_of_real of_real_arccos*)

corollary *Arccos_in_Reals* [*simp*]: $z \in \mathbb{R} \Longrightarrow |Re\ z| \leq 1 \Longrightarrow Arccos\ z \in \mathbb{R}$
by (*metis Im_Arccos_of_real complex_is_Real_iff of_real_Re*)

lemma *arccos_eq_Re_Arccos*: $|x| \leq 1 \Longrightarrow arccos\ x = Re\ (Arccos\ (of\_real\ x))$
by (*metis Re_complex_of_real of_real_arccos*)

6.18.31 Continuity results for arcsin and arccos

lemma *continuous_on_Arcsin_real* [*continuous_intros*]:
continuous_on $\{w \in \mathbb{R}.\ |Re\ w| \leq 1\}$ *Arcsin*

proof −
have *continuous_on* {*w* ∈ ℝ. |*Re w*| ≤ *1*} (λ*x*. *complex_of_real* (*arcsin* (*Re x*))) =
continuous_on {*w* ∈ ℝ. |*Re w*| ≤ *1*} (λ*x*. *complex_of_real* (*Re* (*Arcsin* (*of_real* (*Re x*)))))
by (*rule continuous_on_cong* [*OF refl*]) (*simp add*: *arcsin_eq_Re_Arcsin*)
also have ... = *?thesis*
by (*rule continuous_on_cong* [*OF refl*]) *simp*
finally show *?thesis*
using *continuous_on_arcsin* [*OF continuous_on_Re* [*OF continuous_on_id*], *of* {*w* ∈ ℝ. |*Re w*| ≤ *1*}]
continuous_on_of_real
by *fastforce*
qed

lemma *continuous_within_Arcsin_real*:
continuous (*at z within* {*w* ∈ ℝ. |*Re w*| ≤ *1*}) *Arcsin*
using *closed_real_abs_le continuous_on_Arcsin_real continuous_on_eq_continuous_within*

continuous_within_closed_nontrivial **by** *blast*

lemma *continuous_on_Arccos_real*:
continuous_on {*w* ∈ ℝ. |*Re w*| ≤ *1*} *Arccos*
proof −
have *continuous_on* {*w* ∈ ℝ. |*Re w*| ≤ *1*} (λ*x*. *complex_of_real* (*arccos* (*Re x*))) =
continuous_on {*w* ∈ ℝ. |*Re w*| ≤ *1*} (λ*x*. *complex_of_real* (*Re* (*Arccos* (*of_real* (*Re x*)))))
by (*rule continuous_on_cong* [*OF refl*]) (*simp add*: *arccos_eq_Re_Arccos*)
also have ... = *?thesis*
by (*rule continuous_on_cong* [*OF refl*]) *simp*
finally show *?thesis*
using *continuous_on_arccos* [*OF continuous_on_Re* [*OF continuous_on_id*], *of* {*w* ∈ ℝ. |*Re w*| ≤ *1*}]
continuous_on_of_real
by *fastforce*
qed

lemma *continuous_within_Arccos_real*:
continuous (*at z within* {*w* ∈ ℝ. |*Re w*| ≤ *1*}) *Arccos*
using *closed_real_abs_le continuous_on_Arccos_real continuous_on_eq_continuous_within*

continuous_within_closed_nontrivial **by** *blast*

lemma *sinh_ln_complex*: *x* ≠ *0* ⟹ *sinh* (*ln x* :: *complex*) = (*x* − *inverse x*) / *2*
by (*simp add*: *sinh_def exp_minus scaleR_conv_of_real exp_of_real*)

lemma *cosh_ln_complex*: *x* ≠ *0* ⟹ *cosh* (*ln x* :: *complex*) = (*x* + *inverse x*) / *2*
by (*simp add*: *cosh_def exp_minus scaleR_conv_of_real*)

lemma *tanh_ln_complex*: $x \neq 0 \Longrightarrow tanh\ (ln\ x :: complex) = (x\ \hat{}\ 2 - 1)\ /\ (x\ \hat{}\ 2 + 1)$
by (*simp add*: *tanh_def sinh_ln_complex cosh_ln_complex divide_simps power2_eq_square*)

6.18.32 Roots of unity

theorem *complex_root_unity*:
fixes *j*::*nat*
assumes $n \neq 0$
shows $exp(2 * of\_real\ pi * \mathrm{i} * of\_nat\ j\ /\ of\_nat\ n)\hat{}n = 1$
by (*metis assms bot_nat_0.not_eq_extremum exp_divide_power_eq exp_of_nat2_mult exp_two_pi_i power_one*)

lemma *complex_root_unity_eq*:
fixes *j*::*nat* **and** *k*::*nat*
assumes $1 \leq n$
shows $(exp(2 * of\_real\ pi * \mathrm{i} * of\_nat\ j\ /\ of\_nat\ n) = exp(2 * of\_real\ pi * \mathrm{i} * of\_nat\ k\ /\ of\_nat\ n)$
$\longleftrightarrow j\ mod\ n = k\ mod\ n)$
proof −
have $(\exists z{::}int.\ \mathrm{i} * (of\_nat\ j * (of\_real\ pi * 2)) =$
$\mathrm{i} * (of\_nat\ k * (of\_real\ pi * 2)) + \mathrm{i} * (of\_int\ z * (of\_nat\ n * (of\_real\ pi * 2)))) \longleftrightarrow$
$(\exists z{::}int.\ of\_nat\ j * (\mathrm{i} * (of\_real\ pi * 2)) =$
$(of\_nat\ k + of\_nat\ n * of\_int\ z) * (\mathrm{i} * (of\_real\ pi * 2)))$
by (*simp add*: *algebra_simps*)
also have $\ldots \longleftrightarrow (\exists z{::}int.\ of\_nat\ j = of\_nat\ k + of\_nat\ n * (of\_int\ z :: complex))$
by *simp*
also have $\ldots \longleftrightarrow (\exists z{::}int.\ of\_nat\ j = of\_nat\ k + of\_nat\ n * z)$
by (*metis* (*mono_tags*, *opaque_lifting*) *of_int_add of_int_eq_iff of_int_mult of_int_of_nat_eq*)
also have $\ldots \longleftrightarrow int\ j\ mod\ int\ n = int\ k\ mod\ int\ n$
by (*auto simp*: *mod_eq_dvd_iff dvd_def algebra_simps*)
also have $\ldots \longleftrightarrow j\ mod\ n = k\ mod\ n$
by (*metis of_nat_eq_iff zmod_int*)
finally have $(\exists z.\ \mathrm{i} * (of\_nat\ j * (of\_real\ pi * 2)) =$
$\mathrm{i} * (of\_nat\ k * (of\_real\ pi * 2)) + \mathrm{i} * (of\_int\ z * (of\_nat\ n * (of\_real\ pi * 2)))) \longleftrightarrow j\ mod\ n = k\ mod\ n$ **.**
note $* = this$
show *?thesis*
using *assms* **by** (*simp add*: *exp_eq field_split_simps* $*$)
qed

corollary *bij_betw_roots_unity*:
$bij\_betw\ (\lambda j.\ exp(2 * of\_real\ pi * \mathrm{i} * of\_nat\ j\ /\ of\_nat\ n))$
$\{..<n\}\ \ \{exp(2 * of\_real\ pi * \mathrm{i} * of\_nat\ j\ /\ of\_nat\ n) \mid j.\ j < n\}$
by (*auto simp*: *bij_betw_def inj_on_def complex_root_unity_eq*)

lemma *complex_root_unity_eq_1*:
fixes *j*::*nat* **and** *k*::*nat*
assumes *1 ≤ n*
shows *exp(2 * of_real pi * i * of_nat j / of_nat n) = 1 ⟷ n dvd j*
proof −
have *1 = exp(2 * of_real pi * i * (of_nat n / of_nat n))*
using *assms* **by** *simp*
then have *exp(2 * of_real pi * i * (of_nat j / of_nat n)) = 1 ⟷ j mod n = n mod n*
using *complex_root_unity_eq* [*of n j n*] *assms*
by *simp*
then show *?thesis*
by *auto*
qed

lemma *finite_complex_roots_unity_explicit*:
finite {*exp(2 * of_real pi * i * of_nat j / of_nat n) | j*::*nat. j < n*}
by *simp*

lemma *card_complex_roots_unity_explicit*:
card {*exp(2 * of_real pi * i * of_nat j / of_nat n) | j*::*nat. j < n*} = *n*
by (*simp add*: *Finite_Set.bij_betw_same_card* [*OF bij_betw_roots_unity*, *symmetric*])

lemma *complex_roots_unity*:
assumes *1 ≤ n*
shows {*z*::*complex. z^n = 1*} = {*exp(2 * of_real pi * i * of_nat j / of_nat n) | j. j < n*}
apply (*rule Finite_Set.card_seteq* [*symmetric*])
using *assms*
apply (*auto simp*: *card_complex_roots_unity_explicit finite_roots_unity complex_root_unity card_roots_unity*)
done

lemma *card_complex_roots_unity*: *1 ≤ n ⟹ card* {*z*::*complex. z^n = 1*} = *n*
by (*simp add*: *card_complex_roots_unity_explicit complex_roots_unity*)

lemma *complex_not_root_unity*:
1 ≤ n ⟹ ∃ *u*::*complex. norm u = 1* ∧ *u^n ≠ 1*
apply (*rule_tac x=exp* (*of_real pi * i * of_real* (*1 / n*)) **in** *exI*)
apply (*auto simp*: *Re_complex_div_eq_0 exp_of_nat_mult* [*symmetric*] *mult_ac exp_Euler*)
done

end

6.19 Harmonic Numbers

theory *Harmonic_Numbers*
imports
Complex_Transcendental
Summation_Tests
begin

The definition of the Harmonic Numbers and the Euler-Mascheroni constant. Also provides a reasonably accurate approximation of *ln 2* and the Euler-Mascheroni constant.

6.19.1 The Harmonic numbers

definition *harm* :: *nat* ⇒ *'a* :: *real_normed_field* **where**
harm n = (∑ *k*=*1*..*n*. *inverse* (*of_nat k*))

lemma *harm_altdef*: *harm n* = (∑ *k*<*n*. *inverse* (*of_nat* (*Suc k*)))
unfolding *harm_def* **by** (*induction n*) *simp_all*

lemma *harm_Suc*: *harm* (*Suc n*) = *harm n* + *inverse* (*of_nat* (*Suc n*))
by (*simp add*: *harm_def*)

lemma *harm_nonneg*: *harm n* ≥ (*0* :: *'a* :: {*real_normed_field*,*linordered_field*})
unfolding *harm_def* **by** (*intro sum_nonneg*) *simp_all*

lemma *harm_pos*: *n* > *0* ⟹ *harm n* > (*0* :: *'a* :: {*real_normed_field*,*linordered_field*})
unfolding *harm_def* **by** (*intro sum_pos*) *simp_all*

lemma *harm_mono*: *m* ≤ *n* ⟹ *harm m* ≤ (*harm n* :: *'a* :: {*real_normed_field*,*linordered_field*})
by(*simp add*: *harm_def sum_mono2*)

lemma *of_real_harm*: *of_real* (*harm n*) = *harm n*
unfolding *harm_def* **by** *simp*

lemma *abs_harm* [*simp*]: (*abs* (*harm n*) :: *real*) = *harm n*
using *harm_nonneg*[*of n*] **by** (*rule abs_of_nonneg*)

lemma *norm_harm*: *norm* (*harm n*) = *harm n*
by (*subst of_real_harm* [*symmetric*]) (*simp add*: *harm_nonneg*)

lemma *harm_expand*:
harm 0 = *0*
harm (*Suc 0*) = *1*
harm (*numeral n*) = *harm* (*pred_numeral n*) + *inverse* (*numeral n*)
proof −
have *numeral n* = *Suc* (*pred_numeral n*) **by** *simp*
also have *harm* ... = *harm* (*pred_numeral n*) + *inverse* (*numeral n*)
by (*subst harm_Suc*, *subst numeral_eq_Suc*[*symmetric*]) *simp*

finally show *harm* (*numeral n*) = *harm* (*pred_numeral n*) + *inverse* (*numeral n*) .
qed (*simp_all add*: *harm_def*)

theorem *not_convergent_harm*: ¬*convergent* (*harm* :: *nat* ⇒ 'a :: *real_normed_field*)
proof −
have *convergent* (λ*n*. *norm* (*harm n* :: 'a)) ⟷
convergent (*harm* :: *nat* ⇒ *real*) **by** (*simp add*: *norm_harm*)
also have ... ⟷ *convergent* (λ*n*. ∑ *k*=*Suc 0*..*Suc n*. *inverse* (*of_nat k*) :: *real*)
unfolding *harm_def*[*abs_def*] **by** (*subst convergent_Suc_iff*) *simp_all*
also have ... ⟷ *convergent* (λ*n*. ∑ *k*≤*n*. *inverse* (*of_nat* (*Suc k*)) :: *real*)
by (*subst sum.shift_bounds_cl_Suc_ivl*) (*simp add*: *atLeast0AtMost*)
also have ... ⟷ *summable* (λ*n*. *inverse* (*of_nat n*) :: *real*)
by (*subst summable_Suc_iff* [*symmetric*]) (*simp add*: *summable_iff_convergent'*)
also have ¬... **by** (*rule not_summable_harmonic*)
finally show *?thesis* **by** (*blast dest*: *convergent_norm*)
qed

lemma *harm_pos_iff* [*simp*]: *harm n* > (*0* :: 'a :: {*real_normed_field*,*linordered_field*})
⟷ *n* > *0*
by (*rule iffI*, *cases n*, *simp add*: *harm_expand*, *simp*, *rule harm_pos*)

lemma *ln_diff_le_inverse*:
assumes *x* ≥ (*1*::*real*)
shows *ln* (*x* + *1*) − *ln x* < *1* / *x*
proof −
from *assms* **have** ∃ *z*>*x*. *z* < *x* + *1* ∧ *ln* (*x* + *1*) − *ln x* = (*x* + *1* − *x*) ∗ *inverse z*
by (*intro MVT2*) (*auto intro*!: *derivative_eq_intros simp*: *field_simps*)
then obtain *z* **where** *z*: *z* > *x z* < *x* + *1 ln* (*x* + *1*) − *ln x* = *inverse z* **by** *auto*
have *ln* (*x* + *1*) − *ln x* = *inverse z* **by** *fact*
also from *z*(*1*,*2*) *assms* **have** ... < *1* / *x* **by** (*simp add*: *field_simps*)
finally show *?thesis* .
qed

lemma *ln_le_harm*: *ln* (*real n* + *1*) ≤ (*harm n* :: *real*)
proof (*induction n*)
fix *n* **assume** *IH*: *ln* (*real n* + *1*) ≤ *harm n*
have *ln* (*real* (*Suc n*) + *1*) = *ln* (*real n* + *1*) + (*ln* (*real n* + *2*) − *ln* (*real n* + *1*)) **by** *simp*
also have (*ln* (*real n* + *2*) − *ln* (*real n* + *1*)) ≤ *1* / *real* (*Suc n*)
using *ln_diff_le_inverse*[*of real n* + *1*] **by** (*simp add*: *add_ac*)
also note *IH*
also have *harm n* + *1* / *real* (*Suc n*) = *harm* (*Suc n*) **by** (*simp add*: *harm_Suc field_simps*)
finally show *ln* (*real* (*Suc n*) + *1*) ≤ *harm* (*Suc n*) **by** − *simp*
qed (*simp_all add*: *harm_def*)

lemma *harm_at_top*: *filterlim* (*harm* :: *nat* ⇒ *real*) *at_top sequentially*

proof (*rule filterlim_at_top_mono*)
show *eventually* (λ*n*. *harm n* ≥ *ln* (*real* (*Suc n*))) *at_top*
using *ln_le_harm* **by** (*intro always_eventually allI*) (*simp_all add*: *add_ac*)
show *filterlim* (λ*n*. *ln* (*real* (*Suc n*))) *at_top sequentially*
by (*intro filterlim_compose*[*OF ln_at_top*] *filterlim_compose*[*OF filterlim_real_sequentially*]
filterlim_Suc)
qed

6.19.2 The Euler-Mascheroni constant

The limit of the difference between the partial harmonic sum and the natural logarithm (approximately 0.577216). This value occurs e.g. in the definition of the Gamma function.

definition *euler_mascheroni* :: *'a* :: *real_normed_algebra_1* **where**
euler_mascheroni = *of_real* (*lim* (λ*n*. *harm n* − *ln* (*of_nat n*)))

lemma *of_real_euler_mascheroni* [*simp*]: *of_real euler_mascheroni* = *euler_mascheroni*
by (*simp add*: *euler_mascheroni_def*)

lemma *harm_ge_ln*: *harm n* ≥ *ln* (*real n* + *1*)
proof −
have *ln* (*n* + *1*) = (∑*j*<*n*. *ln* (*real* (*Suc j* + *1*)) − *ln* (*real* (*j* + *1*)))
by (*subst sum_lessThan_telescope*) *auto*
also have ... ≤ (∑*j*<*n*. *1* / (*Suc j*))
proof (*intro sum_mono*, *clarify*)
fix *j* **assume** *j*: *j* < *n*
have ∃ξ. ξ > *real j* + *1* ∧ ξ < *real j* + *2* ∧
ln (*real j* + *2*) − *ln* (*real j* + *1*) = (*real j* + *2* − (*real j* + *1*)) ∗ (*1* / ξ)
by (*intro MVT2*) (*auto intro*!: *derivative_eq_intros*)
then obtain ξ :: *real*
where ξ: ξ ∈ {*real j* + *1*..*real j* + *2*} *ln* (*real j* + *2*) − *ln* (*real j* + *1*) = *1* / ξ
by *auto*
note ξ(*2*)
also have *1* / ξ ≤ *1* / (*Suc j*)
using ξ(*1*) **by** (*auto simp*: *field_simps*)
finally show *ln* (*real* (*Suc j* + *1*)) − *ln* (*real* (*j* + *1*)) ≤ *1* / (*Suc j*)
by (*simp add*: *add_ac*)
qed
also have ... = *harm n*
by (*simp add*: *harm_altdef field_simps*)
finally show *?thesis* **by** (*simp add*: *add_ac*)
qed

lemma *decseq_harm_diff_ln*: *decseq* (λ*n*. *harm* (*Suc n*) − *ln* (*Suc n*))
proof (*rule decseq_SucI*)
fix *m* :: *nat*
define *n* **where** *n* = *Suc m*
have *n* > *0* **by** (*simp add*: *n_def*)

```
  have convex_on {0<..} (λx :: real. −ln x)
    by (rule convex_on_realI[where f' = λx. −1/x])
       (auto intro!: derivative_eq_intros simp: field_simps)
  hence (−1 / (n + 1)) * (real n − real (n + 1)) ≤ (− ln (real n)) − (−ln (real
(n + 1)))
    using ‹n > 0› by (intro convex_on_imp_above_tangent[where A = {0<..}])
                     (auto intro!: derivative_eq_intros simp: interior_open)
  thus harm (Suc n) − ln (Suc n) ≤ harm n − ln n
    by (auto simp: harm_Suc field_simps)
qed

lemma euler_mascheroni_sequence_nonneg:
  assumes n > 0
  shows   harm n − ln (real n) ≥ (0 :: real)
proof −
  have ln (real n) ≤ ln (real n + 1)
    using assms by simp
  also have ... ≤ harm n
    by (rule harm_ge_ln)
  finally show ?thesis by simp
qed

lemma euler_mascheroni_convergent: convergent (λn. harm n − ln n)
proof −
  have harm (Suc n) − ln (real (Suc n)) ≥ 0 for n :: nat
    using euler_mascheroni_sequence_nonneg[of Suc n] by simp
  hence convergent (λn. harm (Suc n) − ln (Suc n))
   by (intro Bseq_monoseq_convergent decseq_bounded[of _ 0] decseq_harm_diff_ln
decseq_imp_monoseq)
      auto
  thus ?thesis
    by (subst (asm) convergent_Suc_iff)
qed

lemma euler_mascheroni_sequence_decreasing:
  m > 0 ⟹ m ≤ n ⟹ harm n − ln (of_nat n) ≤ harm m − ln (of_nat m ::
real)
  using decseqD[OF decseq_harm_diff_ln, of m − 1 n − 1] by simp

lemma euler_mascheroni_LIMSEQ:
  (λn. harm n − ln (of_nat n) :: real) ⟶ euler_mascheroni
  unfolding euler_mascheroni_def
 by (simp add: convergent_LIMSEQ_iff [symmetric] euler_mascheroni_convergent)

lemma euler_mascheroni_LIMSEQ_of_real:
  (λn. of_real (harm n − ln (of_nat n))) ⟶
     (euler_mascheroni :: 'a :: {real_normed_algebra_1, topological_space})
proof −
  have (λn. of_real (harm n − ln (of_nat n))) ⟶ (of_real (euler_mascheroni)
```

```
:: 'a)
  by (intro tendsto_of_real euler_mascheroni_LIMSEQ)
 thus ?thesis by simp
qed

lemma euler_mascheroni_sum_real:
 (λn. inverse (of_nat (n+1)) + ln (of_nat (n+1)) − ln (of_nat (n+2)) :: real)
     sums euler_mascheroni
using sums_add[OF telescope_sums[OF LIMSEQ_Suc[OF euler_mascheroni_LIMSEQ]]
                 telescope_sums'[OF LIMSEQ_inverse_real_of_nat]]
 by (simp_all add: harm_def algebra_simps)

lemma euler_mascheroni_sum:
  (λn. inverse (of_nat (n+1)) + of_real (ln (of_nat (n+1))) − of_real (ln
(of_nat (n+2))))
     sums (euler_mascheroni :: 'a :: {banach, real_normed_field})
proof −
 have (λn. of_real (inverse (of_nat (n+1)) + ln (of_nat (n+1)) − ln (of_nat
(n+2))))
     sums (of_real euler_mascheroni :: 'a :: {banach, real_normed_field})
  by (subst sums_of_real_iff) (rule euler_mascheroni_sum_real)
 thus ?thesis by simp
qed

theorem alternating_harmonic_series_sums: (λk. (−1)^k / real_of_nat (Suc
k)) sums ln 2
proof −
 let ?f = λn. harm n − ln (real_of_nat n)
 let ?g = λn. if even n then 0 else (2::real)
 let ?em = λn. harm n − ln (real_of_nat n)
  have eventually (λn. ?em (2*n) − ?em n + ln 2 = (∑k<2*n. (−1)^k /
real_of_nat (Suc k))) at_top
   using eventually_gt_at_top[of 0::nat]
 proof eventually_elim
   fix n :: nat assume n: n > 0
   have (∑k<2*n. (−1)^k / real_of_nat (Suc k)) =
           (∑k<2*n. ((−1)^k + ?g k) / of_nat (Suc k)) − (∑k<2*n. ?g k /
of_nat (Suc k))
     by (simp add: sum.distrib algebra_simps divide_inverse)
   also have (∑k<2*n. ((−1)^k + ?g k) / real_of_nat (Suc k)) = harm (2*n)
     unfolding harm_altdef by (intro sum.cong) (auto simp: field_simps)
   also have (∑k<2*n. ?g k / real_of_nat (Suc k)) = (∑k|k<2*n ∧ odd k. ?g
k / of_nat (Suc k))
     by (intro sum.mono_neutral_right) auto
   also have ... = (∑k|k<2*n ∧ odd k. 2 / (real_of_nat (Suc k)))
     by (intro sum.cong) auto
   also have (∑k|k<2*n ∧ odd k. 2 / (real_of_nat (Suc k))) = harm n
     unfolding harm_altdef
    by (intro sum.reindex_cong[of λn. 2*n+1]) (auto simp: inj_on_def field_simps
```

```
elim!: oddE)
  also have harm (2*n) - harm n = ?em (2*n) - ?em n + ln 2 using n
    by (simp_all add: algebra_simps ln_mult)
  finally show ?em (2*n) - ?em n + ln 2 = (∑k<2*n. (-1) ^ k / real_of_nat
(Suc k)) ..
 qed
 moreover have (λn. ?em (2*n) - ?em n + ln (2::real))
                  ⟶ euler_mascheroni - euler_mascheroni + ln 2
   by (intro tendsto_intros euler_mascheroni_LIMSEQ filterlim_compose[OF
euler_mascheroni_LIMSEQ]
          filterlim_subseq) (auto simp: strict_mono_def)
 hence (λn. ?em (2*n) - ?em n + ln (2::real)) ⟶ ln 2 by simp
 ultimately have (λn. (∑k<2*n. (-1) ^ k / real_of_nat (Suc k))) ⟶ ln 2
   by (blast intro: Lim_transform_eventually)

 moreover have summable (λk. (-1) ^ k * inverse (real_of_nat (Suc k)))
   using LIMSEQ_inverse_real_of_nat
   by (intro summable_Leibniz(1) decseq_imp_monoseq decseq_SucI) simp_all
 hence A: (λn. ∑k<n. (-1) ^ k / real_of_nat (Suc k)) ⟶ (∑k. (-1) ^ k /
real_of_nat (Suc k))
   by (simp add: summable_sums_iff divide_inverse sums_def)
 from filterlim_compose[OF this filterlim_subseq[of (*) (2::nat)]]
   have (λn. ∑k<2*n. (-1) ^ k / real_of_nat (Suc k)) ⟶ (∑k. (-1) ^ k /
real_of_nat (Suc k))
   by (simp add: strict_mono_def)
 ultimately have (∑k. (- 1) ^ k / real_of_nat (Suc k)) = ln 2 by (intro
LIMSEQ_unique)
 with A show ?thesis by (simp add: sums_def)
qed

lemma alternating_harmonic_series_sums':
 (λk. inverse (real_of_nat (2*k+1)) - inverse (real_of_nat (2*k+2))) sums ln
2
unfolding sums_def
proof (rule Lim_transform_eventually)
 show (λn. ∑k<2*n. (-1) ^ k / (real_of_nat (Suc k))) ⟶ ln 2
   using alternating_harmonic_series_sums unfolding sums_def
   by (rule filterlim_compose) (rule mult_nat_left_at_top, simp)
 show eventually (λn. (∑k<2*n. (-1) ^ k / (real_of_nat (Suc k))) =
              (∑k<n. inverse (real_of_nat (2*k+1)) - inverse (real_of_nat
(2*k+2)))) sequentially
 proof (intro always_eventually allI)
   fix n :: nat
   show (∑k<2*n. (-1) ^ k / (real_of_nat (Suc k))) =
              (∑k<n. inverse (real_of_nat (2*k+1)) - inverse (real_of_nat
(2*k+2)))
     by (induction n) (simp_all add: inverse_eq_divide)
 qed
qed
```

6.19.3 Bounds on the Euler-Mascheroni constant

lemma *ln_inverse_approx_le*:
assumes (*x*::*real*) > *0* *a* > *0*
shows *ln* (*x* + *a*) − *ln* *x* ≤ *a* ∗ (*inverse* *x* + *inverse* (*x* + *a*))/*2* (**is** _ ≤ *?A*)
proof −
define *f'* **where** *f'* = (*inverse* (*x* + *a*) − *inverse* *x*)/*a*
let *?f* = λ*t*. (*t* − *x*) ∗ *f'* + *inverse* *x*
let *?F* = λ*t*. (*t* − *x*)^2 ∗ *f'* / *2* + *t* ∗ *inverse* *x*

have *deriv*: ∃ *D*. ((λ*x*. *?F* *x* − *ln* *x*) *has_field_derivative* *D*) (*at* ξ) ∧ *D* ≥ *0*
if ξ ≥ *x* ξ ≤ *x* + *a* **for** ξ
proof −
from *that* *assms* **have** *t*: *0* ≤ (ξ − *x*) / *a* (ξ − *x*) / *a* ≤ *1* **by** *simp_all*
have *inverse* ξ = *inverse* ((*1* − (ξ − *x*) / *a*) ∗*R* *x* + ((ξ − *x*) / *a*) ∗*R* (*x* + *a*)) (**is** _ = *?A*)
using *assms* **by** (*simp* *add*: *field_simps*)
also from *assms* **have** *convex_on* {*x*..*x*+*a*} *inverse* **by** (*intro* *convex_on_inverse*) *auto*
from *convex_onD_Icc*[*OF* *this* _ *t*] *assms*
have *?A* ≤ (*1* − (ξ − *x*) / *a*) ∗ *inverse* *x* + (ξ − *x*) / *a* ∗ *inverse* (*x* + *a*) **by** *simp*
also have ... = (ξ − *x*) ∗ *f'* + *inverse* *x* **using** *assms*
by (*simp* *add*: *f'_def* *divide_simps*) (*simp* *add*: *field_simps*)
finally have *?f* ξ − *1* / ξ ≥ *0* **by** (*simp* *add*: *field_simps*)
moreover have ((λ*x*. *?F* *x* − *ln* *x*) *has_field_derivative* *?f* ξ − *1* / ξ) (*at* ξ)
using *that* *assms* **by** (*auto* *intro*!: *derivative_eq_intros* *simp*: *field_simps*)
ultimately show *?thesis* **by** *blast*
qed
have *?F* *x* − *ln* *x* ≤ *?F* (*x* + *a*) − *ln* (*x* + *a*)
by (*rule* *DERIV_nonneg_imp_nondecreasing*[*of* *x* *x* + *a*, *OF* _ *deriv*]) (*use* *assms* **in** *auto*)
thus *?thesis*
using *assms* **by** (*simp* *add*: *f'_def* *divide_simps*) (*simp* *add*: *algebra_simps* *power2_eq_square*)*?*
qed

lemma *ln_inverse_approx_ge*:
assumes (*x*::*real*) > *0* *x* < *y*
shows *ln* *y* − *ln* *x* ≥ *2* ∗ (*y* − *x*) / (*x* + *y*) (**is** _ ≥ *?A*)
proof −
define *m* **where** *m* = (*x*+*y*)/*2*
define *f'* **where** *f'* = −*inverse* (*m*^*2*)
from *assms* **have** *m*: *m* > *0* **by** (*simp* *add*: *m_def*)
let *?F* = λ*t*. (*t* − *m*)^*2* ∗ *f'* / *2* + *t* / *m*
let *?f* = λ*t*. (*t* − *m*) ∗ *f'* + *inverse* *m*

have *deriv*: ∃ *D*. ((λ*x*. *ln* *x* − *?F* *x*) *has_field_derivative* *D*) (*at* ξ) ∧ *D* ≥ *0*
if ξ ≥ *x* ξ ≤ *y* **for** ξ
proof −

```
    from that assms have inverse ξ - inverse m ≥ f' * (ξ - m)
      by (intro convex_on_imp_above_tangent[of {0<..}] convex_on_inverse)
         (auto simp: m_def interior_open f'_def power2_eq_square intro!: derivative_eq_intros)
    hence 1 / ξ - ?f ξ ≥ 0 by (simp add: field_simps f'_def)
    moreover have ((λx. ln x - ?F x) has_field_derivative 1 / ξ - ?f ξ) (at ξ)
      using that assms m by (auto intro!: derivative_eq_intros simp: field_simps)
    ultimately show ?thesis by blast
  qed
  have ln x - ?F x ≤ ln y - ?F y
    by (rule DERIV_nonneg_imp_nondecreasing[of x y, OF _ deriv]) (use assms in auto)
  hence ln y - ln x ≥ ?F y - ?F x
    by (simp add: algebra_simps)
  also have ?F y - ?F x = ?A
     using assms by (simp add: f'_def m_def divide_simps) (simp add: algebra_simps power2_eq_square)
  finally show ?thesis .
qed

lemma euler_mascheroni_lower:
            euler_mascheroni ≥ harm (Suc n) - ln (real_of_nat (n + 2)) + 1/real_of_nat (2 * (n + 2))
    and euler_mascheroni_upper:
            euler_mascheroni ≤ harm (Suc n) - ln (real_of_nat (n + 2)) + 1/real_of_nat (2 * (n + 1))
proof -
  define D :: _ ⇒ real
     where D n = inverse (of_nat (n+1)) + ln (of_nat (n+1)) - ln (of_nat (n+2)) for n
  let ?g = λn. ln (of_nat (n+2)) - ln (of_nat (n+1)) - inverse (of_nat (n+1)) :: real
  define inv where [abs_def]: inv n = inverse (real_of_nat n) for n
  fix n :: nat
 note summable = sums_summable[OF euler_mascheroni_sum_real, folded D_def]
  have sums: (λk. (inv (Suc (k + (n+1))) - inv (Suc (Suc k + (n+1))))/2) sums ((inv (Suc (0 + (n+1))) - 0)/2)
    unfolding inv_def
   by (intro sums_divide telescope_sums' LIMSEQ_ignore_initial_segment LIMSEQ_inverse_real_of_nat)
  have sums': (λk. (inv (Suc (k + n)) - inv (Suc (Suc k + n)))/2) sums ((inv (Suc (0 + n)) - 0)/2)
    unfolding inv_def
   by (intro sums_divide telescope_sums' LIMSEQ_ignore_initial_segment LIMSEQ_inverse_real_of_nat)
  from euler_mascheroni_sum_real have euler_mascheroni = (∑ k. D k)
    by (simp add: sums_iff D_def)
  also have ... = (∑ k. D (k + Suc n)) + (∑ k≤n. D k)
    by (subst suminf_split_initial_segment[OF summable, of Suc n],
```

```
      subst lessThan_Suc_atMost) simp
  finally have sum: (∑k≤n. D k) − euler_mascheroni = −(∑k. D (k + Suc n))
by simp

  note sum
  also have ... ≤ −(∑k. (inv (k + Suc n + 1) − inv (k + Suc n + 2)) / 2)
  proof (intro le_imp_neg_le suminf_le allI summable_ignore_initial_segment[OF
summable])
    fix k' :: nat
    define k where k = k' + Suc n
    hence k: k > 0 by (simp add: k_def)
    have real_of_nat (k+1) > 0 by (simp add: k_def)
    with ln_inverse_approx_le[OF this zero_less_one]
      have ln (of_nat k + 2) − ln (of_nat k + 1) ≤ (inv (k+1) + inv (k+2))/2
      by (simp add: inv_def add_ac)
    hence (inv (k+1) − inv (k+2))/2 ≤ inv (k+1) + ln (of_nat (k+1)) − ln
(of_nat (k+2))
      by (simp add: field_simps)
    also have ... = D k unfolding D_def inv_def ..
    finally show D (k' + Suc n) ≥ (inv (k' + Suc n + 1) − inv (k' + Suc n +
2)) / 2
      by (simp add: k_def)
    from sums_summable[OF sums]
      show summable (λk. (inv (k + Suc n + 1) − inv (k + Suc n + 2))/2) by
simp
  qed
  also from sums have ... = −inv (n+2) / 2 by (simp add: sums_iff)
  finally have euler_mascheroni ≥ (∑k≤n. D k) + 1 / (of_nat (2 ∗ (n+2)))
    by (simp add: inv_def field_simps)
  also have (∑k≤n. D k) = harm (Suc n) − (∑k≤n. ln (real_of_nat (Suc k+1))
− ln (of_nat (k+1)))
    unfolding harm_altdef D_def by (subst lessThan_Suc_atMost) (simp add:
sum.distrib sum_subtractf)
   also have (∑k≤n. ln (real_of_nat (Suc k+1)) − ln (of_nat (k+1))) = ln
(of_nat (n+2))
    by (subst atLeast0AtMost [symmetric], subst sum_Suc_diff) simp_all
  finally show euler_mascheroni ≥ harm (Suc n) − ln (real_of_nat (n + 2)) +
1/real_of_nat (2 ∗ (n + 2))
    by simp

  note sum
  also have −(∑k. D (k + Suc n)) ≥ −(∑k. (inv (Suc (k + n)) − inv (Suc (Suc
k + n)))/2)
  proof (intro le_imp_neg_le suminf_le allI summable_ignore_initial_segment[OF
summable])
    fix k' :: nat
    define k where k = k' + Suc n
    hence k: k > 0 by (simp add: k_def)
    have real_of_nat (k+1) > 0 by (simp add: k_def)
```

```
    from ln_inverse_approx_ge[of of_nat k + 1 of_nat k + 2]
      have 2 / (2 * real_of_nat k + 3) ≤ ln (of_nat (k+2)) − ln (real_of_nat
(k+1))
      by (simp add: add_ac)
    hence D k ≤ 1 / real_of_nat (k+1) − 2 / (2 * real_of_nat k + 3)
      by (simp add: D_def inverse_eq_divide inv_def)
     also have ... = inv ((k+1)*(2*k+3)) unfolding inv_def by (simp add:
field_simps)
    also have ... ≤ inv (2*k*(k+1)) unfolding inv_def using k
      by (intro le_imp_inverse_le)
        (simp add: algebra_simps, simp del: of_nat_add)
    also have ... = (inv k − inv (k+1))/2 unfolding inv_def using k
      by (simp add: divide_simps del: of_nat_mult) (simp add: algebra_simps)
   finally show D k ≤ (inv (Suc (k' + n)) − inv (Suc (Suc k' + n)))/2 unfolding
k_def by simp
  next
    from sums_summable[OF sums'']
      show summable (λk. (inv (Suc (k + n)) − inv (Suc (Suc k + n)))/2) by
simp
  qed
  also from sums' have (∑ k. (inv (Suc (k + n)) − inv (Suc (Suc k + n)))/2)
= inv (n+1)/2
    by (simp add: sums_iff)
  finally have euler_mascheroni ≤ (∑ k≤n. D k) + 1 / of_nat (2 * (n+1))
    by (simp add: inv_def field_simps)
 also have (∑ k≤n. D k) = harm (Suc n) − (∑ k≤n. ln (real_of_nat (Suc k+1))
− ln (of_nat (k+1)))
     unfolding harm_altdef D_def by (subst lessThan_Suc_atMost) (simp add:
sum.distrib sum_subtractf)
   also have (∑ k≤n. ln (real_of_nat (Suc k+1)) − ln (of_nat (k+1))) = ln
(of_nat (n+2))
    by (subst atLeast0AtMost [symmetric], subst sum_Suc_diff) simp_all
  finally show euler_mascheroni ≤ harm (Suc n) − ln (real_of_nat (n + 2)) +
1/real_of_nat (2 * (n + 1))
    by simp
qed

lemma euler_mascheroni_pos: euler_mascheroni > (0::real)
  using euler_mascheroni_lower[of 0] ln_2_less_1 by (simp add: harm_def)

context
begin

private lemma ln_approx_aux:
  fixes n :: nat and x :: real
  defines y ≡ (x−1)/(x+1)
  assumes x: x > 0 x ≠ 1
 shows inverse (2*y^(2*n+1)) * (ln x − (∑ k<n. 2*y^(2*k+1) / of_nat (2*k+1)))
∈
```

```
        {0..(1 / (1 - y^2) / of_nat (2*n+1))}
proof -
  from x have norm_y: norm y < 1 unfolding y_def by simp
  from power_strict_mono[OF this, of 2] have norm_y': norm y^2 < 1 by simp

  let ?f = λk. 2 * y ^ (2*k+1) / of_nat (2*k+1)
  note sums = ln_series_quadratic[OF x(1)]
  define c where c = inverse (2*y^(2*n+1))
  let ?d = c * (ln x - (∑ k<n. ?f k))
  have ⋀k. y^2^k / of_nat (2*(k+n)+1) ≤ y^2 ^ k / of_nat (2*n+1)
    by (intro divide_left_mono mult_right_mono mult_pos_pos zero_le_power[of
y^2]) simp_all
  moreover {
    have (λk. ?f (k + n)) sums (ln x - (∑ k<n. ?f k))
      using sums_split_initial_segment[OF sums] by (simp add: y_def)
    hence (λk. c * ?f (k + n)) sums ?d by (rule sums_mult)
    also have (λk. c * (2*y^(2*(k+n)+1) / of_nat (2*(k+n)+1))) =
                (λk. (c * (2*y^(2*n+1))) * ((y^2)^k / of_nat (2*(k+n)+1)))
      by (simp only: ring_distribs power_add power_mult) (simp add: mult_ac)
    also from x have c * (2*y^(2*n+1)) = 1 by (simp add: c_def y_def)
    finally have (λk. (y^2)^k / of_nat (2*(k+n)+1)) sums ?d by simp
  } note sums' = this
  moreover from norm_y' have (λk. (y^2)^k / of_nat (2*n+1)) sums (1 / (1
- y^2) / of_nat (2*n+1))
    by (intro sums_divide geometric_sums) (simp_all add: norm_power)
  ultimately have ?d ≤ (1 / (1 - y^2) / of_nat (2*n+1)) by (rule sums_le)
  moreover have c * (ln x - (∑ k<n. 2 * y ^ (2 * k + 1) / real_of_nat (2 * k
+ 1))) ≥ 0
    by (intro sums_le[OF _ sums_zero sums']) simp_all
  ultimately show ?thesis unfolding c_def by simp
qed

lemma
  fixes n :: nat and x :: real
  defines y ≡ (x-1)/(x+1)
  defines approx ≡ (∑ k<n. 2*y^(2*k+1) / of_nat (2*k+1))
  defines d ≡ y^(2*n+1) / (1 - y^2) / of_nat (2*n+1)
  assumes x: x > 1
  shows   ln_approx_bounds: ln x ∈ {approx..approx + 2*d}
  and     ln_approx_abs:    abs (ln x - (approx + d)) ≤ d
proof -
  define c where c = 2*y^(2*n+1)
  from x have c_pos: c > 0 unfolding c_def y_def
    by (intro mult_pos_pos zero_less_power) simp_all
  have A: inverse c * (ln x - (∑ k<n. 2*y^(2*k+1) / of_nat (2*k+1))) ∈
              {0.. (1 / (1 - y^2) / of_nat (2*n+1))} using assms unfolding
y_def c_def
    by (intro ln_approx_aux) simp_all
  hence inverse c * (ln x - (∑ k<n. 2*y^(2*k+1)/of_nat (2*k+1))) ≤ (1 /
```

```
(1−y^2) / of_nat (2∗n+1))
    by simp
  hence (ln x − (∑k<n. 2∗y^(2∗k+1) / of_nat (2∗k+1))) / c ≤ (1 / (1 −
y^2) / of_nat (2∗n+1))
    by (auto simp add: field_split_simps)
  with c_pos have ln x ≤ c / (1 − y^2) / of_nat (2∗n+1) + approx
    by (subst (asm) pos_divide_le_eq) (simp_all add: mult_ac approx_def)
  moreover {
    from A c_pos have 0 ≤ c ∗ (inverse c ∗ (ln x − (∑k<n. 2∗y^(2∗k+1) /
of_nat (2∗k+1))))
      by (intro mult_nonneg_nonneg[of c]) simp_all
    also have ... = (c ∗ inverse c) ∗ (ln x − (∑k<n. 2∗y^(2∗k+1) / of_nat
(2∗k+1)))
      by (simp add: mult_ac)
    also from c_pos have c ∗ inverse c = 1 by simp
    finally have ln x ≥ approx by (simp add: approx_def)
  }
  ultimately show ln x ∈ {approx..approx + 2∗d} by (simp add: c_def d_def)
  thus abs (ln x − (approx + d)) ≤ d by auto
qed

end

lemma euler_mascheroni_bounds:
  fixes n :: nat assumes n ≥ 1 defines t ≡ harm n − ln (of_nat (Suc n)) :: real
  shows euler_mascheroni ∈ {t + inverse (of_nat (2∗(n+1)))..t + inverse (of_nat
(2∗n))}
  using assms euler_mascheroni_upper[of n−1] euler_mascheroni_lower[of n−1]
  unfolding t_def by (cases n) (simp_all add: harm_Suc t_def inverse_eq_divide)

lemma euler_mascheroni_bounds′:
  fixes n :: nat assumes n ≥ 1 ln (real_of_nat (Suc n)) ∈ {l<..<u}
  shows euler_mascheroni ∈
          {harm n − u + inverse (of_nat (2∗(n+1)))<..<harm n − l + inverse
(of_nat (2∗n))}
  using euler_mascheroni_bounds[OF assms(1)] assms(2) by auto
```

Approximation of *ln* (*2*::′*a*). The lower bound is accurate to about 0.03; the upper bound is accurate to about 0.0015.

```
lemma ln2_ge_two_thirds: 2/3 ≤ ln (2::real)
  and ln2_le_25_over_36: ln (2::real) ≤ 25/36
  using ln_approx_bounds[of 2 1, simplified, simplified eval_nat_numeral, sim-
plified] by simp_all
```

Approximation of the Euler-Mascheroni constant. The lower bound is accurate to about 0.0015; the upper bound is accurate to about 0.015.

```
lemma euler_mascheroni_gt_19_over_33: (euler_mascheroni :: real) > 19/33
(is ?th1)
```

and *euler_mascheroni_less_13_over_22*: (*euler_mascheroni* :: *real*) < *13/22* (**is** *?th2*)
proof −
have *ln* (*real* (*Suc 7*)) = *3* ∗ *ln 2* **by** (*simp add*: *ln_powr* [*symmetric*])
also from *ln_approx_bounds*[*of 2 3*] **have** ... ∈ {*3∗307/443*<..<*3∗4615/6658*}
by (*simp add*: *eval_nat_numeral*)
finally have *ln* (*real* (*Suc 7*)) ∈ ... .
from *euler_mascheroni_bounds'*[*OF* _ *this*] **have** *?th1* ∧ *?th2* **by** (*simp_all add*: *harm_expand*)
thus *?th1 ?th2* **by** *blast*+
qed

end

6.20 The Gamma Function

theory *Gamma_Function*
imports
Equivalence_Lebesgue_Henstock_Integration
Summation_Tests
Harmonic_Numbers
HOL−Library.Nonpos_Ints
HOL−Library.Periodic_Fun
begin

Several equivalent definitions of the Gamma function and its most important properties. Also contains the definition and some properties of the log-Gamma function and the Digamma function and the other Polygamma functions.

Based on the Gamma function, we also prove the Weierstraß product form of the sin function and, based on this, the solution of the Basel problem (the sum over all *1 / real* (n^2).

lemma *pochhammer_eq_0_imp_nonpos_Int*:
pochhammer (*x*::*'a*::*field_char_0*) *n* = *0* ⟹ *x* ∈ $\mathbb{Z}_{\leq 0}$
by (*auto simp*: *pochhammer_eq_0_iff*)

lemma *closed_nonpos_Ints* [*simp*]: *closed* ($\mathbb{Z}_{\leq 0}$ :: *'a* :: *real_normed_algebra_1 set*)
proof −
have $\mathbb{Z}_{\leq 0}$ = (*of_int* ‘ {*n*. *n* ≤ *0*} :: *'a set*)
by (*auto elim*!: *nonpos_Ints_cases intro*!: *nonpos_Ints_of_int*)
also have *closed* ... **by** (*rule closed_of_int_image*)
finally show *?thesis* .
qed

lemma *plus_one_in_nonpos_Ints_imp*: *z* + *1* ∈ $\mathbb{Z}_{\leq 0}$ ⟹ *z* ∈ $\mathbb{Z}_{\leq 0}$
using *nonpos_Ints_diff_Nats*[*of z+1 1*] **by** *simp_all*

lemma *of_int_in_nonpos_Ints_iff*:
(*of_int n* :: *'a* :: *ring_char_0*) ∈ $\mathbb{Z}_{\leq 0}$ ⟷ *n* ≤ *0*
by (*auto simp*: *nonpos_Ints_def*)

lemma *one_plus_of_int_in_nonpos_Ints_iff*:
(*1* + *of_int n* :: *'a* :: *ring_char_0*) ∈ $\mathbb{Z}_{\leq 0}$ ⟷ *n* ≤ −*1*
proof −
have *1* + *of_int n* = (*of_int* (*n* + *1*) :: *'a*) **by** *simp*
also have ... ∈ $\mathbb{Z}_{\leq 0}$ ⟷ *n* + *1* ≤ *0* **by** (*subst of_int_in_nonpos_Ints_iff*) *simp_all*
also have ... ⟷ *n* ≤ −*1* **by** *presburger*
finally show *?thesis* **.**
qed

lemma *one_minus_of_nat_in_nonpos_Ints_iff*:
(*1* − *of_nat n* :: *'a* :: *ring_char_0*) ∈ $\mathbb{Z}_{\leq 0}$ ⟷ *n* > *0*
proof −
have (*1* − *of_nat n* :: *'a*) = *of_int* (*1* − *int n*) **by** *simp*
also have ... ∈ $\mathbb{Z}_{\leq 0}$ ⟷ *n* > *0* **by** (*subst of_int_in_nonpos_Ints_iff*) *presburger*
finally show *?thesis* **.**
qed

lemma *fraction_not_in_ints*:
assumes ¬(*n dvd m*) *n* ≠ *0*
shows *of_int m* / *of_int n* ∉ ($\mathbb{Z}$ :: *'a* :: {*division_ring*,*ring_char_0*} *set*)
proof
assume *of_int m* / (*of_int n* :: *'a*) ∈ $\mathbb{Z}$
then obtain *k* **where** *of_int m* / *of_int n* = (*of_int k* :: *'a*) **by** (*elim Ints_cases*)
with *assms* **have** *of_int m* = (*of_int* (*k* ∗ *n*) :: *'a*) **by** (*auto simp add*: *field_split_simps*)
hence *m* = *k* ∗ *n* **by** (*subst* (*asm*) *of_int_eq_iff*)
hence *n dvd m* **by** *simp*
with *assms*(*1*) **show** *False* **by** *contradiction*
qed

lemma *fraction_not_in_nats*:
assumes ¬*n dvd m n* ≠ *0*
shows *of_int m* / *of_int n* ∉ ($\mathbb{N}$ :: *'a* :: {*division_ring*,*ring_char_0*} *set*)
proof
assume *of_int m* / *of_int n* ∈ ($\mathbb{N}$ :: *'a set*)
also note *Nats_subset_Ints*
finally have *of_int m* / *of_int n* ∈ ($\mathbb{Z}$ :: *'a set*) **.**
moreover have *of_int m* / *of_int n* ∉ ($\mathbb{Z}$ :: *'a set*)
using *assms* **by** (*intro fraction_not_in_ints*)
ultimately show *False* **by** *contradiction*
qed

lemma *not_in_Ints_imp_not_in_nonpos_Ints*: *z* ∉ $\mathbb{Z}$ ⟹ *z* ∉ $\mathbb{Z}_{\leq 0}$
by (*auto simp*: *Ints_def nonpos_Ints_def*)

lemma *double_in_nonpos_Ints_imp*:
assumes $2 * (z :: 'a :: field\_char\_0) \in \mathbb{Z}_{\leq 0}$
shows $z \in \mathbb{Z}_{\leq 0} \vee z + 1/2 \in \mathbb{Z}_{\leq 0}$
proof−
from *assms* **obtain** *k* **where** *k*: $2 * z = -\ of\_nat\ k$ **by** (*elim nonpos_Ints_cases'*)
thus *?thesis* **by** (*cases even k*) (*auto elim*!: *evenE oddE simp*: *field_simps*)
qed

lemma *sin_series*: $(\lambda n.\ ((-1)\hat{}n\ /\ fact\ (2*n+1)) *_R z\hat{}(2*n+1))$ *sums sin z*
proof −
from *sin_converges*[*of z*] **have** $(\lambda n.\ sin\_coeff\ n *_R z\hat{}n)$ *sums sin z* **.**
also have $(\lambda n.\ sin\_coeff\ n *_R z\hat{}n)$ *sums sin z* $\longleftrightarrow$
$(\lambda n.\ ((-1)\hat{}n\ /\ fact\ (2*n+1)) *_R z\hat{}(2*n+1))$ *sums sin z*
by (*subst sums_mono_reindex*[*of* $\lambda n.\ 2*n+1$, *symmetric*])
(*auto simp*: *sin_coeff_def strict_mono_def ac_simps elim*!: *oddE*)
finally show *?thesis* **.**
qed

lemma *cos_series*: $(\lambda n.\ ((-1)\hat{}n\ /\ fact\ (2*n)) *_R z\hat{}(2*n))$ *sums cos z*
proof −
from *cos_converges*[*of z*] **have** $(\lambda n.\ cos\_coeff\ n *_R z\hat{}n)$ *sums cos z* **.**
also have $(\lambda n.\ cos\_coeff\ n *_R z\hat{}n)$ *sums cos z* $\longleftrightarrow$
$(\lambda n.\ ((-1)\hat{}n\ /\ fact\ (2*n)) *_R z\hat{}(2*n))$ *sums cos z*
by (*subst sums_mono_reindex*[*of* $\lambda n.\ 2*n$, *symmetric*])
(*auto simp*: *cos_coeff_def strict_mono_def ac_simps elim*!: *evenE*)
finally show *?thesis* **.**
qed

lemma *sin_z_over_z_series*:
fixes $z :: 'a :: \{real\_normed\_field, banach\}$
assumes $z \neq 0$
shows $(\lambda n.\ (-1)\hat{}n\ /\ fact\ (2*n+1) * z\hat{}(2*n))$ *sums* $(sin\ z\ /\ z)$
proof −
from *sin_series*[*of z*] **have** $(\lambda n.\ z * ((-1)\hat{}n\ /\ fact\ (2*n+1)) * z\hat{}(2*n))$ *sums sin z*
by (*simp add*: *field_simps scaleR_conv_of_real*)
from *sums_mult*[*OF this*, *of inverse z*] **and** *assms* **show** *?thesis*
by (*simp add*: *field_simps*)
qed

lemma *sin_z_over_z_series'*:
fixes $z :: 'a :: \{real\_normed\_field, banach\}$
assumes $z \neq 0$
shows $(\lambda n.\ sin\_coeff\ (n+1) *_R z\hat{}n)$ *sums* $(sin\ z\ /\ z)$
proof −
from *sums_split_initial_segment*[*OF sin_converges*[*of z*], *of 1*]
have $(\lambda n.\ z * (sin\_coeff\ (n+1) *_R z\ \hat{}\ n))$ *sums sin z* **by** *simp*

from *sums_mult*[*OF this*, *of inverse z*] *assms* **show** *?thesis* **by** (*simp add*: *field_simps*)
qed

lemma *has_field_derivative_sin_z_over_z*:
fixes *A* :: *'a* :: {*real_normed_field*,*banach*} *set*
shows ((λ*z*. *if z* = *0 then 1 else sin z* / *z*) *has_field_derivative 0*) (*at 0 within A*)
(**is** (*?f has_field_derivative ?f'*) _)
proof (*rule has_field_derivative_at_within*)
have ((λ*z*::*'a*. ∑ *n*. *of_real* (*sin_coeff* (*n*+*1*)) ∗ *z*^*n*)
has_field_derivative (∑ *n*. *diffs* (λ*n*. *of_real* (*sin_coeff* (*n*+*1*))) *n* ∗ *0*^*n*)) (*at 0*)
proof (*rule termdiffs_strong*)
from *summable_ignore_initial_segment*[*OF sums_summable*[*OF sin_converges*[*of 1*::*'a*]], *of 1*]
show *summable* (λ*n*. *of_real* (*sin_coeff* (*n*+*1*)) ∗ (*1*::*'a*)^*n*) **by** (*simp add*: *of_real_def*)
qed *simp*
also have (λ*z*::*'a*. ∑ *n*. *of_real* (*sin_coeff* (*n*+*1*)) ∗ *z*^*n*) = *?f*
proof
fix *z*
show (∑ *n*. *of_real* (*sin_coeff* (*n*+*1*)) ∗ *z*^*n*) = *?f z*
by (*cases z* = *0*) (*insert sin_z_over_z_series'*[*of z*],
simp_all add: *scaleR_conv_of_real sums_iff sin_coeff_def*)
qed
also have (∑ *n*. *diffs* (λ*n*. *of_real* (*sin_coeff* (*n* + *1*))) *n* ∗ (*0*::*'a*) ^ *n*) =
diffs (λ*n*. *of_real* (*sin_coeff* (*Suc n*))) *0* **by** *simp*
also have ... = *0* **by** (*simp add*: *sin_coeff_def diffs_def*)
finally show ((λ*z*::*'a*. *if z* = *0 then 1 else sin z* / *z*) *has_field_derivative 0*) (*at 0*) .
qed

lemma *round_Re_minimises_norm*:
norm ((*z*::*complex*) − *of_int m*) ≥ *norm* (*z* − *of_int* (*round* (*Re z*)))
proof −
let *?n* = *round* (*Re z*)
have *norm* (*z* − *of_int ?n*) = *sqrt* ((*Re z* − *of_int ?n*)^2 + (*Im z*)^2)
by (*simp add*: *cmod_def*)
also have |*Re z* − *of_int ?n*| ≤ |*Re z* − *of_int m*| **by** (*rule round_diff_minimal*)
hence *sqrt* ((*Re z* − *of_int ?n*)^2 + (*Im z*)^2) ≤ *sqrt* ((*Re z* − *of_int m*)^2 + (*Im z*)^2)
by (*intro real_sqrt_le_mono add_mono*) (*simp_all add*: *abs_le_square_iff*)
also have ... = *norm* (*z* − *of_int m*) **by** (*simp add*: *cmod_def*)
finally show *?thesis* .
qed

lemma *Re_pos_in_ball*:
assumes *Re z* > *0 t* ∈ *ball z* (*Re z*/*2*)

```
  shows   Re t > 0
proof −
  have Re (z − t) ≤ norm (z − t) by (rule complex_Re_le_cmod)
  also from assms have ... < Re z / 2 by (simp add: dist_complex_def)
  finally show Re t > 0 using assms by simp
qed

lemma no_nonpos_Int_in_ball_complex:
  assumes Re z > 0 t ∈ ball z (Re z/2)
  shows   t ∉ ℤ≤0
  using Re_pos_in_ball[OF assms] by (force elim!: nonpos_Ints_cases)

lemma no_nonpos_Int_in_ball:
  assumes t ∈ ball z (dist z (round (Re z)))
  shows   t ∉ ℤ≤0
proof
  assume t ∈ ℤ≤0
  then obtain n where t = of_int n by (auto elim!: nonpos_Ints_cases)
  have dist z (of_int n) ≤ dist z t + dist t (of_int n) by (rule dist_triangle)
  also from assms have dist z t < dist z (round (Re z)) by simp
  also have ... ≤ dist z (of_int n)
    using round_Re_minimises_norm[of z] by (simp add: dist_complex_def)
  finally have dist t (of_int n) > 0 by simp
  with ‹t = of_int n› show False by simp
qed

lemma no_nonpos_Int_in_ball':
  assumes (z :: 'a :: {euclidean_space,real_normed_algebra_1}) ∉ ℤ≤0
  obtains d where d > 0 ⋀t. t ∈ ball z d ⟹ t ∉ ℤ≤0
proof (rule that)
  from assms show setdist {z} ℤ≤0 > 0 by (subst setdist_gt_0_compact_closed)
auto
next
  fix t assume t ∈ ball z (setdist {z} ℤ≤0)
  thus t ∉ ℤ≤0 using setdist_le_dist[of z {z} t ℤ≤0] by force
qed

lemma no_nonpos_Real_in_ball:
  assumes z: z ∉ ℝ≤0 and t: t ∈ ball z (if Im z = 0 then Re z / 2 else abs (Im
z) / 2)
  shows   t ∉ ℝ≤0
using z
proof (cases Im z = 0)
  assume A: Im z = 0
  with z have Re z > 0 by (force simp add: complex_nonpos_Reals_iff)
 with t A Re_pos_in_ball[of z t] show ?thesis by (force simp add: complex_nonpos_Reals_iff)
next
  assume A: Im z ≠ 0
  have abs (Im z) − abs (Im t) ≤ abs (Im z − Im t) by linarith
```

also have ... = *abs* (*Im* (*z* − *t*)) **by** *simp*
also have ... ≤ *norm* (*z* − *t*) **by** (*rule abs_Im_le_cmod*)
also from *A t* **have** ... ≤ *abs* (*Im z*) / *2* **by** (*simp add*: *dist_complex_def*)
finally have *abs* (*Im t*) > *0* **using** *A* **by** *simp*
thus *?thesis* **by** (*force simp add*: *complex_nonpos_Reals_iff*)
qed

6.20.1 The Euler form and the logarithmic Gamma function

We define the Gamma function by first defining its multiplicative inverse *rGamma*. This is more convenient because *rGamma* is entire, which makes proofs of its properties more convenient because one does not have to watch out for discontinuities. (e.g. *rGamma* fulfils *rGamma z* = *z* ∗ *rGamma* (*z* + *1*) everywhere, whereas the Γ function does not fulfil the analogous equation on the non-positive integers)

We define the Γ function (resp. its reciprocale) in the Euler form. This form has the advantage that it is a relatively simple limit that converges everywhere. The limit at the poles is 0 (due to division by 0). The functional equation *Gamma* (*z* + *1*) = *z* ∗ *Gamma z* follows immediately from the definition.

definition *Gamma_series* :: ('*a* :: {*banach*,*real_normed_field*}) ⇒ *nat* ⇒ '*a* **where**
Gamma_series z n = *fact n* ∗ *exp* (*z* ∗ *of_real* (*ln* (*of_nat n*))) / *pochhammer z* (*n*+*1*)

definition *Gamma_series'* :: ('*a* :: {*banach*,*real_normed_field*}) ⇒ *nat* ⇒ '*a* **where**
Gamma_series' z n = *fact* (*n* − *1*) ∗ *exp* (*z* ∗ *of_real* (*ln* (*of_nat n*))) / *pochhammer z n*

definition *rGamma_series* :: ('*a* :: {*banach*,*real_normed_field*}) ⇒ *nat* ⇒ '*a* **where**
rGamma_series z n = *pochhammer z* (*n*+*1*) / (*fact n* ∗ *exp* (*z* ∗ *of_real* (*ln* (*of_nat n*))))

lemma *Gamma_series_altdef*: *Gamma_series z n* = *inverse* (*rGamma_series z n*)
and *rGamma_series_altdef*: *rGamma_series z n* = *inverse* (*Gamma_series z n*)
unfolding *Gamma_series_def rGamma_series_def* **by** *simp_all*

lemma *rGamma_series_minus_of_nat*:
eventually (λ*n*. *rGamma_series* (− *of_nat k*) *n* = *0*) *sequentially*
using *eventually_ge_at_top*[*of k*]
by *eventually_elim* (*auto simp*: *rGamma_series_def pochhammer_of_nat_eq_0_iff*)

lemma *Gamma_series_minus_of_nat*:
eventually (λ*n*. *Gamma_series* (− *of_nat k*) *n* = *0*) *sequentially*

using *eventually_ge_at_top*[*of k*]
by *eventually_elim* (*auto simp*: *Gamma_series_def pochhammer_of_nat_eq_0_iff*)

lemma *Gamma_series'_minus_of_nat*:
eventually (λ*n*. *Gamma_series'* (− *of_nat k*) *n* = *0*) *sequentially*
using *eventually_gt_at_top*[*of k*]
by *eventually_elim* (*auto simp*: *Gamma_series'_def pochhammer_of_nat_eq_0_iff*)

lemma *rGamma_series_nonpos_Ints_LIMSEQ*: *z* ∈ $\mathbb{Z}_{\leq 0}$ ⟹ *rGamma_series z* ⟶ *0*
by (*elim nonpos_Ints_cases'*, *hypsubst*, *subst tendsto_cong*, *rule rGamma_series_minus_of_nat*, *simp*)

lemma *Gamma_series_nonpos_Ints_LIMSEQ*: *z* ∈ $\mathbb{Z}_{\leq 0}$ ⟹ *Gamma_series z* ⟶ *0*
by (*elim nonpos_Ints_cases'*, *hypsubst*, *subst tendsto_cong*, *rule Gamma_series_minus_of_nat*, *simp*)

lemma *Gamma_series'_nonpos_Ints_LIMSEQ*: *z* ∈ $\mathbb{Z}_{\leq 0}$ ⟹ *Gamma_series' z* ⟶ *0*
by (*elim nonpos_Ints_cases'*, *hypsubst*, *subst tendsto_cong*, *rule Gamma_series'_minus_of_nat*, *simp*)

lemma *Gamma_series_Gamma_series'*:
assumes *z*: *z* ∉ $\mathbb{Z}_{\leq 0}$
shows (λ*n*. *Gamma_series' z n* / *Gamma_series z n*) ⟶ *1*
proof (*rule Lim_transform_eventually*)
from *eventually_gt_at_top*[*of 0*::*nat*]
show *eventually* (λ*n*. *z* / *of_nat n* + *1* = *Gamma_series' z n* / *Gamma_series z n*) *sequentially*
proof *eventually_elim*
fix *n* :: *nat* **assume** *n*: *n* > *0*
from *n z* **have** *Gamma_series' z n* / *Gamma_series z n* = (*z* + *of_nat n*) / *of_nat n*
by (*cases n*, *simp*)
(*auto simp add*: *Gamma_series_def Gamma_series'_def pochhammer_rec'*
dest: *pochhammer_eq_0_imp_nonpos_Int plus_of_nat_eq_0_imp*)
also from *n* **have** ... = *z* / *of_nat n* + *1* **by** (*simp add*: *field_split_simps*)
finally show *z* / *of_nat n* + *1* = *Gamma_series' z n* / *Gamma_series z n* **..**
qed
have (λ*x*. *z* / *of_nat x*) ⟶ *0*
by (*rule tendsto_norm_zero_cancel*)
(*insert tendsto_mult*[*OF tendsto_const*[*of norm z*] *lim_inverse_n*],
simp add: *norm_divide inverse_eq_divide*)
from *tendsto_add*[*OF this tendsto_const*[*of 1*]] **show** (λ*n*. *z* / *of_nat n* + *1*) ⟶ *1* **by** *simp*
qed

We now show that the series that defines the Γ function in the Euler form

converges and that the function defined by it is continuous on the complex halfspace with positive real part.

We do this by showing that the logarithm of the Euler series is continuous and converges locally uniformly, which means that the log-Gamma function defined by its limit is also continuous.

This will later allow us to lift holomorphicity and continuity from the log-Gamma function to the inverse of the Gamma function, and from that to the Gamma function itself.

definition *ln_Gamma_series* :: ('*a* :: {*banach*,*real_normed_field*,*ln*}) ⇒ *nat* ⇒ '*a* **where**
ln_Gamma_series z n = *z* ∗ *ln* (*of_nat n*) − *ln z* − (∑ *k*=*1*..*n*. *ln* (*z* / *of_nat k* + *1*))

definition *ln_Gamma_series'* :: ('*a* :: {*banach*,*real_normed_field*,*ln*}) ⇒ *nat* ⇒ '*a* **where**
ln_Gamma_series' z n =
− *euler_mascheroni*∗*z* − *ln z* + (∑ *k*=*1*..*n*. *z* / *of_nat n* − *ln* (*z* / *of_nat k* + *1*))

definition *ln_Gamma* :: ('*a* :: {*banach*,*real_normed_field*,*ln*}) ⇒ '*a* **where**
ln_Gamma z = *lim* (*ln_Gamma_series z*)

We now show that the log-Gamma series converges locally uniformly for all complex numbers except the non-positive integers. We do this by proving that the series is locally Cauchy.

context
begin

private lemma *ln_Gamma_series_complex_converges_aux*:
fixes *z* :: *complex* **and** *k* :: *nat*
assumes *z*: *z* ≠ *0* **and** *k*: *of_nat k* ≥ *2*∗*norm z k* ≥ *2*
shows *norm* (*z* ∗ *ln* (*1* − *1*/*of_nat k*) + *ln* (*z*/*of_nat k* + *1*)) ≤ *2*∗(*norm z* + *norm z*^*2*) / *of_nat k*^*2*
proof −
let *?k* = *of_nat k* :: *complex* **and** *?z* = *norm z*
have *z* ∗*ln* (*1* − *1*/*?k*) + *ln* (*z*/*?k*+*1*) = *z*∗(*ln* (*1* − *1*/*?k* :: *complex*) + *1*/*?k*) + (*ln* (*1*+*z*/*?k*) − *z*/*?k*)
by (*simp add*: *algebra_simps*)
also have *norm* ... ≤ *?z* ∗ *norm* (*ln* (*1*−*1*/*?k*) + *1*/*?k* :: *complex*) + *norm* (*ln* (*1*+*z*/*?k*) − *z*/*?k*)
by (*subst norm_mult* [*symmetric*], *rule norm_triangle_ineq*)
also have *norm* (*Ln* (*1* + −*1*/*?k*) − (−*1*/*?k*)) ≤ (*norm* (−*1*/*?k*))^2 / (*1* − *norm*(−*1*/*?k*))
using *k* **by** (*intro Ln_approx_linear*) (*simp add*: *norm_divide*)
hence *?z* ∗ *norm* (*ln* (*1*−*1*/*?k*) + *1*/*?k*) ≤ *?z* ∗ ((*norm* (*1*/*?k*))^*2* / (*1* − *norm* (*1*/*?k*)))
by (*intro mult_left_mono*) *simp_all*

```
  also have ... ≤ (?z * (of_nat k / (of_nat k − 1))) / of_nat k^2 using k
    by (simp add: field_simps power2_eq_square norm_divide)
  also have ... ≤ (?z * 2) / of_nat k^2 using k
    by (intro divide_right_mono mult_left_mono) (simp_all add: field_simps)
  also have norm (ln (1+z/?k) − z/?k) ≤ norm (z/?k)^2 / (1 − norm (z/?k))
using k
    by (intro Ln_approx_linear) (simp add: norm_divide)
  hence norm (ln (1+z/?k) − z/?k) ≤ ?z^2 / of_nat k^2 / (1 − ?z / of_nat k)
    by (simp add: field_simps norm_divide)
  also have ... ≤ (?z^2 * (of_nat k / (of_nat k − ?z))) / of_nat k^2 using k
    by (simp add: field_simps power2_eq_square)
  also have ... ≤ (?z^2 * 2) / of_nat k^2 using k
    by (intro divide_right_mono mult_left_mono) (simp_all add: field_simps)
  also note add_divide_distrib [symmetric]
  finally show ?thesis by (simp only: distrib_left mult.commute)
qed

lemma ln_Gamma_series_complex_converges:
  assumes z: z ∉ ℤ≤0
  assumes d: d > 0 ⋀n. n ∈ ℤ≤0 ⟹ norm (z − of_int n) > d
  shows uniformly_convergent_on (ball z d) (λn z. ln_Gamma_series z n :: com-
plex)
proof (intro Cauchy_uniformly_convergent uniformly_Cauchy_onI')
  fix e :: real assume e: e > 0
  define e'' where e'' = (SUP t∈ball z d. norm t + norm t^2)
  define e' where e' = e / (2*e'')
  have bounded ((λt. norm t + norm t^2) ` cball z d)
   by (intro compact_imp_bounded compact_continuous_image) (auto intro!: con-
tinuous_intros)
  hence bounded ((λt. norm t + norm t^2) ` ball z d) by (rule bounded_subset)
auto
 hence bdd: bdd_above ((λt. norm t + norm t^2) ` ball z d) by (rule bounded_imp_bdd_above)

  with z d(1) d(2)[of −1] have e''_pos: e'' > 0 unfolding e''_def
    by (subst less_cSUP_iff) (auto intro!: add_pos_nonneg bexI[of _ z])
  have e'': norm t + norm t^2 ≤ e'' if t ∈ ball z d for t unfolding e''_def using
that
    by (rule cSUP_upper[OF _ bdd])
  from e z e''_pos have e': e' > 0 unfolding e'_def
   by (intro divide_pos_pos mult_pos_pos add_pos_pos) (simp_all add: field_simps)

  have summable (λk. inverse ((real_of_nat k)^2))
    by (rule inverse_power_summable) simp
  from summable_partial_sum_bound[OF this e']
  obtain M where M: ⋀m n. M ≤ m ⟹ norm (∑ k = m..n. inverse ((real k)^2))
< e'
    by auto

  define N where N = max 2 (max (nat ⌈2 * (norm z + d)⌉) M)
```

```
  {
    from d have ⌈2 * (cmod z + d)⌉ ≥ ⌈0::real⌉
      by (intro ceiling_mono mult_nonneg_nonneg add_nonneg_nonneg) simp_all
    hence 2 * (norm z + d) ≤ of_nat (nat ⌈2 * (norm z + d)⌉) unfolding N_def
      by (simp_all)
    also have ... ≤ of_nat N unfolding N_def
      by (subst of_nat_le_iff) (rule max.coboundedI2, rule max.cobounded1)
    finally have of_nat N ≥ 2 * (norm z + d) .
    moreover have N ≥ 2 N ≥ M unfolding N_def by simp_all
    moreover have (∑ k=m..n. 1/(of_nat k)^2) < e' if m ≥ N for m n
      using M[OF order.trans[OF ‹N ≥ M› that]] unfolding real_norm_def
    by (subst (asm) abs_of_nonneg) (auto intro: sum_nonneg simp: field_split_simps)
    moreover note calculation
  } note N = this

  show ∃ M. ∀ t∈ball z d. ∀ m≥M. ∀ n>m. dist (ln_Gamma_series t m) (ln_Gamma_series
t n) < e
    unfolding dist_complex_def
  proof (intro exI[of _ N] ballI allI impI)
    fix t m n assume t: t ∈ ball z d and mn: m ≥ N n > m
     from d(2)[of 0] t have 0 < dist z 0 − dist z t by (simp add: field_simps
dist_complex_def)
    also have dist z 0 − dist z t ≤ dist 0 t using dist_triangle[of 0 z t]
      by (simp add: dist_commute)
    finally have t_nz: t ≠ 0 by auto

    have norm t ≤ norm z + norm (t − z) by (rule norm_triangle_sub)
   also from t have norm (t − z) < d by (simp add: dist_complex_def norm_minus_commute)
    also have 2 * (norm z + d) ≤ of_nat N by (rule N)
    also have N ≤ m by (rule mn)
    finally have norm_t: 2 * norm t < of_nat m by simp

    have ln_Gamma_series t m − ln_Gamma_series t n =
            (−(t * Ln (of_nat n)) − (−(t * Ln (of_nat m)))) +
            ((∑ k=1..n. Ln (t / of_nat k + 1)) − (∑ k=1..m. Ln (t / of_nat k
+ 1)))
      by (simp add: ln_Gamma_series_def algebra_simps)
    also have (∑ k=1..n. Ln (t / of_nat k + 1)) − (∑ k=1..m. Ln (t / of_nat
k + 1)) =
                (∑ k∈{1..n}−{1..m}. Ln (t / of_nat k + 1)) using mn
      by (simp add: sum_diff)
    also from mn have {1..n}−{1..m} = {Suc m..n} by fastforce
    also have −(t * Ln (of_nat n)) − (−(t * Ln (of_nat m))) =
                  (∑ k = Suc m..n. t * Ln (of_nat (k − 1)) − t * Ln (of_nat k))
using mn
      by (subst sum_telescope'' [symmetric]) simp_all
    also have ... = (∑ k = Suc m..n. t * Ln (of_nat (k − 1) / of_nat k)) using
mn N
      by (intro sum_cong_Suc)
```

```
    (simp_all del: of_nat_Suc add: field_simps Ln_of_nat Ln_of_nat_over_of_nat)
  also have of_nat (k - 1) / of_nat k = 1 - 1 / (of_nat k :: complex) if k ∈
{Suc m..n} for k
    using that of_nat_eq_0_iff[of Suc i for i] by (cases k) (simp_all add:
field_split_simps)
  hence (∑ k = Suc m..n. t * Ln (of_nat (k - 1) / of_nat k)) =
        (∑ k = Suc m..n. t * Ln (1 - 1 / of_nat k)) using mn N
    by (intro sum.cong) simp_all
  also note sum.distrib [symmetric]
  also have norm (∑ k=Suc m..n. t * Ln (1 - 1/of_nat k) + Ln (t/of_nat k
+ 1)) ≤
    (∑ k=Suc m..n. 2 * (norm t + (norm t)^2) / (real_of_nat k)^2) using t_nz
N(2) mn norm_t
   by (intro order.trans[OF norm_sum sum_mono[OF ln_Gamma_series_complex_converges_aux]])
simp_all
  also have ... ≤ 2 * (norm t + norm t^2) * (∑ k=Suc m..n. 1 / (of_nat k)^2)
    by (simp add: sum_distrib_left)
  also have ... < 2 * (norm t + norm t^2) * e' using mn z t_nz
    by (intro mult_strict_left_mono N mult_pos_pos add_pos_pos) simp_all
  also from e''_pos have ... = e * ((cmod t + (cmod t)^2) / e'')
    by (simp add: e'_def field_simps power2_eq_square)
  also from e''[OF t] e''_pos e
    have ... ≤ e * 1 by (intro mult_left_mono) (simp_all add: field_simps)
  finally show norm (ln_Gamma_series t m - ln_Gamma_series t n) < e by
simp
 qed
qed

end

lemma ln_Gamma_series_complex_converges':
 assumes z: (z :: complex) ∉ ℤ≤0
 shows ∃ d>0. uniformly_convergent_on (ball z d) (λn z. ln_Gamma_series z
n)
proof -
 define d' where d' = Re z
 define d where d = (if d' > 0 then d' / 2 else norm (z - of_int (round d')) /
2)
 have of_int (round d') ∈ ℤ≤0 if d' ≤ 0 using that
   by (intro nonpos_Ints_of_int) (simp_all add: round_def)
 with assms have d_pos: d > 0 unfolding d_def by (force simp: not_less)

 have d < cmod (z - of_int n) if n ∈ ℤ≤0 for n
 proof (cases Re z > 0)
   case True
   from nonpos_Ints_nonpos[OF that] have n: n ≤ 0 by simp
   from True have d = Re z/2 by (simp add: d_def d'_def)
   also from n True have ... < Re (z - of_int n) by simp
   also have ... ≤ norm (z - of_int n) by (rule complex_Re_le_cmod)
```

```
    finally show ?thesis .
  next
    case False
    with assms nonpos_Ints_of_int[of round (Re z)]
      have z ≠ of_int (round d') by (auto simp: not_less)
     with False have d < norm (z − of_int (round d')) by (simp add: d_def
d'_def)
   also have ... ≤ norm (z − of_int n) unfolding d'_def by (rule round_Re_minimises_norm)
    finally show ?thesis .
  qed
  hence conv: uniformly_convergent_on (ball z d) (λn z. ln_Gamma_series z n)
    by (intro ln_Gamma_series_complex_converges d_pos z) simp_all
  from d_pos conv show ?thesis by blast
qed

lemma ln_Gamma_series_complex_converges'': (z :: complex) ∉ ℤ≤0 ⟹ con-
vergent (ln_Gamma_series z)
 by (drule ln_Gamma_series_complex_converges') (auto intro: uniformly_convergent_imp_convergent)

theorem ln_Gamma_complex_LIMSEQ: (z :: complex) ∉ ℤ≤0 ⟹ ln_Gamma_series
z ⟶ ln_Gamma z
 using ln_Gamma_series_complex_converges'' by (simp add: convergent_LIMSEQ_iff
ln_Gamma_def)

lemma exp_ln_Gamma_series_complex:
  assumes n > 0 z ∉ ℤ≤0
  shows   exp (ln_Gamma_series z n :: complex) = Gamma_series z n
proof −
  from assms obtain m where m: n = Suc m by (cases n) blast
  from assms have z ≠ 0 by (intro notI) auto
  with assms have exp (ln_Gamma_series z n) =
        (of_nat n) powr z / (z ∗ (∏k=1..n. exp (Ln (z / of_nat k + 1))))
    unfolding ln_Gamma_series_def powr_def by (simp add: exp_diff exp_sum)
  also from assms have (∏k=1..n. exp (Ln (z / of_nat k + 1))) = (∏k=1..n.
z / of_nat k + 1)
   by (intro prod.cong[OF refl], subst exp_Ln) (auto simp: field_simps plus_of_nat_eq_0_imp)
  also have ... = (∏k=1..n. z + k) / fact n
    by (simp add: fact_prod)
    (subst prod_dividef [symmetric], simp_all add: field_simps)
  also from m have z ∗ ... = (∏k=0..n. z + k) / fact n
   by (simp add: prod.atLeast0_atMost_Suc_shift prod.atLeast_Suc_atMost_Suc_shift
del: prod.cl_ivl_Suc)
  also have (∏k=0..n. z + k) = pochhammer z (Suc n)
    unfolding pochhammer_prod
   by (simp add: prod.atLeast0_atMost_Suc atLeastLessThanSuc_atLeastAtMost)
  also have of_nat n powr z / (pochhammer z (Suc n) / fact n) = Gamma_series
z n
    unfolding Gamma_series_def using assms by (simp add: field_split_simps
powr_def)
```

```
  finally show ?thesis .
qed


lemma ln_Gamma_series'_aux:
  assumes (z::complex) ∉ ℤ≤0
  shows   (λk. z / of_nat (Suc k) − ln (1 + z / of_nat (Suc k))) sums
            (ln_Gamma z + euler_mascheroni ∗ z + ln z) (is ?f sums ?s)
unfolding sums_def
proof (rule Lim_transform)
  show (λn. ln_Gamma_series z n + of_real (harm n − ln (of_nat n)) ∗ z + ln
z) ⟶ ?s
    (is ?g ⟶ _)
   by (intro tendsto_intros ln_Gamma_complex_LIMSEQ euler_mascheroni_LIMSEQ_of_real
assms)

  have A: eventually (λn. (∑k<n. ?f k) − ?g n = 0) sequentially
    using eventually_gt_at_top[of 0::nat]
  proof eventually_elim
    fix n :: nat assume n: n > 0
    have (∑k<n. ?f k) = (∑k=1..n. z / of_nat k − ln (1 + z / of_nat k))
       by (subst atLeast0LessThan [symmetric], subst sum.shift_bounds_Suc_ivl
[symmetric],
          subst atLeastLessThanSuc_atLeastAtMost) simp_all
    also have ... = z ∗ of_real (harm n) − (∑k=1..n. ln (1 + z / of_nat k))
      by (simp add: harm_def sum_subtractf sum_distrib_left divide_inverse)
    also from n have ... − ?g n = 0
      by (simp add: ln_Gamma_series_def sum_subtractf algebra_simps)
    finally show (∑k<n. ?f k) − ?g n = 0 .
  qed
   show (λn. (∑k<n. ?f k) − ?g n) ⟶ 0 by (subst tendsto_cong[OF A])
simp_all
qed


lemma uniformly_summable_deriv_ln_Gamma:
  assumes z: (z :: 'a :: {real_normed_field,banach}) ≠ 0 and d: d > 0 d ≤ norm
z/2
  shows uniformly_convergent_on (ball z d)
          (λk z. ∑i<k. inverse (of_nat (Suc i)) − inverse (z + of_nat (Suc i)))
          (is uniformly_convergent_on _ (λk z. ∑i<k. ?f i z))
proof (rule Weierstrass_m_test'_ev)
  {
    fix t assume t: t ∈ ball z d
    have norm z = norm (t + (z − t)) by simp
   have norm (t + (z − t)) ≤ norm t + norm (z − t) by (rule norm_triangle_ineq)
    also from t d have norm (z − t) < norm z / 2 by (simp add: dist_norm)
    finally have A: norm t > norm z / 2 using z by (simp add: field_simps)
```

```
    have norm t = norm (z + (t - z)) by simp
    also have ... ≤ norm z + norm (t - z) by (rule norm_triangle_ineq)
    also from t d have norm (t - z) ≤ norm z / 2 by (simp add: dist_norm
norm_minus_commute)
    also from z have ... < norm z by simp
    finally have B: norm t < 2 * norm z by simp
    note A B
  } note ball = this

  show eventually (λn. ∀t∈ball z d. norm (?f n t) ≤ 4 * norm z * inverse (of_nat
(Suc n)^2)) sequentially
    using eventually_gt_at_top apply eventually_elim
  proof safe
    fix t :: 'a assume t: t ∈ ball z d
    from z ball[OF t] have t_nz: t ≠ 0 by auto
    fix n :: nat assume n: n > nat ⌈4 * norm z⌉
    from ball[OF t] t_nz have 4 * norm z > 2 * norm t by simp
    also from n have ...  < of_nat n by linarith
    finally have n: of_nat n > 2 * norm t .
    hence of_nat n > norm t by simp
    hence t': t ≠ -of_nat (Suc n) by (intro notI) (simp del: of_nat_Suc)

    with t_nz have ?f n t = 1 / (of_nat (Suc n) * (1 + of_nat (Suc n)/t))
      by (simp add: field_split_simps eq_neg_iff_add_eq_0 del: of_nat_Suc)
    also have norm ... = inverse (of_nat (Suc n)) * inverse (norm (of_nat (Suc
n)/t + 1))
      by (simp add: norm_divide norm_mult field_split_simps del: of_nat_Suc)
    also {
      from z t_nz ball[OF t] have of_nat (Suc n) / (4 * norm z) ≤ of_nat (Suc
n) / (2 * norm t)
        by (intro divide_left_mono mult_pos_pos) simp_all
      also have ... < norm (of_nat (Suc n) / t) - norm (1 :: 'a)
        using t_nz n by (simp add: field_simps norm_divide del: of_nat_Suc)
      also have ... ≤ norm (of_nat (Suc n)/t + 1) by (rule norm_diff_ineq)
      finally have inverse (norm (of_nat (Suc n)/t + 1)) ≤ 4 * norm z / of_nat
(Suc n)
      using z by (simp add: field_split_simps norm_divide mult_ac del: of_nat_Suc)
    }
    also have inverse (real_of_nat (Suc n)) * (4 * norm z / real_of_nat (Suc
n)) =
                4 * norm z * inverse (of_nat (Suc n)^2)
              by (simp add: field_split_simps power2_eq_square del: of_nat_Suc)
    finally show norm (?f n t) ≤ 4 * norm z * inverse (of_nat (Suc n)^2)
      by (simp del: of_nat_Suc)
  qed
next
  show summable (λn. 4 * norm z * inverse ((of_nat (Suc n))^2))
   by (subst summable_Suc_iff) (simp add: summable_mult inverse_power_summable)
qed
```

6.20.2 The Polygamma functions

lemma *summable_deriv_ln_Gamma*:
  $z \neq (0 :: 'a :: \{real\_normed\_field, banach\}) \Longrightarrow$
    summable $(\lambda n.\ inverse\ (of\_nat\ (Suc\ n)) - inverse\ (z + of\_nat\ (Suc\ n)))$
  unfolding *summable_iff_convergent*
  by (*rule uniformly_convergent_imp_convergent*,
    rule uniformly_summable_deriv_ln_Gamma[*of z norm z/2*]) *simp_all*

definition *Polygamma* :: *nat* ⇒ ('*a* :: {*real_normed_field*,*banach*}) ⇒ '*a* **where**
  Polygamma n z = (*if n* = *0 then*
    $(\sum k.\ inverse\ (of\_nat\ (Suc\ k)) - inverse\ (z + of\_nat\ k)) - euler\_mascheroni$
else
    $(-1)\hat{}Suc\ n * fact\ n * (\sum k.\ inverse\ ((z + of\_nat\ k)\hat{}Suc\ n)))$

abbreviation *Digamma* :: ('*a* :: {*real_normed_field*,*banach*}) ⇒ '*a* **where**
  Digamma ≡ *Polygamma 0*

lemma *Digamma_def*:
  $Digamma\ z = (\sum k.\ inverse\ (of\_nat\ (Suc\ k)) - inverse\ (z + of\_nat\ k)) - euler\_mascheroni$
  by (*simp add*: *Polygamma_def*)

lemma *summable_Digamma*:
  assumes $(z :: 'a :: \{real\_normed\_field, banach\}) \neq 0$
  shows *summable* $(\lambda n.\ inverse\ (of\_nat\ (Suc\ n)) - inverse\ (z + of\_nat\ n))$
proof −
  have *sums*: $(\lambda n.\ inverse\ (z + of\_nat\ (Suc\ n)) - inverse\ (z + of\_nat\ n))\ sums$
                $(0 - inverse\ (z + of\_nat\ 0))$
    by (*intro telescope_sums filterlim_compose*[*OF tendsto_inverse_0*]
          tendsto_add_filterlim_at_infinity[*OF tendsto_const*] *tendsto_of_nat*)
  from *summable_add*[*OF summable_deriv_ln_Gamma*[*OF assms*] *sums_summable*[*OF sums*]]
    show *summable* $(\lambda n.\ inverse\ (of\_nat\ (Suc\ n)) - inverse\ (z + of\_nat\ n))$ **by** *simp*
qed

lemma *summable_offset*:
  assumes *summable* $(\lambda n.\ f\ (n + k) :: 'a :: real\_normed\_vector)$
  shows *summable f*
proof −
  from *assms* **have** *convergent* $(\lambda m.\ \sum n{<}m.\ f\ (n + k))$
    using *summable_iff_convergent* **by** *blast*
  hence *convergent* $(\lambda m.\ (\sum n{<}k.\ f\ n) + (\sum n{<}m.\ f\ (n + k)))$
    by (*intro convergent_add convergent_const*)
  also have $(\lambda m.\ (\sum n{<}k.\ f\ n) + (\sum n{<}m.\ f\ (n + k))) = (\lambda m.\ \sum n{<}m{+}k.\ f\ n)$
  proof
    fix *m* :: *nat*
    have $\{..<m+k\} = \{..<k\} \cup \{k..<m+k\}$ **by** *auto*

```
    also have (∑ n∈.... f n) = (∑ n<k. f n) + (∑ n=k..<m+k. f n)
      by (rule sum.union_disjoint) auto
    also have (∑ n=k..<m+k. f n) = (∑ n=0..<m+k−k. f (n + k))
      using sum.shift_bounds_nat_ivl [of f 0 k m] by simp
    finally show (∑ n<k. f n) + (∑ n<m. f (n + k)) = (∑ n<m+k. f n) by
(simp add: atLeast0LessThan)
  qed
  finally have (λa. sum f {..<a}) ⟶ lim (λm. sum f {..<m + k})
    by (auto simp: convergent_LIMSEQ_iff dest: LIMSEQ_offset)
  thus ?thesis by (auto simp: summable_iff_convergent convergent_def)
qed

lemma Polygamma_converges:
  fixes z :: 'a :: {real_normed_field,banach}
  assumes z: z ≠ 0 and n: n ≥ 2
  shows uniformly_convergent_on (ball z d) (λk z. ∑ i<k. inverse ((z + of_nat
i) ^ n))
proof (rule Weierstrass_m_test'_ev)
  define e where e = (1 + d / norm z)
  define m where m = nat ⌈norm z * e⌉
  {
    fix t assume t: t ∈ ball z d
    have norm t = norm (z + (t − z)) by simp
    also have ... ≤ norm z + norm (t − z) by (rule norm_triangle_ineq)
   also from t have norm (t − z) < d by (simp add: dist_norm norm_minus_commute)
    finally have norm t < norm z * e using z by (simp add: divide_simps e_def)
  } note ball = this

  show eventually (λk. ∀ t∈ball z d. norm (inverse ((t + of_nat k) ^ n)) ≤
          inverse (of_nat (k − m) ^ n)) sequentially
    using eventually_gt_at_top[of m] apply eventually_elim
  proof (intro ballI)
    fix k :: nat and t :: 'a assume k: k > m and t: t ∈ ball z d
    from k have real_of_nat (k − m) = of_nat k − of_nat m by (simp add:
of_nat_diff)
    also have ... ≤ norm (of_nat k :: 'a) − norm z * e
      unfolding m_def by (subst norm_of_nat) linarith
    also from ball[OF t] have ... ≤ norm (of_nat k :: 'a) − norm t by simp
    also have ... ≤ norm (of_nat k + t) by (rule norm_diff_ineq)
    finally have inverse ((norm (t + of_nat k)) ^ n) ≤ inverse (real_of_nat (k −
m) ^ n) using k n
        by (intro le_imp_inverse_le power_mono) (simp_all add: add_ac del:
of_nat_Suc)
    thus norm (inverse ((t + of_nat k) ^ n)) ≤ inverse (of_nat (k − m) ^ n)
      by (simp add: norm_inverse norm_power power_inverse)
  qed

  have summable (λk. inverse ((real_of_nat k) ^ n))
    using inverse_power_summable[of n] n by simp
```

hence *summable* (λ*k*. *inverse* ((*real_of_nat* (*k* + *m* − *m*))^*n*)) **by** *simp*
thus *summable* (λ*k*. *inverse* ((*real_of_nat* (*k* − *m*))^*n*)) **by** (*rule summable_offset*)
qed

lemma *Polygamma_converges'*:
fixes *z* :: '*a* :: {*real_normed_field*,*banach*}
assumes *z*: *z* ≠ *0* **and** *n*: *n* ≥ *2*
shows *summable* (λ*k*. *inverse* ((*z* + *of_nat k*)^*n*))
using *uniformly_convergent_imp_convergent*[*OF Polygamma_converges*[*OF assms*, *of 1*], *of z*]
by (*simp add*: *summable_iff_convergent*)

theorem *Digamma_LIMSEQ*:
fixes *z* :: '*a* :: {*banach*,*real_normed_field*}
assumes *z*: *z* ≠ *0*
shows (λ*m*. *of_real* (*ln* (*real m*)) − (∑ *n*<*m*. *inverse* (*z* + *of_nat n*))) ⟶ *Digamma z*
proof −
have (λ*n*. *of_real* (*ln* (*real n* / (*real* (*Suc n*))))) ⟶ (*of_real* (*ln 1*) :: '*a*)
by (*intro tendsto_intros LIMSEQ_n_over_Suc_n*) *simp_all*
hence (λ*n*. *of_real* (*ln* (*real n* / (*real n* + *1*)))) ⟶ (*0* :: '*a*) **by** (*simp add*: *add_ac*)
hence *lim*: (λ*n*. *of_real* (*ln* (*real n*)) − *of_real* (*ln* (*real n* + *1*))) ⟶ (*0*::'*a*)
proof (*rule Lim_transform_eventually*)
show *eventually* (λ*n*. *of_real* (*ln* (*real n* / (*real n* + *1*))) =
of_real (*ln* (*real n*)) − (*of_real* (*ln* (*real n* + *1*)) :: '*a*)) *at_top*
using *eventually_gt_at_top*[*of 0*::*nat*] **by** *eventually_elim* (*simp add*: *ln_div*)
qed

from *summable_Digamma*[*OF z*]
have (λ*n*. *inverse* (*of_nat* (*n*+*1*)) − *inverse* (*z* + *of_nat n*))
sums (*Digamma z* + *euler_mascheroni*)
by (*simp add*: *Digamma_def summable_sums*)
from *sums_diff*[*OF this euler_mascheroni_sum*]
have (λ*n*. *of_real* (*ln* (*real* (*Suc n*) + *1*)) − *of_real* (*ln* (*real n* + *1*)) − *inverse* (*z* + *of_nat n*))
sums Digamma z **by** (*simp add*: *add_ac*)
hence (λ*m*. (∑ *n*<*m*. *of_real* (*ln* (*real* (*Suc n*) + *1*)) − *of_real* (*ln* (*real n* + *1*))) −
(∑ *n*<*m*. *inverse* (*z* + *of_nat n*))) ⟶ *Digamma z*
by (*simp add*: *sums_def sum_subtractf*)
also have (λ*m*. (∑ *n*<*m*. *of_real* (*ln* (*real* (*Suc n*) + *1*)) − *of_real* (*ln* (*real n* + *1*)))) =
(λ*m*. *of_real* (*ln* (*m* + *1*)) :: '*a*)
by (*subst sum_lessThan_telescope*) *simp_all*
finally show *?thesis* **by** (*rule Lim_transform*) (*insert lim*, *simp*)
qed

theorem *Polygamma_LIMSEQ*:

```
  fixes z :: 'a :: {banach,real_normed_field}
  assumes z ≠ 0 and n > 0
  shows   (λk. inverse ((z + of_nat k) ^Suc n)) sums ((−1) ^ Suc n * Polygamma
n z / fact n)
  using Polygamma_converges'[OF assms(1), of Suc n] assms(2)
  by (simp add: sums_iff Polygamma_def)

theorem has_field_derivative_ln_Gamma_complex [derivative_intros]:
  fixes z :: complex
  assumes z: z ∉ ℝ≤0
  shows   (ln_Gamma has_field_derivative Digamma z) (at z)
proof −
  have not_nonpos_Int [simp]: t ∉ ℤ≤0 if Re t > 0 for t
    using that by (auto elim!: nonpos_Ints_cases')
  from z have z': z ∉ ℤ≤0 and z'': z ≠ 0 using nonpos_Ints_subset_nonpos_Reals
nonpos_Reals_zero_I
    by blast+
  let ?f' = λz k. inverse (of_nat (Suc k)) − inverse (z + of_nat (Suc k))
  let ?f = λz k. z / of_nat (Suc k) − ln (1 + z / of_nat (Suc k)) and ?F' = λz.
∑n. ?f' z n
  define d where d = min (norm z/2) (if Im z = 0 then Re z / 2 else abs (Im z)
/ 2)
  from z have d: d > 0 norm z/2 ≥ d by (auto simp add: complex_nonpos_Reals_iff
d_def)
  have ball: Im t = 0 ⟶ Re t > 0 if dist z t < d for t
    using no_nonpos_Real_in_ball[OF z, of t] that unfolding d_def by (force
simp add: complex_nonpos_Reals_iff)
  have sums: (λn. inverse (z + of_nat (Suc n)) − inverse (z + of_nat n)) sums
                  (0 − inverse (z + of_nat 0))
    by (intro telescope_sums filterlim_compose[OF tendsto_inverse_0]
          tendsto_add_filterlim_at_infinity[OF tendsto_const] tendsto_of_nat)

  have ((λz. ∑n. ?f z n) has_field_derivative ?F' z) (at z)
    using d z ln_Gamma_series'_aux[OF z']
   apply (intro has_field_derivative_series'(2)[of ball z d _ _ z] uniformly_summable_deriv_ln_Gamma)
   apply (auto intro!: derivative_eq_intros add_pos_pos mult_pos_pos dest!: ball
          simp: field_simps sums_iff nonpos_Reals_divide_of_nat_iff
          simp del: of_nat_Suc)
    apply (auto simp add: complex_nonpos_Reals_iff)
    done
  with z have ((λz. (∑k. ?f z k) − euler_mascheroni * z − Ln z) has_field_derivative
                ?F' z − euler_mascheroni − inverse z) (at z)
    by (force intro!: derivative_eq_intros simp: Digamma_def)
  also have ?F' z − euler_mascheroni − inverse z = (?F' z + −inverse z) −
euler_mascheroni by simp
  also from sums have −inverse z = (∑n. inverse (z + of_nat (Suc n)) −
inverse (z + of_nat n))
    by (simp add: sums_iff)
  also from sums summable_deriv_ln_Gamma[OF z'']
```

```
    have ?F' z + ... = (∑ n. inverse (of_nat (Suc n)) − inverse (z + of_nat
n))
    by (subst suminf_add) (simp_all add: add_ac sums_iff)
  also have ... − euler_mascheroni = Digamma z by (simp add: Digamma_def)
  finally have ((λz. (∑ k. ?f z k) − euler_mascheroni * z − Ln z)
                  has_field_derivative Digamma z) (at z) .
  moreover from eventually_nhds_ball[OF d(1), of z]
    have eventually (λz. ln_Gamma z = (∑ k. ?f z k) − euler_mascheroni * z −
Ln z) (nhds z)
  proof eventually_elim
    fix t assume t ∈ ball z d
    hence t ∉ ℤ≤0 by (auto dest!: ball elim!: nonpos_Ints_cases)
    from ln_Gamma_series'_aux[OF this]
      show ln_Gamma t = (∑ k. ?f t k) − euler_mascheroni * t − Ln t by (simp
add: sums_iff)
  qed
  ultimately show ?thesis by (subst DERIV_cong_ev[OF refl _ refl])
qed

declare has_field_derivative_ln_Gamma_complex[THEN DERIV_chain2, deriva-
tive_intros]


lemma Digamma_1 [simp]: Digamma (1 :: 'a :: {real_normed_field,banach}) =
− euler_mascheroni
  by (simp add: Digamma_def)

lemma Digamma_plus1:
  assumes z ≠ 0
  shows   Digamma (z+1) = Digamma z + 1/z
proof −
  have sums: (λk. inverse (z + of_nat k) − inverse (z + of_nat (Suc k)))
                sums (inverse (z + of_nat 0) − 0)
    by (intro telescope_sums'[OF filterlim_compose[OF tendsto_inverse_0]]
          tendsto_add_filterlim_at_infinity[OF tendsto_const] tendsto_of_nat)
  have Digamma (z+1) = (∑ k. inverse (of_nat (Suc k)) − inverse (z + of_nat
(Suc k))) −
         euler_mascheroni (is _ = suminf ?f − _) by (simp add: Digamma_def
add_ac)
  also have suminf ?f = (∑ k. inverse (of_nat (Suc k)) − inverse (z + of_nat
k)) +
                        (∑ k. inverse (z + of_nat k) − inverse (z + of_nat (Suc k)))
    using summable_Digamma[OF assms] sums by (subst suminf_add) (simp_all
add: add_ac sums_iff)
  also have (∑ k. inverse (z + of_nat k) − inverse (z + of_nat (Suc k))) = 1/z
    using sums by (simp add: sums_iff inverse_eq_divide)
  finally show ?thesis by (simp add: Digamma_def[of z])
qed
```

theorem *Polygamma_plus1*:
assumes $z \neq 0$
shows $\mathit{Polygamma}\ n\ (z + 1) = \mathit{Polygamma}\ n\ z + (-1)\hat{}n * \mathit{fact}\ n\ /\ (z\ \hat{}\ \mathit{Suc}\ n)$
proof (*cases* $n = 0$)
assume *n*: $n \neq 0$
let *?f* = $\lambda k.\ \mathit{inverse}\ ((z + \mathit{of\_nat}\ k)\ \hat{}\ \mathit{Suc}\ n)$
have $\mathit{Polygamma}\ n\ (z + 1) = (-1)\ \hat{}\ \mathit{Suc}\ n * \mathit{fact}\ n * (\sum k.\ ?f\ (k+1))$
using *n* **by** (*simp add*: *Polygamma_def add_ac*)
also have $(\sum k.\ ?f\ (k+1)) + (\sum k<1.\ ?f\ k) = (\sum k.\ ?f\ k)$
using *Polygamma_converges'*[*OF assms*, *of Suc n*] *n*
by (*subst suminf_split_initial_segment* [*symmetric*]) *simp_all*
hence $(\sum k.\ ?f\ (k+1)) = (\sum k.\ ?f\ k) - \mathit{inverse}\ (z\ \hat{}\ \mathit{Suc}\ n)$ **by** (*simp add*: *algebra_simps*)
also have $(-1)\ \hat{}\ \mathit{Suc}\ n * \mathit{fact}\ n * ((\sum k.\ ?f\ k) - \mathit{inverse}\ (z\ \hat{}\ \mathit{Suc}\ n)) =$
$\mathit{Polygamma}\ n\ z + (-1)\hat{}n * \mathit{fact}\ n\ /\ (z\ \hat{}\ \mathit{Suc}\ n)$ **using** *n*
by (*simp add*: *inverse_eq_divide algebra_simps Polygamma_def*)
finally show *?thesis* **.**
qed (*insert assms*, *simp add*: *Digamma_plus1 inverse_eq_divide*)

theorem *Digamma_of_nat*:
$\mathit{Digamma}\ (\mathit{of\_nat}\ (\mathit{Suc}\ n) :: 'a :: \{\mathit{real\_normed\_field}, \mathit{banach}\}) = \mathit{harm}\ n - \mathit{euler\_mascheroni}$
proof (*induction n*)
case (*Suc n*)
have $\mathit{Digamma}\ (\mathit{of\_nat}\ (\mathit{Suc}\ (\mathit{Suc}\ n)) :: 'a) = \mathit{Digamma}\ (\mathit{of\_nat}\ (\mathit{Suc}\ n) + 1)$ **by** *simp*
also have $\ldots = \mathit{Digamma}\ (\mathit{of\_nat}\ (\mathit{Suc}\ n)) + \mathit{inverse}\ (\mathit{of\_nat}\ (\mathit{Suc}\ n))$
by (*subst Digamma_plus1*) (*simp_all add*: *inverse_eq_divide del*: *of_nat_Suc*)
also have $\mathit{Digamma}\ (\mathit{of\_nat}\ (\mathit{Suc}\ n) :: 'a) = \mathit{harm}\ n - \mathit{euler\_mascheroni}$ **by** (*rule Suc*)
also have $\ldots + \mathit{inverse}\ (\mathit{of\_nat}\ (\mathit{Suc}\ n)) = \mathit{harm}\ (\mathit{Suc}\ n) - \mathit{euler\_mascheroni}$
by (*simp add*: *harm_Suc*)
finally show *?case* **.**
qed (*simp add*: *harm_def*)

lemma *Digamma_numeral*: $\mathit{Digamma}\ (\mathit{numeral}\ n) = \mathit{harm}\ (\mathit{pred\_numeral}\ n) - \mathit{euler\_mascheroni}$
by (*subst of_nat_numeral*[*symmetric*], *subst numeral_eq_Suc*, *subst Digamma_of_nat*) (*rule refl*)

lemma *Polygamma_of_real*: $x \neq 0 \Longrightarrow \mathit{Polygamma}\ n\ (\mathit{of\_real}\ x) = \mathit{of\_real}\ (\mathit{Polygamma}\ n\ x)$
unfolding *Polygamma_def* **using** *summable_Digamma*[*of x*] *Polygamma_converges'*[*of x Suc n*]
by (*simp_all add*: *suminf_of_real*)

lemma *Polygamma_Real*: $z \in \mathbb{R} \Longrightarrow z \neq 0 \Longrightarrow \mathit{Polygamma}\ n\ z \in \mathbb{R}$
by (*elim Reals_cases*, *hypsubst*, *subst Polygamma_of_real*) *simp_all*

lemma *Digamma_half_integer*:
  Digamma (*of_nat n + 1/2* :: *'a* :: {*real_normed_field*,*banach*}) =
     (∑ *k*<*n*. *2* / (*of_nat* (*2*∗*k*+*1*))) − *euler_mascheroni* − *of_real* (*2* ∗ *ln 2*)
proof (*induction n*)
  case *0*
  have *Digamma* (*1/2* :: *'a*) = *of_real* (*Digamma* (*1/2*)) **by** (*simp add*: *Polygamma_of_real* [*symmetric*])
  also have *Digamma* (*1/2*::*real*) =
                (∑ *k*. *inverse* (*of_nat* (*Suc k*)) − *inverse* (*of_nat k* + *1/2*)) − *euler_mascheroni*
    by (*simp add*: *Digamma_def add_ac*)
  also have (∑ *k*. *inverse* (*of_nat* (*Suc k*) :: *real*) − *inverse* (*of_nat k* + *1/2*)) =
            (∑ *k*. *inverse* (*1/2*) ∗ (*inverse* (*2* ∗ *of_nat* (*Suc k*)) − *inverse* (*2* ∗ *of_nat k* + *1*)))
    by (*simp_all add*: *add_ac inverse_mult_distrib*[*symmetric*] *ring_distribs del*: *inverse_divide*)
  also have ... = − *2* ∗ *ln 2* **using** *sums_minus*[*OF alternating_harmonic_series_sums'*]
    by (*subst suminf_mult*) (*simp_all add*: *algebra_simps sums_iff*)
  finally show *?case* **by** *simp*
next
  case (*Suc n*)
  have *nz*: *2* ∗ *of_nat n* + (*1*:: *'a*) ≠ *0*
    using *of_nat_neq_0*[*of 2*∗*n*] **by** (*simp only*: *of_nat_Suc*) (*simp add*: *add_ac*)
  hence *nz'*: *of_nat n* + (*1/2*::*'a*) ≠ *0* **by** (*simp add*: *field_simps*)
  have *Digamma* (*of_nat* (*Suc n*) + *1/2* :: *'a*) = *Digamma* (*of_nat n* + *1/2* + *1*) **by** *simp*
  also from *nz'* **have** ... = *Digamma* (*of_nat n* + *1/2*) + *1* / (*of_nat n* + *1/2*)
    by (*rule Digamma_plus1*)
  also from *nz nz'* **have** *1* / (*of_nat n* + *1/2* :: *'a*) = *2* / (*2* ∗ *of_nat n* + *1*)
    by (*subst divide_eq_eq*) *simp_all*
  also note *Suc*
  finally show *?case* **by** (*simp add*: *add_ac*)
qed

lemma *Digamma_one_half*: *Digamma* (*1/2*) = − *euler_mascheroni* − *of_real* (*2* ∗ *ln 2*)
  using *Digamma_half_integer*[*of 0*] **by** *simp*

lemma *Digamma_real_three_halves_pos*: *Digamma* (*3/2* :: *real*) > *0*
proof −
  have −*Digamma* (*3/2* :: *real*) = −*Digamma* (*of_nat 1* + *1/2*) **by** *simp*
  also have ... = *2* ∗ *ln 2* + *euler_mascheroni* − *2* **by** (*subst Digamma_half_integer*) *simp*
  also note *euler_mascheroni_less_13_over_22*
  also note *ln2_le_25_over_36*
  finally show *?thesis* **by** *simp*
qed

theorem *has_field_derivative_Polygamma* [*derivative_intros*]:
fixes $z :: 'a :: \{real\_normed\_field, euclidean\_space\}$
assumes z: $z \notin \mathbb{Z}_{\leq 0}$
shows (*Polygamma n has_field_derivative Polygamma* (*Suc n*) *z*) (*at z within A*)
proof (*rule has_field_derivative_at_within*, *cases n = 0*)
assume n: $n = 0$
let *?f* = $\lambda k\ z.$ *inverse* (*of_nat* (*Suc k*)) − *inverse* (*z* + *of_nat k*)
let *?F* = $\lambda z. \sum k.$ *?f k z* **and** *?f'* = $\lambda k\ z.$ *inverse* $((z + of\_nat\ k)^2)$
from *no_nonpos_Int_in_ball'*[*OF z*] **obtain** d **where** d: $0 < d$ $\bigwedge t.\ t \in ball\ z\ d \Longrightarrow t \notin \mathbb{Z}_{\leq 0}$
by *auto*
from *z* **have** *summable*: *summable* ($\lambda k.$ *inverse* (*of_nat* (*Suc k*)) − *inverse* (*z* + *of_nat k*))
by (*intro summable_Digamma*) *force*
from *z* **have** *conv*: *uniformly_convergent_on* (*ball z d*) ($\lambda k\ z.$ $\sum i<k.$ *inverse* $((z + of\_nat\ i)^2)$)
by (*intro Polygamma_converges*) *auto*
with *d* **have** *summable* ($\lambda k.$ *inverse* $((z + of\_nat\ k)^2)$) **unfolding** *summable_iff_convergent*
by (*auto dest*!: *uniformly_convergent_imp_convergent simp*: *summable_iff_convergent*)

have (*?F has_field_derivative* $(\sum k.$ *?f' k z*)) (*at z*)
proof (*rule has_field_derivative_series'*[*of ball z d _ _ z*])
fix k :: *nat* **and** t :: $'a$ **assume** t: $t \in ball\ z\ d$
from *t d*(*2*)[*of t*] **show** (($\lambda z.$ *?f k z*) *has_field_derivative ?f' k t*) (*at t within ball z d*)
by (*auto intro*!: *derivative_eq_intros simp*: *power2_eq_square simp del*: *of_nat_Suc*
dest!: *plus_of_nat_eq_0_imp elim*!: *nonpos_Ints_cases*)
qed (*insert d*(*1*) *summable conv*, (*assumption*|*simp*)+)
with *z* **show** (*Polygamma n has_field_derivative Polygamma* (*Suc n*) *z*) (*at z*)
unfolding *Digamma_def* [*abs_def*] *Polygamma_def* [*abs_def*] **using** n
by (*force simp*: *power2_eq_square intro*!: *derivative_eq_intros*)
next
assume n: $n \neq 0$
from *z* **have** *z'*: $z \neq 0$ **by** *auto*
from *no_nonpos_Int_in_ball'*[*OF z*] **obtain** d **where** d: $0 < d$ $\bigwedge t.\ t \in ball\ z\ d \Longrightarrow t \notin \mathbb{Z}_{\leq 0}$
by *auto*
define n' **where** $n' = Suc\ n$
from n **have** n': $n' \geq 2$ **by** (*simp add*: n'_def)
have ($\lambda z.$ $\sum k.$ *inverse* $((z + of\_nat\ k) \mathbin{\hat{}} n')$) *has_field_derivative*
$(\sum k.$ − *of_nat* n' ∗ *inverse* $((z + of\_nat\ k) \mathbin{\hat{}} (n'+1)))$) (*at z*)
proof (*rule has_field_derivative_series'*[*of ball z d _ _ z*])
fix k :: *nat* **and** t :: $'a$ **assume** t: $t \in ball\ z\ d$
with *d* **have** t': $t \notin \mathbb{Z}_{\leq 0}$ $t \neq 0$ **by** *auto*
show (($\lambda a.$ *inverse* $((a + of\_nat\ k) \mathbin{\hat{}} n')$) *has_field_derivative*

$- \mathit{of\_nat}\ n' * \mathit{inverse}\ ((t + \mathit{of\_nat}\ k)\ \hat{}\ (n'+1)))\ (\mathit{at}\ t\ \mathit{within}\ \mathit{ball}\ z\ d)$ **using** t'
by (*fastforce intro*!: *derivative_eq_intros simp*: *divide_simps power_diff dest*: *plus_of_nat_eq_0_imp*)
next
have *uniformly_convergent_on* (*ball z d*)
$(\lambda k\ z.\ (-\ \mathit{of\_nat}\ n' :: 'a) * (\sum i{<}k.\ \mathit{inverse}\ ((z + \mathit{of\_nat}\ i)\ \hat{}\ (n'+1))))$
using z' n **by** (*intro uniformly_convergent_mult Polygamma_converges*) (*simp_all add*: *n'_def*)
thus *uniformly_convergent_on* (*ball z d*)
$(\lambda k\ z.\ \sum i{<}k.\ -\ \mathit{of\_nat}\ n' * \mathit{inverse}\ ((z + \mathit{of\_nat}\ i :: 'a)\ \hat{}\ (n'+1)))$
by (*subst* (*asm*) *sum_distrib_left*) *simp*
qed (*insert Polygamma_converges'*[*OF z' n'*] *d*, *simp_all*)
also have $(\sum k.\ -\ \mathit{of\_nat}\ n' * \mathit{inverse}\ ((z + \mathit{of\_nat}\ k)\ \hat{}\ (n' + 1))) =$
$(-\ \mathit{of\_nat}\ n') * (\sum k.\ \mathit{inverse}\ ((z + \mathit{of\_nat}\ k)\ \hat{}\ (n' + 1)))$
using *Polygamma_converges'*[*OF z'*, *of n'+1*] n' **by** (*subst suminf_mult*) *simp_all*
finally have $((\lambda z.\ \sum k.\ \mathit{inverse}\ ((z + \mathit{of\_nat}\ k)\ \hat{}\ n'))\ \mathit{has\_field\_derivative}$
$-\ \mathit{of\_nat}\ n' * (\sum k.\ \mathit{inverse}\ ((z + \mathit{of\_nat}\ k)\ \hat{}\ (n' + 1))))\ (\mathit{at}\ z)$ **.**
from *DERIV_cmult*[*OF this*, *of* $(-1)\hat{}$*Suc n * fact n* :: *'a*]
show (*Polygamma n has_field_derivative Polygamma* (*Suc n*) *z*) (*at z*)
unfolding *n'_def Polygamma_def*[*abs_def*] **using** n **by** (*simp add*: *algebra_simps*)
qed

declare *has_field_derivative_Polygamma*[*THEN DERIV_chain2*, *derivative_intros*]

lemma *isCont_Polygamma* [*continuous_intros*]:
fixes $f :: \_ \Rightarrow 'a :: \{\mathit{real\_normed\_field}, \mathit{euclidean\_space}\}$
shows $\mathit{isCont}\ f\ z \Longrightarrow f\ z \notin \mathbb{Z}_{\leq 0} \Longrightarrow \mathit{isCont}\ (\lambda x.\ \mathit{Polygamma}\ n\ (f\ x))\ z$
by (*rule isCont_o2*[*OF _ DERIV_isCont*[*OF has_field_derivative_Polygamma*]])

lemma *continuous_on_Polygamma*:
$A \cap \mathbb{Z}_{\leq 0} = \{\} \Longrightarrow \mathit{continuous\_on}\ A\ (\mathit{Polygamma}\ n :: \_ \Rightarrow 'a :: \{\mathit{real\_normed\_field}, \mathit{euclidean\_space}\})$
by (*intro continuous_at_imp_continuous_on isCont_Polygamma*[*OF continuous_ident*] *ballI*) *blast*

lemma *isCont_ln_Gamma_complex* [*continuous_intros*]:
fixes $f :: 'a{::}\mathit{t2\_space} \Rightarrow \mathit{complex}$
shows $\mathit{isCont}\ f\ z \Longrightarrow f\ z \notin \mathbb{R}_{\leq 0} \Longrightarrow \mathit{isCont}\ (\lambda z.\ \mathit{ln\_Gamma}\ (f\ z))\ z$
by (*rule isCont_o2*[*OF _ DERIV_isCont*[*OF has_field_derivative_ln_Gamma_complex*]])

lemma *continuous_on_ln_Gamma_complex* [*continuous_intros*]:
fixes $A :: \mathit{complex\ set}$
shows $A \cap \mathbb{R}_{\leq 0} = \{\} \Longrightarrow \mathit{continuous\_on}\ A\ \mathit{ln\_Gamma}$
by (*intro continuous_at_imp_continuous_on ballI isCont_ln_Gamma_complex*[*OF continuous_ident*])
fastforce

lemma *deriv_Polygamma*:
assumes $z \notin \mathbb{Z}_{\leq 0}$
shows *deriv* (*Polygamma m*) *z* =
Polygamma (*Suc m*) (*z* :: *'a* :: {*real_normed_field*,*euclidean_space*})
by (*intro DERIV_imp_deriv has_field_derivative_Polygamma assms*)
thm *has_field_derivative_Polygamma*

lemma *higher_deriv_Polygamma*:
assumes $z \notin \mathbb{Z}_{\leq 0}$
shows (*deriv* ^^ *n*) (*Polygamma m*) *z* =
Polygamma (*m* + *n*) (*z* :: *'a* :: {*real_normed_field*,*euclidean_space*})
proof −
have *eventually* (λ*u*. (*deriv* ^^ *n*) (*Polygamma m*) *u* = *Polygamma* (*m* + *n*) *u*) (*nhds z*)
proof (*induction n*)
case (*Suc n*)
from *Suc.IH* **have** *eventually* (λ*z*. *eventually* (λ*u*. (*deriv* ^^ *n*) (*Polygamma m*) *u* = *Polygamma* (*m* + *n*) *u*) (*nhds z*)) (*nhds z*)
by (*simp add*: *eventually_eventually*)
hence *eventually* (λ*z*. *deriv* ((*deriv* ^^ *n*) (*Polygamma m*)) *z* =
deriv (*Polygamma* (*m* + *n*)) *z*) (*nhds z*)
by *eventually_elim* (*intro deriv_cong_ev refl*)
moreover have *eventually* (λ*z*. *z* ∈ *UNIV* − $\mathbb{Z}_{\leq 0}$) (*nhds z*) **using** *assms*
by (*intro eventually_nhds_in_open open_Diff open_UNIV*) *auto*
ultimately show *?case* **by** *eventually_elim* (*simp_all add*: *deriv_Polygamma*)
qed *simp_all*
thus *?thesis* **by** (*rule eventually_nhds_x_imp_x*)
qed

lemma *deriv_ln_Gamma_complex*:
assumes $z \notin \mathbb{R}_{\leq 0}$
shows *deriv ln_Gamma z* = *Digamma* (*z* :: *complex*)
by (*intro DERIV_imp_deriv has_field_derivative_ln_Gamma_complex assms*)

lemma *higher_deriv_ln_Gamma_complex*:
assumes (*x*::*complex*) $\notin \mathbb{R}_{\leq 0}$
shows (*deriv* ^^ *j*) *ln_Gamma x* = (*if j* = *0 then ln_Gamma x else Polygamma* (*j* − *1*) *x*)
proof (*cases j*)
case (*Suc j'*)
have (*deriv* ^^ *j'*) (*deriv ln_Gamma*) *x* = (*deriv* ^^ *j'*) *Digamma x*
using *eventually_nhds_in_open*[*of UNIV* − $\mathbb{R}_{\leq 0}$ *x*] *assms*
by (*intro higher_deriv_cong_ev refl*)
(*auto elim*!: *eventually_mono simp*: *open_Diff deriv_ln_Gamma_complex*)
also have ... = *Polygamma j' x* **using** *assms*
by (*subst higher_deriv_Polygamma*)
(*auto elim*!: *nonpos_Ints_cases simp*: *complex_nonpos_Reals_iff*)
finally show *?thesis* **using** *Suc* **by** (*simp del*: *funpow.simps add*: *funpow_Suc_right*)

qed *simp_all*

We define a type class that captures all the fundamental properties of the inverse of the Gamma function and defines the Gamma function upon that. This allows us to instantiate the type class both for the reals and for the complex numbers with a minimal amount of proof duplication.

class *Gamma* = *real_normed_field* + *complete_space* +
fixes *rGamma* :: *'a* ⇒ *'a*
assumes *rGamma_eq_zero_iff_aux*: *rGamma z* = *0* ⟷ (∃ *n*. *z* = − *of_nat n*)
assumes *differentiable_rGamma_aux1*:
(⋀*n*. *z* ≠ − *of_nat n*) ⟹
let d = (*THE d*. (λ*n*. ∑ *k*<*n*. *inverse* (*of_nat* (*Suc k*)) − *inverse* (*z* + *of_nat k*)) ⟶ *d*) − *scaleR euler_mascheroni 1*
in filterlim (λ*y*. (*rGamma y* − *rGamma z* + *rGamma z* ∗ *d* ∗ (*y* − *z*)) /$_R$ *norm* (*y* − *z*)) (*nhds 0*) (*at z*)
assumes *differentiable_rGamma_aux2*:
let z = − *of_nat n*
in filterlim (λ*y*. (*rGamma y* − *rGamma z* − (−*1*)^*n* ∗ (*prod of_nat* {*1*..*n*}) ∗ (*y* − *z*)) /$_R$ *norm* (*y* − *z*)) (*nhds 0*) (*at z*)
assumes *rGamma_series_aux*: (⋀*n*. *z* ≠ − *of_nat n*) ⟹
let fact' = (λ*n*. *prod of_nat* {*1*..*n*});
exp = (λ*x*. *THE e*. (λ*n*. ∑ *k*<*n*. *x*^*k* /$_R$ *fact k*) ⟶ *e*);
pochhammer' = (λ*a n*. (∏ *n* = *0*..*n*. *a* + *of_nat n*))
in filterlim (λ*n*. *pochhammer' z n* / (*fact' n* ∗ *exp* (*z* ∗ (*ln* (*of_nat n*) ∗$_R$ *1*)))) (*nhds* (*rGamma z*)) *sequentially*
begin
subclass *banach* **..**
end

definition *Gamma z* = *inverse* (*rGamma z*)

6.20.3 Basic properties

lemma *Gamma_nonpos_Int*: *z* ∈ ℤ$_{\leq 0}$ ⟹ *Gamma z* = *0*
and *rGamma_nonpos_Int*: *z* ∈ ℤ$_{\leq 0}$ ⟹ *rGamma z* = *0*
using *rGamma_eq_zero_iff_aux*[*of z*] **unfolding** *Gamma_def* **by** (*auto elim*!: *nonpos_Ints_cases'*)

lemma *Gamma_nonzero*: *z* ∉ ℤ$_{\leq 0}$ ⟹ *Gamma z* ≠ *0*
and *rGamma_nonzero*: *z* ∉ ℤ$_{\leq 0}$ ⟹ *rGamma z* ≠ *0*
using *rGamma_eq_zero_iff_aux*[*of z*] **unfolding** *Gamma_def* **by** (*auto elim*!: *nonpos_Ints_cases'*)

lemma *Gamma_eq_zero_iff*: *Gamma z* = *0* ⟷ *z* ∈ ℤ$_{\leq 0}$
and *rGamma_eq_zero_iff*: *rGamma z* = *0* ⟷ *z* ∈ ℤ$_{\leq 0}$

using *rGamma_eq_zero_iff_aux*[*of z*] **unfolding** *Gamma_def* **by** (*auto elim*!: *nonpos_Ints_cases'*)

lemma *rGamma_inverse_Gamma*: *rGamma z* = *inverse* (*Gamma z*)
unfolding *Gamma_def* **by** *simp*

lemma *rGamma_series_LIMSEQ* [*tendsto_intros*]:
rGamma_series z ⟶ *rGamma z*
proof (*cases z* ∈ ℤ$_{\leq 0}$)
case *False*
hence *z* ≠ − *of_nat n* **for** *n* **by** *auto*
from *rGamma_series_aux*[*OF this*] **show** *?thesis*
by (*simp add*: *rGamma_series_def*[*abs_def*] *fact_prod pochhammer_Suc_prod exp_def of_real_def*[*symmetric*] *suminf_def sums_def*[*abs_def*] *atLeast0AtMost*)
qed (*insert rGamma_eq_zero_iff*[*of z*], *simp_all add*: *rGamma_series_nonpos_Ints_LIMSEQ*)

theorem *Gamma_series_LIMSEQ* [*tendsto_intros*]:
Gamma_series z ⟶ *Gamma z*
proof (*cases z* ∈ ℤ$_{\leq 0}$)
case *False*
hence (λ*n*. *inverse* (*rGamma_series z n*)) ⟶ *inverse* (*rGamma z*)
by (*intro tendsto_intros*) (*simp_all add*: *rGamma_eq_zero_iff*)
also have (λ*n*. *inverse* (*rGamma_series z n*)) = *Gamma_series z*
by (*simp add*: *rGamma_series_def Gamma_series_def*[*abs_def*])
finally show *?thesis* **by** (*simp add*: *Gamma_def*)
qed (*insert Gamma_eq_zero_iff*[*of z*], *simp_all add*: *Gamma_series_nonpos_Ints_LIMSEQ*)

lemma *Gamma_altdef*: *Gamma z* = *lim* (*Gamma_series z*)
using *Gamma_series_LIMSEQ*[*of z*] **by** (*simp add*: *limI*)

lemma *rGamma_1* [*simp*]: *rGamma 1* = *1*
proof −
have *A*: *eventually* (λ*n*. *rGamma_series 1 n* = *of_nat* (*Suc n*) / *of_nat n*) *sequentially*
using *eventually_gt_at_top*[*of 0*::*nat*]
by (*force elim*!: *eventually_mono simp*: *rGamma_series_def exp_of_real pochhammer_fact field_split_simps pochhammer_rec' dest*!: *pochhammer_eq_0_imp_nonpos_Int*)
have *rGamma_series 1* ⟶ *1* **by** (*subst tendsto_cong*[*OF A*]) (*rule LIMSEQ_Suc_n_over_n*)
moreover have *rGamma_series 1* ⟶ *rGamma 1* **by** (*rule tendsto_intros*)
ultimately show *?thesis* **by** (*intro LIMSEQ_unique*)
qed

lemma *rGamma_plus1*: *z* ∗ *rGamma* (*z* + *1*) = *rGamma z*
proof −
let *?f* = λ*n*. (*z* + *1*) ∗ *inverse* (*of_nat n*) + *1*
have *eventually* (λ*n*. *?f n* ∗ *rGamma_series z n* = *z* ∗ *rGamma_series* (*z* + *1*)

```
n) sequentially
    using eventually_gt_at_top[of 0::nat]
  proof eventually_elim
    fix n :: nat assume n: n > 0
    hence z * rGamma_series (z + 1) n = inverse (of_nat n) *
            pochhammer z (Suc (Suc n)) / (fact n * exp (z * of_real (ln (of_nat
n))))
        by (subst pochhammer_rec) (simp add: rGamma_series_def field_simps
exp_add exp_of_real)
    also from n have ... = ?f n * rGamma_series z n
    by (subst pochhammer_rec') (simp_all add: field_split_simps rGamma_series_def)
    finally show ?f n * rGamma_series z n = z * rGamma_series (z + 1) n ..
  qed
  moreover have (λn. ?f n * rGamma_series z n) ⟶ ((z+1) * 0 + 1) *
rGamma z
    by (intro tendsto_intros lim_inverse_n)
  hence (λn. ?f n * rGamma_series z n) ⟶ rGamma z by simp
  ultimately have (λn. z * rGamma_series (z + 1) n) ⟶ rGamma z
    by (blast intro: Lim_transform_eventually)
  moreover have (λn. z * rGamma_series (z + 1) n) ⟶ z * rGamma (z +
1)
    by (intro tendsto_intros)
  ultimately show z * rGamma (z + 1) = rGamma z using LIMSEQ_unique
by blast
qed


lemma pochhammer_rGamma: rGamma z = pochhammer z n * rGamma (z +
of_nat n)
proof (induction n arbitrary: z)
  case (Suc n z)
  have rGamma z = pochhammer z n * rGamma (z + of_nat n) by (rule Suc.IH)
  also note rGamma_plus1 [symmetric]
  finally show ?case by (simp add: add_ac pochhammer_rec')
qed simp_all

theorem Gamma_plus1: z ∉ ℤ≤0 ⟹ Gamma (z + 1) = z * Gamma z
  using rGamma_plus1[of z] by (simp add: rGamma_inverse_Gamma field_simps
Gamma_eq_zero_iff)

theorem pochhammer_Gamma: z ∉ ℤ≤0 ⟹ pochhammer z n = Gamma (z +
of_nat n) / Gamma z
  using pochhammer_rGamma[of z]
  by (simp add: rGamma_inverse_Gamma Gamma_eq_zero_iff field_simps)

lemma Gamma_0 [simp]: Gamma 0 = 0
  and rGamma_0 [simp]: rGamma 0 = 0
  and Gamma_neg_1 [simp]: Gamma (− 1) = 0
  and rGamma_neg_1 [simp]: rGamma (− 1) = 0
```

and *Gamma_neg_numeral* [*simp*]: *Gamma* (− *numeral n*) = *0*
and *rGamma_neg_numeral* [*simp*]: *rGamma* (− *numeral n*) = *0*
and *Gamma_neg_of_nat* [*simp*]: *Gamma* (− *of_nat m*) = *0*
and *rGamma_neg_of_nat* [*simp*]: *rGamma* (− *of_nat m*) = *0*
by (*simp_all add*: *rGamma_eq_zero_iff Gamma_eq_zero_iff*)

lemma *Gamma_1* [*simp*]: *Gamma 1* = *1* **unfolding** *Gamma_def* **by** *simp*

theorem *Gamma_fact*: *Gamma* (*1* + *of_nat n*) = *fact n*
by (*simp add*: *pochhammer_fact pochhammer_Gamma of_nat_in_nonpos_Ints_iff flip*: *of_nat_Suc*)

lemma *Gamma_numeral*: *Gamma* (*numeral n*) = *fact* (*pred_numeral n*)
by (*subst of_nat_numeral*[*symmetric*], *subst numeral_eq_Suc*,
subst of_nat_Suc, *subst Gamma_fact*) (*rule refl*)

lemma *Gamma_of_int*: *Gamma* (*of_int n*) = (*if n* > *0 then fact* (*nat* (*n* − *1*)) *else 0*)
proof (*cases n* > *0*)
case *True*
hence *Gamma* (*of_int n*) = *Gamma* (*of_nat* (*Suc* (*nat* (*n* − *1*)))) **by** (*subst of_nat_Suc*) *simp_all*
with *True* **show** *?thesis* **by** (*subst* (*asm*) *of_nat_Suc*, *subst* (*asm*) *Gamma_fact*) *simp*
qed (*simp_all add*: *Gamma_eq_zero_iff nonpos_Ints_of_int*)

lemma *rGamma_of_int*: *rGamma* (*of_int n*) = (*if n* > *0 then inverse* (*fact* (*nat* (*n* − *1*))) *else 0*)
by (*simp add*: *Gamma_of_int rGamma_inverse_Gamma*)

lemma *Gamma_seriesI*:
assumes (λ*n*. *g n* / *Gamma_series z n*) ⟶ *1*
shows *g* ⟶ *Gamma z*
proof (*rule Lim_transform_eventually*)
have *1*/*2* > (*0*::*real*) **by** *simp*
from *tendstoD*[*OF assms*, *OF this*]
show *eventually* (λ*n*. *g n* / *Gamma_series z n* ∗ *Gamma_series z n* = *g n*) *sequentially*
by (*force elim*!: *eventually_mono simp*: *dist_real_def*)
from *assms* **have** (λ*n*. *g n* / *Gamma_series z n* ∗ *Gamma_series z n*) ⟶ *1* ∗ *Gamma z*
by (*intro tendsto_intros*)
thus (λ*n*. *g n* / *Gamma_series z n* ∗ *Gamma_series z n*) ⟶ *Gamma z* **by** *simp*
qed

lemma *Gamma_seriesI′*:
assumes *f* ⟶ *rGamma z*
assumes (λ*n*. *g n* ∗ *f n*) ⟶ *1*

assumes $z \notin \mathbb{Z}_{\leq 0}$
shows $g \longrightarrow Gamma\ z$
proof (*rule Lim_transform_eventually*)
have $1/2 > (0::real)$ **by** *simp*
from *tendstoD*[*OF assms*(2), *OF this*] **show** *eventually* ($\lambda n.\ g\ n * f\ n\ /\ f\ n = g\ n$) *sequentially*
by (*force elim*!: *eventually_mono simp*: *dist_real_def*)
from *assms* **have** $(\lambda n.\ g\ n * f\ n\ /\ f\ n) \longrightarrow 1\ /\ rGamma\ z$
by (*intro tendsto_divide assms*) (*simp_all add*: *rGamma_eq_zero_iff*)
thus $(\lambda n.\ g\ n * f\ n\ /\ f\ n) \longrightarrow Gamma\ z$ **by** (*simp add*: *Gamma_def divide_inverse*)
qed

lemma *Gamma_series'_LIMSEQ*: $Gamma\_series'\ z \longrightarrow Gamma\ z$
by (*cases* $z \in \mathbb{Z}_{\leq 0}$) (*simp_all add*: *Gamma_nonpos_Int Gamma_seriesI*[*OF Gamma_series_Gamma_series'*]
Gamma_series'_nonpos_Ints_LIMSEQ[*of z*])

6.20.4 Differentiability

lemma *has_field_derivative_rGamma_no_nonpos_int*:
assumes $z \notin \mathbb{Z}_{\leq 0}$
shows (*rGamma has_field_derivative* $-rGamma\ z * Digamma\ z$) (*at z within A*)
proof (*rule has_field_derivative_at_within*)
from *assms* **have** $z \neq -\ of\_nat\ n$ **for** *n* **by** *auto*
from *differentiable_rGamma_aux1*[*OF this*]
show (*rGamma has_field_derivative* $-rGamma\ z * Digamma\ z$) (*at z*)
unfolding *Digamma_def suminf_def sums_def*[*abs_def*]
has_field_derivative_def has_derivative_def netlimit_at
by (*simp add*: *Let_def bounded_linear_mult_right mult_ac of_real_def* [*symmetric*])
qed

lemma *has_field_derivative_rGamma_nonpos_int*:
(*rGamma has_field_derivative* $(-1)\hat{}n * fact\ n$) (*at* $(-\ of\_nat\ n)$ *within A*)
apply (*rule has_field_derivative_at_within*)
using *differentiable_rGamma_aux2*[*of n*]
unfolding *Let_def has_field_derivative_def has_derivative_def netlimit_at*
by (*simp only*: *bounded_linear_mult_right mult_ac of_real_def* [*symmetric*] *fact_prod*) *simp*

lemma *has_field_derivative_rGamma* [*derivative_intros*]:
(*rGamma has_field_derivative* (*if* $z \in \mathbb{Z}_{\leq 0}$ *then* $(-1)\hat{}(nat\ \lfloor norm\ z \rfloor) * fact\ (nat\ \lfloor norm\ z \rfloor)$
else $-rGamma\ z * Digamma\ z$)) (*at z within A*)
using *has_field_derivative_rGamma_no_nonpos_int*[*of z A*]
has_field_derivative_rGamma_nonpos_int[*of nat* $\lfloor norm\ z \rfloor$ *A*]
by (*auto elim*!: *nonpos_Ints_cases'*)

declare *has_field_derivative_rGamma_no_nonpos_int* [*THEN DERIV_chain2*, *derivative_intros*]
declare *has_field_derivative_rGamma* [*THEN DERIV_chain2*, *derivative_intros*]
declare *has_field_derivative_rGamma_nonpos_int* [*derivative_intros*]
declare *has_field_derivative_rGamma_no_nonpos_int* [*derivative_intros*]
declare *has_field_derivative_rGamma* [*derivative_intros*]

theorem *has_field_derivative_Gamma* [*derivative_intros*]:
  $z \notin \mathbb{Z}_{\leq 0} \Longrightarrow$ (*Gamma has_field_derivative Gamma z* $*$ *Digamma z*) (*at z within A*)
  unfolding *Gamma_def* [*abs_def*]
  by (*fastforce intro*!: *derivative_eq_intros simp*: *rGamma_eq_zero_iff*)

declare *has_field_derivative_Gamma*[*THEN DERIV_chain2*, *derivative_intros*]

hide_fact *rGamma_eq_zero_iff_aux differentiable_rGamma_aux1 differentiable_rGamma_aux2 differentiable_rGamma_aux2 rGamma_series_aux Gamma_class.rGamma_eq_zero_iff_aux*

lemma *continuous_on_rGamma* [*continuous_intros*]: *continuous_on A rGamma*
  by (*rule DERIV_continuous_on has_field_derivative_rGamma*)+

lemma *continuous_on_Gamma* [*continuous_intros*]: $A \cap \mathbb{Z}_{\leq 0} = \{\} \Longrightarrow$ *continuous_on A Gamma*
  by (*rule DERIV_continuous_on has_field_derivative_Gamma*)+ *blast*

lemma *isCont_rGamma* [*continuous_intros*]:
  isCont f z $\Longrightarrow$ *isCont* (λx. *rGamma* (*f x*)) *z*
  by (*rule isCont_o2*[*OF _ DERIV_isCont*[*OF has_field_derivative_rGamma*]])

lemma *isCont_Gamma* [*continuous_intros*]:
  isCont f z $\Longrightarrow$ *f z* $\notin \mathbb{Z}_{\leq 0} \Longrightarrow$ *isCont* (λx. *Gamma* (*f x*)) *z*
  by (*rule isCont_o2*[*OF _ DERIV_isCont*[*OF has_field_derivative_Gamma*]])

6.20.5 The complex Gamma function

instantiation *complex* :: *Gamma*
begin

definition *rGamma_complex* :: *complex* $\Rightarrow$ *complex* **where**
  rGamma_complex z = *lim* (*rGamma_series z*)

lemma *rGamma_series_complex_converges*:
    convergent (*rGamma_series* (*z* :: *complex*)) (**is** *?thesis1*)
  and *rGamma_complex_altdef*:
    rGamma z = (*if z* $\in \mathbb{Z}_{\leq 0}$ *then 0 else exp* (−*ln_Gamma z*)) (**is** *?thesis2*)
proof −
  have *?thesis1* $\wedge$ *?thesis2*
  proof (*cases z* $\in \mathbb{Z}_{\leq 0}$)

```
    case False
    have rGamma_series z ⟶ exp (− ln_Gamma z)
    proof (rule Lim_transform_eventually)
      from ln_Gamma_series_complex_converges'[OF False]
    obtain d where 0 < d uniformly_convergent_on (ball z d) (λn z. ln_Gamma_series
z n)
        by auto
      from this(1) uniformly_convergent_imp_convergent[OF this(2), of z]
       have ln_Gamma_series z ⟶ lim (ln_Gamma_series z) by (simp add:
convergent_LIMSEQ_iff)
      thus (λn. exp (−ln_Gamma_series z n)) ⟶ exp (− ln_Gamma z)
          unfolding convergent_def ln_Gamma_def by (intro tendsto_exp tend-
sto_minus)
     from eventually_gt_at_top[of 0::nat] exp_ln_Gamma_series_complex False
        show eventually (λn. exp (−ln_Gamma_series z n) = rGamma_series z
n) sequentially
          by (force elim!: eventually_mono simp: exp_minus Gamma_series_def
rGamma_series_def)
    qed
    with False show ?thesis
      by (auto simp: convergent_def rGamma_complex_def intro!: limI)
  next
    case True
    then obtain k where z = − of_nat k by (erule nonpos_Ints_cases')
    also have rGamma_series ... ⟶ 0
     by (subst tendsto_cong[OF rGamma_series_minus_of_nat]) (simp_all add:
convergent_const)
    finally show ?thesis using True
      by (auto simp: rGamma_complex_def convergent_def intro!: limI)
  qed
  thus ?thesis1 ?thesis2 by blast+
qed

context
begin


private lemma rGamma_complex_plus1: z * rGamma (z + 1) = rGamma (z ::
complex)
proof −
  let ?f = λn. (z + 1) * inverse (of_nat n) + 1
  have eventually (λn. ?f n * rGamma_series z n = z * rGamma_series (z + 1)
n) sequentially
    using eventually_gt_at_top[of 0::nat]
  proof eventually_elim
    fix n :: nat assume n: n > 0
    hence z * rGamma_series (z + 1) n = inverse (of_nat n) *
            pochhammer z (Suc (Suc n)) / (fact n * exp (z * of_real (ln (of_nat
n))))
```

```
      by (subst pochhammer_rec) (simp add: rGamma_series_def field_simps
exp_add exp_of_real)
    also from n have ... = ?f n * rGamma_series z n
    by (subst pochhammer_rec') (simp_all add: field_split_simps rGamma_series_def
add_ac)
    finally show ?f n * rGamma_series z n = z * rGamma_series (z + 1) n ..
  qed
  moreover have (λn. ?f n * rGamma_series z n) ⟶ ((z+1) * 0 + 1) *
rGamma z
    using rGamma_series_complex_converges
    by (intro tendsto_intros lim_inverse_n)
       (simp_all add: convergent_LIMSEQ_iff rGamma_complex_def)
  hence (λn. ?f n * rGamma_series z n) ⟶ rGamma z by simp
  ultimately have (λn. z * rGamma_series (z + 1) n) ⟶ rGamma z
    by (blast intro: Lim_transform_eventually)
  moreover have (λn. z * rGamma_series (z + 1) n) ⟶ z * rGamma (z +
1)
    using rGamma_series_complex_converges
   by (auto intro!: tendsto_mult simp: rGamma_complex_def convergent_LIMSEQ_iff)
  ultimately show z * rGamma (z + 1) = rGamma z using LIMSEQ_unique
by blast
qed

private lemma has_field_derivative_rGamma_complex_no_nonpos_Int:
  assumes (z :: complex) ∉ ℤ≤0
  shows   (rGamma has_field_derivative - rGamma z * Digamma z) (at z)
proof -
  have diff: (rGamma has_field_derivative - rGamma z * Digamma z) (at z) if
Re z > 0 for z
  proof (subst DERIV_cong_ev[OF refl _ refl])
    from that have eventually (λt. t ∈ ball z (Re z/2)) (nhds z)
      by (intro eventually_nhds_in_nhd) simp_all
    thus eventually (λt. rGamma t = exp (- ln_Gamma t)) (nhds z)
      using no_nonpos_Int_in_ball_complex[OF that]
      by (auto elim!: eventually_mono simp: rGamma_complex_altdef)
  next
    have z ∉ ℝ≤0 using that by (simp add: complex_nonpos_Reals_iff)
    with that show ((λt. exp (- ln_Gamma t)) has_field_derivative (-rGamma
z * Digamma z)) (at z)
   by (force elim!: nonpos_Ints_cases intro!: derivative_eq_intros simp: rGamma_complex_altdef)
  qed

  from assms show (rGamma has_field_derivative - rGamma z * Digamma z)
(at z)
  proof (induction nat ⌊1 - Re z⌋ arbitrary: z)
    case (Suc n z)
    from Suc.prems have z: z ≠ 0 by auto
    from Suc.hyps have n = nat ⌊- Re z⌋ by linarith
    hence A: n = nat ⌊1 - Re (z + 1)⌋ by simp
```

```
  from Suc.prems have B: z + 1 ∉ ℤ≤0 by (force dest: plus_one_in_nonpos_Ints_imp)

   have ((λz. z * (rGamma ∘ (λz. z + 1)) z) has_field_derivative
     −rGamma (z + 1) * (Digamma (z + 1) * z − 1)) (at z)
      by (rule derivative_eq_intros DERIV_chain Suc refl A B)+ (simp add:
algebra_simps)
   also have (λz. z * (rGamma ∘ (λz. z + 1 :: complex)) z) = rGamma
     by (simp add: rGamma_complex_plus1)
   also from z have Digamma (z + 1) * z − 1 = z * Digamma z
     by (subst Digamma_plus1) (simp_all add: field_simps)
   also have −rGamma (z + 1) * (z * Digamma z) = −rGamma z * Digamma z
     by (simp add: rGamma_complex_plus1[of z, symmetric])
   finally show ?case .
 qed (intro diff, simp)
qed

private lemma rGamma_complex_1: rGamma (1 :: complex) = 1
proof −
  have A: eventually (λn. rGamma_series 1 n = of_nat (Suc n) / of_nat n)
sequentially
   using eventually_gt_at_top[of 0::nat]
  by (force elim!: eventually_mono simp: rGamma_series_def exp_of_real pochham-
mer_fact
           field_split_simps pochhammer_rec′ dest!: pochhammer_eq_0_imp_nonpos_Int)
  have rGamma_series 1 ⟶ 1 by (subst tendsto_cong[OF A]) (rule LIM-
SEQ_Suc_n_over_n)
  thus rGamma 1 = (1 :: complex) unfolding rGamma_complex_def by (rule
limI)
qed

private lemma has_field_derivative_rGamma_complex_nonpos_Int:
  (rGamma has_field_derivative (−1)^n * fact n) (at (− of_nat n :: complex))
proof (induction n)
  case 0
  have A: (0::complex) + 1 ∉ ℤ≤0 by simp
  have ((λz. z * (rGamma ∘ (λz. z + 1 :: complex)) z) has_field_derivative 1)
(at 0)
   by (rule derivative_eq_intros DERIV_chain refl
         has_field_derivative_rGamma_complex_no_nonpos_Int A)+ (simp
add: rGamma_complex_1)
   thus ?case by (simp add: rGamma_complex_plus1)
next
  case (Suc n)
  hence A: (rGamma has_field_derivative (−1)^n * fact n)
          (at (− of_nat (Suc n) + 1 :: complex)) by simp
  have ((λz. z * (rGamma ∘ (λz. z + 1 :: complex)) z) has_field_derivative
        (− 1) ^ Suc n * fact (Suc n)) (at (− of_nat (Suc n)))
    by (rule derivative_eq_intros refl A DERIV_chain)+
      (simp add: algebra_simps rGamma_complex_altdef)
```

thus *?case* **by** (*simp add*: *rGamma_complex_plus1*)
qed

instance proof
fix *z* :: *complex* **show** (*rGamma z* = *0*) ⟷ (∃ *n*. *z* = − *of_nat n*)
by (*auto simp*: *rGamma_complex_altdef elim*!: *nonpos_Ints_cases*′)
next
fix *z* :: *complex* **assume** ⋀*n*. *z* ≠ − *of_nat n*
hence *z* ∉ ℤ≤0 **by** (*auto elim*!: *nonpos_Ints_cases*′)
from *has_field_derivative_rGamma_complex_no_nonpos_Int*[*OF this*]
show *let d* = (*THE d*. (λ*n*. ∑ *k*<*n*. *inverse* (*of_nat* (*Suc k*)) − *inverse* (*z* + *of_nat k*)) ⟶ *d*) − *euler_mascheroni* ∗*R* *1* *in* (λ*y*. (*rGamma y* − *rGamma z* + *rGamma z* ∗ *d* ∗ (*y* − *z*)) /*R* *cmod* (*y* − *z*)) −*z*→ *0*
by (*simp add*: *has_field_derivative_def has_derivative_def Digamma_def sums_def* [*abs_def*] *of_real_def*[*symmetric*] *suminf_def*)
next
fix *n* :: *nat*
from *has_field_derivative_rGamma_complex_nonpos_Int*[*of n*]
show *let z* = − *of_nat n in* (λ*y*. (*rGamma y* − *rGamma z* − (− *1*) ^ *n* ∗ *prod of_nat* {*1*..*n*} ∗ (*y* − *z*)) /*R* *cmod* (*y* − *z*)) −*z*→ *0*
by (*simp add*: *has_field_derivative_def has_derivative_def fact_prod Let_def*)
next
fix *z* :: *complex*
from *rGamma_series_complex_converges*[*of z*] **have** *rGamma_series z* ⟶ *rGamma z*
by (*simp add*: *convergent_LIMSEQ_iff rGamma_complex_def*)
thus *let fact*′ = λ*n*. *prod of_nat* {*1*..*n*}; *exp* = λ*x*. *THE e*. (λ*n*. ∑ *k*<*n*. *x* ^ *k* /*R* *fact k*) ⟶ *e*; *pochhammer*′ = λ*a n*. ∏ *n* = *0*..*n*. *a* + *of_nat n* *in* (λ*n*. *pochhammer*′ *z n* / (*fact*′ *n* ∗ *exp* (*z* ∗ *ln* (*real_of_nat n*) ∗*R* *1*))) ⟶ *rGamma z*
by (*simp add*: *fact_prod pochhammer_Suc_prod rGamma_series_def* [*abs_def*] *exp_def* *of_real_def* [*symmetric*] *suminf_def sums_def* [*abs_def*] *atLeast0AtMost*)
qed

end
end

lemma *Gamma_complex_altdef*:
Gamma z = (*if z* ∈ ℤ≤0 *then 0 else exp* (*ln_Gamma* (*z* :: *complex*)))
unfolding *Gamma_def rGamma_complex_altdef* **by** (*simp add*: *exp_minus*)

lemma *cnj_rGamma*: *cnj* (*rGamma z*) = *rGamma* (*cnj z*)
proof −
have *rGamma_series* (*cnj z*) = (λ*n*. *cnj* (*rGamma_series z n*))
by (*intro ext*) (*simp_all add*: *rGamma_series_def exp_cnj*)
also have ... ⟶ *cnj* (*rGamma z*) **by** (*intro tendsto_cnj tendsto_intros*)
finally show *?thesis* **unfolding** *rGamma_complex_def* **by** (*intro sym*[*OF limI*])
qed

lemma *cnj_Gamma*: *cnj* (*Gamma z*) = *Gamma* (*cnj z*)
unfolding *Gamma_def* **by** (*simp add*: *cnj_rGamma*)

lemma *Gamma_complex_real*:
z ∈ ℝ ⟹ *Gamma z* ∈ (ℝ :: *complex set*) **and** *rGamma_complex_real*: *z* ∈ ℝ
⟹ *rGamma z* ∈ ℝ
by (*simp_all add*: *Reals_cnj_iff cnj_Gamma cnj_rGamma*)

lemma *field_differentiable_rGamma*: *rGamma field_differentiable* (*at z within A*)
using *has_field_derivative_rGamma*[*of z*] **unfolding** *field_differentiable_def*
by *blast*

lemma *holomorphic_rGamma* [*holomorphic_intros*]: *rGamma holomorphic_on A*
unfolding *holomorphic_on_def* **by** (*auto intro*!: *field_differentiable_rGamma*)

lemma *holomorphic_rGamma′* [*holomorphic_intros*]:
assumes *f holomorphic_on A*
shows (λ*x*. *rGamma* (*f x*)) *holomorphic_on A*
proof −
have *rGamma* ∘ *f holomorphic_on A* **using** *assms*
by (*intro holomorphic_on_compose assms holomorphic_rGamma*)
thus *?thesis* **by** (*simp only*: *o_def*)
qed

lemma *analytic_rGamma*: *rGamma analytic_on A*
unfolding *analytic_on_def* **by** (*auto intro*!: *exI*[*of _ 1*] *holomorphic_rGamma*)

lemma *field_differentiable_Gamma*: *z* ∉ ℤ≤0 ⟹ *Gamma field_differentiable* (*at z within A*)
using *has_field_derivative_Gamma*[*of z*] **unfolding** *field_differentiable_def* **by** *auto*

lemma *holomorphic_Gamma* [*holomorphic_intros*]: *A* ∩ ℤ≤0 = {} ⟹ *Gamma holomorphic_on A*
unfolding *holomorphic_on_def* **by** (*auto intro*!: *field_differentiable_Gamma*)

lemma *holomorphic_Gamma′* [*holomorphic_intros*]:
assumes *f holomorphic_on A* **and** ⋀*x*. *x* ∈ *A* ⟹ *f x* ∉ ℤ≤0
shows (λ*x*. *Gamma* (*f x*)) *holomorphic_on A*
proof −

have *Gamma* ∘ *f holomorphic_on A* **using** *assms*
by (*intro holomorphic_on_compose assms holomorphic_Gamma*) *auto*
thus *?thesis* **by** (*simp only*: *o_def*)
qed

lemma *analytic_Gamma*: $A \cap \mathbb{Z}_{\leq 0} = \{\} \Longrightarrow$ *Gamma analytic_on A*
by (*rule analytic_on_subset*[*of* _ *UNIV* − $\mathbb{Z}_{\leq 0}$], *subst analytic_on_open*)
(*auto intro*!: *holomorphic_Gamma*)

lemma *field_differentiable_ln_Gamma_complex*:
$z \notin \mathbb{R}_{\leq 0} \Longrightarrow$ *ln_Gamma field_differentiable* (*at* (*z*::*complex*) *within A*)
by (*rule field_differentiable_within_subset*[*of* _ _ *UNIV*])
(*force simp*: *field_differentiable_def intro*!: *derivative_intros*)+

lemma *holomorphic_ln_Gamma* [*holomorphic_intros*]: $A \cap \mathbb{R}_{\leq 0} = \{\} \Longrightarrow$ *ln_Gamma holomorphic_on A*
unfolding *holomorphic_on_def* **by** (*auto intro*!: *field_differentiable_ln_Gamma_complex*)

lemma *analytic_ln_Gamma*: $A \cap \mathbb{R}_{\leq 0} = \{\} \Longrightarrow$ *ln_Gamma analytic_on A*
by (*rule analytic_on_subset*[*of* _ *UNIV* − $\mathbb{R}_{\leq 0}$], *subst analytic_on_open*)
(*auto intro*!: *holomorphic_ln_Gamma*)

lemma *has_field_derivative_rGamma_complex′* [*derivative_intros*]:
(*rGamma has_field_derivative* (*if* $z \in \mathbb{Z}_{\leq 0}$ *then* $(-1)\hat{}(nat\ \lfloor -Re\ z \rfloor) * fact\ (nat\ \lfloor -Re\ z \rfloor)$ *else*
$-rGamma\ z * Digamma\ z$)) (*at z within A*)
using *has_field_derivative_rGamma*[*of z*] **by** (*auto elim*!: *nonpos_Ints_cases′*)

declare *has_field_derivative_rGamma_complex′*[*THEN DERIV_chain2*, *derivative_intros*]

lemma *field_differentiable_Polygamma*:
fixes *z* :: *complex*
shows
$z \notin \mathbb{Z}_{\leq 0} \Longrightarrow$ *Polygamma n field_differentiable* (*at z within A*)
using *has_field_derivative_Polygamma*[*of z n*] **unfolding** *field_differentiable_def*
by *auto*

lemma *holomorphic_on_Polygamma* [*holomorphic_intros*]: $A \cap \mathbb{Z}_{\leq 0} = \{\} \Longrightarrow$ *Polygamma n holomorphic_on A*
unfolding *holomorphic_on_def* **by** (*auto intro*!: *field_differentiable_Polygamma*)

lemma *analytic_on_Polygamma*: $A \cap \mathbb{Z}_{\leq 0} = \{\} \Longrightarrow$ *Polygamma n analytic_on A*
by (*rule analytic_on_subset*[*of* _ *UNIV* − $\mathbb{Z}_{\leq 0}$], *subst analytic_on_open*)
(*auto intro*!: *holomorphic_on_Polygamma*)

6.20.6 The real Gamma function

lemma *rGamma_series_real*:
eventually (λ*n*. *rGamma_series x n* = *Re* (*rGamma_series* (*of_real x*) *n*)) *sequentially*
using *eventually_gt_at_top*[*of 0* :: *nat*]
proof *eventually_elim*
fix *n* :: *nat* **assume** *n*: *n* > *0*
have *Re* (*rGamma_series* (*of_real x*) *n*) =
Re (*of_real* (*pochhammer x* (*Suc n*)) / (*fact n* ∗ *exp* (*of_real* (*x* ∗ *ln* (*real_of_nat n*)))))
using *n* **by** (*simp add*: *rGamma_series_def powr_def pochhammer_of_real*)
also from *n* **have** ... = *Re* (*of_real* ((*pochhammer x* (*Suc n*)) /
(*fact n* ∗ (*exp* (*x* ∗ *ln* (*real_of_nat n*))))))
by (*subst exp_of_real*) *simp*
also from *n* **have** ... = *rGamma_series x n*
by (*subst Re_complex_of_real*) (*simp add*: *rGamma_series_def powr_def*)
finally show *rGamma_series x n* = *Re* (*rGamma_series* (*of_real x*) *n*) **..**
qed

instantiation *real* :: *Gamma*
begin

definition *rGamma_real x* = *Re* (*rGamma* (*of_real x* :: *complex*))

instance proof
fix *x* :: *real*
have *rGamma x* = *Re* (*rGamma* (*of_real x*)) **by** (*simp add*: *rGamma_real_def*)
also have *of_real* ... = *rGamma* (*of_real x* :: *complex*)
by (*intro of_real_Re rGamma_complex_real*) *simp_all*
also have ... = *0* ⟷ *x* ∈ ℤ≤0 **by** (*simp add*: *rGamma_eq_zero_iff of_real_in_nonpos_Ints_iff*)
also have ... ⟷ (∃ *n*. *x* = − *of_nat n*) **by** (*auto elim*!: *nonpos_Ints_cases*′)
finally show (*rGamma x*) = *0* ⟷ (∃ *n*. *x* = − *real_of_nat n*) **by** *simp*
next
fix *x* :: *real* **assume** ⋀*n*. *x* ≠ − *of_nat n*
hence *x*: *complex_of_real x* ∉ ℤ≤0
by (*subst of_real_in_nonpos_Ints_iff*) (*auto elim*!: *nonpos_Ints_cases*′)
then have *x* ≠ *0* **by** *auto*
with *x* **have** (*rGamma has_field_derivative* − *rGamma x* ∗ *Digamma x*) (*at x*)
by (*fastforce intro*!: *derivative_eq_intros has_vector_derivative_real_field*
simp: *Polygamma_of_real rGamma_real_def* [*abs_def*])
thus *let d* = (*THE d*. (λ*n*. ∑*k*<*n*. *inverse* (*of_nat* (*Suc k*)) − *inverse* (*x* + *of_nat k*))
⟶ *d*) − *euler_mascheroni* ∗*R* *1 in* (λ*y*. (*rGamma y* − *rGamma x* +
rGamma x ∗ *d* ∗ (*y* − *x*)) /*R* *norm* (*y* − *x*)) −*x*→ *0*
by (*simp add*: *has_field_derivative_def has_derivative_def Digamma_def sums_def* [*abs_def*]
of_real_def[*symmetric*] *suminf_def*)
next

fix n :: *nat*
have (*rGamma has_field_derivative* $(-1)\hat{}n * fact\ n$) (*at* $(-\ of\_nat\ n :: real)$)
by (*fastforce intro*!: *derivative_eq_intros has_vector_derivative_real_field*
simp: *Polygamma_of_real rGamma_real_def* [*abs_def*])
thus *let* $x = -\ of\_nat\ n$ *in* $(\lambda y.\ (rGamma\ y - rGamma\ x - (-\ 1)\ \hat{}\ n * prod\ of\_nat\ \{1..n\} *$
$(y - x))\ /_R\ norm\ (y - x))\ -x::real\to\ 0$
by (*simp add*: *has_field_derivative_def has_derivative_def fact_prod Let_def*)
next
fix x :: *real*
have *rGamma_series* $x \longrightarrow rGamma\ x$
proof (*rule Lim_transform_eventually*)
show $(\lambda n.\ Re\ (rGamma\_series\ (of\_real\ x)\ n)) \longrightarrow rGamma\ x$ **unfolding** *rGamma_real_def*
by (*intro tendsto_intros*)
qed (*insert rGamma_series_real*, *simp add*: *eq_commute*)
thus *let* $fact' = \lambda n.\ prod\ of\_nat\ \{1..n\}$;
$exp = \lambda x.\ THE\ e.\ (\lambda n.\ \sum k<n.\ x\ \hat{}\ k\ /_R\ fact\ k) \longrightarrow e$;
$pochhammer' = \lambda a\ n.\ \prod n = 0..n.\ a + of\_nat\ n$
in $(\lambda n.\ pochhammer'\ x\ n\ /\ (fact'\ n * exp\ (x * ln\ (real\_of\_nat\ n) *_R\ 1)))$
$\longrightarrow rGamma\ x$
by (*simp add*: *fact_prod pochhammer_Suc_prod rGamma_series_def* [*abs_def*] *exp_def*
of_real_def [*symmetric*] *suminf_def sums_def* [*abs_def*] *atLeast0AtMost*)
qed

end

lemma *rGamma_complex_of_real*: $rGamma\ (complex\_of\_real\ x) = complex\_of\_real\ (rGamma\ x)$
unfolding *rGamma_real_def* **using** *rGamma_complex_real* **by** *simp*

lemma *Gamma_complex_of_real*: $Gamma\ (complex\_of\_real\ x) = complex\_of\_real\ (Gamma\ x)$
unfolding *Gamma_def* **by** (*simp add*: *rGamma_complex_of_real*)

lemma *rGamma_real_altdef*: $rGamma\ x = lim\ (rGamma\_series\ (x :: real))$
by (*rule sym*, *rule limI*, *rule tendsto_intros*)

lemma *Gamma_real_altdef1*: $Gamma\ x = lim\ (Gamma\_series\ (x :: real))$
by (*rule sym*, *rule limI*, *rule tendsto_intros*)

lemma *Gamma_real_altdef2*: $Gamma\ x = Re\ (Gamma\ (of\_real\ x))$
using *rGamma_complex_real*[*OF Reals_of_real*[*of x*]]
by (*elim Reals_cases*)
(*simp only*: *Gamma_def rGamma_real_def of_real_inverse*[*symmetric*] *Re_complex_of_real*)

lemma *ln_Gamma_series_complex_of_real*:
  $x > 0 \Longrightarrow n > 0 \Longrightarrow$ *ln_Gamma_series* (*complex_of_real x*) *n* = *of_real* (*ln_Gamma_series x n*)
proof −
 assume *xn*: *x* > *0 n* > *0*
 have *Ln* (*complex_of_real x* / *of_nat k* + *1*) = *of_real* (*ln* (*x* / *of_nat k* + *1*))
if *k* ≥ *1* **for** *k*
  using *that xn* **by** (*subst Ln_of_real* [*symmetric*]) (*auto intro*!: *add_nonneg_pos simp*: *field_simps*)
 with *xn* **show** *?thesis* **by** (*simp add*: *ln_Gamma_series_def Ln_of_real*)
qed

lemma *ln_Gamma_real_converges*:
 assumes (*x*::*real*) > *0*
 shows *convergent* (*ln_Gamma_series x*)
proof −
 have (λ*n*. *ln_Gamma_series* (*complex_of_real x*) *n*) ⟶ *ln_Gamma* (*of_real x*) **using** *assms*
  by (*intro ln_Gamma_complex_LIMSEQ*) (*auto simp*: *of_real_in_nonpos_Ints_iff*)
 moreover from *eventually_gt_at_top*[*of 0*::*nat*]
  have *eventually* (λ*n*. *complex_of_real* (*ln_Gamma_series x n*) =
      ln_Gamma_series (*complex_of_real x*) *n*) *sequentially*
  by *eventually_elim* (*simp add*: *ln_Gamma_series_complex_of_real assms*)
 ultimately have (λ*n*. *complex_of_real* (*ln_Gamma_series x n*)) ⟶ *ln_Gamma* (*of_real x*)
  by (*subst tendsto_cong*) *assumption*+
 from *tendsto_Re*[*OF this*] **show** *?thesis* **by** (*auto simp*: *convergent_def*)
qed

lemma *ln_Gamma_real_LIMSEQ*: (*x*::*real*) > *0* ⟹ *ln_Gamma_series x* ⟶ *ln_Gamma x*
  using *ln_Gamma_real_converges*[*of x*] **unfolding** *ln_Gamma_def* **by** (*simp add*: *convergent_LIMSEQ_iff*)

lemma *ln_Gamma_complex_of_real*: *x* > *0* ⟹ *ln_Gamma* (*complex_of_real x*) = *of_real* (*ln_Gamma x*)
proof (*unfold ln_Gamma_def*, *rule limI*, *rule Lim_transform_eventually*)
 assume *x*: *x* > *0*
 show *eventually* (λ*n*. *of_real* (*ln_Gamma_series x n*) =
      ln_Gamma_series (*complex_of_real x*) *n*) *sequentially*
  using *eventually_gt_at_top*[*of 0*::*nat*]
  by *eventually_elim* (*simp add*: *ln_Gamma_series_complex_of_real x*)
qed (*intro tendsto_of_real*, *insert ln_Gamma_real_LIMSEQ*[*of x*], *simp add*: *ln_Gamma_def*)

lemma *Gamma_real_pos_exp*: *x* > (*0* :: *real*) ⟹ *Gamma x* = *exp* (*ln_Gamma x*)
 by (*auto simp*: *Gamma_real_altdef2 Gamma_complex_altdef of_real_in_nonpos_Ints_iff*
      ln_Gamma_complex_of_real exp_of_real)

lemma *ln_Gamma_real_pos*: $x > 0 \Longrightarrow ln\_Gamma\ x = ln\ (Gamma\ x :: real)$
unfolding *Gamma_real_pos_exp* **by** *simp*

lemma *ln_Gamma_complex_conv_fact*: $n > 0 \Longrightarrow ln\_Gamma\ (of\_nat\ n :: complex) = ln\ (fact\ (n - 1))$
using *ln_Gamma_complex_of_real*[*of real n*] *Gamma_fact*[*of n − 1*, **where** *'a = real*]
by (*simp add*: *ln_Gamma_real_pos of_nat_diff Ln_of_real* [*symmetric*])

lemma *ln_Gamma_real_conv_fact*: $n > 0 \Longrightarrow ln\_Gamma\ (real\ n) = ln\ (fact\ (n - 1))$
using *Gamma_fact*[*of n − 1*, **where** *'a = real*]
by (*simp add*: *ln_Gamma_real_pos of_nat_diff Ln_of_real* [*symmetric*])

lemma *Gamma_real_pos* [*simp*, *intro*]: $x > (0::real) \Longrightarrow Gamma\ x > 0$
by (*simp add*: *Gamma_real_pos_exp*)

lemma *Gamma_real_nonneg* [*simp*, *intro*]: $x > (0::real) \Longrightarrow Gamma\ x \geq 0$
by (*simp add*: *Gamma_real_pos_exp*)

lemma *has_field_derivative_ln_Gamma_real* [*derivative_intros*]:
assumes *x*: $x > (0::real)$
shows (*ln_Gamma has_field_derivative Digamma x*) (*at x*)
proof (*subst DERIV_cong_ev*[*OF refl _ refl*])
from *assms* **show** ((*Re* ∘ *ln_Gamma* ∘ *complex_of_real*) *has_field_derivative Digamma x*) (*at x*)
by (*auto intro*!: *derivative_eq_intros has_vector_derivative_real_field*
simp: *Polygamma_of_real o_def*)
from *eventually_nhds_in_nhd*[*of x* {*0*<..}] *assms*
show *eventually* (λ*y*. *ln_Gamma y* = (*Re* ∘ *ln_Gamma* ∘ *of_real*) *y*) (*nhds x*)
by (*auto elim*!: *eventually_mono simp*: *ln_Gamma_complex_of_real interior_open*)
qed

lemma *field_differentiable_ln_Gamma_real*:
$x > 0 \Longrightarrow ln\_Gamma\ field\_differentiable\ (at\ (x::real)\ within\ A)$
by (*rule field_differentiable_within_subset*[*of _ _ UNIV*])
(*auto simp*: *field_differentiable_def intro*!: *derivative_intros*)+

declare *has_field_derivative_ln_Gamma_real*[*THEN DERIV_chain2*, *derivative_intros*]

lemma *deriv_ln_Gamma_real*:
assumes $z > 0$
shows *deriv ln_Gamma z* = *Digamma* (*z* :: *real*)
by (*intro DERIV_imp_deriv has_field_derivative_ln_Gamma_real assms*)

lemma *higher_deriv_ln_Gamma_real*:
assumes $(x::real) > 0$

```
  shows   (deriv ^^ j) ln_Gamma x = (if j = 0 then ln_Gamma x else Polygamma
(j - 1) x)
proof (cases j)
  case (Suc j')
  have (deriv ^^ j') (deriv ln_Gamma) x = (deriv ^^ j') Digamma x
    using eventually_nhds_in_open[of {0<..} x] assms
    by (intro higher_deriv_cong_ev refl)
       (auto elim!: eventually_mono simp: open_Diff deriv_ln_Gamma_real)
  also have ... = Polygamma j' x using assms
    by (subst higher_deriv_Polygamma)
       (auto elim!: nonpos_Ints_cases simp: complex_nonpos_Reals_iff)
  finally show ?thesis using Suc by (simp del: funpow.simps add: funpow_Suc_right)
qed simp_all

lemma higher_deriv_ln_Gamma_complex_of_real:
  assumes (x :: real) > 0
  shows   (deriv ^^ j) ln_Gamma (complex_of_real x) = of_real ((deriv ^^ j)
ln_Gamma x)
    using assms
   by (auto simp: higher_deriv_ln_Gamma_real higher_deriv_ln_Gamma_complex
                  ln_Gamma_complex_of_real Polygamma_of_real)

lemma has_field_derivative_rGamma_real' [derivative_intros]:
  (rGamma has_field_derivative (if x ∈ ℤ≤0 then (-1)^(nat ⌊-x⌋) * fact (nat
⌊-x⌋) else
       -rGamma x * Digamma x)) (at x within A)
  using has_field_derivative_rGamma[of x] by (force elim!: nonpos_Ints_cases')

declare has_field_derivative_rGamma_real'[THEN DERIV_chain2, derivative_intros]

lemma Polygamma_real_odd_pos:
  assumes (x::real) ∉ ℤ≤0 odd n
  shows   Polygamma n x > 0
proof -
  from assms have x ≠ 0 by auto
  with assms show ?thesis
    unfolding Polygamma_def using Polygamma_converges'[of x Suc n]
    by (auto simp: zero_less_power_eq simp del: power_Suc
          dest: plus_of_nat_eq_0_imp intro!: mult_pos_pos suminf_pos)
qed

lemma Polygamma_real_even_neg:
  assumes (x::real) > 0 n > 0 even n
  shows   Polygamma n x < 0
  using assms unfolding Polygamma_def using Polygamma_converges'[of x Suc
n]
  by (auto intro!: mult_pos_pos suminf_pos)

lemma Polygamma_real_strict_mono:
```

assumes $x > 0$ $x < (y::real)$ *even n*
shows *Polygamma n x < Polygamma n y*
proof −
have $\exists\xi.\ x < \xi \wedge \xi < y \wedge$ *Polygamma n y − Polygamma n x* $= (y - x) *$ *Polygamma (Suc n)* ξ
using *assms* **by** (*intro MVT2 derivative_intros impI allI*) (*auto elim*!: *nonpos_Ints_cases*)
then obtain ξ
where ξ: $x < \xi$ $\xi < y$
and *Polygamma*: *Polygamma n y − Polygamma n x* $= (y - x) *$ *Polygamma (Suc n)* ξ
by *auto*
note *Polygamma*
also from ξ *assms* **have** $(y - x) *$ *Polygamma (Suc n)* $\xi > 0$
by (*intro mult_pos_pos Polygamma_real_odd_pos*) (*auto elim*!: *nonpos_Ints_cases*)
finally show *?thesis* **by** *simp*
qed

lemma *Polygamma_real_strict_antimono*:
assumes $x > 0$ $x < (y::real)$ *odd n*
shows *Polygamma n x > Polygamma n y*
proof −
have $\exists\xi.\ x < \xi \wedge \xi < y \wedge$ *Polygamma n y − Polygamma n x* $= (y - x) *$ *Polygamma (Suc n)* ξ
using *assms* **by** (*intro MVT2 derivative_intros impI allI*) (*auto elim*!: *nonpos_Ints_cases*)
then obtain ξ
where ξ: $x < \xi$ $\xi < y$
and *Polygamma*: *Polygamma n y − Polygamma n x* $= (y - x) *$ *Polygamma (Suc n)* ξ
by *auto*
note *Polygamma*
also from ξ *assms* **have** $(y - x) *$ *Polygamma (Suc n)* $\xi < 0$
by (*intro mult_pos_neg Polygamma_real_even_neg*) *simp_all*
finally show *?thesis* **by** *simp*
qed

lemma *Polygamma_real_mono*:
assumes $x > 0$ $x \leq (y::real)$ *even n*
shows *Polygamma n x ≤ Polygamma n y*
using *Polygamma_real_strict_mono*[*OF assms*(*1*) _ *assms*(*3*), *of y*] *assms*(*2*)
by (*cases x = y*) *simp_all*

lemma *Digamma_real_strict_mono*: $(0::real) < x \Longrightarrow x < y \Longrightarrow$ *Digamma x < Digamma y*
by (*rule Polygamma_real_strict_mono*) *simp_all*

lemma *Digamma_real_mono*: $(0::real) < x \Longrightarrow x \leq y \Longrightarrow$ *Digamma x ≤ Digamma y*

by (*rule Polygamma_real_mono*) *simp_all*

lemma *Digamma_real_ge_three_halves_pos*:
assumes $x \geq 3/2$
shows *Digamma* (*x* :: *real*) > *0*
proof −
have *0* < *Digamma* (*3/2* :: *real*) **by** (*fact Digamma_real_three_halves_pos*)
also from *assms* **have** ... ≤ *Digamma x* **by** (*intro Polygamma_real_mono*) *simp_all*
finally show *?thesis* **.**
qed

lemma *ln_Gamma_real_strict_mono*:
assumes $x \geq 3/2$ $x < y$
shows *ln_Gamma* (*x* :: *real*) < *ln_Gamma y*
proof −
have ∃ξ. *x* < ξ ∧ ξ < *y* ∧ *ln_Gamma y* − *ln_Gamma x* = (*y* − *x*) ∗ *Digamma* ξ
using *assms* **by** (*intro MVT2 derivative_intros impI allI*) (*auto elim*!: *nonpos_Ints_cases*)
then obtain ξ **where** ξ: *x* < ξ ξ < *y*
and *ln_Gamma*: *ln_Gamma y* − *ln_Gamma x* = (*y* − *x*) ∗ *Digamma* ξ
by *auto*
note *ln_Gamma*
also from ξ *assms* **have** (*y* − *x*) ∗ *Digamma* ξ > *0*
by (*intro mult_pos_pos Digamma_real_ge_three_halves_pos*) *simp_all*
finally show *?thesis* **by** *simp*
qed

lemma *Gamma_real_strict_mono*:
assumes $x \geq 3/2$ $x < y$
shows *Gamma* (*x* :: *real*) < *Gamma y*
proof −
from *Gamma_real_pos_exp*[*of x*] *assms* **have** *Gamma x* = *exp* (*ln_Gamma x*)
by *simp*
also have ... < *exp* (*ln_Gamma y*) **by** (*intro exp_less_mono ln_Gamma_real_strict_mono assms*)
also from *Gamma_real_pos_exp*[*of y*] *assms* **have** ... = *Gamma y* **by** *simp*
finally show *?thesis* **.**
qed

theorem *log_convex_Gamma_real*: *convex_on* {*0*<..} (*ln* ∘ *Gamma* :: *real* ⇒ *real*)
by (*rule convex_on_realI*[*of* _ _ *Digamma*])
(*auto intro*!: *derivative_eq_intros Polygamma_real_mono Gamma_real_pos*
simp: *o_def Gamma_eq_zero_iff elim*!: *nonpos_Ints_cases*′)

6.20.7 The uniqueness of the real Gamma function

The following is a proof of the Bohr–Mollerup theorem, which states that any log-convex function G on the positive reals that fulfils $G(1) = 1$ and satisfies the functional equation $G(x + 1) = x\ G(x)$ must be equal to the Gamma function. In principle, if G is a holomorphic complex function, one could then extend this from the positive reals to the entire complex plane (minus the non-positive integers, where the Gamma function is not defined).

```
context
  fixes G :: real ⇒ real
  assumes G_1: G 1 = 1
  assumes G_plus1: x > 0 ⟹ G (x + 1) = x * G x
  assumes G_pos: x > 0 ⟹ G x > 0
  assumes log_convex_G: convex_on {0<..} (ln ∘ G)
begin

private lemma G_fact: G (of_nat n + 1) = fact n
  using G_plus1[of real n + 1 for n]
  by (induction n) (simp_all add: G_1 G_plus1)

private definition S :: real ⇒ real ⇒ real where
  S x y = (ln (G y) - ln (G x)) / (y - x)

private lemma S_eq:
  n ≥ 2 ⟹ S (of_nat n) (of_nat n + x) = (ln (G (real n + x)) - ln (fact (n -
1))) / x
  by (subst G_fact [symmetric]) (simp add: S_def add_ac of_nat_diff)

private lemma G_lower:
  assumes x: x > 0 and n: n ≥ 1
  shows  Gamma_series x n ≤ G x
proof -
  have (ln ∘ G) (real (Suc n)) ≤ ((ln ∘ G) (real (Suc n) + x) -
        (ln ∘ G) (real (Suc n) - 1)) / (real (Suc n) + x - (real (Suc n) - 1)) *
         (real (Suc n) - (real (Suc n) - 1)) + (ln ∘ G) (real (Suc n) - 1)
    using x n by (intro convex_onD_Icc' convex_on_subset[OF log_convex_G])
auto
  hence S (of_nat n) (of_nat (Suc n)) ≤ S (of_nat (Suc n)) (of_nat (Suc n) +
x)
    unfolding S_def using x by (simp add: field_simps)
  also have S (of_nat n) (of_nat (Suc n)) = ln (fact n) - ln (fact (n-1))
    unfolding S_def using n
    by (subst (1 2) G_fact [symmetric]) (simp_all add: add_ac of_nat_diff)
  also have ... = ln (fact n / fact (n-1)) by (subst ln_div) simp_all
  also from n have fact n / fact (n - 1) = n by (cases n) simp_all
  finally have x * ln (real n) + ln (fact n) ≤ ln (G (real (Suc n) + x))
    using x n by (subst (asm) S_eq) (simp_all add: field_simps)
  also have x * ln (real n) + ln (fact n) = ln (exp (x * ln (real n)) * fact n)
```

```
    using x by (simp add: ln_mult)
  finally have exp (x * ln (real n)) * fact n ≤ G (real (Suc n) + x) using x
    by (subst (asm) ln_le_cancel_iff) (simp_all add: G_pos)
  also have G (real (Suc n) + x) = pochhammer x (Suc n) * G x
    using G_plus1[of real (Suc n) + x for n] G_plus1[of x] x
    by (induction n) (simp_all add: pochhammer_Suc add_ac)
  finally show Gamma_series x n ≤ G x
    using x by (simp add: field_simps pochhammer_pos Gamma_series_def)
qed

private lemma G_upper:
  assumes x: x > 0 x ≤ 1 and n: n ≥ 2
  shows  G x ≤ Gamma_series x n * (1 + x / real n)
proof −
  have (ln ∘ G) (real n + x) ≤ ((ln ∘ G) (real n + 1) −
        (ln ∘ G) (real n)) / (real n + 1 − (real n)) *
         ((real n + x) − real n) + (ln ∘ G) (real n)
    using x n by (intro convex_onD_Icc' convex_on_subset[OF log_convex_G])
auto
  hence S (of_nat n) (of_nat n + x) ≤ S (of_nat n) (of_nat n + 1)
    unfolding S_def using x by (simp add: field_simps)
  also from n have S (of_nat n) (of_nat n + 1) = ln (fact n) − ln (fact (n−1))
    by (subst (1 2) G_fact [symmetric]) (simp add: S_def add_ac of_nat_diff)
  also have ... = ln (fact n / (fact (n−1))) using n by (subst ln_div) simp_all
  also from n have fact n / fact (n − 1) = n by (cases n) simp_all
  finally have ln (G (real n + x)) ≤ x * ln (real n) + ln (fact (n − 1))
    using x n by (subst (asm) S_eq) (simp_all add: field_simps)
  also have ... = ln (exp (x * ln (real n)) * fact (n − 1)) using x
    by (simp add: ln_mult)
  finally have G (real n + x) ≤ exp (x * ln (real n)) * fact (n − 1) using x
    by (subst (asm) ln_le_cancel_iff) (simp_all add: G_pos)
  also have G (real n + x) = pochhammer x n * G x
    using G_plus1[of real n + x for n] x
    by (induction n) (simp_all add: pochhammer_Suc add_ac)
  finally have G x ≤ exp (x * ln (real n)) * fact (n− 1) / pochhammer x n
    using x by (simp add: field_simps pochhammer_pos)
  also from n have fact (n − 1) = fact n / n by (cases n) simp_all
  also have exp (x * ln (real n)) * ... / pochhammer x n =
              Gamma_series x n * (1 + x / real n) using n x
    by (simp add: Gamma_series_def divide_simps pochhammer_Suc)
  finally show ?thesis .
qed

private lemma G_eq_Gamma_aux:
  assumes x: x > 0 x ≤ 1
  shows   G x = Gamma x
proof (rule antisym)
  show G x ≥ Gamma x
  proof (rule tendsto_upperbound)
```

```
    from G_lower[of x] show eventually (λn. Gamma_series x n ≤ G x) sequentially
        using x by (auto intro: eventually_mono[OF eventually_ge_at_top[of 1::nat]])
  qed (simp_all add: Gamma_series_LIMSEQ)
next
  show G x ≤ Gamma x
  proof (rule tendsto_lowerbound)
    have (λn. Gamma_series x n * (1 + x / real n)) ⟶ Gamma x * (1 + 0)
      by (rule tendsto_intros real_tendsto_divide_at_top
            Gamma_series_LIMSEQ filterlim_real_sequentially)+
    thus (λn. Gamma_series x n * (1 + x / real n)) ⟶ Gamma x by simp
  next
    from G_upper[of x] show eventually (λn. Gamma_series x n * (1 + x / real n) ≥ G x) sequentially
    using x by (auto intro: eventually_mono[OF eventually_ge_at_top[of 2::nat]])
  qed simp_all
qed

theorem Gamma_pos_real_unique:
  assumes x: x > 0
  shows   G x = Gamma x
proof -
  have G_eq: G (real n + x) = Gamma (real n + x) if x ∈ {0<..1} for n x using that
  proof (induction n)
    case (Suc n)
    from Suc have x + real n > 0 by simp
    hence x + real n ∉ ℤ≤0 by auto
    with Suc show ?case using G_plus1[of real n + x] Gamma_plus1[of real n + x]
      by (auto simp: add_ac)
  qed (simp_all add: G_eq_Gamma_aux)

  show ?thesis
  proof (cases frac x = 0)
    case True
    hence x = of_int (floor x) by (simp add: frac_def)
    with x have x_eq: x = of_nat (nat (floor x) - 1) + 1 by simp
    show ?thesis by (subst (1 2) x_eq, rule G_eq) simp_all
  next
    case False
    from assms have x_eq: x = of_nat (nat (floor x)) + frac x
      by (simp add: frac_def)
    have frac_le_1: frac x ≤ 1 unfolding frac_def by linarith
    show ?thesis
      by (subst (1 2) x_eq, rule G_eq, insert False frac_le_1) simp_all
  qed
qed
```

end

6.20.8 The Beta function

definition *Beta* **where** *Beta a b* = *Gamma a* ∗ *Gamma b* / *Gamma* (*a* + *b*)

lemma *Beta_altdef*: *Beta a b* = *Gamma a* ∗ *Gamma b* ∗ *rGamma* (*a* + *b*)
by (*simp add*: *inverse_eq_divide Beta_def Gamma_def*)

lemma *Beta_commute*: *Beta a b* = *Beta b a*
unfolding *Beta_def* **by** (*simp add*: *ac_simps*)

lemma *has_field_derivative_Beta1* [*derivative_intros*]:
assumes *x* ∉ ℤ≤0 *x* + *y* ∉ ℤ≤0
shows ((λ*x*. *Beta x y*) *has_field_derivative* (*Beta x y* ∗ (*Digamma x* − *Digamma* (*x* + *y*))))
(*at x within A*) **unfolding** *Beta_altdef*
by (*rule DERIV_cong*, (*rule derivative_intros assms*)+) (*simp add*: *algebra_simps*)

lemma *Beta_pole1*: *x* ∈ ℤ≤0 ⟹ *Beta x y* = *0*
by (*auto simp add*: *Beta_def elim*!: *nonpos_Ints_cases*′)

lemma *Beta_pole2*: *y* ∈ ℤ≤0 ⟹ *Beta x y* = *0*
by (*auto simp add*: *Beta_def elim*!: *nonpos_Ints_cases*′)

lemma *Beta_zero*: *x* + *y* ∈ ℤ≤0 ⟹ *Beta x y* = *0*
by (*auto simp add*: *Beta_def elim*!: *nonpos_Ints_cases*′)

lemma *has_field_derivative_Beta2* [*derivative_intros*]:
assumes *y* ∉ ℤ≤0 *x* + *y* ∉ ℤ≤0
shows ((λ*y*. *Beta x y*) *has_field_derivative* (*Beta x y* ∗ (*Digamma y* − *Digamma* (*x* + *y*))))
(*at y within A*)
using *has_field_derivative_Beta1*[*of y x A*] *assms* **by** (*simp add*: *Beta_commute add_ac*)

theorem *Beta_plus1_plus1*:
assumes *x* ∉ ℤ≤0 *y* ∉ ℤ≤0
shows *Beta* (*x* + *1*) *y* + *Beta x* (*y* + *1*) = *Beta x y*
proof −
have *Beta* (*x* + *1*) *y* + *Beta x* (*y* + *1*) =
(*Gamma* (*x* + *1*) ∗ *Gamma y* + *Gamma x* ∗ *Gamma* (*y* + *1*)) ∗ *rGamma* ((*x* + *y*) + *1*)
by (*simp add*: *Beta_altdef add_divide_distrib algebra_simps*)
also have ... = (*Gamma x* ∗ *Gamma y*) ∗ ((*x* + *y*) ∗ *rGamma* ((*x* + *y*) + *1*))
by (*subst assms*[*THEN Gamma_plus1*])+ (*simp add*: *algebra_simps*)
also from *assms* **have** ... = *Beta x y* **unfolding** *Beta_altdef* **by** (*subst rGamma_plus1*) *simp*

finally show *?thesis* **.**
qed

theorem *Beta_plus1_left*:
assumes $x \notin \mathbb{Z}_{\leq 0}$
shows $(x + y) * Beta\ (x + 1)\ y = x * Beta\ x\ y$
proof −
have $(x + y) * Beta\ (x + 1)\ y = Gamma\ (x + 1) * Gamma\ y * ((x + y) * rGamma\ ((x + y) + 1))$
unfolding *Beta_altdef* **by** (*simp only*: *ac_simps*)
also have $\ldots = x * Beta\ x\ y$ **unfolding** *Beta_altdef*
by (*subst assms*[*THEN Gamma_plus1*] *rGamma_plus1*)+ (*simp only*: *ac_simps*)
finally show *?thesis* **.**
qed

theorem *Beta_plus1_right*:
assumes $y \notin \mathbb{Z}_{\leq 0}$
shows $(x + y) * Beta\ x\ (y + 1) = y * Beta\ x\ y$
using *Beta_plus1_left*[*of y x*] *assms* **by** (*simp_all add*: *Beta_commute add.commute*)

lemma *Gamma_Gamma_Beta*:
assumes $x + y \notin \mathbb{Z}_{\leq 0}$
shows $Gamma\ x * Gamma\ y = Beta\ x\ y * Gamma\ (x + y)$
unfolding *Beta_altdef* **using** *assms Gamma_eq_zero_iff*[*of x+y*]
by (*simp add*: *rGamma_inverse_Gamma*)

6.20.9 Legendre duplication theorem

context
begin

private lemma *Gamma_legendre_duplication_aux*:
fixes $z :: 'a :: Gamma$
assumes $z \notin \mathbb{Z}_{\leq 0}\ z + 1/2 \notin \mathbb{Z}_{\leq 0}$
shows $Gamma\ z * Gamma\ (z + 1/2) = exp\ ((1 - 2*z) * of\_real\ (ln\ 2)) * Gamma\ (1/2) * Gamma\ (2*z)$
proof −
let *?powr* $= \lambda b\ a.\ exp\ (a * of\_real\ (ln\ (of\_nat\ b)))$
let *?h* $= \lambda n.\ (fact\ (n-1))^2\ /\ fact\ (2*n-1) * of\_nat\ (2\hat{}(2*n)) * exp\ (1/2 * of\_real\ (ln\ (real\_of\_nat\ n)))$
{
fix $z :: 'a$ **assume** *z*: $z \notin \mathbb{Z}_{\leq 0}\ z + 1/2 \notin \mathbb{Z}_{\leq 0}$
let *?g* $= \lambda n.$ *?powr 2* $(2*z) * Gamma\_series'\ z\ n * Gamma\_series'\ (z + 1/2)\ n\ /$
$Gamma\_series'\ (2*z)\ (2*n)$
have *eventually* $(\lambda n.$ *?g* $n =$ *?h* $n)$ *sequentially* **using** *eventually_gt_at_top*
proof *eventually_elim*
fix $n :: nat$ **assume** *n*: $n > 0$
let *?f* $= fact\ (n - 1) :: 'a$ **and** *?f'* $= fact\ (2*n - 1) :: 'a$

have *A*: *exp t ∗ exp t = exp (2∗t :: 'a)* **for** *t* **by** (*subst exp_add* [*symmetric*]) *simp*
have *A*: *Gamma_series' z n ∗ Gamma_series' (z + 1/2) n = ?f^2 ∗ ?powr n (2∗z + 1/2) /*
(pochhammer z n ∗ pochhammer (z + 1/2) n)
by (*simp add*: *Gamma_series'_def exp_add ring_distribs power2_eq_square A mult_ac*)
have *B*: *Gamma_series' (2∗z) (2∗n) =*
?f' ∗ ?powr 2 (2∗z) ∗ ?powr n (2∗z) /
(of_nat (2^(2∗n)) ∗ pochhammer z n ∗ pochhammer (z+1/2) n) **using** *n*
by (*simp add*: *Gamma_series'_def ln_mult exp_add ring_distribs pochhammer_double*)
from *z* **have** *pochhammer z n ≠ 0* **by** (*auto dest*: *pochhammer_eq_0_imp_nonpos_Int*)
moreover from *z* **have** *pochhammer (z + 1/2) n ≠ 0* **by** (*auto dest*: *pochhammer_eq_0_imp_nonpos_Int*)
ultimately have *?powr 2 (2∗z) ∗ (Gamma_series' z n ∗ Gamma_series' (z + 1/2) n) / Gamma_series' (2∗z) (2∗n) =*
?f^2 / ?f' ∗ of_nat (2^(2∗n)) ∗ (?powr n ((4∗z + 1)/2) ∗ ?powr n (−2∗z))
using *n* **unfolding** *A B* **by** (*simp add*: *field_split_simps exp_minus*)
also have *?powr n ((4∗z + 1)/2) ∗ ?powr n (−2∗z) = ?powr n (1/2)*
by (*simp add*: *algebra_simps exp_add*[*symmetric*] *add_divide_distrib*)
finally show *?g n = ?h n* **by** (*simp only*: *mult_ac*)
qed

moreover from *z double_in_nonpos_Ints_imp*[*of z*] **have** $2 * z \notin \mathbb{Z}_{\leq 0}$ **by** *auto*
hence *?g ⟶ ?powr 2 (2∗z) ∗ Gamma z ∗ Gamma (z+1/2) / Gamma (2∗z)*
using *LIMSEQ_subseq_LIMSEQ*[*OF Gamma_series'_LIMSEQ*, *of* (∗)*2 2∗z*]
by (*intro tendsto_intros Gamma_series'_LIMSEQ*)
(*simp_all add*: *o_def strict_mono_def Gamma_eq_zero_iff*)
ultimately have *?h ⟶ ?powr 2 (2∗z) ∗ Gamma z ∗ Gamma (z+1/2) / Gamma (2∗z)*
by (*blast intro*: *Lim_transform_eventually*)
} **note** *lim = this*

from *assms double_in_nonpos_Ints_imp*[*of z*] **have** *z'*: $2 * z \notin \mathbb{Z}_{\leq 0}$ **by** *auto*
from *fraction_not_in_ints*[*of 2 1*] **have** $(1/2 :: 'a) \notin \mathbb{Z}_{\leq 0}$
by (*intro not_in_Ints_imp_not_in_nonpos_Ints*) *simp_all*
with *lim*[*of 1/2 :: 'a*] **have** *?h ⟶ 2 ∗ Gamma (1/2 :: 'a)* **by** (*simp add*: *exp_of_real*)
from *LIMSEQ_unique*[*OF this lim*[*OF assms*]] *z'* **show** *?thesis*
by (*simp add*: *field_split_simps Gamma_eq_zero_iff ring_distribs exp_diff exp_of_real*)
qed

The following lemma is somewhat annoying. With a little bit of complex analysis (Cauchy's integral theorem, to be exact), this would be completely

trivial. However, we want to avoid depending on the complex analysis session at this point, so we prove it the hard way.

private lemma *Gamma_reflection_aux*:
defines $h \equiv \lambda z{::}complex.\ if\ z \in \mathbb{Z}\ then\ 0\ else$
$(of\_real\ pi * cot\ (of\_real\ pi{*}z) + Digamma\ z - Digamma\ (1 - z))$
defines $a \equiv complex\_of\_real\ pi$
obtains h' **where** $continuous\_on\ UNIV\ h' \bigwedge z.\ (h\ has\_field\_derivative\ (h'\ z))\ (at\ z)$
proof −
define f **where** $f\ n = a * of\_real\ (cos\_coeff\ (n{+}1) - sin\_coeff\ (n{+}2))$ **for** n
define F **where** $F\ z = (if\ z = 0\ then\ 0\ else\ (cos\ (a{*}z) - sin\ (a{*}z)/(a{*}z))\ /\ z)$ **for** z
define g **where** $g\ n = complex\_of\_real\ (sin\_coeff\ (n{+}1))$ **for** n
define G **where** $G\ z = (if\ z = 0\ then\ 1\ else\ sin\ (a{*}z)/(a{*}z))$ **for** z
have a_nz: $a \neq 0$ **unfolding** a_def **by** $simp$

have $(\lambda n.\ f\ n * (a{*}z)\hat{}n)\ sums\ (F\ z) \wedge (\lambda n.\ g\ n * (a{*}z)\hat{}n)\ sums\ (G\ z)$
if $abs\ (Re\ z) < 1$ **for** z
proof $(cases\ z = 0;\ rule\ conjI)$
assume $z \neq 0$
note $z = this\ that$

from z **have** sin_nz: $sin\ (a{*}z) \neq 0$ **unfolding** a_def **by** $(auto\ simp{:}\ sin\_eq\_0)$
have $(\lambda n.\ of\_real\ (sin\_coeff\ n) * (a{*}z)\hat{}n)\ sums\ (sin\ (a{*}z))$ **using** $sin\_converges[of\ a{*}z]$
by $(simp\ add{:}\ scaleR\_conv\_of\_real)$
from $sums\_split\_initial\_segment[OF\ this,\ of\ 1]$
have $(\lambda n.\ (a{*}z) * of\_real\ (sin\_coeff\ (n{+}1)) * (a{*}z)\hat{}n)\ sums\ (sin\ (a{*}z))$ **by** $(simp\ add{:}\ mult\_ac)$
from $sums\_mult[OF\ this,\ of\ inverse\ (a{*}z)]\ z\ a\_nz$
have A: $(\lambda n.\ g\ n * (a{*}z)\hat{}n)\ sums\ (sin\ (a{*}z)/(a{*}z))$
by $(simp\ add{:}\ field\_simps\ g\_def)$
with z **show** $(\lambda n.\ g\ n * (a{*}z)\hat{}n)\ sums\ (G\ z)$ **by** $(simp\ add{:}\ G\_def)$
from $A\ z\ a\_nz\ sin\_nz$ **have** g_nz: $(\sum n.\ g\ n * (a{*}z)\hat{}n) \neq 0$ **by** $(simp\ add{:}\ sums\_iff\ g\_def)$

have $[simp]$: $sin\_coeff\ (Suc\ 0) = 1$ **by** $(simp\ add{:}\ sin\_coeff\_def)$
from $sums\_split\_initial\_segment[OF\ sums\_diff[OF\ cos\_converges[of\ a{*}z]\ A],\ of\ 1]$
have $(\lambda n.\ z * f\ n * (a{*}z)\hat{}n)\ sums\ (cos\ (a{*}z) - sin\ (a{*}z)\ /\ (a{*}z))$
by $(simp\ add{:}\ mult\_ac\ scaleR\_conv\_of\_real\ ring\_distribs\ f\_def\ g\_def)$
from $sums\_mult[OF\ this,\ of\ inverse\ z]\ z\ assms$
show $(\lambda n.\ f\ n * (a{*}z)\hat{}n)\ sums\ (F\ z)$ **by** $(simp\ add{:}\ divide\_simps\ mult\_ac\ f\_def\ F\_def)$
next
assume $z{:}\ z = 0$
have $(\lambda n.\ f\ n * (a * z)\ \hat{}\ n)\ sums\ f\ 0$ **using** $powser\_sums\_zero[of\ f]\ z$ **by** $simp$
with z **show** $(\lambda n.\ f\ n * (a * z)\ \hat{}\ n)\ sums\ (F\ z)$
by $(simp\ add{:}\ f\_def\ F\_def\ sin\_coeff\_def\ cos\_coeff\_def)$

have $(\lambda n.\ g\ n * (a * z)\ \hat{}\ n)$ *sums* $g\ 0$ **using** *powser_sums_zero*[*of* g] z **by** *simp*
with z **show** $(\lambda n.\ g\ n * (a * z)\ \hat{}\ n)$ *sums* $(G\ z)$
by (*simp add*: *g_def G_def sin_coeff_def cos_coeff_def*)
qed
note *sums* = *conjunct1*[*OF this*] *conjunct2*[*OF this*]

define *h2* **where** [*abs_def*]:
h2 $z = (\sum n.\ f\ n * (a*z)\hat{}n)\ /\ (\sum n.\ g\ n * (a*z)\hat{}n) + Digamma\ (1 + z) - Digamma\ (1 - z)$ **for** z
define *POWSER* **where** [*abs_def*]: *POWSER* $f\ z = (\sum n.\ f\ n * (z\hat{}n :: complex))$ **for** $f\ z$
define *POWSER'* **where** [*abs_def*]: *POWSER'* $f\ z = (\sum n.\ diffs\ f\ n * (z\hat{}n))$ **for** f **and** $z :: complex$
define *h2'* **where** [*abs_def*]:
h2' $z = a * (POWSER\ g\ (a*z) * POWSER'\ f\ (a*z) - POWSER\ f\ (a*z) * POWSER'\ g\ (a*z))\ /$
$(POWSER\ g\ (a*z))\hat{}2 + Polygamma\ 1\ (1 + z) + Polygamma\ 1\ (1 - z)$ **for** z

have *h_eq*: $h\ t = h2\ t$ **if** $abs\ (Re\ t) < 1$ **for** t
proof −
from *that* **have** *t*: $t \in \mathbb{Z} \longleftrightarrow t = 0$ **by** (*auto elim*!: *Ints_cases*)
hence $h\ t = a*cot\ (a*t) - 1/t + Digamma\ (1 + t) - Digamma\ (1 - t)$
unfolding *h_def* **using** *Digamma_plus1*[*of t*] **by** (*force simp*: *field_simps a_def*)
also have $a*cot\ (a*t) - 1/t = (F\ t)\ /\ (G\ t)$
using t **by** (*auto simp add*: *divide_simps sin_eq_0 cot_def a_def F_def G_def*)
also have $\ldots = (\sum n.\ f\ n * (a*t)\hat{}n)\ /\ (\sum n.\ g\ n * (a*t)\hat{}n)$
using *sums*[*of t*] *that* **by** (*simp add*: *sums_iff*)
finally show $h\ t = h2\ t$ **by** (*simp only*: *h2_def*)
qed

let *?A* = $\{z.\ abs\ (Re\ z) < 1\}$
have *open* $(\{z.\ Re\ z < 1\} \cap \{z.\ Re\ z > -1\})$
using *open_halfspace_Re_gt open_halfspace_Re_lt* **by** *auto*
also have $(\{z.\ Re\ z < 1\} \cap \{z.\ Re\ z > -1\}) = \{z.\ abs\ (Re\ z) < 1\}$ **by** *auto*
finally have *open_A*: *open ?A* **.**
hence [*simp*]: *interior ?A* = *?A* **by** (*simp add*: *interior_open*)

have *summable_f*: *summable* $(\lambda n.\ f\ n * z\hat{}n)$ **for** z
by (*rule powser_inside*, *rule sums_summable*, *rule sums*[*of* $\mathrm{i} * of\_real\ (norm\ z + 1)\ /\ a$])
(*simp_all add*: *norm_mult a_def del*: *of_real_add*)
have *summable_g*: *summable* $(\lambda n.\ g\ n * z\hat{}n)$ **for** z
by (*rule powser_inside*, *rule sums_summable*, *rule sums*[*of* $\mathrm{i} * of\_real\ (norm\ z + 1)\ /\ a$])
(*simp_all add*: *norm_mult a_def del*: *of_real_add*)

```
  have summable_fg': summable (λn. diffs f n * z^n) summable (λn. diffs g n *
z^n) for z
    by (intro termdiff_converges_all summable_f summable_g)+
  have (POWSER f has_field_derivative (POWSER' f z)) (at z)
             (POWSER g has_field_derivative (POWSER' g z)) (at z) for z
    unfolding POWSER_def POWSER'_def
    by (intro termdiffs_strong_converges_everywhere summable_f summable_g)+
  note derivs = this[THEN DERIV_chain2[OF _ DERIV_cmult[OF DERIV_ident]],
unfolded POWSER_def]
  have isCont (POWSER f) z isCont (POWSER g) z isCont (POWSER' f) z
isCont (POWSER' g) z
    for z unfolding POWSER_def POWSER'_def
   by (intro isCont_powser_converges_everywhere summable_f summable_g summable_fg')+
  note cont = this[THEN isCont_o2[rotated], unfolded POWSER_def POWSER'_def]

  {
    fix z :: complex assume z: abs (Re z) < 1
    define d where d = i * of_real (norm z + 1)
    have d: abs (Re d) < 1 norm z < norm d by (simp_all add: d_def norm_mult
del: of_real_add)
    have eventually (λz. h z = h2 z) (nhds z)
      using eventually_nhds_in_nhd[of z ?A] using h_eq z
      by (auto elim!: eventually_mono)

    moreover from sums(2)[OF z] z have nz: (∑ n. g n * (a * z) ^ n) ≠ 0
      unfolding G_def by (auto simp: sums_iff sin_eq_0 a_def)
    have A: z ∈ ℤ ⟷ z = 0 using z by (auto elim!: Ints_cases)
    have no_int: 1 + z ∈ ℤ ⟷ z = 0 using z Ints_diff[of 1+z 1] A
      by (auto elim!: nonpos_Ints_cases)
    have no_int': 1 - z ∈ ℤ ⟷ z = 0 using z Ints_diff[of 1 1-z] A
      by (auto elim!: nonpos_Ints_cases)
    from no_int no_int' have no_int: 1 - z ∉ ℤ≤0 1 + z ∉ ℤ≤0 by auto
    have (h2 has_field_derivative h2' z) (at z) unfolding h2_def
    by (rule DERIV_cong, (rule derivative_intros refl derivs[unfolded POWSER_def]
nz no_int)+)
         (auto simp: h2'_def POWSER_def field_simps power2_eq_square)
    ultimately have deriv: (h has_field_derivative h2' z) (at z)
      by (subst DERIV_cong_ev[OF refl _ refl])

    from sums(2)[OF z] z have (∑ n. g n * (a * z) ^ n) ≠ 0
      unfolding G_def by (auto simp: sums_iff a_def sin_eq_0)
   hence isCont h2' z using no_int unfolding h2'_def[abs_def] POWSER_def
POWSER'_def
      by (intro continuous_intros cont
            continuous_on_compose2[OF _ continuous_on_Polygamma[of {z. Re
z > 0}]]) auto
    note deriv and this
  } note A = this
```

```
  interpret h: periodic_fun_simple' h
  proof
    fix z :: complex
    show h (z + 1) = h z
    proof (cases z ∈ ℤ)
      assume z: z ∉ ℤ
      hence A: z + 1 ∉ ℤ z ≠ 0 using Ints_diff[of z+1 1] by auto
      hence Digamma (z + 1) - Digamma (-z) = Digamma z - Digamma (-z
+ 1)
        by (subst (1 2) Digamma_plus1) simp_all
      with A z show h (z + 1) = h z
        by (simp add: h_def sin_plus_pi cos_plus_pi ring_distribs cot_def)
    qed (simp add: h_def)
  qed

  have h2'_eq: h2' (z - 1) = h2' z if z: Re z > 0 Re z < 1 for z
  proof -
    have ((λz. h (z - 1)) has_field_derivative h2' (z - 1)) (at z)
      by (rule DERIV_cong, rule DERIV_chain'[OF _ A(1)])
         (insert z, auto intro!: derivative_eq_intros)
    hence (h has_field_derivative h2' (z - 1)) (at z) by (subst (asm) h.minus_1)
    moreover from z have (h has_field_derivative h2' z) (at z) by (intro A)
simp_all
    ultimately show h2' (z - 1) = h2' z by (rule DERIV_unique)
  qed

  define h2'' where h2'' z = h2' (z - of_int ⌊Re z⌋) for z
  have deriv: (h has_field_derivative h2'' z) (at z) for z
  proof -
    fix z :: complex
    have B: |Re z - real_of_int ⌊Re z⌋| < 1 by linarith
    have ((λt. h (t - of_int ⌊Re z⌋)) has_field_derivative h2'' z) (at z)
      unfolding h2''_def by (rule DERIV_cong, rule DERIV_chain'[OF _ A(1)])
                            (insert B, auto intro!: derivative_intros)
    thus (h has_field_derivative h2'' z) (at z) by (simp add: h.minus_of_int)
  qed

  have cont: continuous_on UNIV h2''
  proof (intro continuous_at_imp_continuous_on ballI)
    fix z :: complex
    define r where r = ⌊Re z⌋
    define A where A = {t. of_int r - 1 < Re t ∧ Re t < of_int r + 1}
    have continuous_on A (λt. h2' (t - of_int r)) unfolding A_def
      by (intro continuous_at_imp_continuous_on isCont_o2[OF _ A(2)] ballI
continuous_intros)
         (simp_all add: abs_real_def)
    moreover have h2'' t = h2' (t - of_int r) if t: t ∈ A for t
    proof (cases Re t ≥ of_int r)
      case True
```

```
    from t have of_int r − 1 < Re t Re t < of_int r + 1 by (simp_all add:
A_def)
    with True have ⌊Re t⌋ = ⌊Re z⌋ unfolding r_def by linarith
    thus ?thesis by (auto simp: r_def h2''_def)
   next
    case False
    from t have t: of_int r − 1 < Re t Re t < of_int r + 1 by (simp_all add:
A_def)
    with False have t': ⌊Re t⌋ = ⌊Re z⌋ − 1 unfolding r_def by linarith
    moreover from t False have h2' (t − of_int r + 1 − 1) = h2' (t − of_int
r + 1)
      by (intro h2'_eq) simp_all
    ultimately show ?thesis by (auto simp: r_def h2''_def algebra_simps t')
   qed
   ultimately have continuous_on A h2'' by (subst continuous_on_cong[OF
refl])
   moreover {
    have open ({t. of_int r − 1 < Re t} ∩ {t. of_int r + 1 > Re t})
      by (intro open_Int open_halfspace_Re_gt open_halfspace_Re_lt)
    also have {t. of_int r − 1 < Re t} ∩ {t. of_int r + 1 > Re t} = A
      unfolding A_def by blast
    finally have open A .
   }
   ultimately have C: isCont h2'' t if t ∈ A for t using that
    by (subst (asm) continuous_on_eq_continuous_at) auto
  have of_int r − 1 < Re z Re z < of_int r + 1 unfolding r_def by linarith+
   thus isCont h2'' z by (intro C) (simp_all add: A_def)
 qed

 from that[OF cont deriv] show ?thesis .
qed

lemma Gamma_reflection_complex:
 fixes z :: complex
 shows Gamma z * Gamma (1 − z) = of_real pi / sin (of_real pi * z)
proof −
 let ?g = λz::complex. Gamma z * Gamma (1 − z) * sin (of_real pi * z)
 define g where [abs_def]: g z = (if z ∈ ℤ then of_real pi else ?g z) for z ::
complex
 let ?h = λz::complex. (of_real pi * cot (of_real pi*z) + Digamma z − Digamma
(1 − z))
 define h where [abs_def]: h z = (if z ∈ ℤ then 0 else ?h z) for z :: complex

 — g is periodic with period 1.
 interpret g: periodic_fun_simple' g
 proof
  fix z :: complex
  show g (z + 1) = g z
  proof (cases z ∈ ℤ)
```

```
    case False
    hence z * g z = z * Beta z (- z + 1) * sin (of_real pi * z) by (simp add:
g_def Beta_def)
    also have z * Beta z (- z + 1) = (z + 1 + -z) * Beta (z + 1) (- z + 1)
      using False Ints_diff[of 1 1 - z] nonpos_Ints_subset_Ints
      by (subst Beta_plus1_left [symmetric]) auto
    also have ... * sin (of_real pi * z) = z * (Beta (z + 1) (-z) * sin (of_real
pi * (z + 1)))
     using False Ints_diff[of z+1 1] Ints_minus[of -z] nonpos_Ints_subset_Ints
      by (subst Beta_plus1_right) (auto simp: ring_distribs sin_plus_pi)
    also from False have Beta (z + 1) (-z) * sin (of_real pi * (z + 1)) = g (z
+ 1)
      using Ints_diff[of z+1 1] by (auto simp: g_def Beta_def)
    finally show g (z + 1) = g z using False by (subst (asm) mult_left_cancel)
auto
   qed (simp add: g_def)
 qed

 — g is entire.
 have g_g': (g has_field_derivative (h z * g z)) (at z) for z :: complex
 proof (cases z ∈ ℤ)
   let ?h' = λz. Beta z (1 - z) * ((Digamma z - Digamma (1 - z)) * sin (z *
of_real pi) +
                   of_real pi * cos (z * of_real pi))
   case False
   from False have eventually (λt. t ∈ UNIV - ℤ) (nhds z)
     by (intro eventually_nhds_in_open) (auto simp: open_Diff)
    hence eventually (λt. g t = ?g t) (nhds z) by eventually_elim (simp add:
g_def)
   moreover {
     from False Ints_diff[of 1 1-z] have 1 - z ∉ ℤ by auto
    hence (?g has_field_derivative ?h' z) (at z) using nonpos_Ints_subset_Ints
      by (auto intro!: derivative_eq_intros simp: algebra_simps Beta_def)
     also from False have sin (of_real pi * z) ≠ 0 by (subst sin_eq_0) auto
     hence ?h' z = h z * g z
       using False unfolding g_def h_def cot_def by (simp add: field_simps
Beta_def)
     finally have (?g has_field_derivative (h z * g z)) (at z) .
   }
   ultimately show ?thesis by (subst DERIV_cong_ev[OF refl _ refl])
 next
   case True
   then obtain n where z: z = of_int n by (auto elim!: Ints_cases)
   let ?t = (λz::complex. if z = 0 then 1 else sin z / z) ∘ (λz. of_real pi * z)
   have deriv_0: (g has_field_derivative 0) (at 0)
   proof (subst DERIV_cong_ev[OF refl _ refl])
     show eventually (λz. g z = of_real pi * Gamma (1 + z) * Gamma (1 - z)
* ?t z) (nhds 0)
       using eventually_nhds_ball[OF zero_less_one, of 0::complex]
```

```
      proof eventually_elim
        fix z :: complex assume z: z ∈ ball 0 1
        show g z = of_real pi * Gamma (1 + z) * Gamma (1 - z) * ?t z
        proof (cases z = 0)
          assume z': z ≠ 0
          with z have z'': z ∉ ℤ≤0 z ∉ ℤ by (auto elim!: Ints_cases)
          from Gamma_plus1[OF this(1)] have Gamma z = Gamma (z + 1) / z
by simp
          with z'' z' show ?thesis by (simp add: g_def ac_simps)
        qed (simp add: g_def)
      qed
      have (?t has_field_derivative (0 * of_real pi)) (at 0)
        using has_field_derivative_sin_z_over_z[of UNIV :: complex set]
        by (intro DERIV_chain) simp_all
    thus ((λz. of_real pi * Gamma (1 + z) * Gamma (1 - z) * ?t z) has_field_derivative
0) (at 0)
        by (auto intro!: derivative_eq_intros simp: o_def)
    qed

    have ((g ∘ (λx. x - of_int n)) has_field_derivative 0 * 1) (at (of_int n))
      using deriv_0 by (intro DERIV_chain) (auto intro!: derivative_eq_intros)
   also have g ∘ (λx. x - of_int n) = g by (intro ext) (simp add: g.minus_of_int)
   finally show (g has_field_derivative (h z * g z)) (at z) by (simp add: z h_def)
  qed

  have g_eq: g (z/2) * g ((z+1)/2) = Gamma (1/2)^2 * g z if Re z > -1 Re z
< 2 for z
  proof (cases z ∈ ℤ)
    case True
    with that have z = 0 ∨ z = 1 by (force elim!: Ints_cases)
    moreover have g 0 * g (1/2) = Gamma (1/2)^2 * g 0
     using fraction_not_in_ints[where 'a = complex, of 2 1] by (simp add: g_def
power2_eq_square)
    moreover have g (1/2) * g 1 = Gamma (1/2)^2 * g 1
      using fraction_not_in_ints[where 'a = complex, of 2 1]
      by (simp add: g_def power2_eq_square Beta_def algebra_simps)
    ultimately show ?thesis by force
  next
    case False
     hence z: z/2 ∉ ℤ (z+1)/2 ∉ ℤ using Ints_diff[of z+1 1] by (auto elim!:
Ints_cases)
    hence z': z/2 ∉ ℤ≤0 (z+1)/2 ∉ ℤ≤0 by (auto elim!: nonpos_Ints_cases)
    from z have 1-z/2 ∉ ℤ 1-((z+1)/2) ∉ ℤ
      using Ints_diff[of 1 1-z/2] Ints_diff[of 1 1-((z+1)/2)] by auto
   hence z'': 1-z/2 ∉ ℤ≤0 1-((z+1)/2) ∉ ℤ≤0 by (auto elim!: nonpos_Ints_cases)
    from z have g (z/2) * g ((z+1)/2) =
    (Gamma (z/2) * Gamma ((z+1)/2)) * (Gamma (1-z/2) * Gamma (1-((z+1)/2)))
*
      (sin (of_real pi * z/2) * sin (of_real pi * (z+1)/2))
```

```
    by (simp add: g_def)
  also from z' Gamma_legendre_duplication_aux[of z/2]
    have Gamma (z/2) * Gamma ((z+1)/2) = exp ((1-z) * of_real (ln 2)) *
Gamma (1/2) * Gamma z
    by (simp add: add_divide_distrib)
  also from z'' Gamma_legendre_duplication_aux[of 1-(z+1)/2]
    have Gamma (1-z/2) * Gamma (1-(z+1)/2) =
          Gamma (1-z) * Gamma (1/2) * exp (z * of_real (ln 2))
    by (simp add: add_divide_distrib ac_simps)
  finally have g (z/2) * g ((z+1)/2) = Gamma (1/2)^2 * (Gamma z * Gamma
(1-z) *
                (2 * (sin (of_real pi*z/2) * sin (of_real pi*(z+1)/2))))
    by (simp add: add_ac power2_eq_square exp_add ring_distribs exp_diff
exp_of_real)
  also have sin (of_real pi*(z+1)/2) = cos (of_real pi*z/2)
    using cos_sin_eq[of - of_real pi * z/2, symmetric]
    by (simp add: ring_distribs add_divide_distrib ac_simps)
  also have 2 * (sin (of_real pi*z/2) * cos (of_real pi*z/2)) = sin (of_real pi
* z)
    by (subst sin_times_cos) (simp add: field_simps)
  also have Gamma z * Gamma (1 - z) * sin (complex_of_real pi * z) = g z
    using ‹z ∉ ℤ› by (simp add: g_def)
  finally show ?thesis .
 qed
 have g_eq: g (z/2) * g ((z+1)/2) = Gamma (1/2)^2 * g z for z
 proof -
  define r where r = ⌊Re z / 2⌋
  have Gamma (1/2)^2 * g z = Gamma (1/2)^2 * g (z - of_int (2*r)) by
(simp only: g.minus_of_int)
  also have of_int (2*r) = 2 * of_int r by simp
  also have Re z - 2 * of_int r > -1 Re z - 2 * of_int r < 2 unfolding
r_def by linarith+
  hence Gamma (1/2)^2 * g (z - 2 * of_int r) =
            g ((z - 2 * of_int r)/2) * g ((z - 2 * of_int r + 1)/2)
    unfolding r_def by (intro g_eq[symmetric]) simp_all
  also have (z - 2 * of_int r) / 2 = z/2 - of_int r by simp
  also have g ... = g (z/2) by (rule g.minus_of_int)
  also have (z - 2 * of_int r + 1) / 2 = (z + 1)/2 - of_int r by simp
  also have g ... = g ((z+1)/2) by (rule g.minus_of_int)
  finally show ?thesis ..
 qed

 have g_nz [simp]: g z ≠ 0 for z :: complex
 unfolding g_def using Ints_diff[of 1 1 - z]
  by (auto simp: Gamma_eq_zero_iff sin_eq_0 dest!: nonpos_Ints_Int)

 have h_eq: h z = (h (z/2) + h ((z+1)/2)) / 2 for z
 proof -
  have ((λt. g (t/2) * g ((t+1)/2)) has_field_derivative
```

```
                (g (z/2) * g ((z+1)/2)) * ((h (z/2) + h ((z+1)/2)) / 2)) (at z)
      by (auto intro!: derivative_eq_intros g_g'[THEN DERIV_chain2] simp:
field_simps)
    hence ((λt. Gamma (1/2)^2 * g t) has_field_derivative
          Gamma (1/2)^2 * g z * ((h (z/2) + h ((z+1)/2)) / 2)) (at z)
      by (subst (1 2) g_eq[symmetric]) simp
    from DERIV_cmult[OF this, of inverse ((Gamma (1/2))^2)]
      have (g has_field_derivative (g z * ((h (z/2) + h ((z+1)/2))/2))) (at z)
      using fraction_not_in_ints[where 'a = complex, of 2 1]
    by (simp add: divide_simps Gamma_eq_zero_iff not_in_Ints_imp_not_in_nonpos_Ints)
    moreover have (g has_field_derivative (g z * h z)) (at z)
      using g_g'[of z] by (simp add: ac_simps)
    ultimately have g z * h z = g z * ((h (z/2) + h ((z+1)/2))/2)
      by (intro DERIV_unique)
    thus h z = (h (z/2) + h ((z+1)/2)) / 2 by simp
  qed

  obtain h' where h'_cont: continuous_on UNIV h' and
              h_h': ⋀z. (h has_field_derivative h' z) (at z)
    unfolding h_def by (erule Gamma_reflection_aux)

  have h'_eq: h' z = (h' (z/2) + h' ((z+1)/2)) / 4 for z
  proof -
    have ((λt. (h (t/2) + h ((t+1)/2)) / 2) has_field_derivative
                ((h' (z/2) + h' ((z+1)/2)) / 4)) (at z)
      by (fastforce intro!: derivative_eq_intros h_h'[THEN DERIV_chain2])
    hence (h has_field_derivative ((h' (z/2) + h' ((z+1)/2))/4)) (at z)
      by (subst (asm) h_eq[symmetric])
    from h_h' and this show h' z = (h' (z/2) + h' ((z+1)/2)) / 4 by (rule
DERIV_unique)
  qed

  have h'_zero: h' z = 0 for z
  proof -
    define m where m = max 1 |Re z|
    define B where B = {t. abs (Re t) ≤ m ∧ abs (Im t) ≤ abs (Im z)}
    have closed ({t. Re t ≥ -m} ∩ {t. Re t ≤ m} ∩
              {t. Im t ≥ -|Im z|} ∩ {t. Im t ≤ |Im z|})
     (is closed ?B) by (intro closed_Int closed_halfspace_Re_ge closed_halfspace_Re_le
                        closed_halfspace_Im_ge closed_halfspace_Im_le)
    also have ?B = B unfolding B_def by fastforce
    finally have closed B .
    moreover have bounded B unfolding bounded_iff
    proof (intro ballI exI)
      fix t assume t: t ∈ B
      have norm t ≤ |Re t| + |Im t| by (rule cmod_le)
      also from t have |Re t| ≤ m unfolding B_def by blast
      also from t have |Im t| ≤ |Im z| unfolding B_def by blast
      finally show norm t ≤ m + |Im z| by - simp
```

```
  qed
  ultimately have compact: compact B by (subst compact_eq_bounded_closed)
blast

  define M where M = (SUP z∈B. norm (h' z))
  have compact (h' ‘ B)
    by (intro compact_continuous_image continuous_on_subset[OF h'_cont]
compact) blast+
  hence bdd: bdd_above ((λz. norm (h' z)) ‘ B)
    using bdd_above_norm[of h' ‘ B] by (simp add: image_comp o_def com-
pact_imp_bounded)
  have norm (h' z) ≤ M unfolding M_def by (intro cSUP_upper bdd) (simp_all
add: B_def m_def)
  also have M ≤ M/2
  proof (subst M_def, subst cSUP_le_iff)
    have z ∈ B unfolding B_def m_def by simp
    thus B ≠ {} by auto
  next
    show ∀ z∈B. norm (h' z) ≤ M/2
    proof
      fix t :: complex assume t: t ∈ B
      from h'_eq[of t] t have h' t = (h' (t/2) + h' ((t+1)/2)) / 4 by (simp)
      also have norm ... = norm (h' (t/2) + h' ((t+1)/2)) / 4 by simp
      also have norm (h' (t/2) + h' ((t+1)/2)) ≤ norm (h' (t/2)) + norm (h'
((t+1)/2))
        by (rule norm_triangle_ineq)
      also from t have abs (Re ((t + 1)/2)) ≤ m unfolding m_def B_def by
auto
      with t have t/2 ∈ B (t+1)/2 ∈ B unfolding B_def by auto
     hence norm (h' (t/2)) + norm (h' ((t+1)/2)) ≤ M + M unfolding M_def
        by (intro add_mono cSUP_upper bdd) (auto simp: B_def)
      also have (M + M) / 4 = M / 2 by simp
      finally show norm (h' t) ≤ M/2 by − simp_all
    qed
  qed (insert bdd, auto)
  hence M ≤ 0 by simp
  finally show h' z = 0 by simp
 qed
 have h_h'_2: (h has_field_derivative 0) (at z) for z
  using h_h'[of z] h'_zero[of z] by simp

 have g_real: g z ∈ ℝ if z ∈ ℝ for z
  unfolding g_def using that by (auto intro!: Reals_mult Gamma_complex_real)
 have h_real: h z ∈ ℝ if z ∈ ℝ for z
  unfolding h_def using that by (auto intro!: Reals_mult Reals_add Reals_diff
Polygamma_Real)
 have g_nz: g z ≠ 0 for z unfolding g_def using Ints_diff[of 1 1−z]
  by (auto simp: Gamma_eq_zero_iff sin_eq_0)
```

```
  from h'_zero h_h'_2 have ∃ c. ∀ z∈UNIV. h z = c
    by (intro has_field_derivative_zero_constant) (simp_all add: dist_0_norm)
  then obtain c where c: ⋀z. h z = c by auto
  have ∃ u. u ∈ closed_segment 0 1 ∧ Re (g 1) − Re (g 0) = Re (h u ∗ g u ∗ (1
− 0))
    by (intro complex_mvt_line g_g')
  then obtain u where u: u ∈ closed_segment 0 1 Re (g 1) − Re (g 0) = Re (h
u ∗ g u)
    by auto
  from u(1) have u': u ∈ ℝ unfolding closed_segment_def
    by (auto simp: scaleR_conv_of_real)
  from u' g_real[of u] g_nz[of u] have Re (g u) ≠ 0 by (auto elim!: Reals_cases)
  with u(2) c[of u] g_real[of u] g_nz[of u] u'
    have Re c = 0 by (simp add: complex_is_Real_iff g.of_1)
  with h_real[of 0] c[of 0] have c = 0 by (auto elim!: Reals_cases)
  with c have A: h z ∗ g z = 0 for z by simp
  hence (g has_field_derivative 0) (at z) for z using g_g'[of z] by simp
  hence ∃ c'. ∀ z∈UNIV. g z = c' by (intro has_field_derivative_zero_constant)
simp_all
  then obtain c' where c: ⋀z. g z = c' by (force)
  from this[of 0] have c' = pi unfolding g_def by simp
  with c have g z = pi by simp

  show ?thesis
  proof (cases z ∈ ℤ)
    case False
    with ‹g z = pi› show ?thesis by (auto simp: g_def divide_simps)
  next
    case True
    then obtain n where n: z = of_int n by (elim Ints_cases)
    with sin_eq_0[of of_real pi ∗ z] have sin (of_real pi ∗ z) = 0 by force
     moreover have of_int (1 − n) ∈ ℤ≤0 if n > 0 using that by (intro non-
pos_Ints_of_int) simp
    ultimately show ?thesis using n
      by (cases n ≤ 0) (auto simp: Gamma_eq_zero_iff nonpos_Ints_of_int)
  qed
qed

lemma rGamma_reflection_complex:
  rGamma z ∗ rGamma (1 − z :: complex) = sin (of_real pi ∗ z) / of_real pi
  using Gamma_reflection_complex[of z]
    by (simp add: Gamma_def field_split_simps split: if_split_asm)

lemma rGamma_reflection_complex':
  rGamma z ∗ rGamma (− z :: complex) = −z ∗ sin (of_real pi ∗ z) / of_real pi
proof −
  have rGamma z ∗ rGamma (−z) = −z ∗ (rGamma z ∗ rGamma (1 − z))
    using rGamma_plus1[of −z, symmetric] by simp
  also have rGamma z ∗ rGamma (1 − z) = sin (of_real pi ∗ z) / of_real pi
```

by (*rule rGamma_reflection_complex*)
finally show *?thesis* **by** *simp*
qed

lemma *Gamma_reflection_complex'*:
*Gamma z * Gamma (− z :: complex) = − of_real pi / (z * sin (of_real pi * z))*
using *rGamma_reflection_complex'*[*of z*] **by** (*force simp add*: *Gamma_def field_split_simps*)

lemma *Gamma_one_half_real*: *Gamma (1/2 :: real) = sqrt pi*
proof −
from *Gamma_reflection_complex*[*of 1/2*] *fraction_not_in_ints*[**where** *'a = complex*, *of 2 1*]
have *Gamma (1/2 :: complex)^2 = of_real pi* **by** (*simp add*: *power2_eq_square*)
hence *of_real pi = Gamma (complex_of_real (1/2))^2* **by** *simp*
also have *... = of_real ((Gamma (1/2))^2)* **by** (*subst Gamma_complex_of_real*) *simp_all*
finally have *Gamma (1/2)^2 = pi* **by** (*subst* (*asm*) *of_real_eq_iff*) *simp_all*
moreover have *Gamma (1/2 :: real) ≥ 0* **using** *Gamma_real_pos*[*of 1/2*] **by** *simp*
ultimately show *?thesis* **by** (*rule real_sqrt_unique* [*symmetric*])
qed

lemma *Gamma_one_half_complex*: *Gamma (1/2 :: complex) = of_real (sqrt pi)*
proof −
have *Gamma (1/2 :: complex) = Gamma (of_real (1/2))* **by** *simp*
also have *... = of_real (sqrt pi)* **by** (*simp only*: *Gamma_complex_of_real Gamma_one_half_real*)
finally show *?thesis* **.**
qed

theorem *Gamma_legendre_duplication*:
fixes *z* :: *complex*
assumes *z ∉ ℤ≤0 z + 1/2 ∉ ℤ≤0*
shows *Gamma z * Gamma (z + 1/2) =*
*exp ((1 − 2*z) * of_real (ln 2)) * of_real (sqrt pi) * Gamma (2*z)*
using *Gamma_legendre_duplication_aux*[*OF assms*] **by** (*simp add*: *Gamma_one_half_complex*)

end

6.20.10 Limits and residues

The inverse of the Gamma function has simple zeros:

lemma *rGamma_zeros*:
*(λz. rGamma z / (z + of_nat n)) − (− of_nat n) → ((−1)^n * fact n :: 'a :: Gamma)*
proof (*subst tendsto_cong*)
let *?f = λz. pochhammer z n * rGamma (z + of_nat (Suc n)) :: 'a*

```
  from eventually_at_ball'[OF zero_less_one, of − of_nat n :: 'a UNIV]
    show eventually (λz. rGamma z / (z + of_nat n) = ?f z) (at (− of_nat n))
    by (subst pochhammer_rGamma[of _ Suc n])
       (auto elim!: eventually_mono simp: field_split_simps pochhammer_rec'
eq_neg_iff_add_eq_0)
  have isCont ?f (− of_nat n) by (intro continuous_intros)
  thus ?f − (− of_nat n) → (− 1) ^ n * fact n unfolding isCont_def
    by (simp add: pochhammer_same)
qed
```

The simple zeros of the inverse of the Gamma function correspond to simple poles of the Gamma function, and their residues can easily be computed from the limit we have just proven:

```
lemma Gamma_poles: filterlim Gamma at_infinity (at (− of_nat n :: 'a :: Gamma))
proof −
  from eventually_at_ball'[OF zero_less_one, of − of_nat n :: 'a UNIV]
    have eventually (λz. rGamma z ≠ (0 :: 'a)) (at (− of_nat n))
    by (auto elim!: eventually_mono nonpos_Ints_cases'
          simp: rGamma_eq_zero_iff dist_of_nat dist_minus)
  with isCont_rGamma[of − of_nat n :: 'a, OF continuous_ident]
    have filterlim (λz. inverse (rGamma z) :: 'a) at_infinity (at (− of_nat n))
   unfolding isCont_def by (intro filterlim_compose[OF filterlim_inverse_at_infinity])
                           (simp_all add: filterlim_at)
  moreover have (λz. inverse (rGamma z) :: 'a) = Gamma
    by (intro ext) (simp add: rGamma_inverse_Gamma)
  ultimately show ?thesis by (simp only: )
qed

lemma Gamma_residues:
  (λz. Gamma z * (z + of_nat n)) − (− of_nat n) → ((−1) ^ n / fact n :: 'a ::
Gamma)
proof (subst tendsto_cong)
  let ?c = (− 1) ^ n / fact n :: 'a
  from eventually_at_ball'[OF zero_less_one, of − of_nat n :: 'a UNIV]
    show eventually (λz. Gamma z * (z + of_nat n) = inverse (rGamma z / (z
+ of_nat n)))
         (at (− of_nat n))
   by (auto elim!: eventually_mono simp: field_split_simps rGamma_inverse_Gamma)
  have (λz. inverse (rGamma z / (z + of_nat n))) − (− of_nat n) →
        inverse ((− 1) ^ n * fact n :: 'a)
    by (intro tendsto_intros rGamma_zeros) simp_all
  also have inverse ((− 1) ^ n * fact n) = ?c
    by (simp_all add: field_simps flip: power_mult_distrib)
  finally show (λz. inverse (rGamma z / (z + of_nat n))) − (− of_nat n) → ?c
.
qed
```

6.20.11 Alternative definitions

Variant of the Euler form

definition *Gamma_series_euler'* **where**
Gamma_series_euler' z n =
inverse z ∗ (∏ *k*=*1..n. exp* (*z* ∗ *of_real* (*ln* (*1* + *inverse* (*of_nat k*)))) / (*1* + *z* / *of_nat k*))

context
begin
private lemma *Gamma_euler'_aux1*:
fixes *z* :: *'a* :: {*real_normed_field*,*banach*}
assumes *n*: *n* > *0*
shows *exp* (*z* ∗ *of_real* (*ln* (*of_nat n* + *1*))) = (∏ *k*=*1..n. exp* (*z* ∗ *of_real* (*ln* (*1* + *1* / *of_nat k*))))
proof −
have (∏ *k*=*1..n. exp* (*z* ∗ *of_real* (*ln* (*1* + *1* / *of_nat k*)))) =
exp (*z* ∗ *of_real* (∑ *k* = *1..n. ln* (*1* + *1* / *real_of_nat k*)))
by (*subst exp_sum* [*symmetric*]) (*simp_all add*: *sum_distrib_left*)
also have (∑ *k*=*1..n. ln* (*1* + *1* / *of_nat k*) :: *real*) = *ln* (∏ *k*=*1..n. 1* + *1* / *real_of_nat k*)
by (*subst ln_prod* [*symmetric*]) (*auto intro*!: *add_pos_nonneg*)
also have (∏ *k*=*1..n. 1* + *1* / *of_nat k* :: *real*) = (∏ *k*=*1..n.* (*of_nat k* + *1*) / *of_nat k*)
by (*intro prod.cong*) (*simp_all add*: *field_split_simps*)
also have (∏ *k*=*1..n.* (*of_nat k* + *1*) / *of_nat k* :: *real*) = *of_nat n* + *1*
by (*induction n*) (*simp_all add*: *prod.nat_ivl_Suc' field_split_simps*)
finally show *?thesis* **..**
qed

theorem *Gamma_series_euler'*:
assumes *z*: (*z* :: *'a* :: *Gamma*) ∉ $\mathbb{Z}_{\leq 0}$
shows (λ*n. Gamma_series_euler' z n*) ⟶ *Gamma z*
proof (*rule Gamma_seriesI*, *rule Lim_transform_eventually*)
let *?f* = λ*n. fact n* ∗ *exp* (*z* ∗ *of_real* (*ln* (*of_nat n* + *1*))) / *pochhammer z* (*n* + *1*)
let *?r* = λ*n. ?f n* / *Gamma_series z n*
let *?r'* = λ*n. exp* (*z* ∗ *of_real* (*ln* (*of_nat* (*Suc n*) / *of_nat n*)))
from *z* **have** *z'*: *z* ≠ *0* **by** *auto*

have *eventually* (λ*n. ?r' n* = *?r n*) *sequentially*
using *z* **by** (*auto simp*: *field_split_simps Gamma_series_def ring_distribs exp_diff ln_div*
intro: *eventually_mono eventually_gt_at_top*[*of 0*::*nat*] *dest*: *pochhammer_eq_0_imp_nonpos_Int*)
moreover have *?r'* ⟶ *exp* (*z* ∗ *of_real* (*ln 1*))
by (*intro tendsto_intros LIMSEQ_Suc_n_over_n*) *simp_all*
ultimately show *?r* ⟶ *1* **by** (*force intro*: *Lim_transform_eventually*)

```
  from eventually_gt_at_top[of 0::nat]
    show eventually (λn. ?r n = Gamma_series_euler' z n / Gamma_series z n)
sequentially
  proof eventually_elim
    fix n :: nat assume n: n > 0
    from n z' have Gamma_series_euler' z n =
      exp (z * of_real (ln (of_nat n + 1))) / (z * (∏k=1..n. (1 + z / of_nat k)))
      by (subst Gamma_euler'_aux1)
         (simp_all add: Gamma_series_euler'_def prod.distrib
                        prod_inversef[symmetric] divide_inverse)
    also have (∏k=1..n. (1 + z / of_nat k)) = pochhammer (z + 1) n / fact n
    proof (cases n)
      case (Suc n')
      then show ?thesis
        unfolding pochhammer_prod fact_prod
       by (simp add:  atLeastLessThanSuc_atLeastAtMost field_simps prod_dividef

                  prod.atLeast_Suc_atMost_Suc_shift del: prod.cl_ivl_Suc)
    qed auto
    also have z * ... = pochhammer z (Suc n) / fact n by (simp add: pochham-
mer_rec)
    finally show ?r n = Gamma_series_euler' z n / Gamma_series z n by simp
  qed
qed

end
```

Weierstrass form

```
definition Gamma_series_Weierstrass :: 'a :: {banach,real_normed_field} ⇒ nat
⇒ 'a where
  Gamma_series_Weierstrass z n =
     exp (−euler_mascheroni * z) / z * (∏k=1..n. exp (z / of_nat k) / (1 + z /
of_nat k))

definition
  rGamma_series_Weierstrass :: 'a :: {banach,real_normed_field} ⇒ nat ⇒ 'a
where
  rGamma_series_Weierstrass z n =
     exp (euler_mascheroni * z) * z * (∏k=1..n. (1 + z / of_nat k) * exp (−z /
of_nat k))

lemma Gamma_series_Weierstrass_nonpos_Ints:
  eventually (λk. Gamma_series_Weierstrass (− of_nat n) k = 0) sequentially
 using eventually_ge_at_top[of n] by eventually_elim (auto simp: Gamma_series_Weierstrass_def)

lemma rGamma_series_Weierstrass_nonpos_Ints:
  eventually (λk. rGamma_series_Weierstrass (− of_nat n) k = 0) sequentially
 using eventually_ge_at_top[of n] by eventually_elim (auto simp: rGamma_series_Weierstrass_def)
```

theorem *Gamma_Weierstrass_complex*: *Gamma_series_Weierstrass z* ⟶ *Gamma* (*z* :: *complex*)
proof (*cases z* ∈ $\mathbb{Z}_{\leq 0}$)
case *True*
then obtain *n* **where** *z* = − *of_nat n* **by** (*elim nonpos_Ints_cases'*)
also from *True* **have** *Gamma_series_Weierstrass* ... ⟶ *Gamma z*
by (*simp add*: *tendsto_cong*[*OF Gamma_series_Weierstrass_nonpos_Ints*] *Gamma_nonpos_Int*)
finally show *?thesis* **.**
next
case *False*
hence *z*: *z* ≠ *0* **by** *auto*
let *?f* = (λ*x*. ∏ *x* = *Suc 0..x*. *exp* (*z* / *of_nat x*) / (*1* + *z* / *of_nat x*))
have *A*: *exp* (*ln* (*1* + *z* / *of_nat n*)) = (*1* + *z* / *of_nat n*) **if** *n* ≥ *1* **for** *n* :: *nat*
using *False that* **by** (*subst exp_Ln*) (*auto simp*: *field_simps dest*!: *plus_of_nat_eq_0_imp*)
have (λ*n*. ∑ *k*=*1..n*. *z* / *of_nat k* − *ln* (*1* + *z* / *of_nat k*)) ⟶ *ln_Gamma z* + *euler_mascheroni* ∗ *z* + *ln z*
using *ln_Gamma_series'_aux*[*OF False*]
by (*simp only*: *atLeastLessThanSuc_atLeastAtMost* [*symmetric*] *One_nat_def* *sum.shift_bounds_Suc_ivl sums_def atLeast0LessThan*)
from *tendsto_exp*[*OF this*] *False z* **have** *?f* ⟶ *z* ∗ *exp* (*euler_mascheroni* ∗ *z*) ∗ *Gamma z*
by (*simp add*: *exp_add exp_sum exp_diff mult_ac Gamma_complex_altdef A*)
from *tendsto_mult*[*OF tendsto_const*[*of exp* (−*euler_mascheroni* ∗ *z*) / *z*] *this*] *z*
show *Gamma_series_Weierstrass z* ⟶ *Gamma z*
by (*simp add*: *exp_minus field_split_simps Gamma_series_Weierstrass_def* [*abs_def*])
qed

lemma *tendsto_complex_of_real_iff*: ((λ*x*. *complex_of_real* (*f x*)) ⟶ *of_real c*) *F* = (*f* ⟶ *c*) *F*
by (*rule tendsto_of_real_iff*)

lemma *Gamma_Weierstrass_real*: *Gamma_series_Weierstrass x* ⟶ *Gamma* (*x* :: *real*)
using *Gamma_Weierstrass_complex*[*of of_real x*] **unfolding** *Gamma_series_Weierstrass_def*[*abs_de*
by (*subst tendsto_complex_of_real_iff* [*symmetric*])
(*simp_all add*: *exp_of_real*[*symmetric*] *Gamma_complex_of_real*)

lemma *rGamma_Weierstrass_complex*: *rGamma_series_Weierstrass z* ⟶ *rGamma* (*z* :: *complex*)
proof (*cases z* ∈ $\mathbb{Z}_{\leq 0}$)
case *True*
then obtain *n* **where** *z* = − *of_nat n* **by** (*elim nonpos_Ints_cases'*)
also from *True* **have** *rGamma_series_Weierstrass* ... ⟶ *rGamma z*
by (*simp add*: *tendsto_cong*[*OF rGamma_series_Weierstrass_nonpos_Ints*] *rGamma_nonpos_Int*)

```
  finally show ?thesis .
next
  case False
 have rGamma_series_Weierstrass z = (λn. inverse (Gamma_series_Weierstrass
z n))
   by (simp add: rGamma_series_Weierstrass_def[abs_def] Gamma_series_Weierstrass_def
                 exp_minus divide_inverse prod_inversef[symmetric] mult_ac)
  also from False have ... ⟶ inverse (Gamma z)
   by (intro tendsto_intros Gamma_Weierstrass_complex) (simp add: Gamma_eq_zero_iff)
  finally show ?thesis by (simp add: Gamma_def)
qed
```

Binomial coefficient form

```
lemma Gamma_gbinomial:
  (λn. ((z + of_nat n) gchoose n) * exp (-z * of_real (ln (of_nat n)))) ⟶
rGamma (z+1)
proof (cases z = 0)
  case False
  show ?thesis
  proof (rule Lim_transform_eventually)
    let ?powr = λa b. exp (b * of_real (ln (of_nat a)))
    show eventually (λn. rGamma_series z n / z =
          ((z + of_nat n) gchoose n) * ?powr n (-z)) sequentially
    proof (intro always_eventually allI)
      fix n :: nat
      from False have ((z + of_nat n) gchoose n) = pochhammer z (Suc n) / z /
fact n
        by (simp add: gbinomial_pochhammer' pochhammer_rec)
     also have pochhammer z (Suc n) / z / fact n * ?powr n (-z) = rGamma_series
z n / z
        by (simp add: rGamma_series_def field_split_simps exp_minus)
      finally show rGamma_series z n / z = ((z + of_nat n) gchoose n) * ?powr
n (-z) ..
    qed

   from False have (λn. rGamma_series z n / z) ⟶ rGamma z / z by (intro
tendsto_intros)
   also from False have rGamma z / z = rGamma (z + 1) using rGamma_plus1[of
z]
     by (simp add: field_simps)
    finally show (λn. rGamma_series z n / z) ⟶ rGamma (z+1) .
  qed
qed (simp_all add: binomial_gbinomial [symmetric])

lemma gbinomial_minus': (a + of_nat b) gchoose b = (- 1) ^ b * (- (a + 1)
gchoose b)
  by (subst gbinomial_minus) (simp add: power_mult_distrib [symmetric])
```

lemma *gbinomial_asymptotic*:
fixes $z :: 'a :: Gamma$
shows $(\lambda n.\ (z\ gchoose\ n)\ /\ ((-1)\hat{}n\ /\ exp\ ((z+1) * of\_real\ (ln\ (real\ n)))))$
$\longrightarrow$
$inverse\ (Gamma\ (-\ z))$
unfolding *rGamma_inverse_Gamma* [*symmetric*] **using** *Gamma_gbinomial*[*of* $-z-1$]
by (*subst* (*asm*) *gbinomial_minus'*)
(*simp add*: *add_ac mult_ac divide_inverse power_inverse* [*symmetric*])

lemma *fact_binomial_limit*:
$(\lambda n.\ of\_nat\ ((k + n)\ choose\ n)\ /\ of\_nat\ (n\ \hat{}\ k) :: 'a :: Gamma) \longrightarrow 1\ /\ fact\ k$
proof (*rule Lim_transform_eventually*)
have $(\lambda n.\ of\_nat\ ((k + n)\ choose\ n)\ /\ of\_real\ (exp\ (of\_nat\ k * ln\ (real\_of\_nat\ n))))$
$\longrightarrow 1\ /\ Gamma\ (of\_nat\ (Suc\ k) :: 'a)$ (**is** *?f* $\longrightarrow$ _)
using *Gamma_gbinomial*[*of of_nat k* :: $'a$]
by (*simp add*: *binomial_gbinomial Gamma_def field_split_simps exp_of_real* [*symmetric*] *exp_minus*)
also have $Gamma\ (of\_nat\ (Suc\ k)) = fact\ k$ **by** (*simp add*: *Gamma_fact*)
finally show *?f* $\longrightarrow 1\ /\ fact\ k$ **.**

show *eventually* $(\lambda n.\ ?f\ n = of\_nat\ ((k + n)\ choose\ n)\ /\ of\_nat\ (n\ \hat{}\ k))$ *sequentially*
using *eventually_gt_at_top*[*of* 0::*nat*]
proof *eventually_elim*
fix $n :: nat$ **assume** n: $n > 0$
from n **have** $exp\ (real\_of\_nat\ k * ln\ (real\_of\_nat\ n)) = real\_of\_nat\ (n\hat{}k)$
by (*simp add*: *exp_of_nat_mult*)
thus $?f\ n = of\_nat\ ((k + n)\ choose\ n)\ /\ of\_nat\ (n\ \hat{}\ k)$ **by** *simp*
qed
qed

lemma *binomial_asymptotic'*:
$(\lambda n.\ of\_nat\ ((k + n)\ choose\ n)\ /\ (of\_nat\ (n\ \hat{}\ k)\ /\ fact\ k) :: 'a :: Gamma) \longrightarrow 1$
using *tendsto_mult*[*OF fact_binomial_limit*[*of k*] *tendsto_const*[*of fact k* :: $'a$]]
by *simp*

lemma *gbinomial_Beta*:
assumes $z + 1 \notin \mathbb{Z}_{\leq 0}$
shows $((z::'a::Gamma)\ gchoose\ n) = inverse\ ((z + 1) * Beta\ (z - of\_nat\ n + 1)\ (of\_nat\ n + 1))$
using *assms*
proof (*induction n arbitrary*: z)
case 0
hence $z + 2 \notin \mathbb{Z}_{\leq 0}$
using *plus_one_in_nonpos_Ints_imp*[*of z+1*] **by** (*auto simp*: *add.commute*)

```
  with 0 show ?case
    by (auto simp: Beta_def Gamma_eq_zero_iff Gamma_plus1 [symmetric]
add.commute)
next
  case (Suc n z)
  show ?case
  proof (cases z ∈ ℤ≤0)
    case True
    with Suc.prems have z = 0
    by (auto elim!: nonpos_Ints_cases simp: algebra_simps one_plus_of_int_in_nonpos_Ints_iff)
    show ?thesis
    proof (cases n = 0)
      case True
      with ‹z = 0› show ?thesis
        by (simp add: Beta_def Gamma_eq_zero_iff Gamma_plus1 [symmetric])
    next
      case False
      with ‹z = 0› show ?thesis
        by (simp_all add: Beta_pole1 one_minus_of_nat_in_nonpos_Ints_iff)
    qed
  next
    case False
    have (z gchoose (Suc n)) = ((z - 1 + 1) gchoose (Suc n)) by simp
    also have ... = (z - 1 gchoose n) * ((z - 1) + 1) / of_nat (Suc n)
      by (subst gbinomial_factors) (simp add: field_simps)
    also from False have ... = inverse (of_nat (Suc n) * Beta (z - of_nat n)
(of_nat (Suc n)))
      (is _ = inverse ?x) by (subst Suc.IH) (simp_all add: field_simps Beta_pole1)
   also have of_nat (Suc n) ∉ (ℤ≤0 :: 'a set) by (subst of_nat_in_nonpos_Ints_iff)
simp_all
    hence ?x = (z + 1) * Beta (z - of_nat (Suc n) + 1) (of_nat (Suc n) + 1)
      by (subst Beta_plus1_right [symmetric]) simp_all
    finally show ?thesis .
  qed
qed

theorem gbinomial_Gamma:
  assumes z + 1 ∉ ℤ≤0
  shows   (z gchoose n) = Gamma (z + 1) / (fact n * Gamma (z - of_nat n +
1))
proof -
  have (z gchoose n) = Gamma (z + 2) / (z + 1) / (fact n * Gamma (z - of_nat
n + 1))
    by (subst gbinomial_Beta[OF assms]) (simp_all add: Beta_def Gamma_fact
[symmetric] add_ac)
  also from assms have Gamma (z + 2) / (z + 1) = Gamma (z + 1)
    using Gamma_plus1[of z+1] by (auto simp add: field_split_simps)
  finally show ?thesis .
qed
```

Integral form

lemma *integrable_on_powr_from_0'*:
assumes *a*: $a > (-1::real)$ **and** *c*: $c \geq 0$
shows $(\lambda x.\ x\ powr\ a)$ *integrable_on* $\{0<..c\}$
proof −
from *c* **have** ∗: $\{0<..c\} - \{0..c\} = \{\}$ $\{0..c\} - \{0<..c\} = \{0\}$ **by** *auto*
show *?thesis*
by (*rule integrable_spike_set* [*OF integrable_on_powr_from_0*[*OF a c*]]) (*simp_all add*: ∗)
qed

lemma *absolutely_integrable_Gamma_integral*:
assumes $Re\ z > 0$ $a > 0$
shows $(\lambda t.\ complex\_of\_real\ t\ powr\ (z - 1) / of\_real\ (exp\ (a * t)))$
absolutely_integrable_on $\{0<..\}$ (**is** *?f absolutely_integrable_on* _)
proof −
have $((\lambda x.\ (Re\ z - 1) * (ln\ x / x)) \longrightarrow (Re\ z - 1) * 0)$ *at_top*
by (*intro tendsto_intros ln_x_over_x_tendsto_0*)
hence $((\lambda x.\ ((Re\ z - 1) * ln\ x) / x) \longrightarrow 0)$ *at_top* **by** *simp*
from *order_tendstoD*(*2*)[*OF this*, *of a/2*] **and** ‹$a > 0$›
have *eventually* $(\lambda x.\ (Re\ z - 1) * ln\ x / x < a/2)$ *at_top* **by** *simp*
from *eventually_conj*[*OF this eventually_gt_at_top*[*of 0*]]
obtain *x0* **where** $\forall x{\geq}x0.\ (Re\ z - 1) * ln\ x / x < a/2 \wedge x > 0$
by (*auto simp*: *eventually_at_top_linorder*)
hence $x0 > 0$ **by** *simp*
have $x\ powr\ (Re\ z - 1) / exp\ (a * x) < exp\ (-(a/2) * x)$ **if** $x \geq x0$ **for** *x*
proof −
from *that* **and** ‹$\forall x{\geq}x0.\ \_$› **have** *x*: $(Re\ z - 1) * ln\ x / x < a / 2\ x > 0$ **by** *auto*
have $x\ powr\ (Re\ z - 1) = exp\ ((Re\ z - 1) * ln\ x)$
using ‹$x > 0$› **by** (*simp add*: *powr_def*)
also from *x* **have** $(Re\ z - 1) * ln\ x < (a * x) / 2$ **by** (*simp add*: *field_simps*)
finally show *?thesis* **by** (*simp add*: *field_simps exp_add* [*symmetric*])
qed
note $x0 = ‹x0 > 0›\ this$

have *?f absolutely_integrable_on* $(\{0<..x0\} \cup \{x0..\})$
proof (*rule set_integrable_Un*)
show *?f absolutely_integrable_on* $\{0<..x0\}$
unfolding *set_integrable_def*
proof (*rule Bochner_Integration.integrable_bound* [*OF _ _ AE_I2*])
show *integrable lebesgue* $(\lambda x.\ indicat\_real\ \{0<..x0\}\ x *_R x\ powr\ (Re\ z - 1))$

using *x0*(*1*) *assms*
by (*intro nonnegative_absolutely_integrable_1* [*unfolded set_integrable_def*] *integrable_on_powr_from_0'*) *auto*
show $(\lambda x.\ indicat\_real\ \{0<..x0\}\ x *_R (x\ powr\ (z - 1) / exp\ (a * x))) \in$ *borel_measurable lebesgue*
by (*intro measurable_completion*)

```
      (auto intro!: borel_measurable_continuous_on_indicator continuous_intros)
    fix x :: real
    have x powr (Re z − 1) / exp (a ∗ x) ≤ x powr (Re z − 1) / 1 if x ≥ 0
      using that assms by (intro divide_left_mono) auto
    thus norm (indicator {0<..x0} x ∗R ?f x) ≤
          norm (indicator {0<..x0} x ∗R x powr (Re z − 1))
      by (simp_all add: norm_divide norm_powr_real_powr indicator_def)
  qed
 next
  show ?f absolutely_integrable_on {x0..}
    unfolding set_integrable_def
  proof (rule Bochner_Integration.integrable_bound [OF _ _ AE_I2])
    show integrable lebesgue (λx. indicat_real {x0..} x ∗R exp (− (a / 2) ∗ x))
using assms
      by (intro nonnegative_absolutely_integrable_1 [unfolded set_integrable_def]
integrable_on_exp_minus_to_infinity) auto
      show (λx. indicat_real {x0..} x ∗R (x powr (z − 1) / exp (a ∗ x))) ∈
borel_measurable lebesgue using x0(1)
      by (intro measurable_completion)
      (auto intro!: borel_measurable_continuous_on_indicator continuous_intros)
    fix x :: real
    show norm (indicator {x0..} x ∗R ?f x) ≤
          norm (indicator {x0..} x ∗R exp (−(a/2) ∗ x)) using x0
    by (auto simp: norm_divide norm_powr_real_powr indicator_def less_imp_le)
  qed
 qed auto
 also have {0<..x0} ∪ {x0..} = {0<..} using x0(1) by auto
 finally show ?thesis .
qed

lemma integrable_Gamma_integral_bound:
 fixes a c :: real
 assumes a: a > −1 and c: c ≥ 0
 defines f ≡ λx. if x ∈ {0..c} then x powr a else exp (−x/2)
 shows   f integrable_on {0..}
proof −
 have f integrable_on {0..c}
  by (rule integrable_spike_finite[of {}, OF _ _ integrable_on_powr_from_0[of
a c]])
     (insert a c, simp_all add: f_def)
 moreover have A: (λx. exp (−x/2)) integrable_on {c..}
  using integrable_on_exp_minus_to_infinity[of 1/2] by simp
 have f integrable_on {c..}
  by (rule integrable_spike_finite[of {c}, OF _ _ A]) (simp_all add: f_def)
 ultimately show f integrable_on {0..}
  by (rule integrable_Un') (insert c, auto simp: max_def)
qed

theorem Gamma_integral_complex:
```

```
  assumes z: Re z > 0
  shows   ((λt. of_real t powr (z − 1) / of_real (exp t)) has_integral Gamma z)
{0..}
proof −
  have A: ((λt. (of_real t) powr (z − 1) ∗ of_real ((1 − t) ^ n))
        has_integral (fact n / pochhammer z (n+1))) {0..1}
    if Re z > 0 for n z using that
  proof (induction n arbitrary: z)
    case 0
    have ((λt. complex_of_real t powr (z − 1)) has_integral
          (of_real 1 powr z / z − of_real 0 powr z / z)) {0..1} using 0
      by (intro fundamental_theorem_of_calculus_interior)
       (auto intro!: continuous_intros derivative_eq_intros has_vector_derivative_real_field)
    thus ?case by simp
  next
    case (Suc n)
    let ?f = λt. complex_of_real t powr z / z
    let ?f′ = λt. complex_of_real t powr (z − 1)
    let ?g = λt. (1 − complex_of_real t) ^ Suc n
    let ?g′ = λt. − ((1 − complex_of_real t) ^ n) ∗ of_nat (Suc n)
    have ((λt. ?f′ t ∗ ?g t) has_integral
          (of_nat (Suc n)) ∗ fact n / pochhammer z (n+2)) {0..1}
      (is (_ has_integral ?I) _)
    proof (rule integration_by_parts_interior[where f′ = ?f′ and g = ?g])
      from Suc.prems show continuous_on {0..1} ?f continuous_on {0..1} ?g
        by (auto intro!: continuous_intros)
    next
      fix t :: real assume t: t ∈ {0<..<1}
      show (?f has_vector_derivative ?f′ t) (at t) using t Suc.prems
        by (auto intro!: derivative_eq_intros has_vector_derivative_real_field)
      show (?g has_vector_derivative ?g′ t) (at t)
          by (rule has_vector_derivative_real_field derivative_eq_intros refl)+
simp_all
    next
      from Suc.prems have [simp]: z ≠ 0 by auto
      from Suc.prems have A: Re (z + of_nat n) > 0 for n by simp
      have [simp]: z + of_nat n ≠ 0 z + 1 + of_nat n ≠ 0 for n
      using A[of n] A[of Suc n] by (auto simp add: add.assoc simp del: plus_complex.sel)
      have ((λx. of_real x powr z ∗ of_real ((1 − x) ^ n) ∗ (− of_nat (Suc n) /
z)) has_integral
            fact n / pochhammer (z+1) (n+1) ∗ (− of_nat (Suc n) / z)) {0..1}
        (is (?A has_integral ?B) _)
      using Suc.IH[of z+1] Suc.prems by (intro has_integral_mult_left) (simp_all
add: add_ac pochhammer_rec)
      also have ?A = (λt. ?f t ∗ ?g′ t) by (intro ext) (simp_all add: field_simps)
      also have ?B = − (of_nat (Suc n) ∗ fact n / pochhammer z (n+2))
        by (simp add: field_split_simps pochhammer_rec
            prod.shift_bounds_cl_Suc_ivl del: of_nat_Suc)
      finally show ((λt. ?f t ∗ ?g′ t) has_integral (?f 1 ∗ ?g 1 − ?f 0 ∗ ?g 0 −
```

```
?I)) {0..1}
      by simp
   qed (simp_all add: bounded_bilinear_mult)
   thus ?case by simp
 qed

 have B: ((λt. if t ∈ {0..of_nat n} then
          of_real t powr (z − 1) ∗ (1 − of_real t / of_nat n) ^ n else 0)
         has_integral (of_nat n powr z ∗ fact n / pochhammer z (n+1))) {0..}
for n
 proof (cases n > 0)
   case [simp]: True
   hence [simp]: n ≠ 0 by auto
   with has_integral_affinity01[OF A[OF z, of n], of inverse (of_nat n) 0]
     have ((λx. (of_nat n − of_real x) ^ n ∗ (of_real x / of_nat n) powr (z −
1) / of_nat n ^ n)
           has_integral fact n ∗ of_nat n / pochhammer z (n+1)) ((λx. real n ∗
x) ‘{0..1})
     (is (?f has_integral ?I) ?ivl) by (simp add: field_simps scaleR_conv_of_real)
   also from True have ((λx. real n∗x) ‘{0..1}) = {0..real n}
     by (subst image_mult_atLeastAtMost) simp_all
   also have ?f = (λx. (of_real x / of_nat n) powr (z − 1) ∗ (1 − of_real x /
of_nat n) ^ n)
     using True by (intro ext) (simp add: field_simps)
   finally have ((λx. (of_real x / of_nat n) powr (z − 1) ∗ (1 − of_real x /
of_nat n) ^ n)
                 has_integral ?I) {0..real n} (is ?P) .
   also have ?P ⟷ ((λx. exp ((z − 1) ∗ of_real (ln (x / of_nat n))) ∗ (1 −
of_real x / of_nat n) ^ n)
                   has_integral ?I) {0..real n}
   by (intro has_integral_spike_finite_eq[of {0}]) (auto simp: powr_def Ln_of_real
[symmetric])
   also have ... ⟷ ((λx. exp ((z − 1) ∗ of_real (ln x − ln (of_nat n))) ∗ (1
− of_real x / of_nat n) ^ n)
                   has_integral ?I) {0..real n}
     by (intro has_integral_spike_finite_eq[of {0}]) (simp_all add: ln_div)
   finally have ... .
   note B = has_integral_mult_right[OF this, of exp ((z − 1) ∗ ln (of_nat n))]
   have ((λx. exp ((z − 1) ∗ of_real (ln x)) ∗ (1 − of_real x / of_nat n) ^ n)
         has_integral (?I ∗ exp ((z − 1) ∗ ln (of_nat n)))) {0..real n} (is ?P)
       by (insert B, subst (asm) mult.assoc [symmetric], subst (asm) exp_add
[symmetric])
         (simp add: algebra_simps)
   also have ?P ⟷ ((λx. of_real x powr (z − 1) ∗ (1 − of_real x / of_nat n)
^ n)
         has_integral (?I ∗ exp ((z − 1) ∗ ln (of_nat n)))) {0..real n}
       by (intro has_integral_spike_finite_eq[of {0}]) (simp_all add: powr_def
Ln_of_real)
    also have fact n ∗ of_nat n / pochhammer z (n+1) ∗ exp ((z − 1) ∗ Ln
```

```
(of_nat n)) =
                (of_nat n powr z * fact n / pochhammer z (n+1))
      by (auto simp add: powr_def algebra_simps exp_diff exp_of_real)
    finally show ?thesis by (subst has_integral_restrict) simp_all
  next
    case False
   thus ?thesis by (subst has_integral_restrict) (simp_all add: has_integral_refl)
  qed

  have eventually (λn. Gamma_series z n =
        of_nat n powr z * fact n / pochhammer z (n+1)) sequentially
    using eventually_gt_at_top[of 0::nat]
    by eventually_elim (simp add: powr_def algebra_simps Gamma_series_def)
  from this and Gamma_series_LIMSEQ[of z]
    have C: (λk. of_nat k powr z * fact k / pochhammer z (k+1)) ⟶ Gamma
z
    by (blast intro: Lim_transform_eventually)
  {
    fix x :: real assume x: x ≥ 0
    have lim_exp: (λk. (1 - x / real k) ^ k) ⟶ exp (-x)
      using tendsto_exp_limit_sequentially[of -x] by simp
    have (λk. of_real x powr (z - 1) * of_real ((1 - x / of_nat k) ^ k))
          ⟶ of_real x powr (z - 1) * of_real (exp (-x)) (is ?P)
      by (intro tendsto_intros lim_exp)
    also from eventually_gt_at_top[of nat ⌈x⌉]
      have eventually (λk. of_nat k > x) sequentially by eventually_elim linarith
    hence ?P ⟷ (λk. if x ≤ of_nat k then
                of_real x powr (z - 1) * of_real ((1 - x / of_nat k) ^ k) else 0)
                  ⟶ of_real x powr (z - 1) * of_real (exp (-x))
      by (intro tendsto_cong) (auto elim!: eventually_mono)
    finally have ... .
  }
  hence D: ∀x∈{0..}. (λk. if x ∈ {0..real k} then
            of_real x powr (z - 1) * (1 - of_real x / of_nat k) ^ k else 0)
           ⟶ of_real x powr (z - 1) / of_real (exp x)
    by (simp add: exp_minus field_simps cong: if_cong)

  have ((λx. (Re z - 1) * (ln x / x)) ⟶ (Re z - 1) * 0) at_top
    by (intro tendsto_intros ln_x_over_x_tendsto_0)
  hence ((λx. ((Re z - 1) * ln x) / x) ⟶ 0) at_top by simp
  from order_tendstoD(2)[OF this, of 1/2]
    have eventually (λx. (Re z - 1) * ln x / x < 1/2) at_top by simp
  from eventually_conj[OF this eventually_gt_at_top[of 0]]
    obtain x0 where ∀x≥x0. (Re z - 1) * ln x / x < 1/2 ∧ x > 0
    by (auto simp: eventually_at_top_linorder)
  hence x0: x0 > 0 ⋀x. x ≥ x0 ⟹ (Re z - 1) * ln x < x / 2 by auto

  define h where h = (λx. if x ∈ {0..x0} then x powr (Re z - 1) else exp (-x/2))
  have le_h: x powr (Re z - 1) * exp (-x) ≤ h x if x: x ≥ 0 for x
```

```
  proof (cases x > x0)
    case True
    from True x0(1) have x powr (Re z - 1) * exp (-x) = exp ((Re z - 1) * ln
x - x)
      by (simp add: powr_def exp_diff exp_minus field_simps exp_add)
    also from x0(2)[of x] True have ... < exp (-x/2)
      by (simp add: field_simps)
    finally show ?thesis using True by (auto simp add: h_def)
  next
    case False
    from x have x powr (Re z - 1) * exp (- x) ≤ x powr (Re z - 1) * 1
      by (intro mult_left_mono) simp_all
    with False show ?thesis by (auto simp add: h_def)
  qed

  have E: ∀x∈{0..}. cmod (if x ∈ {0..real k} then of_real x powr (z - 1) *
                (1 - complex_of_real x / of_nat k) ^ k else 0) ≤ h x
    (is ∀x∈_. ?f x ≤ _) for k
  proof safe
    fix x :: real assume x: x ≥ 0
    {
      fix x :: real and n :: nat assume x: x ≤ of_nat n
       have (1 - complex_of_real x / of_nat n) = complex_of_real ((1 - x /
of_nat n)) by simp
      also have norm ... = |(1 - x / real n)| by (subst norm_of_real) (rule refl)
      also from x have ... = (1 - x / real n) by (intro abs_of_nonneg) (simp_all
add: field_split_simps)
      finally have cmod (1 - complex_of_real x / of_nat n) = 1 - x / real n .
    } note D = this
    from D[of x k] x
      have ?f x ≤ (if of_nat k ≥ x ∧ k > 0 then x powr (Re z - 1) * (1 - x /
real k) ^ k else 0)
      by (auto simp: norm_mult norm_powr_real_powr norm_power intro!: mult_nonneg_nonneg)
    also have ... ≤ x powr (Re z - 1) * exp (-x)
      by (auto intro!: mult_left_mono exp_ge_one_minus_x_over_n_power_n)
    also from x have ... ≤ h x by (rule le_h)
    finally show ?f x ≤ h x .
  qed

  have F: h integrable_on {0..} unfolding h_def
    by (rule integrable_Gamma_integral_bound) (insert assms x0(1), simp_all)
  show ?thesis
    by (rule has_integral_dominated_convergence[OF B F E D C])
qed

lemma Gamma_integral_real:
  assumes x: x > (0 :: real)
  shows   ((λt. t powr (x - 1) / exp t) has_integral Gamma x) {0..}
proof -
```

```
  have A: ((λt. complex_of_real t powr (complex_of_real x − 1) /
        complex_of_real (exp t)) has_integral complex_of_real (Gamma x)) {0..}
   using Gamma_integral_complex[of x] assms by (simp_all add: Gamma_complex_of_real
powr_of_real)
   have ((λt. complex_of_real (t powr (x − 1) / exp t)) has_integral of_real
(Gamma x)) {0..}
    by (rule has_integral_eq[OF _ A]) (simp_all add: powr_of_real [symmetric])
  from has_integral_linear[OF this bounded_linear_Re] show ?thesis by (simp
add: o_def)
qed

lemma absolutely_integrable_Gamma_integral':
  assumes Re z > 0
 shows  (λt. complex_of_real t powr (z − 1) / of_real (exp t)) absolutely_integrable_on
{0<..}
 using absolutely_integrable_Gamma_integral [OF assms zero_less_one] by simp

lemma Gamma_integral_complex':
  assumes z: Re z > 0
  shows  ((λt. of_real t powr (z − 1) / of_real (exp t)) has_integral Gamma z)
{0<..}
proof −
  have ((λt. of_real t powr (z − 1) / of_real (exp t)) has_integral Gamma z)
{0..}
    by (rule Gamma_integral_complex) fact+
  hence ((λt. if t ∈ {0<..} then of_real t powr (z − 1) / of_real (exp t) else 0)
        has_integral Gamma z) {0..}
    by (rule has_integral_spike [of {0}, rotated 2]) auto
  also have ?this = ?thesis
    by (subst has_integral_restrict) auto
  finally show ?thesis .
qed

lemma Gamma_conv_nn_integral_real:
  assumes s > (0::real)
  shows   Gamma s = nn_integral lborel (λt. ennreal (indicator {0..} t ∗ t powr
(s − 1) / exp t))
 using nn_integral_has_integral_lebesgue[OF _ Gamma_integral_real[OF assms]]
by simp

lemma integrable_Beta:
  assumes a > 0 b > (0::real)
  shows   set_integrable lborel {0..1} (λt. t powr (a − 1) ∗ (1 − t) powr (b −
1))
proof −
  define C where C = max 1 ((1/2) powr (b − 1))
  define D where D = max 1 ((1/2) powr (a − 1))
  have C: (1 − x) powr (b − 1) ≤ C if x ∈ {0..1/2} for x
  proof (cases b < 1)
```

```
    case False
      with that have (1 − x) powr (b − 1) ≤ (1 powr (b − 1)) by (intro
powr_mono2) auto
    thus ?thesis by (auto simp: C_def)
  qed (insert that, auto simp: max.coboundedI1 max.coboundedI2 powr_mono2'
powr_mono2 C_def)
  have D: x powr (a − 1) ≤ D if x ∈ {1/2..1} for x
  proof (cases a < 1)
    case False
    with that have x powr (a − 1) ≤ (1 powr (a − 1)) by (intro powr_mono2)
auto
    thus ?thesis by (auto simp: D_def)
  next
    case True
  qed (insert that, auto simp: max.coboundedI1 max.coboundedI2 powr_mono2'
powr_mono2 D_def)
  have [simp]: C ≥ 0 D ≥ 0 by (simp_all add: C_def D_def)

  have I1: set_integrable lborel {0..1/2} (λt. t powr (a − 1) ∗ (1 − t) powr (b
− 1))
    unfolding set_integrable_def
  proof (rule Bochner_Integration.integrable_bound[OF _ _ AE_I2])
    have (λt. t powr (a − 1)) integrable_on {0..1/2}
      by (rule integrable_on_powr_from_0) (use assms in auto)
    hence (λt. t powr (a − 1)) absolutely_integrable_on {0..1/2}
      by (subst absolutely_integrable_on_iff_nonneg) auto
    from integrable_mult_right[OF this [unfolded set_integrable_def], of C]
    show integrable lborel (λx. indicat_real {0..1/2} x ∗R (C ∗ x powr (a − 1)))
      by (subst (asm) integrable_completion) (auto simp: mult_ac)
  next
    fix x :: real
    have x powr (a − 1) ∗ (1 − x) powr (b − 1) ≤ x powr (a − 1) ∗ C if x ∈
{0..1/2}
      using that by (intro mult_left_mono powr_mono2 C) auto
    thus norm (indicator {0..1/2} x ∗R (x powr (a − 1) ∗ (1 − x) powr (b −
1))) ≤
          norm (indicator {0..1/2} x ∗R (C ∗ x powr (a − 1)))
      by (auto simp: indicator_def abs_mult mult_ac)
  qed (auto intro!: AE_I2 simp: indicator_def)

  have I2: set_integrable lborel {1/2..1} (λt. t powr (a − 1) ∗ (1 − t) powr (b
− 1))
    unfolding set_integrable_def
  proof (rule Bochner_Integration.integrable_bound[OF _ _ AE_I2])
    have (λt. t powr (b − 1)) integrable_on {0..1/2}
      by (rule integrable_on_powr_from_0) (use assms in auto)
    hence (λt. t powr (b − 1)) integrable_on (cbox 0 (1/2)) by simp
    from integrable_affinity[OF this, of −1 1]
      have (λt. (1 − t) powr (b − 1)) integrable_on {1/2..1} by simp
```

hence $(\lambda t.\ (1 - t)\ powr\ (b - 1))$ *absolutely_integrable_on* $\{1/2..1\}$
by (*subst absolutely_integrable_on_iff_nonneg*) *auto*
from *integrable_mult_right*[*OF this* [*unfolded set_integrable_def*], *of D*]
show *integrable lborel* $(\lambda x.\ indicat\_real\ \{1/2..1\}\ x *_R (D * (1 - x)\ powr\ (b - 1)))$
by (*subst* (*asm*) *integrable_completion*) (*auto simp*: *mult_ac*)
next
fix x :: *real*
have $x\ powr\ (a - 1) * (1 - x)\ powr\ (b - 1) \leq D * (1 - x)\ powr\ (b - 1)$ **if** $x \in \{1/2..1\}$
using *that* **by** (*intro mult_right_mono powr_mono2 D*) *auto*
thus $norm\ (indicator\ \{1/2..1\}\ x *_R (x\ powr\ (a - 1) * (1 - x)\ powr\ (b - 1))) \leq$
$norm\ (indicator\ \{1/2..1\}\ x *_R (D * (1 - x)\ powr\ (b - 1)))$
by (*auto simp*: *indicator_def abs_mult mult_ac*)
qed (*auto intro*!: *AE_I2 simp*: *indicator_def*)

have *set_integrable lborel* $(\{0..1/2\} \cup \{1/2..1\})\ (\lambda t.\ t\ powr\ (a - 1) * (1 - t)\ powr\ (b - 1))$
by (*intro set_integrable_Un I1 I2*) *auto*
also have $\{0..1/2\} \cup \{1/2..1\} = \{0..(1::real)\}$ **by** *auto*
finally show *?thesis* **.**
qed

lemma *integrable_Beta'*:
assumes $a > 0\ b > (0::real)$
shows $(\lambda t.\ t\ powr\ (a - 1) * (1 - t)\ powr\ (b - 1))\ integrable\_on\ \{0..1\}$
using *integrable_Beta*[*OF assms*] **by** (*rule set_borel_integral_eq_integral*)

theorem *has_integral_Beta_real*:
assumes *a*: $a > 0$ **and** *b*: $b > (0 :: real)$
shows $((\lambda t.\ t\ powr\ (a - 1) * (1 - t)\ powr\ (b - 1))\ has\_integral\ Beta\ a\ b)\ \{0..1\}$
proof −
define B **where** $B = integral\ \{0..1\}\ (\lambda x.\ x\ powr\ (a - 1) * (1 - x)\ powr\ (b - 1))$
have [*simp*]: $B \geq 0$ **unfolding** *B_def* **using** *a b*
by (*intro integral_nonneg integrable_Beta'*) *auto*
from *a b* **have** $ennreal\ (Gamma\ a * Gamma\ b) =$
$(\int^+ t.\ ennreal\ (indicator\ \{0..\}\ t * t\ powr\ (a - 1) / exp\ t)\ \partial lborel) *$
$(\int^+ t.\ ennreal\ (indicator\ \{0..\}\ t * t\ powr\ (b - 1) / exp\ t)\ \partial lborel)$
by (*subst ennreal_mult'*) (*simp_all add*: *Gamma_conv_nn_integral_real*)
also have $\ldots = (\int^+ t.\ \int^+ u.\ ennreal\ (indicator\ \{0..\}\ t * t\ powr\ (a - 1) / exp\ t) *$
$ennreal\ (indicator\ \{0..\}\ u * u\ powr\ (b - 1) / exp\ u)\ \partial lborel\ \partial lborel)$
by (*simp add*: *nn_integral_cmult nn_integral_multc*)
also have $\ldots = (\int^+ t.\ \int^+ u.\ ennreal\ (indicator\ (\{0..\}\times\{0..\})\ (t,u) * t\ powr\ (a - 1) * u\ powr\ (b - 1)$

```
                    / exp (t + u)) ∂lborel ∂lborel)
  by (intro nn_integral_cong)
    (auto simp: indicator_def divide_ennreal ennreal_mult' [symmetric] exp_add)
 also have ... = (∫⁺t. ∫⁺u. ennreal (indicator ({0..}×{t..}) (t,u) * t powr (a
- 1) *
                    (u - t) powr (b - 1) / exp u) ∂lborel ∂lborel)
 proof (rule nn_integral_cong, goal_cases)
  case (1 t)
  have (∫⁺u. ennreal (indicator ({0..}×{0..}) (t,u) * t powr (a - 1) *
                      u powr (b - 1) / exp (t + u)) ∂distr lborel borel ((+)
(−t))) =
        (∫⁺u. ennreal (indicator ({0..}×{t..}) (t,u) * t powr (a - 1) *
                    (u - t) powr (b - 1) / exp u) ∂lborel)
    by (subst nn_integral_distr) (auto intro!: nn_integral_cong simp: indica-
tor_def)
  thus ?case by (subst (asm) lborel_distr_plus)
 qed
 also have ... = (∫⁺u. ∫⁺t. ennreal (indicator ({0..}×{t..}) (t,u) * t powr (a
- 1) *
                    (u - t) powr (b - 1) / exp u) ∂lborel ∂lborel)
  by (subst lborel_pair.Fubini')
    (auto simp: case_prod_unfold indicator_def cong: measurable_cong_sets)
 also have ... = (∫⁺u. ∫⁺t. ennreal (indicator {0..u} t * t powr (a - 1) * (u
- t) powr (b - 1)) *
                    ennreal (indicator {0..} u / exp u) ∂lborel ∂lborel)
  by (intro nn_integral_cong) (auto simp: indicator_def ennreal_mult' [symmetric])
 also have ... = (∫⁺u. (∫⁺t. ennreal (indicator {0..u} t * t powr (a - 1) * (u
- t) powr (b - 1))
                    ∂lborel) * ennreal (indicator {0..} u / exp u) ∂lborel)
  by (subst nn_integral_multc [symmetric]) auto
 also have ... = (∫⁺u. (∫⁺t. ennreal (indicator {0..u} t * t powr (a - 1) * (u
- t) powr (b - 1))
                    ∂lborel) * ennreal (indicator {0<..} u / exp u) ∂lborel)
  by (intro nn_integral_cong_AE eventually_mono[OF AE_lborel_singleton[of
0]])
    (auto simp: indicator_def)
 also have ... = (∫⁺u. ennreal B * ennreal (indicator {0..} u / exp u * u powr
(a + b - 1)) ∂lborel)
 proof (intro nn_integral_cong, goal_cases)
  case (1 u)
  show ?case
  proof (cases u > 0)
   case True
   have (∫⁺t. ennreal (indicator {0..u} t * t powr (a - 1) * (u - t) powr (b
- 1)) ∂lborel) =
         (∫⁺t. ennreal (indicator {0..1} t * (u * t) powr (a - 1) * (u - u *
t) powr (b - 1))
          ∂distr lborel borel ((*) (1 / u))) (is _ = nn_integral _ ?f)
     using True
```

by (*subst nn_integral_distr*) (*auto simp*: *indicator_def field_simps intro*!: *nn_integral_cong*)
also have *distr lborel borel* ((∗) (*1 / u*)) = *density lborel* (λ_. *u*)
using ‹*u > 0*› **by** (*subst lborel_distr_mult*) *auto*
also have *nn_integral ... ?f* = (∫⁺*x*. *ennreal* (*indicator* {*0..1*} *x* ∗ (*u* ∗ (*u* ∗ *x*) *powr* (*a* − *1*) ∗ (*u* ∗ (*1* − *x*)) *powr* (*b* − *1*))) *∂lborel*) **using** ‹*u > 0*›
by (*subst nn_integral_density*) (*auto simp*: *ennreal_mult'* [*symmetric*] *algebra_simps*)
also have ... = (∫⁺*x*. *ennreal* (*u powr* (*a* + *b* − *1*)) ∗ *ennreal* (*indicator* {*0..1*} *x* ∗ *x powr* (*a* − *1*) ∗ (*1* − *x*) *powr* (*b* − *1*)) *∂lborel*) **using** ‹*u > 0*› *a b*
by (*intro nn_integral_cong*) (*auto simp*: *indicator_def powr_mult powr_add powr_diff mult_ac ennreal_mult'* [*symmetric*])
also have ... = *ennreal* (*u powr* (*a* + *b* − *1*)) ∗ (∫⁺*x*. *ennreal* (*indicator* {*0..1*} *x* ∗ *x powr* (*a* − *1*) ∗ (*1* − *x*) *powr* (*b* − *1*)) *∂lborel*)
by (*subst nn_integral_cmult*) *auto*
also have ((λ*x*. *x powr* (*a* − *1*) ∗ (*1* − *x*) *powr* (*b* − *1*)) *has_integral* *integral* {*0..1*} (λ*x*. *x powr* (*a* − *1*) ∗ (*1* − *x*) *powr* (*b* − *1*))) {*0..1*}
using *a b* **by** (*intro integrable_integral integrable_Beta'*)
from *nn_integral_has_integral_lebesgue*[*OF _ this*] *a b*
have (∫⁺*x*. *ennreal* (*indicator* {*0..1*} *x* ∗ *x powr* (*a* − *1*) ∗ (*1* − *x*) *powr* (*b* − *1*)) *∂lborel*) = *B* **by** (*simp add*: *mult_ac B_def*)
finally show *?thesis* **using** ‹*u > 0*› **by** (*simp add*: *ennreal_mult'* [*symmetric*] *mult_ac*)
qed *auto*
qed
also have ... = *ennreal B* ∗ *ennreal* (*Gamma* (*a* + *b*))
using *a b* **by** (*subst nn_integral_cmult*) (*auto simp*: *Gamma_conv_nn_integral_real*)
also have ... = *ennreal* (*B* ∗ *Gamma* (*a* + *b*))
by (*subst* (*1 2*) *mult.commute*, *intro ennreal_mult'* [*symmetric*]) (*use a b* **in** *auto*)
finally have *B* = *Beta a b* **using** *a b Gamma_real_pos*[*of a* + *b*]
by (*subst* (*asm*) *ennreal_inj*) (*auto simp*: *field_simps Beta_def Gamma_eq_zero_iff*)
moreover have (λ*t*. *t powr* (*a* − *1*) ∗ (*1* − *t*) *powr* (*b* − *1*)) *integrable_on* {*0..1*}
by (*intro integrable_Beta' a b*)
ultimately show *?thesis* **by** (*simp add*: *has_integral_iff B_def*)
qed

6.20.12 The Weierstraß product formula for the sine

theorem *sin_product_formula_complex*:
fixes *z* :: *complex*

shows $(\lambda n.\ of\_real\ pi * z * (\prod k{=}1..n.\ 1 - z\hat{}2 \ /\ of\_nat\ k\hat{}2)) \longrightarrow sin\ (of\_real\ pi * z)$
proof −
let *?f* = *rGamma_series_Weierstrass*
have $(\lambda n.\ (-\ of\_real\ pi * inverse\ z) * (?f\ z\ n * ?f\ (-\ z)\ n))$
$\longrightarrow (-\ of\_real\ pi * inverse\ z) * (rGamma\ z * rGamma\ (-\ z))$
by (*intro tendsto_intros rGamma_Weierstrass_complex*)
also have $(\lambda n.\ (-\ of\_real\ pi * inverse\ z) * (?f\ z\ n * ?f\ (-z)\ n)) =$
$(\lambda n.\ of\_real\ pi * z * (\prod k{=}1..n.\ 1 - z\hat{}2\ /\ of\_nat\ k\ \hat{}\ 2))$
proof
fix *n* :: *nat*
have $(-\ of\_real\ pi * inverse\ z) * (?f\ z\ n * ?f\ (-z)\ n) =$
$of\_real\ pi * z * (\prod k{=}1..n.\ (of\_nat\ k - z) * (of\_nat\ k + z)\ /\ of\_nat\ k\ \hat{}\ 2)$
by (*simp add*: *rGamma_series_Weierstrass_def mult_ac exp_minus divide_simps prod.distrib*[*symmetric*] *power2_eq_square*)
also have $(\prod k{=}1..n.\ (of\_nat\ k - z) * (of\_nat\ k + z)\ /\ of\_nat\ k\ \hat{}\ 2) =$
$(\prod k{=}1..n.\ 1 - z\hat{}2\ /\ of\_nat\ k\ \hat{}\ 2)$
by (*intro prod.cong*) (*simp_all add*: *power2_eq_square field_simps*)
finally show $(-\ of\_real\ pi * inverse\ z) * (?f\ z\ n * ?f\ (-z)\ n) = of\_real\ pi * z * \ldots$
by (*simp add*: *field_split_simps*)
qed
also have $(-\ of\_real\ pi * inverse\ z) * (rGamma\ z * rGamma\ (-\ z)) = sin\ (of\_real\ pi * z)$
by (*subst rGamma_reflection_complex'*) (*simp add*: *field_split_simps*)
finally show *?thesis* **.**
qed

lemma *sin_product_formula_real*:
$(\lambda n.\ pi * (x{::}real) * (\prod k{=}1..n.\ 1 - x\hat{}2\ /\ of\_nat\ k\hat{}2)) \longrightarrow sin\ (pi * x)$
proof −
from *sin_product_formula_complex*[*of of_real x*]
have $(\lambda n.\ of\_real\ pi * of\_real\ x * (\prod k{=}1..n.\ 1 - (of\_real\ x)\hat{}2\ /\ (of\_nat\ k)\hat{}2))$
$\longrightarrow sin\ (of\_real\ pi * of\_real\ x :: complex)$ (**is** *?f* $\longrightarrow$ *?y*) **.**
also have *?f* = $(\lambda n.\ of\_real\ (pi * x * (\prod k{=}1..n.\ 1 - x\hat{}2\ /\ (of\_nat\ k\hat{}2))))$
by *simp*
also have *?y* = *of_real* (*sin* (*pi* ∗ *x*)) **by** (*simp only*: *sin_of_real* [*symmetric*] *of_real_mult*)
finally show *?thesis* **by** (*subst* (*asm*) *tendsto_of_real_iff*)
qed

lemma *sin_product_formula_real'*:
assumes $x \neq (0{::}real)$
shows $(\lambda n.\ (\prod k{=}1..n.\ 1 - x\hat{}2\ /\ of\_nat\ k\hat{}2)) \longrightarrow sin\ (pi * x)\ /\ (pi * x)$
using *tendsto_divide*[*OF sin_product_formula_real*[*of x*] *tendsto_const*[*of pi * x*]] *assms*
by *simp*

theorem *wallis*: $(\lambda n.\ \prod k{=}1..n.\ (4{*}real\ k\hat{}2)\ /\ (4{*}real\ k\hat{}2\ -\ 1)) \longrightarrow pi\ /\ 2$
proof −
from *tendsto_inverse*[*OF tendsto_mult*[*OF*
sin_product_formula_real[*of 1/2*] *tendsto_const*[*of 2/pi*]]]
have $(\lambda n.\ (\prod k{=}1..n.\ inverse\ (1\ -\ (1/2)^2\ /\ (real\ k)^2))) \longrightarrow pi/2$
by (*simp add*: *prod_inversef* [*symmetric*])
also have $(\lambda n.\ (\prod k{=}1..n.\ inverse\ (1\ -\ (1/2)^2\ /\ (real\ k)^2))) =$
$(\lambda n.\ (\prod k{=}1..n.\ (4{*}real\ k\hat{}2)/(4{*}real\ k\hat{}2\ -\ 1)))$
by (*intro ext prod.cong refl*) (*simp add*: *field_split_simps*)
finally show *?thesis* **.**
qed

6.20.13 The Solution to the Basel problem

theorem *inverse_squares_sums*: $(\lambda n.\ 1\ /\ (n\ +\ 1)^2)\ sums\ (pi^2\ /\ 6)$
proof −
define *P* **where** $P\ x\ n = (\prod k{=}1..n.\ 1\ -\ x\hat{}2\ /\ of\_nat\ k\hat{}2)$ **for** $x :: real$ **and** *n*
define *K* **where** $K = (\sum n.\ inverse\ (real\_of\_nat\ (Suc\ n))\hat{}2)$
define *f* **where** [*abs_def*]: $f\ x = (\sum n.\ P\ x\ n\ /\ of\_nat\ (Suc\ n)\hat{}2)$ **for** *x*
define *g* **where** [*abs_def*]: $g\ x = (1\ -\ sin\ (pi\ *\ x)\ /\ (pi\ *\ x))$ **for** *x*

have *sums*: $(\lambda n.\ P\ x\ n\ /\ of\_nat\ (Suc\ n)\hat{}2)\ sums\ (if\ x = 0\ then\ K\ else\ g\ x\ /\ x\hat{}2)$ **for** *x*
proof (*cases* $x = 0$)
assume *x*: $x = 0$
have *summable* $(\lambda n.\ inverse\ ((real\_of\_nat\ (Suc\ n))^2))$
using *inverse_power_summable*[*of 2*] **by** (*subst summable_Suc_iff*) *simp*
thus *?thesis* **by** (*simp add*: *x g_def P_def K_def inverse_eq_divide power_divide summable_sums*)
next
assume *x*: $x \neq 0$
have $(\lambda n.\ P\ x\ n\ -\ P\ x\ (Suc\ n))\ sums\ (P\ x\ 0\ -\ sin\ (pi\ *\ x)\ /\ (pi\ *\ x))$
unfolding *P_def* **using** *x* **by** (*intro telescope_sums' sin_product_formula_real'*)
also have $(\lambda n.\ P\ x\ n\ -\ P\ x\ (Suc\ n)) = (\lambda n.\ (x\hat{}2\ /\ of\_nat\ (Suc\ n)\hat{}2)\ *\ P\ x\ n)$
unfolding *P_def* **by** (*simp add*: *prod.nat_ivl_Suc' algebra_simps*)
also have $P\ x\ 0 = 1$ **by** (*simp add*: *P_def*)
finally have $(\lambda n.\ x^2\ /\ (of\_nat\ (Suc\ n))^2\ *\ P\ x\ n)\ sums\ (1\ -\ sin\ (pi\ *\ x)\ /\ (pi\ *\ x))$ **.**
from *sums_divide*[*OF this*, *of* $x\hat{}2$] *x* **show** *?thesis* **unfolding** *g_def* **by** *simp*
qed

have *continuous_on* (*ball 0 1*) *f*
proof (*rule uniform_limit_theorem*; (*intro always_eventually allI*)*?*)
show *uniform_limit* (*ball 0 1*) $(\lambda n\ x.\ \sum k{<}n.\ P\ x\ k\ /\ of\_nat\ (Suc\ k)\hat{}2)\ f\ sequentially$
proof (*unfold f_def*, *rule Weierstrass_m_test*)

```
    fix n :: nat and x :: real assume x: x ∈ ball 0 1
    {
      fix k :: nat assume k: k ≥ 1
      from x have x^2 < 1 by (auto simp: abs_square_less_1)
      also from k have ... ≤ of_nat k^2 by simp
      finally have (1 - x^2 / of_nat k^2) ∈ {0..1} using k
        by (simp_all add: field_simps del: of_nat_Suc)
    }
    hence (∏k=1..n. abs (1 - x^2 / of_nat k^2)) ≤ (∏k=1..n. 1) by (intro
prod_mono) simp
    thus norm (P x n / (of_nat (Suc n)^2)) ≤ 1 / of_nat (Suc n)^2
      unfolding P_def by (simp add: field_simps abs_prod del: of_nat_Suc)
   qed (subst summable_Suc_iff, insert inverse_power_summable[of 2], simp
add: inverse_eq_divide)
  qed (auto simp: P_def intro!: continuous_intros)
  hence isCont f 0 by (subst (asm) continuous_on_eq_continuous_at) simp_all
  hence (f - 0 → f 0) by (simp add: isCont_def)
 also have f 0 = K unfolding f_def P_def K_def by (simp add: inverse_eq_divide
power_divide)
  finally have f - 0 → K .

  moreover have f - 0 → pi^2 / 6
  proof (rule Lim_transform_eventually)
    define f' where [abs_def]: f' x = (∑ n. - sin_coeff (n+3) * pi ^ (n+2) *
x^n) for x
    have eventually (λx. x ≠ (0::real)) (at 0)
      by (auto simp add: eventually_at intro!: exI[of _ 1])
    thus eventually (λx. f' x = f x) (at 0)
    proof eventually_elim
      fix x :: real assume x: x ≠ 0
       have sin_coeff 1 = (1 :: real) sin_coeff 2 = (0::real) by (simp_all add:
sin_coeff_def)
       with sums_split_initial_segment[OF sums_minus[OF sin_converges], of 3
pi*x]
      have (λn. - (sin_coeff (n+3) * (pi*x)^(n+3))) sums (pi * x - sin (pi*x))
        by (simp add: eval_nat_numeral)
      from sums_divide[OF this, of x^3 * pi] x
       have (λn. - (sin_coeff (n+3) * pi^(n+2) * x^n)) sums ((1 - sin (pi*x)
/ (pi*x)) / x^2)
        by (simp add: field_split_simps eval_nat_numeral)
      with x have (λn. - (sin_coeff (n+3) * pi^(n+2) * x^n)) sums (g x / x^2)
        by (simp add: g_def)
      hence f' x = g x / x^2 by (simp add: sums_iff f'_def)
     also have ... = f x using sums[of x] x by (simp add: sums_iff g_def f_def)
      finally show f' x = f x .
    qed

    have isCont f' 0 unfolding f'_def
    proof (intro isCont_powser_converges_everywhere)
```

```
    fix x :: real show summable (λn. -sin_coeff (n+3) * pi^(n+2) * x^n)
    proof (cases x = 0)
      assume x: x ≠ 0
    from summable_divide[OF sums_summable[OF sums_split_initial_segment[OF
          sin_converges[of pi*x]], of 3], of -pi*x^3] x
      show ?thesis by (simp add: field_split_simps eval_nat_numeral)
    qed (simp only: summable_0_powser)
  qed
  hence f' - 0 → f' 0 by (simp add: isCont_def)
  also have f' 0 = pi * pi / fact 3 unfolding f'_def
    by (subst powser_zero) (simp add: sin_coeff_def)
  finally show f' - 0 → pi^2 / 6 by (simp add: eval_nat_numeral)
 qed

 ultimately have K = pi^2 / 6 by (rule LIM_unique)
 moreover from inverse_power_summable[of 2]
  have summable (λn. (inverse (real_of_nat (Suc n)))^2)
  by (subst summable_Suc_iff) (simp add: power_inverse)
 ultimately show ?thesis unfolding K_def
  by (auto simp add: sums_iff power_divide inverse_eq_divide)
qed

end


theory Interval_Integral
 imports Equivalence_Lebesgue_Henstock_Integration
begin

definition einterval a b = {x. a < ereal x ∧ ereal x < b}

lemma einterval_eq[simp]:
 shows einterval_eq_Icc: einterval (ereal a) (ereal b) = {a <..< b}
   and einterval_eq_Ici: einterval (ereal a) ∞ = {a <..}
   and einterval_eq_Iic: einterval (- ∞) (ereal b) = {..< b}
   and einterval_eq_UNIV: einterval (- ∞) ∞ = UNIV
 by (auto simp: einterval_def)

lemma einterval_same: einterval a a = {}
 by (auto simp: einterval_def)

lemma einterval_iff: x ∈ einterval a b ⟷ a < ereal x ∧ ereal x < b
 by (simp add: einterval_def)

lemma einterval_nonempty: a < b ⟹ ∃ c. c ∈ einterval a b
  by (cases a b rule: ereal2_cases, auto simp: einterval_def intro!: dense gt_ex
lt_ex)

lemma open_einterval[simp]: open (einterval a b)
```

by (*cases a b rule*: *ereal2_cases*)
(*auto simp*: *einterval_def intro*!: *open_Collect_conj open_Collect_less continuous_intros*)

lemma *borel_einterval*[*measurable*]: *einterval a b* ∈ *sets borel*
unfolding *einterval_def* **by** *measurable*

6.20.14 Approximating a (possibly infinite) interval

lemma *filterlim_sup1*: (*LIM x F. f x :> G1*) ⟹ (*LIM x F. f x :> (sup G1 G2)*)
unfolding *filterlim_def* **by** (*auto intro*: *le_supI1*)

lemma *ereal_incseq_approx*:
fixes *a b* :: *ereal*
assumes *a* < *b*
obtains *X* :: *nat* ⇒ *real* **where** *incseq X* ⋀*i*. *a* < *X i* ⋀*i*. *X i* < *b X* ⟶ *b*
proof (*cases b*)
case *PInf*
with ‹*a* < *b*› **have** *a* = −∞ ∨ (∃ *r*. *a* = *ereal r*)
by (*cases a*) *auto*
moreover have (λ*x*. *ereal* (*real* (*Suc x*))) ⟶ ∞
by (*simp add*: *Lim_PInfty filterlim_sequentially_Suc*) (*metis le_SucI of_nat_Suc of_nat_mono order_trans real_arch_simple*)
moreover have ⋀*r*. (λ*x*. *ereal* (*r* + *real* (*Suc x*))) ⟶ ∞
by (*simp add*: *filterlim_sequentially_Suc Lim_PInfty*) (*metis add.commute diff_le_eq nat_ceiling_le_eq*)
ultimately show *thesis*
by (*intro that*[*of* λ*i*. *real_of_ereal a* + *Suc i*])
(*auto simp*: *incseq_def PInf*)
next
case (*real b*′)
define *d* **where** *d* = *b*′ − (*if a* = −∞ *then b*′ − *1 else real_of_ereal a*)
with ‹*a* < *b*› **have** *a*′: *0* < *d*
by (*cases a*) (*auto simp*: *real*)
moreover
have ⋀*i r*. *r* < *b*′ ⟹ (*b*′ − *r*) ∗ *1* < (*b*′ − *r*) ∗ *real* (*Suc* (*Suc i*))
by (*intro mult_strict_left_mono*) *auto*
with ‹*a* < *b*› *a*′ **have** ⋀*i*. *a* < *ereal* (*b*′ − *d* / *real* (*Suc* (*Suc i*)))
by (*cases a*) (*auto simp*: *real d_def field_simps*)
moreover
have (λ*i*. *b*′ − *d* / *real i*) ⟶ *b*′
by (*force intro*: *tendsto_eq_intros tendsto_divide_0*[*OF tendsto_const*] *filterlim_sup1*
simp: *at_infinity_eq_at_top_bot filterlim_real_sequentially*)
then have (λ*i*. *b*′ − *d* / *Suc* (*Suc i*)) ⟶ *b*′
by (*blast intro*: *dest*: *filterlim_sequentially_Suc* [*THEN iffD2*])
ultimately show *thesis*
by (*intro that*[*of* λ*i*. *b*′ − *d* / *Suc* (*Suc i*)])
(*auto simp*: *real incseq_def intro*!: *divide_left_mono*)

qed (*use* ‹$a < b$› **in** *auto*)

lemma *ereal_decseq_approx*:
fixes $a\ b$ :: *ereal*
assumes $a < b$
obtains X :: $nat \Rightarrow real$ **where**
$decseq\ X\ \bigwedge i.\ a < X\ i\ \bigwedge i.\ X\ i < b\ X \longrightarrow a$
proof −
have $-b < -a$ **using** ‹$a < b$› **by** *simp*
from *ereal_incseq_approx*[*OF this*] **obtain** X **where**
$incseq\ X$
$\bigwedge i.\ -b < ereal\ (X\ i)$
$\bigwedge i.\ ereal\ (X\ i) < -a$
$(\lambda x.\ ereal\ (X\ x)) \longrightarrow -a$
by *auto*
then show *thesis*
apply (*intro that*[*of* $\lambda i.\ -X\ i$])
apply (*auto simp*: *decseq_def incseq_def simp flip*: *uminus_ereal.simps*)
apply (*metis ereal_minus_less_minus ereal_uminus_uminus ereal_Lim_uminus*)+
done
qed

proposition *einterval_Icc_approximation*:
fixes $a\ b$ :: *ereal*
assumes $a < b$
obtains $u\ l$ :: $nat \Rightarrow real$ **where**
$einterval\ a\ b = (\bigcup i.\ \{l\ i\ ..\ u\ i\})$
$incseq\ u\ decseq\ l\ \bigwedge i.\ l\ i < u\ i\ \bigwedge i.\ a < l\ i\ \bigwedge i.\ u\ i < b$
$l \longrightarrow a\ u \longrightarrow b$
proof −
from *dense*[*OF* ‹$a < b$›] **obtain** c **where** $a < c\ c < b$ **by** *safe*
from *ereal_incseq_approx*[*OF* ‹$c < b$›] **obtain** u **where** u:
$incseq\ u$
$\bigwedge i.\ c < ereal\ (u\ i)$
$\bigwedge i.\ ereal\ (u\ i) < b$
$(\lambda x.\ ereal\ (u\ x)) \longrightarrow b$
by *auto*
from *ereal_decseq_approx*[*OF* ‹$a < c$›] **obtain** l **where** l:
$decseq\ l$
$\bigwedge i.\ a < ereal\ (l\ i)$
$\bigwedge i.\ ereal\ (l\ i) < c$
$(\lambda x.\ ereal\ (l\ x)) \longrightarrow a$
by *auto*
have $einterval\ a\ b = (\bigcup i.\ \{l\ i\ ..\ u\ i\})$
proof (*auto simp*: *einterval_iff*)
fix x **assume** $a < ereal\ x\ ereal\ x < b$
have *eventually* $(\lambda i.\ ereal\ (l\ i) < ereal\ x)$ *sequentially*
using $l(4)$ ‹$a < ereal\ x$› **by** (*rule order_tendstoD*)
moreover

have *eventually* (λ*i*. *ereal x* < *ereal* (*u i*)) *sequentially*
using *u*(*4*) ‹*ereal x*< *b*› **by** (*rule order_tendstoD*)
ultimately have *eventually* (λ*i*. *l i* < *x* ∧ *x* < *u i*) *sequentially*
by *eventually_elim auto*
then show ∃ *i*. *l i* ≤ *x* ∧ *x* ≤ *u i*
by (*auto intro*: *less_imp_le simp*: *eventually_sequentially*)
next
fix *x i* **assume** *l i* ≤ *x x* ≤ *u i*
with ‹*a* < *ereal* (*l i*)› ‹*ereal* (*u i*) < *b*›
show *a* < *ereal x ereal x* < *b*
by (*auto simp flip*: *ereal_less_eq*(*3*))
qed
moreover { fix *i* **from** *less_trans*[*OF* ‹*l i* < *c*› ‹*c* < *u i*›] **have** *l i* < *u i* **by** *simp* **}**
ultimately show *thesis*
by (*simp add*: *l that u*)
qed

definition *interval_lebesgue_integral* :: *real measure* ⇒ *ereal* ⇒ *ereal* ⇒ (*real* ⇒ *'a*) ⇒ *'a*::{*banach*, *second_countable_topology*} **where**
interval_lebesgue_integral M a b f =
(*if a* ≤ *b then* (*LINT x*:*einterval a b*|*M*. *f x*) *else* − (*LINT x*:*einterval b a*|*M*. *f x*))

syntax
_ascii_interval_lebesgue_integral :: *pttrn* ⇒ *real* ⇒ *real* ⇒ *real measure* ⇒ *real* ⇒ *real*
((*5LINT* _=_..*_*|_. _) [*0*,*60*,*60*,*61*,*100*] *60*)

translations
LINT x=*a*..*b*|*M*. *f* == *CONST interval_lebesgue_integral M a b* (λ*x*. *f*)

definition *interval_lebesgue_integrable* :: *real measure* ⇒ *ereal* ⇒ *ereal* ⇒ (*real* ⇒ *'a*::{*banach*, *second_countable_topology*}) ⇒ *bool* **where**
interval_lebesgue_integrable M a b f =
(*if a* ≤ *b then set_integrable M* (*einterval a b*) *f else set_integrable M* (*einterval b a*) *f*)

syntax
_ascii_interval_lebesgue_borel_integral :: *pttrn* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *real*
((*4LBINT* _=_..*_*. _) [*0*,*60*,*60*,*61*] *60*)

translations
LBINT x=*a*..*b*. *f* == *CONST interval_lebesgue_integral CONST lborel a b* (λ*x*. *f*)

6.20.15 Basic properties of integration over an interval

lemma *interval_lebesgue_integral_cong*:
$a \leq b \Longrightarrow (\bigwedge x.\ x \in einterval\ a\ b \Longrightarrow f\ x = g\ x) \Longrightarrow einterval\ a\ b \in sets\ M \Longrightarrow$
interval_lebesgue_integral M a b f = interval_lebesgue_integral M a b g
by (*auto intro*: *set_lebesgue_integral_cong simp*: *interval_lebesgue_integral_def*)

lemma *interval_lebesgue_integral_cong_AE*:
$f \in borel\_measurable\ M \Longrightarrow g \in borel\_measurable\ M \Longrightarrow$
$a \leq b \Longrightarrow AE\ x \in einterval\ a\ b\ in\ M.\ f\ x = g\ x \Longrightarrow einterval\ a\ b \in sets\ M \Longrightarrow$
interval_lebesgue_integral M a b f = interval_lebesgue_integral M a b g
by (*auto intro*: *set_lebesgue_integral_cong_AE simp*: *interval_lebesgue_integral_def*)

lemma *interval_integrable_mirror*:
shows *interval_lebesgue_integrable lborel a b* $(\lambda x.\ f\ (-x)) \longleftrightarrow$
interval_lebesgue_integrable lborel $(-b)$ $(-a)$ *f*
proof −
have $*$: *indicator* $(einterval\ a\ b)\ (-\ x) = (indicator\ (einterval\ (-b)\ (-a))\ x :: real)$
for *a b* :: *ereal* **and** *x* :: *real*
by (*cases a b rule*: *ereal2_cases*) (*auto simp*: *einterval_def split*: *split_indicator*)
show *?thesis*
unfolding *interval_lebesgue_integrable_def*
using *lborel_integrable_real_affine_iff*[*symmetric*, *of* -1 $\lambda x.\ indicator\ (einterval\ \_\ \_)\ x *_R f\ x\ 0$]
by (*simp add*: $*$ *set_integrable_def*)
qed

lemma *interval_lebesgue_integral_add* [*intro*, *simp*]:
fixes *M a b f*
assumes *interval_lebesgue_integrable M a b f interval_lebesgue_integrable M a b g*
shows *interval_lebesgue_integrable M a b* $(\lambda x.\ f\ x + g\ x)$
and *interval_lebesgue_integral M a b* $(\lambda x.\ f\ x + g\ x) =$
interval_lebesgue_integral M a b f + interval_lebesgue_integral M a b g
using *assms* **by** (*auto simp*: *interval_lebesgue_integral_def interval_lebesgue_integrable_def field_simps*)

lemma *interval_lebesgue_integral_diff* [*intro*, *simp*]:
fixes *M a b f*
assumes *interval_lebesgue_integrable M a b f*
interval_lebesgue_integrable M a b g
shows *interval_lebesgue_integrable M a b* $(\lambda x.\ f\ x - g\ x)$ **and**
interval_lebesgue_integral M a b $(\lambda x.\ f\ x - g\ x) =$
interval_lebesgue_integral M a b f − interval_lebesgue_integral M a b g
using *assms* **by** (*auto simp*: *interval_lebesgue_integral_def interval_lebesgue_integrable_def field_simps*)

lemma *interval_lebesgue_integrable_mult_right* [*intro*, *simp*]:
fixes *M a b c* **and** *f* :: *real* ⇒ *'a*::{*banach*, *real_normed_field*, *second_countable_topology*}

shows $(c \neq 0 \Longrightarrow \mathit{interval\_lebesgue\_integrable}\ M\ a\ b\ f) \Longrightarrow$
$\mathit{interval\_lebesgue\_integrable}\ M\ a\ b\ (\lambda x.\ c * f\ x)$
by (*simp add*: *interval_lebesgue_integrable_def*)

lemma *interval_lebesgue_integrable_mult_left* [*intro*, *simp*]:
fixes $M\ a\ b\ c$ **and** $f :: \mathit{real} \Rightarrow 'a::\{\mathit{banach}, \mathit{real\_normed\_field}, \mathit{second\_countable\_topology}\}$
shows $(c \neq 0 \Longrightarrow \mathit{interval\_lebesgue\_integrable}\ M\ a\ b\ f) \Longrightarrow$
$\mathit{interval\_lebesgue\_integrable}\ M\ a\ b\ (\lambda x.\ f\ x * c)$
by (*simp add*: *interval_lebesgue_integrable_def*)

lemma *interval_lebesgue_integrable_divide* [*intro*, *simp*]:
fixes $M\ a\ b\ c$ **and** $f :: \mathit{real} \Rightarrow 'a::\{\mathit{banach}, \mathit{real\_normed\_field}, \mathit{field}, \mathit{second\_countable\_topology}\}$
shows $(c \neq 0 \Longrightarrow \mathit{interval\_lebesgue\_integrable}\ M\ a\ b\ f) \Longrightarrow$
$\mathit{interval\_lebesgue\_integrable}\ M\ a\ b\ (\lambda x.\ f\ x\ /\ c)$
by (*simp add*: *interval_lebesgue_integrable_def*)

lemma *interval_lebesgue_integral_mult_right* [*simp*]:
fixes $M\ a\ b\ c$ **and** $f :: \mathit{real} \Rightarrow 'a::\{\mathit{banach}, \mathit{real\_normed\_field}, \mathit{second\_countable\_topology}\}$
shows $\mathit{interval\_lebesgue\_integral}\ M\ a\ b\ (\lambda x.\ c * f\ x) =$
$c * \mathit{interval\_lebesgue\_integral}\ M\ a\ b\ f$
by (*simp add*: *interval_lebesgue_integral_def*)

lemma *interval_lebesgue_integral_mult_left* [*simp*]:
fixes $M\ a\ b\ c$ **and** $f :: \mathit{real} \Rightarrow 'a::\{\mathit{banach}, \mathit{real\_normed\_field}, \mathit{second\_countable\_topology}\}$
shows $\mathit{interval\_lebesgue\_integral}\ M\ a\ b\ (\lambda x.\ f\ x * c) =$
$\mathit{interval\_lebesgue\_integral}\ M\ a\ b\ f * c$
by (*simp add*: *interval_lebesgue_integral_def*)

lemma *interval_lebesgue_integral_divide* [*simp*]:
fixes $M\ a\ b\ c$ **and** $f :: \mathit{real} \Rightarrow 'a::\{\mathit{banach}, \mathit{real\_normed\_field}, \mathit{field}, \mathit{second\_countable\_topology}\}$
shows $\mathit{interval\_lebesgue\_integral}\ M\ a\ b\ (\lambda x.\ f\ x\ /\ c) =$
$\mathit{interval\_lebesgue\_integral}\ M\ a\ b\ f\ /\ c$
by (*simp add*: *interval_lebesgue_integral_def*)

lemma *interval_lebesgue_integral_uminus*:
$\mathit{interval\_lebesgue\_integral}\ M\ a\ b\ (\lambda x.\ -\ f\ x) = -\ \mathit{interval\_lebesgue\_integral}\ M$
$a\ b\ f$
by (*auto simp*: *interval_lebesgue_integral_def interval_lebesgue_integrable_def*
set_lebesgue_integral_def)

lemma *interval_lebesgue_integral_of_real*:
$\mathit{interval\_lebesgue\_integral}\ M\ a\ b\ (\lambda x.\ \mathit{complex\_of\_real}\ (f\ x)) =$
$\mathit{of\_real}\ (\mathit{interval\_lebesgue\_integral}\ M\ a\ b\ f)$
unfolding *interval_lebesgue_integral_def*
by (*auto simp*: *interval_lebesgue_integral_def set_integral_complex_of_real*)

lemma *interval_lebesgue_integral_le_eq*:
fixes $a\ b\ f$
assumes $a \leq b$

shows *interval_lebesgue_integral M a b f* = (*LINT x* : *einterval a b* | *M*. *f x*)
using *assms* **by** (*auto simp*: *interval_lebesgue_integral_def*)

lemma *interval_lebesgue_integral_gt_eq*:
fixes *a b f*
assumes *a* > *b*
shows *interval_lebesgue_integral M a b f* = −(*LINT x* : *einterval b a* | *M*. *f x*)
using *assms* **by** (*auto simp*: *interval_lebesgue_integral_def less_imp_le einterval_def*)

lemma *interval_lebesgue_integral_gt_eq′*:
fixes *a b f*
assumes *a* > *b*
shows *interval_lebesgue_integral M a b f* = − *interval_lebesgue_integral M b a f*
using *assms* **by** (*auto simp*: *interval_lebesgue_integral_def less_imp_le einterval_def*)

lemma *interval_integral_endpoints_same* [*simp*]: (*LBINT x*=*a*..*a*. *f x*) = *0*
by (*simp add*: *interval_lebesgue_integral_def set_lebesgue_integral_def einterval_same*)

lemma *interval_integral_endpoints_reverse*: (*LBINT x*=*a*..*b*. *f x*) = −(*LBINT x*=*b*..*a*. *f x*)
by (*cases a b rule*: *linorder_cases*) (*auto simp*: *interval_lebesgue_integral_def set_lebesgue_integral_def einterval_same*)

lemma *interval_integrable_endpoints_reverse*:
interval_lebesgue_integrable lborel a b f ⟷
interval_lebesgue_integrable lborel b a f
by (*cases a b rule*: *linorder_cases*) (*auto simp*: *interval_lebesgue_integrable_def einterval_same*)

lemma *interval_integral_reflect*:
(*LBINT x*=*a*..*b*. *f x*) = (*LBINT x*=−*b*..−*a*. *f* (−*x*))
proof (*induct a b rule*: *linorder_wlog*)
case (*sym a b*) **then show** *?case*
by (*auto simp*: *interval_lebesgue_integral_def interval_integrable_endpoints_reverse*
split: *if_split_asm*)
next
case (*le a b*)
have *LBINT x*:{*x*. − *x* ∈ *einterval a b*}. *f* (− *x*) = *LBINT x*:*einterval* (− *b*) (− *a*). *f* (− *x*)
unfolding *interval_lebesgue_integrable_def set_lebesgue_integral_def einterval_def*
by (*metis* (*lifting*) *ereal_less_uminus_reorder ereal_uminus_less_reorder indicator_simps mem_Collect_eq uminus_ereal.simps*(*1*))
then show *?case*
unfolding *interval_lebesgue_integral_def*

by (*subst set_integral_reflect*) (*simp add*: *le*)
qed

lemma *interval_lebesgue_integral_0_infty*:
interval_lebesgue_integrable M 0 ∞ f ⟷ set_integrable M {0<..} f
interval_lebesgue_integral M 0 ∞ f = (LINT x:{0<..}|M. f x)
unfolding *zero_ereal_def*
by (*auto simp*: *interval_lebesgue_integral_le_eq interval_lebesgue_integrable_def*)

lemma *interval_integral_to_infinity_eq*: (*LINT x=ereal a..∞ | M. f x*) = (*LINT x : {a<..} | M. f x*)
unfolding *interval_lebesgue_integral_def* **by** *auto*

proposition *interval_integrable_to_infinity_eq*: (*interval_lebesgue_integrable M a ∞ f*) =
(*set_integrable M {a<..} f*)
unfolding *interval_lebesgue_integrable_def* **by** *auto*

6.20.16 Basic properties of integration over an interval wrt lebesgue measure

lemma *interval_integral_zero* [*simp*]:
fixes *a b* :: *ereal*
shows *LBINT x=a..b. 0 = 0*
unfolding *interval_lebesgue_integral_def set_lebesgue_integral_def einterval_eq*
by *simp*

lemma *interval_integral_const* [*intro*, *simp*]:
fixes *a b c* :: *real*
shows *interval_lebesgue_integrable lborel a b* (*λx. c*) **and** *LBINT x=a..b. c = c ∗ (b − a)*
unfolding *interval_lebesgue_integral_def interval_lebesgue_integrable_def einterval_eq*
by (*auto simp*: *less_imp_le field_simps measure_def set_integrable_def set_lebesgue_integral_def*)

lemma *interval_integral_cong_AE*:
assumes [*measurable*]: *f ∈ borel_measurable borel g ∈ borel_measurable borel*
assumes *AE x ∈ einterval* (*min a b*) (*max a b*) *in lborel. f x = g x*
shows *interval_lebesgue_integral lborel a b f = interval_lebesgue_integral lborel a b g*
using *assms*
by (*auto simp*: *interval_lebesgue_integral_def max_def min_def intro*!: *set_lebesgue_integral_cong_AE*)

lemma *interval_integral_cong*:
assumes ⋀*x. x ∈ einterval* (*min a b*) (*max a b*) ⟹ *f x = g x*
shows *interval_lebesgue_integral lborel a b f = interval_lebesgue_integral lborel a b g*
using *assms* **by** (*simp add*: *interval_lebesgue_integral_def set_lebesgue_integral_cong*)

lemma *interval_lebesgue_integrable_cong_AE*:
  $f \in$ *borel_measurable lborel* $\Longrightarrow$ $g \in$ *borel_measurable lborel* $\Longrightarrow$
  AE $x \in$ *einterval* (*min a b*) (*max a b*) *in lborel*. $f\ x = g\ x \Longrightarrow$
  interval_lebesgue_integrable lborel a b f = *interval_lebesgue_integrable lborel a b g*
 apply (*simp add*: *interval_lebesgue_integrable_def*)
 apply (*intro conjI impI set_integrable_cong_AE*)
 apply (*auto simp*: *min_def max_def*)
 done

lemma *interval_integrable_abs_iff*:
 fixes f :: *real* $\Rightarrow$ *real*
 shows $f \in$ *borel_measurable lborel* $\Longrightarrow$
  interval_lebesgue_integrable lborel a b (λx. $|f\ x|$) = *interval_lebesgue_integrable lborel a b f*
 unfolding *interval_lebesgue_integrable_def*
 by (*simp add*: *set_integrable_abs_iff'*)

lemma *interval_integral_Icc*:
 fixes *a b* :: *real*
 shows $a \leq b \Longrightarrow$ (*LBINT x=a..b. f x*) = (*LBINT x* : {*a..b*}. *f x*)
 by (*auto intro*!: *set_integral_discrete_difference*[**where** *X*={*a*, *b*}]
  simp add: *interval_lebesgue_integral_def*)

lemma *interval_integral_Icc'*:
 $a \leq b \Longrightarrow$ (*LBINT x=a..b. f x*) = (*LBINT x* : {*x*. $a \leq$ *ereal x* $\wedge$ *ereal x* $\leq b$}. *f x*)
  by (*auto intro*!: *set_integral_discrete_difference*[**where** *X*={*real_of_ereal a*, *real_of_ereal b*}]
  simp add: *interval_lebesgue_integral_def einterval_iff*)

lemma *interval_integral_Ioc*:
 $a \leq b \Longrightarrow$ (*LBINT x=a..b. f x*) = (*LBINT x* : {*a<..b*}. *f x*)
 by (*auto intro*!: *set_integral_discrete_difference*[**where** *X*={*a*, *b*}]
  simp add: *interval_lebesgue_integral_def einterval_iff*)

lemma *interval_integral_Ioc'*:
 $a \leq b \Longrightarrow$ (*LBINT x=a..b. f x*) = (*LBINT x* : {*x*. $a <$ *ereal x* $\wedge$ *ereal x* $\leq b$}. *f x*)
  by (*auto intro*!: *set_integral_discrete_difference*[**where** *X*={*real_of_ereal a*, *real_of_ereal b*}]
  simp add: *interval_lebesgue_integral_def einterval_iff*)

lemma *interval_integral_Ico*:
 $a \leq b \Longrightarrow$ (*LBINT x=a..b. f x*) = (*LBINT x* : {*a..<b*}. *f x*)
 by (*auto intro*!: *set_integral_discrete_difference*[**where** *X*={*a*, *b*}]
  simp add: *interval_lebesgue_integral_def einterval_iff*)

lemma *interval_integral_Ioi*:
  $|a| < \infty \Longrightarrow (LBINT\ x{=}a..\infty.\ f\ x) = (LBINT\ x : \{real\_of\_ereal\ a <..\}.\ f\ x)$
  by (*auto simp*: *interval_lebesgue_integral_def einterval_iff*)

lemma *interval_integral_Ioo*:
  $a \leq b \Longrightarrow |a| < \infty ==> |b| < \infty \Longrightarrow (LBINT\ x{=}a..b.\ f\ x) = (LBINT\ x : \{real\_of\_ereal\ a <..< real\_of\_ereal\ b\}.\ f\ x)$
  by (*auto simp*: *interval_lebesgue_integral_def einterval_iff*)

lemma *interval_integral_discrete_difference*:
  fixes $f :: real \Rightarrow 'b{::}\{banach, second\_countable\_topology\}$ **and** $a\ b :: ereal$
  assumes *countable X*
  and *eq*: $\bigwedge x.\ a \leq b \Longrightarrow a < x \Longrightarrow x < b \Longrightarrow x \notin X \Longrightarrow f\ x = g\ x$
  and *anti_eq*: $\bigwedge x.\ b \leq a \Longrightarrow b < x \Longrightarrow x < a \Longrightarrow x \notin X \Longrightarrow f\ x = g\ x$
  assumes $\bigwedge x.\ x \in X \Longrightarrow emeasure\ M\ \{x\} = 0\ \bigwedge x.\ x \in X \Longrightarrow \{x\} \in sets\ M$
  shows *interval_lebesgue_integral M a b f* = *interval_lebesgue_integral M a b g*
  unfolding *interval_lebesgue_integral_def set_lebesgue_integral_def*
 apply (*intro if_cong refl arg_cong*[**where** $f{=}\lambda x.\ -\ x$] *integral_discrete_difference*[*of X*] *assms*)
  apply (*auto simp*: *eq anti_eq einterval_iff split*: *split_indicator*)
  done

lemma *interval_integral_sum*:
  fixes *a b c* :: *ereal*
  assumes *integrable*: *interval_lebesgue_integrable lborel* (*min a* (*min b c*)) (*max a* (*max b c*)) *f*
  shows (*LBINT x*=*a*..*b*. *f x*) + (*LBINT x*=*b*..*c*. *f x*) = (*LBINT x*=*a*..*c*. *f x*)
proof −
  let *?I* = $\lambda a\ b.$ *LBINT x*=*a*..*b*. *f x*
  { fix *a b c* :: *ereal* **assume** *interval_lebesgue_integrable lborel a c f a* ≤ *b b* ≤ *c*
    then have *ord*: *a* ≤ *b b* ≤ *c a* ≤ *c* **and** *f'*: *set_integrable lborel* (*einterval a c*) *f*
      by (*auto simp*: *interval_lebesgue_integrable_def*)
    then have *f*: *set_borel_measurable borel* (*einterval a c*) *f*
      unfolding *set_integrable_def set_borel_measurable_def*
      by (*drule_tac borel_measurable_integrable*) *simp*
    have (*LBINT x*:*einterval a c*. *f x*) = (*LBINT x*:*einterval a b* ∪ *einterval b c*. *f x*)
    proof (*rule set_integral_cong_set*)
      show *AE x in lborel*. (*x* ∈ *einterval a b* ∪ *einterval b c*) = (*x* ∈ *einterval a c*)
        using *AE_lborel_singleton*[*of real_of_ereal b*] *ord*
        by (*cases a b c rule*: *ereal3_cases*) (*auto simp*: *einterval_iff*)
     show *set_borel_measurable lborel* (*einterval a c*) *f set_borel_measurable lborel* (*einterval a b* ∪ *einterval b c*) *f*
        unfolding *set_borel_measurable_def*
      using *ord* **by** (*auto simp*: *einterval_iff intro*!: *set_borel_measurable_subset*[*OF f*, *unfolded set_borel_measurable_def*])
    qed
    also have ... = (*LBINT x*:*einterval a b*. *f x*) + (*LBINT x*:*einterval b c*. *f x*)

```
    using ord
     by (intro set_integral_Un_AE) (auto intro!: set_integrable_subset[OF f'']
simp: einterval_iff not_less)
   finally have ?I a b + ?I b c = ?I a c
    using ord by (simp add: interval_lebesgue_integral_def)
 } note 1 = this
 { fix a b c :: ereal assume interval_lebesgue_integrable lborel a c f a ≤ b b ≤ c
   from 1[OF this] have ?I b c + ?I a b = ?I a c
    by (metis add.commute)
 } note 2 = this
 have 3: ⋀a b. b ≤ a ⟹ (LBINT x=a..b. f x) = - (LBINT x=b..a. f x)
  by (rule interval_integral_endpoints_reverse)
 show ?thesis
  using integrable
  apply (cases a b b c a c rule: linorder_le_cases[case_product linorder_le_cases
linorder_cases])
   apply simp_all
  by (simp_all add: min_absorb1 min_absorb2 max_absorb1 max_absorb2 field_simps
1 2 3)
qed

lemma interval_integrable_isCont:
 fixes a b and f :: real ⇒ 'a::{banach, second_countable_topology}
 shows (⋀x. min a b ≤ x ⟹ x ≤ max a b ⟹ isCont f x) ⟹
   interval_lebesgue_integrable lborel a b f
proof (induct a b rule: linorder_wlog)
 case (le a b) then show ?case
   unfolding interval_lebesgue_integrable_def set_integrable_def
   by (auto simp: interval_lebesgue_integrable_def
     intro!: set_integrable_subset[unfolded set_integrable_def, OF borel_integrable_compact[of
{a .. b}]]
       continuous_at_imp_continuous_on)
qed (auto intro: interval_integrable_endpoints_reverse[THEN iffD1])

lemma interval_integrable_continuous_on:
 fixes a b :: real and f
 assumes a ≤ b and continuous_on {a..b} f
 shows interval_lebesgue_integrable lborel a b f
using assms unfolding interval_lebesgue_integrable_def apply simp
  by (rule set_integrable_subset, rule borel_integrable_atLeastAtMost' [of a b],
auto)

lemma interval_integral_eq_integral:
 fixes f :: real ⇒ 'a::euclidean_space
 shows a ≤ b ⟹ set_integrable lborel {a..b} f ⟹ LBINT x=a..b. f x = integral
{a..b} f
 by (subst interval_integral_Icc, simp) (rule set_borel_integral_eq_integral)

lemma interval_integral_eq_integral':
```

fixes $f :: real \Rightarrow 'a::euclidean\_space$
shows $a \leq b \Longrightarrow set\_integrable\ lborel\ (einterval\ a\ b)\ f \Longrightarrow LBINT\ x{=}a..b.\ f\ x = integral\ (einterval\ a\ b)\ f$
by (*subst interval_lebesgue_integral_le_eq*, *simp*) (*rule set_borel_integral_eq_integral*)

6.20.17 General limit approximation arguments

proposition *interval_integral_Icc_approx_nonneg*:
fixes $a\ b :: ereal$
assumes $a < b$
fixes $u\ l :: nat \Rightarrow real$
assumes *approx*: $einterval\ a\ b = (\bigcup i.\ \{l\ i\ ..\ u\ i\})$
$incseq\ u\ decseq\ l\ \bigwedge i.\ l\ i < u\ i\ \bigwedge i.\ a < l\ i\ \bigwedge i.\ u\ i < b$
$l \longrightarrow a\ u \longrightarrow b$
fixes $f :: real \Rightarrow real$
assumes *f_integrable*: $\bigwedge i.\ set\_integrable\ lborel\ \{l\ i..u\ i\}\ f$
assumes *f_nonneg*: $AE\ x\ in\ lborel.\ a < ereal\ x \longrightarrow ereal\ x < b \longrightarrow 0 \leq f\ x$
assumes *f_measurable*: $set\_borel\_measurable\ lborel\ (einterval\ a\ b)\ f$
assumes *lbint_lim*: $(\lambda i.\ LBINT\ x{=}l\ i..\ u\ i.\ f\ x) \longrightarrow C$
shows
$set\_integrable\ lborel\ (einterval\ a\ b)\ f$
$(LBINT\ x{=}a..b.\ f\ x) = C$
proof −
have *1* [*unfolded set_integrable_def*]: $\bigwedge i.\ set\_integrable\ lborel\ \{l\ i..u\ i\}\ f$ **by** (*rule f_integrable*)
have *2*: $AE\ x\ in\ lborel.\ mono\ (\lambda n.\ indicator\ \{l\ n..u\ n\}\ x *_R f\ x)$
proof −
from *f_nonneg* **have** $AE\ x\ in\ lborel.\ \forall i.\ l\ i \leq x \longrightarrow x \leq u\ i \longrightarrow 0 \leq f\ x$
by *eventually_elim*
(*metis approx*(*5*) *approx*(*6*) *dual_order.strict_trans1 ereal_less_eq*(*3*) *le_less_trans*)
then show *?thesis*
apply *eventually_elim*
apply (*auto simp*: *mono_def split*: *split_indicator*)
apply (*metis approx*(*3*) *decseqD order_trans*)
apply (*metis approx*(*2*) *incseqD order_trans*)
done
qed
have *3*: $AE\ x\ in\ lborel.\ (\lambda i.\ indicator\ \{l\ i..u\ i\}\ x *_R f\ x) \longrightarrow indicator\ (einterval\ a\ b)\ x *_R f\ x$
proof −
{ fix $x\ i$ **assume** $l\ i \leq x\ x \leq u\ i$
then have $eventually\ (\lambda i.\ l\ i \leq x \wedge x \leq u\ i)\ sequentially$
apply (*auto simp*: *eventually_sequentially intro*!: *exI*[*of* _ *i*])
apply (*metis approx*(*3*) *decseqD order_trans*)
apply (*metis approx*(*2*) *incseqD order_trans*)
done
then have $eventually\ (\lambda i.\ f\ x * indicator\ \{l\ i..u\ i\}\ x = f\ x)\ sequentially$
by *eventually_elim auto* **}**

```
    then show ?thesis
     unfolding approx(1) by (auto intro!: AE_I2 tendsto_eventually split: split_indicator)
  qed
  have 4: (λi. ∫ x. indicator {l i..u i} x *R f x ∂lborel) ⟶ C
   using lbint_lim by (simp add: interval_integral_Icc [unfolded set_lebesgue_integral_def]
approx less_imp_le)
  have 5: (λx. indicat_real (einterval a b) x *R f x) ∈ borel_measurable lborel
    using f_measurable set_borel_measurable_def by blast
  have (LBINT x=a..b. f x) = lebesgue_integral lborel (λx. indicator (einterval a
b) x *R f x)
   using assms by (simp add: interval_lebesgue_integral_def set_lebesgue_integral_def
less_imp_le)
  also have ... = C
    by (rule integral_monotone_convergence [OF 1 2 3 4 5])
  finally show (LBINT x=a..b. f x) = C .
  show set_integrable lborel (einterval a b) f
    unfolding set_integrable_def
    by (rule integrable_monotone_convergence[OF 1 2 3 4 5])
qed

proposition interval_integral_Icc_approx_integrable:
  fixes u l :: nat ⇒ real and a b :: ereal
  fixes f :: real ⇒ 'a::{banach, second_countable_topology}
  assumes a < b
  assumes  approx: einterval a b = (⋃ i. {l i .. u i})
    incseq u decseq l ⋀i. l i < u i ⋀i. a < l i ⋀i. u i < b
    l ⟶ a u ⟶ b
  assumes f_integrable: set_integrable lborel (einterval a b) f
  shows (λi. LBINT x=l i.. u i. f x) ⟶ (LBINT x=a..b. f x)
proof −
  have (λi. LBINT x:{l i.. u i}. f x) ⟶ (LBINT x:einterval a b. f x)
    unfolding set_lebesgue_integral_def
  proof (rule integral_dominated_convergence)
    show integrable lborel (λx. norm (indicator (einterval a b) x *R f x))
      using f_integrable integrable_norm set_integrable_def by blast
    show (λx. indicat_real (einterval a b) x *R f x) ∈ borel_measurable lborel
      using f_integrable by (simp add: set_integrable_def)
    then show ⋀i. (λx. indicat_real {l i..u i} x *R f x) ∈ borel_measurable lborel
      by (rule set_borel_measurable_subset [unfolded set_borel_measurable_def])
(auto simp: approx)
   show ⋀i. AE x in lborel. norm (indicator {l i..u i} x *R f x) ≤ norm (indicator
(einterval a b) x *R f x)
      by (intro AE_I2) (auto simp: approx split: split_indicator)
   show AE x in lborel. (λi. indicator {l i..u i} x *R f x) ⟶ indicator (einterval
a b) x *R f x
    proof (intro AE_I2 tendsto_intros tendsto_eventually)
      fix x
      { fix i assume l i ≤ x x ≤ u i
        with ‹incseq u›[THEN incseqD, of i] ‹decseq l›[THEN decseqD, of i]
```

have *eventually* ($\lambda i.\ l\ i \le x \wedge x \le u\ i$) *sequentially*
by (*auto simp*: *eventually_sequentially decseq_def incseq_def intro*: *order_trans*) **}**
then show *eventually* ($\lambda xa.$ *indicator* $\{l\ xa..u\ xa\}\ x$ = (*indicator* (*einterval a b*) *x*::*real*)) *sequentially*
using *approx order_tendstoD*(*2*)[*OF* ‹$l \longrightarrow a$›, *of x*] *order_tendstoD*(*1*)[*OF* ‹$u \longrightarrow b$›, *of x*]
by (*auto split*: *split_indicator*)
qed
qed
with ‹$a < b$› ‹$\bigwedge i.\ l\ i < u\ i$› **show** *?thesis*
by (*simp add*: *interval_lebesgue_integral_le_eq*[*symmetric*] *interval_integral_Icc less_imp_le*)
qed

6.20.18 A slightly stronger Fundamental Theorem of Calculus

Three versions: first, for finite intervals, and then two versions for arbitrary intervals.

lemma *interval_integral_FTC_finite*:
fixes *f F* :: *real* ⇒ *'a*::*euclidean_space* **and** *a b* :: *real*
assumes *f*: *continuous_on* $\{min\ a\ b..max\ a\ b\}$ *f*
assumes *F*: $\bigwedge x.\ min\ a\ b \le x \Longrightarrow x \le max\ a\ b \Longrightarrow$ (*F has_vector_derivative* (*f x*)) (*at x within* $\{min\ a\ b..max\ a\ b\}$)
shows (*LBINT x*=*a*..*b*. *f x*) = *F b* − *F a*
proof (*cases a* ≤ *b*)
case *True*
have (*LBINT x*=*a*..*b*. *f x*) = (*LBINT x*. *indicat_real* $\{a..b\}\ x *_R f\ x$)
by (*simp add*: *True interval_integral_Icc set_lebesgue_integral_def*)
also have ... = *F b* − *F a*
proof (*rule integral_FTC_atLeastAtMost* [*OF True*])
show *continuous_on* $\{a..b\}$ *f*
using *True f* **by** *linarith*
show $\bigwedge x.\ [\![a \le x; x \le b]\!] \Longrightarrow$ (*F has_vector_derivative f x*) (*at x within* $\{a..b\}$)
by (*metis F True max.commute max_absorb1 min_def*)
qed
finally show *?thesis* **.**
next
case *False*
then have *b* ≤ *a*
by *simp*
have − *interval_lebesgue_integral lborel* (*ereal b*) (*ereal a*) *f* = − (*LBINT x*. *indicat_real* $\{b..a\}\ x *_R f\ x$)
by (*simp add*: ‹$b \le a$› *interval_integral_Icc set_lebesgue_integral_def*)
also have ... = *F b* − *F a*
proof (*subst integral_FTC_atLeastAtMost* [*OF* ‹$b \le a$›])

```
    show continuous_on {b..a} f
      using False f by linarith
    show ⋀x. ⟦b ≤ x; x ≤ a⟧
        ⟹ (F has_vector_derivative f x) (at x within {b..a})
      by (metis F False max_def min_def)
  qed auto
  finally show ?thesis
    by (metis interval_integral_endpoints_reverse)
qed


lemma interval_integral_FTC_nonneg:
  fixes f F :: real ⇒ real and a b :: ereal
  assumes a < b
  assumes F: ⋀x. a < ereal x ⟹ ereal x < b ⟹ DERIV F x :> f x
  assumes f: ⋀x. a < ereal x ⟹ ereal x < b ⟹ isCont f x
  assumes f_nonneg: AE x in lborel. a < ereal x ⟶ ereal x < b ⟶ 0 ≤ f x
  assumes A: ((F ∘ real_of_ereal) ⟶⟶ A) (at_right a)
  assumes B: ((F ∘ real_of_ereal) ⟶⟶ B) (at_left b)
  shows
    set_integrable lborel (einterval a b) f
    (LBINT x=a..b. f x) = B − A
proof −
  obtain u l where approx:
    einterval a b = (⋃i. {l i .. u i})
    incseq u decseq l ⋀i. l i < u i ⋀i. a < l i ⋀i. u i < b
    l ⟶⟶ a u ⟶⟶ b
    by (blast intro: einterval_Icc_approximation[OF ‹a < b›])
  have aless[simp]: ⋀x i. l i ≤ x ⟹ a < ereal x
    by (rule order_less_le_trans, rule approx, force)
  have lessb[simp]: ⋀x i. x ≤ u i ⟹ ereal x < b
   by (rule order_le_less_trans, subst ereal_less_eq(3), assumption, rule approx)
  have cf: ⋀i. continuous_on {min (l i) (u i)..max (l i) (u i)} f
    using approx f by (intro continuous_at_imp_continuous_on strip) auto
  have FTCi: ⋀i. (LBINT x=l i..u i. f x) = F (u i) − F (l i)
    apply (intro interval_integral_FTC_finite cf DERIV_subset [OF F])
   by (smt (verit) F aless approx(4) has_real_derivative_iff_has_vector_derivative
has_vector_derivative_at_within lessb)
  have 1: ⋀i. set_integrable lborel {l i..u i} f
   by (meson aless lessb assms(3) atLeastAtMost_iff borel_integrable_atLeastAtMost'
continuous_at_imp_continuous_on)
  have 2: set_borel_measurable lborel (einterval a b) f
    unfolding set_borel_measurable_def
   by (auto simp del: real_scaleR_def intro!: borel_measurable_continuous_on_indicator
            simp: continuous_on_eq_continuous_at einterval_iff f)
  have (λx. F (l x)) ⟶⟶ A
    using A approx unfolding tendsto_at_iff_sequentially comp_def
    by (force elim!: allE[of _ λi. ereal (l i)])
  moreover have (λx. F (u x)) ⟶⟶ B
```

```
    using B approx unfolding tendsto_at_iff_sequentially comp_def
    by (force elim!: allE[of _ λi. ereal (u i)])
  ultimately have 3: (λi. LBINT x=l i..u i. f x) ⟶ B − A
    by (simp add: FTCi tendsto_diff)
  show (LBINT x=a..b. f x) = B − A
    by (rule interval_integral_Icc_approx_nonneg [OF ‹a < b› approx 1 f_nonneg
2 3])
  show set_integrable lborel (einterval a b) f
    by (rule interval_integral_Icc_approx_nonneg [OF ‹a < b› approx 1 f_nonneg
2 3])
qed

theorem interval_integral_FTC_integrable:
  fixes f F :: real ⇒ 'a::euclidean_space and a b :: ereal
  assumes a < b
  assumes F: ⋀x. a < ereal x ⟹ ereal x < b ⟹ (F has_vector_derivative f x)
(at x)
  assumes f: ⋀x. a < ereal x ⟹ ereal x < b ⟹ isCont f x
  assumes f_integrable: set_integrable lborel (einterval a b) f
  assumes A: ((F ∘ real_of_ereal) ⟶ A) (at_right a)
  assumes B: ((F ∘ real_of_ereal) ⟶ B) (at_left b)
  shows (LBINT x=a..b. f x) = B − A
proof −
  obtain u l where approx:
    einterval a b = (⋃i. {l i .. u i})
    incseq u decseq l ⋀i. l i < u i ⋀i. a < l i ⋀i. u i < b
    l ⟶ a u ⟶ b
    by (blast intro: einterval_Icc_approximation[OF ‹a < b›])
  have [simp]: ⋀x i. l i ≤ x ⟹ a < ereal x
    by (rule order_less_le_trans, rule approx, force)
  have [simp]: ⋀x i. x ≤ u i ⟹ ereal x < b
   by (rule order_le_less_trans, subst ereal_less_eq(3), assumption, rule approx)
  have FTCi: ⋀i. (LBINT x=l i..u i. f x) = F (u i) − F (l i)
    using assms approx
    by (auto simp: less_imp_le min_def max_def
        intro!: f continuous_at_imp_continuous_on interval_integral_FTC_finite
            intro: has_vector_derivative_at_within)
  have (λi. LBINT x=l i..u i. f x) ⟶ B − A
    unfolding FTCi
  proof (intro tendsto_intros)
    show (λx. F (l x)) ⟶ A
      using A approx unfolding tendsto_at_iff_sequentially comp_def
      by (elim allE[of _ λi. ereal (l i)], auto)
    show (λx. F (u x)) ⟶ B
      using B approx unfolding tendsto_at_iff_sequentially comp_def
      by (elim allE[of _ λi. ereal (u i)], auto)
  qed
  moreover have (λi. LBINT x=l i..u i. f x) ⟶ (LBINT x=a..b. f x)
   by (rule interval_integral_Icc_approx_integrable [OF ‹a < b› approx f_integrable])
```

ultimately show *?thesis*
by (*elim LIMSEQ_unique*)
qed

theorem *interval_integral_FTC2*:
fixes *a b c* :: *real* **and** *f* :: *real* ⇒ *'a*::*euclidean_space*
assumes *a* ≤ *c c* ≤ *b*
and *contf*: *continuous_on* {*a..b*} *f*
fixes *x* :: *real*
assumes *a* ≤ *x* **and** *x* ≤ *b*
shows ((λ*u*. *LBINT y=c..u*. *f y*) *has_vector_derivative* (*f x*)) (*at x within* {*a..b*})
proof −
let *?F* = (λ*u*. *LBINT y=a..u*. *f y*)
have *intf*: *set_integrable lborel* {*a..b*} *f*
by (*rule borel_integrable_atLeastAtMost'*, *rule contf*)
have ((λ*u*. *integral* {*a..u*} *f*) *has_vector_derivative f x*) (*at x within* {*a..b*})
using ‹*a* ≤ *x*› ‹*x* ≤ *b*›
by (*auto intro*: *integral_has_vector_derivative continuous_on_subset* [*OF contf*])
then have ((λ*u*. *integral* {*a..u*} *f*) *has_vector_derivative* (*f x*)) (*at x within* {*a..b*})
by *simp*
then have (*?F has_vector_derivative* (*f x*)) (*at x within* {*a..b*})
by (*rule has_vector_derivative_weaken*)
(*auto intro*!: *assms interval_integral_eq_integral*[*symmetric*] *set_integrable_subset* [*OF intf*])
then have ((λ*x*. (*LBINT y=c..a*. *f y*) + *?F x*) *has_vector_derivative* (*f x*)) (*at x within* {*a..b*})
by (*auto intro*!: *derivative_eq_intros*)
then show *?thesis*
proof (*rule has_vector_derivative_weaken*)
fix *u* **assume** *u* ∈ {*a* .. *b*}
then show (*LBINT y=c..a*. *f y*) + (*LBINT y=a..u*. *f y*) = (*LBINT y=c..u*. *f y*)
using *assms*
apply (*intro interval_integral_sum*)
apply (*auto simp*: *interval_lebesgue_integrable_def simp del*: *real_scaleR_def*)
by (*rule set_integrable_subset* [*OF intf*], *auto simp*: *min_def max_def*)
qed (*insert assms*, *auto*)
qed

proposition *einterval_antiderivative*:
fixes *a b* :: *ereal* **and** *f* :: *real* ⇒ *'a*::*euclidean_space*
assumes *a* < *b* **and** *contf*: ⋀*x* :: *real*. *a* < *x* ⟹ *x* < *b* ⟹ *isCont f x*
shows ∃ *F*. ∀ *x* :: *real*. *a* < *x* ⟶ *x* < *b* ⟶ (*F has_vector_derivative f x*) (*at x*)
proof −
from *einterval_nonempty* [*OF* ‹*a* < *b*›] **obtain** *c* :: *real* **where** [*simp*]: *a* < *c c*

```
< b
    by (auto simp: einterval_def)
  let ?F = (λu. LBINT y=c..u. f y)
  show ?thesis
  proof (rule exI, clarsimp)
    fix x :: real
    assume [simp]: a < x x < b
    have 1: a < min c x by simp
    from einterval_nonempty [OF 1] obtain d :: real where [simp]: a < d d < c
d < x
      by (auto simp: einterval_def)
    have 2: max c x < b by simp
    from einterval_nonempty [OF 2] obtain e :: real where [simp]: c < e x < e
e < b
      by (auto simp: einterval_def)
    have (?F has_vector_derivative f x) (at x within {d<..<e})
    proof (rule has_vector_derivative_within_subset [of _ _ _ {d..e}])
      have continuous_on {d..e} f
      proof (intro continuous_at_imp_continuous_on ballI contf; clarsimp)
        show ⋀x. ⟦d ≤ x; x ≤ e⟧ ⟹ a < ereal x
          using ‹a < ereal d› ereal_less_ereal_Ex by auto
        show ⋀x. ⟦d ≤ x; x ≤ e⟧ ⟹ ereal x < b
          using ‹ereal e < b› ereal_less_eq(3) le_less_trans by blast
      qed
      then show (?F has_vector_derivative f x) (at x within {d..e})
        by (intro interval_integral_FTC2) (use ‹d < c› ‹c < e› ‹d < x› ‹x < e›
in ‹linarith+›)
    qed auto
    then show (?F has_vector_derivative f x) (at x)
      by (force simp: has_vector_derivative_within_open [of _ {d<..<e}])
  qed
qed
```

6.20.19 The substitution theorem

Once again, three versions: first, for finite intervals, and then two versions for arbitrary intervals.

```
theorem interval_integral_substitution_finite:
  fixes a b :: real and f :: real ⇒ 'a::euclidean_space
  assumes a ≤ b
  and derivg: ⋀x. a ≤ x ⟹ x ≤ b ⟹ (g has_real_derivative (g' x)) (at x within
{a..b})
  and contf : continuous_on (g ` {a..b}) f
  and contg': continuous_on {a..b} g'
  shows LBINT x=a..b. g' x *R f (g x) = LBINT y=g a..g b. f y
proof−
  have v_derivg: ⋀x. a ≤ x ⟹ x ≤ b ⟹ (g has_vector_derivative (g' x)) (at x
within {a..b})
    using derivg unfolding has_real_derivative_iff_has_vector_derivative .
```

then have *contg* [*simp*]: *continuous_on* {*a..b*} *g*
by (*rule continuous_on_vector_derivative*) *auto*
have *1*: ∃ *x*∈{*a..b*}. *u* = *g x* **if** *min* (*g a*) (*g b*) ≤ *u u* ≤ *max* (*g a*) (*g b*) **for** *u*
by (*cases g a* ≤ *g b*) (*use that assms IVT′* [*of g a u b*] *IVT2′* [*of g b u a*] **in** ‹*auto simp*: *min_def max_def*›)
obtain *c d* **where** *g_im*: *g* ‘ {*a..b*} = {*c..d*} **and** *c* ≤ *d*
by (*metis continuous_image_closed_interval contg* ‹*a* ≤ *b*›)
obtain *F* **where** *derivF*:
⋀*x*. ⟦*a* ≤ *x*; *x* ≤ *b*⟧ ⟹ (*F has_vector_derivative* (*f* (*g x*))) (*at* (*g x*) *within* (*g* ‘ {*a..b*}))
using *continuous_on_subset* [*OF contf*] *g_im*
by (*metis antiderivative_continuous atLeastAtMost_iff image_subset_iff set_eq_subset*)
have *contfg*: *continuous_on* {*a..b*} (λ*x*. *f* (*g x*))
by (*blast intro*: *continuous_on_compose2 contf contg*)
have *continuous_on* {*a..b*} (λ*x*. *g′ x* ∗*R f* (*g x*))
by (*auto intro*!: *continuous_on_scaleR contg′ contfg*)
then have *LBINT x. indicat_real* {*a..b*} *x* ∗*R g′ x* ∗*R f* (*g x*) = *F* (*g b*) − *F* (*g a*)
using *integral_FTC_atLeastAtMost* [*OF* ‹*a* ≤ *b*› *vector_diff_chain_within*[*OF v_derivg derivF*]]
by *force*
then have *LBINT x*=*a..b. g′ x* ∗*R f* (*g x*) = *F* (*g b*) − *F* (*g a*)
by (*simp add*: *assms interval_integral_Icc set_lebesgue_integral_def*)
moreover have *LBINT y*=(*g a*)..(*g b*). *f y* = *F* (*g b*) − *F* (*g a*)
proof (*rule interval_integral_FTC_finite*)
show *continuous_on* {*min* (*g a*) (*g b*)..*max* (*g a*) (*g b*)} *f*
by (*rule continuous_on_subset* [*OF contf*]) (*auto simp*: *image_def 1*)
show (*F has_vector_derivative f y*) (*at y within* {*min* (*g a*) (*g b*)..*max* (*g a*) (*g b*)})
if *y*: *min* (*g a*) (*g b*) ≤ *y y* ≤ *max* (*g a*) (*g b*) **for** *y*
proof −
obtain *x* **where** *a* ≤ *x x* ≤ *b y* = *g x*
using *1 y* **by** *force*
then show *?thesis*
by (*auto simp*: *image_def intro*!: *1 has_vector_derivative_within_subset* [*OF derivF*])
qed
qed
ultimately show *?thesis* **by** *simp*
qed

theorem *interval_integral_substitution_integrable*:
fixes *f* :: *real* ⇒ *′a*::*euclidean_space* **and** *a b u v* :: *ereal*
assumes *a* < *b*
and *deriv_g*: ⋀*x*. *a* < *ereal x* ⟹ *ereal x* < *b* ⟹ *DERIV g x* :> *g′ x*
and *contf*: ⋀*x*. *a* < *ereal x* ⟹ *ereal x* < *b* ⟹ *isCont f* (*g x*)

and *contg'*: $\bigwedge x.\ a < ereal\ x \Longrightarrow ereal\ x < b \Longrightarrow isCont\ g'\ x$
and *g'_nonneg*: $\bigwedge x.\ a \leq ereal\ x \Longrightarrow ereal\ x \leq b \Longrightarrow 0 \leq g'\ x$
and *A*: $((ereal \circ g \circ real\_of\_ereal) \longrightarrow A)\ (at\_right\ a)$
and *B*: $((ereal \circ g \circ real\_of\_ereal) \longrightarrow B)\ (at\_left\ b)$
and *integrable*: *set_integrable lborel* $(einterval\ a\ b)\ (\lambda x.\ g'\ x *_R f\ (g\ x))$
and *integrable2*: *set_integrable lborel* $(einterval\ A\ B)\ (\lambda x.\ f\ x)$
shows $(LBINT\ x{=}A..B.\ f\ x) = (LBINT\ x{=}a..b.\ g'\ x *_R f\ (g\ x))$
proof −
obtain *u l* **where** *approx* [*simp*]:
$einterval\ a\ b = (\bigcup i.\ \{l\ i\ ..\ u\ i\})$
$incseq\ u\ decseq\ l\ \bigwedge i.\ l\ i < u\ i\ \bigwedge i.\ a < l\ i\ \bigwedge i.\ u\ i < b$
$l \longrightarrow a\ u \longrightarrow b$
by (*blast intro*: *einterval_Icc_approximation*[*OF* ‹$a < b$›])
note *less_imp_le* [*simp*]
have [*simp*]: $\bigwedge x\ i.\ l\ i \leq x \Longrightarrow a < ereal\ x$
by (*rule order_less_le_trans*, *rule approx*, *force*)
have [*simp*]: $\bigwedge x\ i.\ x \leq u\ i \Longrightarrow ereal\ x < b$
by (*rule order_le_less_trans*, *subst ereal_less_eq*(*3*), *assumption*, *rule approx*)
then have *lessb*[*simp*]: $\bigwedge i.\ l\ i < b$
using *approx*(*4*) *less_eq_real_def* **by** *blast*
have [*simp*]: $\bigwedge i.\ a < u\ i$
by (*rule order_less_trans*, *rule approx*, *auto*, *rule approx*)
have *lle*[*simp*]: $\bigwedge i\ j.\ i \leq j \Longrightarrow l\ j \leq l\ i$ **by** (*rule decseqD*, *rule approx*)
have [*simp*]: $\bigwedge i\ j.\ i \leq j \Longrightarrow u\ i \leq u\ j$ **by** (*rule incseqD*, *rule approx*)
have *g_nondec* [*simp*]: $g\ x \leq g\ y$ **if** $a < x\ x \leq y\ y < b$ **for** $x\ y$
proof (*rule DERIV_nonneg_imp_nondecreasing* [*OF* ‹$x \leq y$›], *intro exI conjI*)
show $\bigwedge u.\ x \leq u \Longrightarrow u \leq y \Longrightarrow (g\ has\_real\_derivative\ g'\ u)\ (at\ u)$
by (*meson deriv_g ereal_less_eq*(*3*) *le_less_trans less_le_trans that*)
show $\bigwedge u.\ x \leq u \Longrightarrow u \leq y \Longrightarrow 0 \leq g'\ u$
by (*meson assms*(*5*) *dual_order.trans le_ereal_le less_imp_le order_refl that*)
qed
have $A \leq B$ **and** *un*: $einterval\ A\ B = (\bigcup i.\ \{g(l\ i)<..<g(u\ i)\})$
proof −
have *A2*: $(\lambda i.\ g\ (l\ i)) \longrightarrow A$
using *A* **apply** (*auto simp*: *einterval_def tendsto_at_iff_sequentially comp_def*)
by (*drule_tac* $x = \lambda i.\ ereal\ (l\ i)$ **in** *spec*, *auto*)
hence *A3*: $\bigwedge i.\ g\ (l\ i) \geq A$
by (*intro decseq_ge*, *auto simp*: *decseq_def*)
have *B2*: $(\lambda i.\ g\ (u\ i)) \longrightarrow B$
using *B* **apply** (*auto simp*: *einterval_def tendsto_at_iff_sequentially comp_def*)
by (*drule_tac* $x = \lambda i.\ ereal\ (u\ i)$ **in** *spec*, *auto*)
hence *B3*: $\bigwedge i.\ g\ (u\ i) \leq B$
by (*intro incseq_le*, *auto simp*: *incseq_def*)
have $ereal\ (g\ (l\ 0)) \leq ereal\ (g\ (u\ 0))$
by *auto*
then show $A \leq B$
by (*meson A3 B3 order.trans*)
{ fix $x :: real$

```
    assume A < x and x < B
   then have eventually (λi. ereal (g (l i)) < x ∧ x < ereal (g (u i))) sequentially
     by (fast intro: eventually_conj order_tendstoD A2 B2)
    hence ∃ i. g (l i) < x ∧ x < g (u i)
     by (simp add: eventually_sequentially, auto)
  } note AB = this
  show einterval A B = (⋃ i. {g(l i)<..<g(u i)})
  proof
    show einterval A B ⊆ (⋃ i. {g(l i)<..<g(u i)})
     by (auto simp: einterval_def AB)
    show (⋃ i. {g(l i)<..<g(u i)}) ⊆ einterval A B
    proof (clarsimp simp add: einterval_def, intro conjI)
      show ⋀x i. ⟦g (l i) < x; x < g (u i)⟧ ⟹ A < ereal x
       using A3 le_ereal_less by blast
      show ⋀x i. ⟦g (l i) < x; x < g (u i)⟧ ⟹ ereal x < B
       using B3 ereal_le_less by blast
    qed
  qed
qed

  have eq1: (LBINT x=l i.. u i. g' x *R f (g x)) = (LBINT y=g (l i)..g (u i). f
y) for i
    apply (rule interval_integral_substitution_finite [OF _ DERIV_subset [OF
deriv_g]])
   unfolding has_real_derivative_iff_has_vector_derivative[symmetric]
       apply (auto intro!: continuous_at_imp_continuous_on contf contg')
   done
  have (λi. LBINT x=l i..u i. g' x *R f (g x)) ⟶ (LBINT x=a..b. g' x *R f
(g x))
   using approx(4) ‹a < b› integrable interval_integral_Icc_approx_integrable by
fastforce
  hence 2: (λi. (LBINT y=g (l i)..g (u i). f y)) ⟶ (LBINT x=a..b. g' x *R f
(g x))
    by (simp add: eq1)
  have incseq: incseq (λi. {g (l i)<..<g (u i)})
   apply (clarsimp simp: incseq_def, intro conjI)
   using lessb lle approx(5) g_nondec le_less_trans apply blast
   by (force intro: less_le_trans)
  have (λi. set_lebesgue_integral lborel {g (l i)<..<g (u i)} f)
       ⟶ set_lebesgue_integral lborel (einterval A B) f
    unfolding un  by (rule set_integral_cont_up) (use incseq  integrable2 un in
auto)
  then have (λi. (LBINT y=g (l i)..g (u i). f y)) ⟶ (LBINT x = A..B. f x)
   by (simp add: interval_lebesgue_integral_le_eq ‹A ≤ B›)
  thus ?thesis by (intro LIMSEQ_unique [OF _ 2])
qed
```

theorem *interval_integral_substitution_nonneg*:
fixes $f\ g\ g'$:: *real* ⇒ *real* **and** $a\ b\ u\ v$ :: *ereal*
assumes $a < b$
and *deriv_g*: $\bigwedge x.\ a < ereal\ x \Longrightarrow ereal\ x < b \Longrightarrow DERIV\ g\ x :> g'\ x$
and *contf*: $\bigwedge x.\ a < ereal\ x \Longrightarrow ereal\ x < b \Longrightarrow isCont\ f\ (g\ x)$
and *contg'*: $\bigwedge x.\ a < ereal\ x \Longrightarrow ereal\ x < b \Longrightarrow isCont\ g'\ x$
and *f_nonneg*: $\bigwedge x.\ a < ereal\ x \Longrightarrow ereal\ x < b \Longrightarrow 0 \le f\ (g\ x)$
and *g'_nonneg*: $\bigwedge x.\ a \le ereal\ x \Longrightarrow ereal\ x \le b \Longrightarrow 0 \le g'\ x$
and *A*: $((ereal \circ g \circ real\_of\_ereal) \longrightarrow A)\ (at\_right\ a)$
and *B*: $((ereal \circ g \circ real\_of\_ereal) \longrightarrow B)\ (at\_left\ b)$
and *integrable_fg*: *set_integrable lborel* $(einterval\ a\ b)\ (\lambda x.\ f\ (g\ x) * g'\ x)$
shows
set_integrable lborel $(einterval\ A\ B)\ f$
$(LBINT\ x{=}A..B.\ f\ x) = (LBINT\ x{=}a..b.\ (f\ (g\ x) * g'\ x))$
proof −
from *einterval_Icc_approximation*[*OF* ‹$a < b$›] **obtain** $u\ l$ **where** *approx* [*simp*]:
$einterval\ a\ b = (\bigcup i.\ \{l\ i..u\ i\})$
incseq u
decseq l
$\bigwedge i.\ l\ i < u\ i$
$\bigwedge i.\ a < ereal\ (l\ i)$
$\bigwedge i.\ ereal\ (u\ i) < b$
$(\lambda x.\ ereal\ (l\ x)) \longrightarrow a$
$(\lambda x.\ ereal\ (u\ x)) \longrightarrow b$ **by** *this auto*
have *aless*[*simp*]: $\bigwedge x\ i.\ l\ i \le x \Longrightarrow a < ereal\ x$
by (*rule order_less_le_trans*, *rule approx*, *force*)
have *lessb*[*simp*]: $\bigwedge x\ i.\ x \le u\ i \Longrightarrow ereal\ x < b$
by (*rule order_le_less_trans*, *subst ereal_less_eq*(*3*), *assumption*, *rule approx*)
have *llb*[*simp*]: $\bigwedge i.\ l\ i < b$
using *lessb approx*(*4*) *less_eq_real_def* **by** *blast*
have *alu*[*simp*]: $\bigwedge i.\ a < u\ i$
by (*rule order_less_trans*, *rule approx*, *auto*, *rule approx*)
have [*simp*]: $\bigwedge i\ j.\ i \le j \Longrightarrow l\ j \le l\ i$ **by** (*rule decseqD*, *rule approx*)
have *uleu*[*simp*]: $\bigwedge i\ j.\ i \le j \Longrightarrow u\ i \le u\ j$ **by** (*rule incseqD*, *rule approx*)
have *g_nondec* [*simp*]: $g\ x \le g\ y$ **if** $a < x\ x \le y\ y < b$ **for** $x\ y$
proof (*rule DERIV_nonneg_imp_nondecreasing* [*OF* ‹$x \le y$›], *intro exI conjI*)
show $\bigwedge u.\ x \le u \Longrightarrow u \le y \Longrightarrow (g\ has\_real\_derivative\ g'\ u)\ (at\ u)$
by (*meson deriv_g ereal_less_eq*(*3*) *le_less_trans less_le_trans that*)
show $\bigwedge u.\ x \le u \Longrightarrow u \le y \Longrightarrow 0 \le g'\ u$
by (*meson g'_nonneg less_ereal.simps*(*1*) *less_trans not_less that*)
qed
have $A \le B$ **and** *un*: $einterval\ A\ B = (\bigcup i.\ \{g(l\ i)<..<g(u\ i)\})$
proof −
have *A2*: $(\lambda i.\ g\ (l\ i)) \longrightarrow A$
using *A* **by** (*force simp*: *einterval_def tendsto_at_iff_sequentially comp_def elim*!: *allE*[**where** $x = \lambda i.\ ereal\ (l\ i)$])
hence *A3*: $\bigwedge i.\ g\ (l\ i) \ge A$
by (*intro decseq_ge*, *auto simp*: *decseq_def*)
have *B2*: $(\lambda i.\ g\ (u\ i)) \longrightarrow B$

```
    using B by (force simp: einterval_def tendsto_at_iff_sequentially comp_def
elim!: allE[where x = λi. ereal (u i)])
   hence B3: ⋀i. g (u i) ≤ B
    by (intro incseq_le, auto simp: incseq_def)
   have ereal (g (l 0)) ≤ ereal (g (u 0))
    by (auto simp: less_imp_le)
   then show A ≤ B
    by (meson A3 B3 order.trans)
   { fix x :: real
    assume A < x and x < B
   then have eventually (λi. ereal (g (l i)) < x ∧ x < ereal (g (u i))) sequentially
     by (fast intro: eventually_conj order_tendstoD A2 B2)
    hence ∃ i. g (l i) < x ∧ x < g (u i)
     by (simp add: eventually_sequentially, auto)
   } note AB = this
   show einterval A B = (⋃ i. {g(l i)<..<g(u i)})
   proof
    show einterval A B ⊆ (⋃ i. {g (l i)<..<g (u i)})
     by (auto simp: einterval_def AB)
    show (⋃ i. {g (l i)<..<g (u i)}) ⊆ einterval A B
    using A3 B3 by (force simp: einterval_def intro: le_ereal_less ereal_le_less)
   qed
  qed

  have eq1: (LBINT x=l i.. u i. (f (g x) * g' x)) = (LBINT y=g (l i)..g (u i). f
y) for i
  proof −
   have (LBINT x=l i.. u i. g' x *R f (g x)) = (LBINT y=g (l i)..g (u i). f y)
    apply (rule interval_integral_substitution_finite [OF _ DERIV_subset [OF
deriv_g]])
    unfolding has_real_derivative_iff_has_vector_derivative[symmetric]
      apply (auto simp: less_imp_le intro!: continuous_at_imp_continuous_on
contf contg')
    done
   then show ?thesis
    by (simp add: ac_simps)
  qed
  have incseq: incseq (λi. {g (l i)<..<g (u i)})
   apply (clarsimp simp: incseq_def, intro conjI)
   apply (meson llb antimono_def approx(3) approx(5) g_nondec le_less_trans)
   using alu uleu approx(6) g_nondec less_le_trans by blast
  have img: ∃ c ≥ l i. c ≤ u i ∧ x = g c if g (l i) ≤ x x ≤ g (u i) for x i
  proof −
   have continuous_on {l i..u i} g
    by (force intro!: DERIV_isCont deriv_g continuous_at_imp_continuous_on)
   with that show ?thesis
    using IVT' [of g] approx(4) dual_order.strict_implies_order by blast
  qed
  have continuous_on {g (l i)..g (u i)} f for i
```

using *contf img* **by** (*force simp add*: *intro*!: *continuous_at_imp_continuous_on*)
then have *int_f*: $\bigwedge i.$ *set_integrable lborel* $\{g\ (l\ i)<..<g\ (u\ i)\}\ f$
by (*rule set_integrable_subset* [*OF borel_integrable_atLeastAtMost'*]) (*auto intro*: *less_imp_le*)
have *integrable*: *set_integrable lborel* $(\bigcup i.\ \{g\ (l\ i)<..<g\ (u\ i)\})\ f$
proof (*intro pos_integrable_to_top incseq int_f*)
let *?l* = (*LBINT x=a..b. f* (*g x*) $*$ g' *x*)
have $(\lambda i.$ *LBINT x=l i..u i. f* (*g x*) $*$ g' *x*) $\longrightarrow$ *?l*
by (*intro assms interval_integral_Icc_approx_integrable* [*OF* ‹$a < b$› *approx*])
hence $(\lambda i.$ (*LBINT y=g* (*l i*)..*g* (*u i*). *f y*)) $\longrightarrow$ *?l*
by (*simp add*: *eq1*)
then show $(\lambda i.$ *set_lebesgue_integral lborel* $\{g\ (l\ i)<..<g\ (u\ i)\}\ f) \longrightarrow$ *?l*
unfolding *interval_lebesgue_integral_def* **by** (*auto simp*: *less_imp_le*)
have $\bigwedge x\ i.\ g\ (l\ i) \le x \Longrightarrow x \le g\ (u\ i) \Longrightarrow 0 \le f\ x$
using *aless f_nonneg img lessb* **by** *blast*
then show $\bigwedge x\ i.\ x \in \{g\ (l\ i)<..<g\ (u\ i)\} \Longrightarrow 0 \le f\ x$
using *less_eq_real_def* **by** *auto*
qed (*auto simp*: *greaterThanLessThan_borel*)
thus *set_integrable lborel* (*einterval A B*) *f*
by (*simp add*: *un*)

have (*LBINT x=A..B. f x*) = (*LBINT x=a..b.* g' *x* $*_R$ *f* (*g x*))
proof (*rule interval_integral_substitution_integrable*)
show *set_integrable lborel* (*einterval a b*) $(\lambda x.$ g' *x* $*_R$ *f* (*g x*))
using *integrable_fg* **by** (*simp add*: *ac_simps*)
qed *fact+*
then show (*LBINT x=A..B. f x*) = (*LBINT x=a..b.* (*f* (*g x*) $*$ g' *x*))
by (*simp add*: *ac_simps*)
qed

syntax *_complex_lebesgue_borel_integral* :: *pttrn* $\Rightarrow$ *real* $\Rightarrow$ *complex*
((2*CLBINT* *_*. *_*) [*0*,*60*] *60*)

translations *CLBINT x. f* == *CONST complex_lebesgue_integral CONST lborel* $(\lambda x.$ *f*)

syntax *_complex_set_lebesgue_borel_integral* :: *pttrn* $\Rightarrow$ *real set* $\Rightarrow$ *real* $\Rightarrow$ *complex*
((3*CLBINT* *_*:*_*. *_*) [*0*,*60*,*61*] *60*)

translations
CLBINT x:*A. f* == *CONST complex_set_lebesgue_integral CONST lborel A* $(\lambda x.$ *f*)

abbreviation *complex_interval_lebesgue_integral* ::
real measure $\Rightarrow$ *ereal* $\Rightarrow$ *ereal* $\Rightarrow$ (*real* $\Rightarrow$ *complex*) $\Rightarrow$ *complex* **where**
complex_interval_lebesgue_integral M a b f $\equiv$ *interval_lebesgue_integral M a b f*

abbreviation *complex_interval_lebesgue_integrable* ::
real measure ⇒ *ereal* ⇒ *ereal* ⇒ (*real* ⇒ *complex*) ⇒ *bool* **where**
complex_interval_lebesgue_integrable M a b f ≡ *interval_lebesgue_integrable M a b f*

syntax
_ascii_complex_interval_lebesgue_borel_integral :: *pttrn* ⇒ *ereal* ⇒ *ereal* ⇒ *real* ⇒ *complex*
((*4CLBINT _=_..*_. _) [*0*,*60*,*60*,*61*] *60*)

translations
CLBINT x=a..b. f == *CONST complex_interval_lebesgue_integral CONST lborel a b* (λ*x. f*)

proposition *interval_integral_norm*:
fixes *f* :: *real* ⇒ *'a* :: {*banach*, *second_countable_topology*}
shows *interval_lebesgue_integrable lborel a b f* ⟹ *a* ≤ *b* ⟹
norm (*LBINT t=a..b. f t*) ≤ *LBINT t=a..b. norm* (*f t*)
using *integral_norm_bound*[*of lborel* λ*x. indicator* (*einterval a b*) *x* ∗*R f x*]
by (*auto simp*: *interval_lebesgue_integral_def interval_lebesgue_integrable_def set_lebesgue_integral_def*)

proposition *interval_integral_norm2*:
interval_lebesgue_integrable lborel a b f ⟹
norm (*LBINT t=a..b. f t*) ≤ |*LBINT t=a..b. norm* (*f t*)|
proof (*induct a b rule*: *linorder_wlog*)
case (*sym a b*) **then show** *?case*
by (*simp add*: *interval_integral_endpoints_reverse*[*of a b*] *interval_integrable_endpoints_reverse*[*of a b*])
next
case (*le a b*)
then have |*LBINT t=a..b. norm* (*f t*)| = *LBINT t=a..b. norm* (*f t*)
using *integrable_norm*[*of lborel* λ*x. indicator* (*einterval a b*) *x* ∗*R f x*]
by (*auto simp*: *interval_lebesgue_integral_def interval_lebesgue_integrable_def set_lebesgue_integral_def*
intro!: *integral_nonneg_AE abs_of_nonneg*)
then show *?case*
using *le* **by** (*simp add*: *interval_integral_norm*)
qed

lemma *integral_cos*: *t* ≠ *0* ⟹ *LBINT x=a..b. cos* (*t* ∗ *x*) = *sin* (*t* ∗ *b*) / *t* − *sin* (*t* ∗ *a*) / *t*
apply (*intro interval_integral_FTC_finite continuous_intros*)
by (*auto intro*!: *derivative_eq_intros simp*: *has_real_derivative_iff_has_vector_derivative*[*symmetric*])

end

6.21 Integration by Substition for the Lebesgue Integral

theory *Lebesgue_Integral_Substitution*
imports *Interval_Integral*
begin

lemma *nn_integral_substitution_aux*:
fixes $f :: real \Rightarrow ennreal$
assumes *Mf*: $f \in borel\_measurable\ borel$
assumes *nonnegf*: $\bigwedge x.\ f\ x \geq 0$
assumes *derivg*: $\bigwedge x.\ x \in \{a..b\} \Longrightarrow (g\ has\_real\_derivative\ g'\ x)\ (at\ x)$
assumes *contg'*: $continuous\_on\ \{a..b\}\ g'$
assumes *derivg_nonneg*: $\bigwedge x.\ x \in \{a..b\} \Longrightarrow g'\ x \geq 0$
assumes $a < b$
shows $(\int^+ x.\ f\ x * indicator\ \{g\ a..g\ b\}\ x\ \partial lborel) =$
$(\int^+ x.\ f\ (g\ x) * g'\ x * indicator\ \{a..b\}\ x\ \partial lborel)$
proof−
from ‹$a < b$› **have** [*simp*]: $a \leq b$ **by** *simp*
from *derivg* **have** *contg*: $continuous\_on\ \{a..b\}\ g$ **by** (*rule has_real_derivative_imp_continuous_on*)
from *this* **and** *contg'* **have** *Mg*: $set\_borel\_measurable\ borel\ \{a..b\}\ g$ **and**
Mg': $set\_borel\_measurable\ borel\ \{a..b\}\ g'$
by (*simp_all only*: *set_measurable_continuous_on_ivl*)
from *derivg* **have** *derivg'*: $\bigwedge x.\ x \in \{a..b\} \Longrightarrow (g\ has\_vector\_derivative\ g'\ x)\ (at\ x)$
by (*simp only*: *has_real_derivative_iff_has_vector_derivative*)

have *real_ind*[*simp*]: $\bigwedge A\ x.\ enn2real\ (indicator\ A\ x) = indicator\ A\ x$
by (*auto split*: *split_indicator*)
have *ennreal_ind*[*simp*]: $\bigwedge A\ x.\ ennreal\ (indicator\ A\ x) = indicator\ A\ x$
by (*auto split*: *split_indicator*)
have [*simp*]: $\bigwedge x\ A.\ indicator\ A\ (g\ x) = indicator\ (g -` A)\ x$
by (*auto split*: *split_indicator*)

from *derivg derivg_nonneg* **have** *monog*: $\bigwedge x\ y.\ a \leq x \Longrightarrow x \leq y \Longrightarrow y \leq b \Longrightarrow g\ x \leq g\ y$
by (*rule deriv_nonneg_imp_mono*) *simp_all*
with *monog* **have** [*simp*]: $g\ a \leq g\ b$ **by** (*auto intro*: *mono_onD*)

show *?thesis*
proof (*induction rule*: *borel_measurable_induct*[*OF Mf*, *case_names cong set mult add sup*])
case (*cong f1 f2*)
from *cong.hyps*(*3*) **have** *f1* = *f2* **by** *auto*
with *cong* **show** *?case* **by** *simp*
next
case (*set A*)
from *set.hyps* **show** *?case*

```
  proof (induction rule: borel_set_induct)
    case empty
    thus ?case by simp
  next
    case (interval c d)
    {
      fix u v :: real assume asm: {u..v} ⊆ {g a..g b} u ≤ v

      obtain u' v' where u'v': {a..b} ∩ g-`{u..v} = {u'..v'} u' ≤ v' g u' = u g
v' = v
          using asm by (rule_tac continuous_interval_vimage_Int[OF contg
monog, of u v]) simp_all
      hence {u'..v'} ⊆ {a..b} {u'..v'} ⊆ g -` {u..v} by blast+
      with u'v'(2) have u' ∈ g -` {u..v} v' ∈ g -` {u..v} by auto
      from u'v'(1) have [simp]: {a..b} ∩ g -` {u..v} ∈ sets borel by simp

      have A: continuous_on {min u' v'..max u' v'} g'
        by (simp only: u'v' max_absorb2 min_absorb1)
          (intro continuous_on_subset[OF contg'], insert u'v', auto)
      have ⋀x. x ∈ {u'..v'} ⟹ (g has_real_derivative g' x) (at x within {u'..v'})
        using asm by (intro has_field_derivative_subset[OF derivg] subsetD[OF
‹{u'..v'} ⊆ {a..b}›]) auto
      hence B: ⋀x. min u' v' ≤ x ⟹ x ≤ max u' v' ⟹
                  (g has_vector_derivative g' x) (at x within {min u' v'..max u'
v'})
        by (simp only: u'v' max_absorb2 min_absorb1)
          (auto simp: has_real_derivative_iff_has_vector_derivative)
       have integrable lborel (λx. indicator ({a..b} ∩ g -` {u..v}) x *R g' x)
        using set_integrable_subset borel_integrable_atLeastAtMost'[OF contg']
        by (metis ‹{u'..v'} ⊆ {a..b}› eucl_ivals(5) set_integrable_def sets_lborel
u'v'(1))
      hence (∫+x. ennreal (g' x) * indicator ({a..b} ∩ g-` {u..v}) x ∂lborel) =
              LBINT x:{a..b} ∩ g-`{u..v}. g' x
        unfolding set_lebesgue_integral_def
        by (subst nn_integral_eq_integral[symmetric])
          (auto intro!: derivg_nonneg nn_integral_cong split: split_indicator)
      also from interval_integral_FTC_finite[OF A B]
        have LBINT x:{a..b} ∩ g-`{u..v}. g' x = v - u
          by (simp add: u'v' interval_integral_Icc ‹u ≤ v›)
       finally have (∫+ x. ennreal (g' x) * indicator ({a..b} ∩ g -` {u..v}) x
∂lborel) =
                      ennreal (v - u) .
    } note A = this

    have (∫+x. indicator {c..d} (g x) * ennreal (g' x) * indicator {a..b} x ∂lborel)
=
            (∫+ x. ennreal (g' x) * indicator ({a..b} ∩ g -` {c..d}) x ∂lborel)
      by (intro nn_integral_cong) (simp split: split_indicator)
    also have {a..b} ∩ g-`{c..d} = {a..b} ∩ g-`{max (g a) c..min (g b) d}
```

```
    using ‹a ≤ b› ‹c ≤ d›
    by (auto intro!: monog intro: order.trans)
  also have (∫⁺ x. ennreal (g' x) * indicator ... x ∂lborel) =
    (if max (g a) c ≤ min (g b) d then min (g b) d − max (g a) c else 0)
     using ‹c ≤ d› by (simp add: A)
  also have ... = (∫⁺ x. indicator ({g a..g b} ∩ {c..d}) x ∂lborel)
    by (subst nn_integral_indicator) (auto intro!: measurable_sets Mg simp:)
  also have ... = (∫⁺ x. indicator {c..d} x * indicator {g a..g b} x ∂lborel)
    by (intro nn_integral_cong) (auto split: split_indicator)
  finally show ?case ..

  next

  case (compl A)
  note ‹A ∈ sets borel›[measurable]
  from emeasure_mono[of A ∩ {g a..g b} {g a..g b} lborel]
 have [simp]: emeasure lborel (A ∩ {g a..g b}) ≠ top by (auto simp: top_unique)
  have [simp]: g −‘ A ∩ {a..b} ∈ sets borel
    by (rule set_borel_measurable_sets[OF Mg]) auto
  have [simp]: g −‘ (−A) ∩ {a..b} ∈ sets borel
    by (rule set_borel_measurable_sets[OF Mg]) auto

  have (∫⁺x. indicator (−A) x * indicator {g a..g b} x ∂lborel) =
          (∫⁺x. indicator (−A ∩ {g a..g b}) x ∂lborel)
    by (rule nn_integral_cong) (simp split: split_indicator)
    also from compl have ... = emeasure lborel ({g a..g b} − A) using de-
rivg_nonneg
    by (simp add: vimage_Compl diff_eq Int_commute[of −A])
  also have {g a..g b} − A = {g a..g b} − A ∩ {g a..g b} by blast
  also have emeasure lborel ... = g b − g a − emeasure lborel (A ∩ {g a..g b})
        using ‹A ∈ sets borel› by (subst emeasure_Diff) auto
 also have emeasure lborel (A ∩ {g a..g b}) =
              ∫⁺x. indicator A x * indicator {g a..g b} x ∂lborel
   using ‹A ∈ sets borel›
   by (subst nn_integral_indicator[symmetric], simp, intro nn_integral_cong)
     (simp split: split_indicator)
  also have ... = ∫⁺ x. indicator (g−‘A ∩ {a..b}) x * ennreal (g' x * indicator
{a..b} x) ∂lborel (is _ = ?I)
    by (subst compl.IH, intro nn_integral_cong) (simp split: split_indicator)
  also have g b − g a = LBINT x:{a..b}. g' x using derivg'
    unfolding set_lebesgue_integral_def
    by (intro integral_FTC_atLeastAtMost[symmetric])
    (auto intro: continuous_on_subset[OF contg'] has_field_derivative_subset[OF
derivg]
          has_vector_derivative_at_within)
  also have ennreal ... = ∫⁺ x. g' x * indicator {a..b} x ∂lborel
  using borel_integrable_atLeastAtMost'[OF contg'] unfolding set_lebesgue_integral_def
    by (subst nn_integral_eq_integral)
       (simp_all add: mult.commute derivg_nonneg set_integrable_def split:
```

split_indicator)
also have *Mg''*: (λ*x*. *indicator* (*g* −‘ *A* ∩ {*a*..*b*}) *x* ∗ *ennreal* (*g'* *x* ∗ *indicator* {*a*..*b*} *x*))
∈ *borel_measurable borel* **using** *Mg'*
by (*intro borel_measurable_times_ennreal borel_measurable_indicator*)
(*simp_all add*: *mult.commute set_borel_measurable_def*)
have *le*: (∫ +*x*. *indicator* (*g*−‘*A* ∩ {*a*..*b*}) *x* ∗ *ennreal* (*g'* *x* ∗ *indicator* {*a*..*b*} *x*) *∂lborel*) ≤
(∫ +*x*. *ennreal* (*g'* *x*) ∗ *indicator* {*a*..*b*} *x* *∂lborel*)
by (*intro nn_integral_mono*) (*simp split*: *split_indicator add*: *derivg_nonneg*)
note *integrable* = *borel_integrable_atLeastAtMost'*[*OF contg'*]
with *le* **have** *notinf*: (∫ +*x*. *indicator* (*g*−‘*A* ∩ {*a*..*b*}) *x* ∗ *ennreal* (*g'* *x* ∗ *indicator* {*a*..*b*} *x*) *∂lborel*) ≠ *top*
by (*auto simp*: *real_integrable_def nn_integral_set_ennreal mult.commute top_unique set_integrable_def*)
have (∫ + *x*. *g'* *x* ∗ *indicator* {*a*..*b*} *x* *∂lborel*) − *?I* =
∫ + *x*. *ennreal* (*g'* *x* ∗ *indicator* {*a*..*b*} *x*) −
indicator (*g* −‘ *A* ∩ {*a*..*b*}) *x* ∗ *ennreal* (*g'* *x* ∗ *indicator* {*a*..*b*} *x*) *∂lborel*
apply (*intro nn_integral_diff*[*symmetric*])
apply (*insert Mg'*, *simp add*: *mult.commute set_borel_measurable_def*) []
apply (*insert Mg''*, *simp*) []
apply (*simp split*: *split_indicator add*: *derivg_nonneg*)
apply (*rule notinf*)
apply (*simp split*: *split_indicator add*: *derivg_nonneg*)
done
also have ... = ∫ + *x*. *indicator* (−*A*) (*g x*) ∗ *ennreal* (*g'* *x*) ∗ *indicator* {*a*..*b*} *x* *∂lborel*
by (*intro nn_integral_cong*) (*simp split*: *split_indicator*)
finally show *?case* **.**

next
case (*union f*)
then have [*simp*]: ⋀*i*. {*a*..*b*} ∩ *g* −‘ *f i* ∈ *sets borel*
by (*subst Int_commute*, *intro set_borel_measurable_sets*[*OF Mg*]) *auto*
have *g* −‘ (⋃ *i*. *f i*) ∩ {*a*..*b*} = (⋃ *i*. {*a*..*b*} ∩ *g* −‘ *f i*) **by** *auto*
hence *g* −‘ (⋃ *i*. *f i*) ∩ {*a*..*b*} ∈ *sets borel* **by** (*auto simp del*: *UN_simps*)

have (∫ +*x*. *indicator* (⋃ *i*. *f i*) *x* ∗ *indicator* {*g a*..*g b*} *x* *∂lborel*) =
∫ +*x*. *indicator* (⋃ *i*. {*g a*..*g b*} ∩ *f i*) *x* *∂lborel*
by (*intro nn_integral_cong*) (*simp split*: *split_indicator*)
also from *union* **have** ... = *emeasure lborel* (⋃ *i*. {*g a*..*g b*} ∩ *f i*) **by** *simp*
also from *union* **have** ... = (∑ *i*. *emeasure lborel* ({*g a*..*g b*} ∩ *f i*))
by (*intro suminf_emeasure*[*symmetric*]) (*auto simp*: *disjoint_family_on_def*)
also from *union* **have** ... = (∑ *i*. ∫ +*x*. *indicator* ({*g a*..*g b*} ∩ *f i*) *x* *∂lborel*) **by** *simp*
also have (λ*i*. ∫ +*x*. *indicator* ({*g a*..*g b*} ∩ *f i*) *x* *∂lborel*) =
(λ*i*. ∫ +*x*. *indicator* (*f i*) *x* ∗ *indicator* {*g a*..*g b*} *x* *∂lborel*)
by (*intro ext nn_integral_cong*) (*simp split*: *split_indicator*)

also from *union.IH* **have** $(\sum i.\ \int^+ x.\ indicator\ (f\ i)\ x * indicator\ \{g\ a..g\ b\}\ x\ \partial lborel) =$
$(\sum i.\ \int^+ x.\ indicator\ (f\ i)\ (g\ x) * ennreal\ (g'\ x) * indicator\ \{a..b\}\ x\ \partial lborel)$ **by** *simp*
also have $(\lambda i.\ \int^+ x.\ indicator\ (f\ i)\ (g\ x) * ennreal\ (g'\ x) * indicator\ \{a..b\}\ x\ \partial lborel) =$
$(\lambda i.\ \int^+ x.\ ennreal\ (g'\ x * indicator\ \{a..b\}\ x) * indicator\ (\{a..b\} \cap g\ -`\ f\ i)\ x\ \partial lborel)$
by (*intro ext nn_integral_cong*) (*simp split*: *split_indicator*)
also have $(\sum i.\ ...\ i) = \int^+ x.\ (\sum i.\ ennreal\ (g'\ x * indicator\ \{a..b\}\ x) * indicator\ (\{a..b\} \cap g\ -`\ f\ i)\ x)\ \partial lborel$
using *Mg'*
apply (*intro nn_integral_suminf*[*symmetric*])
apply (*rule borel_measurable_times_ennreal*, *simp add*: *mult.commute set_borel_measurable_def*)
apply (*rule borel_measurable_indicator*, *subst sets_lborel*)
apply (*simp_all split*: *split_indicator add*: *derivg_nonneg*)
done
also have $(\lambda x\ i.\ ennreal\ (g'\ x * indicator\ \{a..b\}\ x) * indicator\ (\{a..b\} \cap g\ -`\ f\ i)\ x) =$
$(\lambda x\ i.\ ennreal\ (g'\ x * indicator\ \{a..b\}\ x) * indicator\ (g\ -`\ f\ i)\ x)$
by (*intro ext*) (*simp split*: *split_indicator*)
also have $(\int^+ x.\ (\sum i.\ ennreal\ (g'\ x * indicator\ \{a..b\}\ x) * indicator\ (g\ -`\ f\ i)\ x)\ \partial lborel) =$
$\int^+ x.\ ennreal\ (g'\ x * indicator\ \{a..b\}\ x) * (\sum i.\ indicator\ (g\ -`\ f\ i)\ x)\ \partial lborel$
by (*intro nn_integral_cong*) (*auto split*: *split_indicator simp*: *derivg_nonneg*)
also from *union* **have** $(\lambda x.\ \sum i.\ indicator\ (g\ -`\ f\ i)\ x :: ennreal) = (\lambda x.\ indicator\ (\bigcup i.\ g\ -`\ f\ i)\ x)$
by (*intro ext suminf_indicator*) (*auto simp*: *disjoint_family_on_def*)
also have $(\int^+ x.\ ennreal\ (g'\ x * indicator\ \{a..b\}\ x) * ...\ x\ \partial lborel) =$
$(\int^+ x.\ indicator\ (\bigcup i.\ f\ i)\ (g\ x) * ennreal\ (g'\ x) * indicator\ \{a..b\}\ x\ \partial lborel)$
by (*intro nn_integral_cong*) (*simp split*: *split_indicator*)
finally show *?case* **.**
qed

next
case (*mult f c*)
note *Mf*[*measurable*] = ‹*f* ∈ *borel_measurable borel*›
let *?I* = *indicator* {*a*..*b*}
have $(\lambda x.\ f\ (g\ x * ?I\ x) * ennreal\ (g'\ x * ?I\ x)) \in borel\_measurable\ borel$ **using** *Mg Mg'*
by (*intro borel_measurable_times_ennreal measurable_compose*[*OF _ Mf*])
(*simp_all add*: *mult.commute set_borel_measurable_def*)
also have $(\lambda x.\ f\ (g\ x * ?I\ x) * ennreal\ (g'\ x * ?I\ x)) = (\lambda x.\ f\ (g\ x) * ennreal\ (g'\ x) * ?I\ x)$
by (*intro ext*) (*simp split*: *split_indicator*)
finally have *Mf'*: $(\lambda x.\ f\ (g\ x) * ennreal\ (g'\ x) * ?I\ x) \in borel\_measurable\ borel$

```
.
    with mult show ?case
      by (subst (1 2 3) mult_ac, subst (1 2) nn_integral_cmult) (simp_all add:
mult_ac)

next
  case (add f2 f1)
    let ?I = indicator {a..b}
    {
      fix f :: real ⇒ ennreal assume Mf: f ∈ borel_measurable borel
       have (λx. f (g x * ?I x) * ennreal (g' x * ?I x)) ∈ borel_measurable borel
using Mg Mg'
        by (intro borel_measurable_times_ennreal measurable_compose[OF _ Mf])
           (simp_all add:  mult.commute set_borel_measurable_def)
      also have (λx. f (g x * ?I x) * ennreal (g' x * ?I x)) = (λx. f (g x) * ennreal
(g' x) * ?I x)
        by (intro ext) (simp split: split_indicator)
      finally have (λx. f (g x) * ennreal (g' x) * ?I x) ∈ borel_measurable borel .
    } note Mf' = this[OF ‹f1 ∈ borel_measurable borel›] this[OF ‹f2 ∈ borel_measurable
borel›]

    have (∫⁺ x. (f1 x + f2 x) * indicator {g a..g b} x ∂lborel) =
              (∫⁺ x. f1 x * indicator {g a..g b} x + f2 x * indicator {g a..g b} x
∂lborel)
      by (intro nn_integral_cong) (simp split: split_indicator)
    also from add have ... = (∫⁺ x. f1 (g x) * ennreal (g' x) * indicator {a..b}
x ∂lborel) +
                                  (∫⁺ x. f2 (g x) * ennreal (g' x) * indicator {a..b} x
∂lborel)
      by (simp_all add: nn_integral_add)
    also from add have ... = (∫⁺ x. f1 (g x) * ennreal (g' x) * indicator {a..b}
x +
                                    f2 (g x) * ennreal (g' x) * indicator {a..b} x ∂lborel)
      by (intro nn_integral_add[symmetric])
         (auto simp add: Mf' derivg_nonneg split: split_indicator)
    also have ... = ∫⁺ x. (f1 (g x) + f2 (g x)) * ennreal (g' x) * indicator {a..b}
x ∂lborel
      by (intro nn_integral_cong) (simp split: split_indicator add: distrib_right)
    finally show ?case .

next
  case (sup F)
  {
    fix i
    let ?I = indicator {a..b}
     have (λx. F i (g x * ?I x) * ennreal (g' x * ?I x)) ∈ borel_measurable borel
using Mg Mg'
    by (rule_tac borel_measurable_times_ennreal, rule_tac measurable_compose[OF
_ sup.hyps(1)])
```

```
      (simp_all add: mult.commute set_borel_measurable_def)
    also have (λx. F i (g x * ?I x) * ennreal (g' x * ?I x)) = (λx. F i (g x) *
ennreal (g' x) * ?I x)
     by (intro ext) (simp split: split_indicator)
    finally have ... ∈ borel_measurable borel .
  } note Mf' = this

    have (∫⁺x. (SUP i. F i x) * indicator {g a..g b} x ∂lborel) =
            ∫⁺x. (SUP i. F i x* indicator {g a..g b} x) ∂lborel
      by (intro nn_integral_cong) (simp split: split_indicator)
    also from sup have ... = (SUP i. ∫⁺x. F i x* indicator {g a..g b} x ∂lborel)
      by (intro nn_integral_monotone_convergence_SUP)
        (auto simp: incseq_def le_fun_def split: split_indicator)
    also from sup have ... = (SUP i. ∫⁺x. F i (g x) * ennreal (g' x) * indicator
{a..b} x ∂lborel)
      by simp
    also from sup have ... =  ∫⁺x. (SUP i. F i (g x) * ennreal (g' x) * indicator
{a..b} x) ∂lborel
      by (intro nn_integral_monotone_convergence_SUP[symmetric])
        (auto simp: incseq_def le_fun_def derivg_nonneg Mf' split: split_indicator
            intro!: mult_right_mono)
    also from sup have ... = ∫⁺x. (SUP i. F i (g x)) * ennreal (g' x) * indicator
{a..b} x ∂lborel
      by (subst mult.assoc, subst mult.commute, subst SUP_mult_left_ennreal)
        (auto split: split_indicator simp: derivg_nonneg mult_ac)
    finally show ?case by (simp add: image_comp)
  qed
qed

theorem nn_integral_substitution:
  fixes f :: real ⇒ real
  assumes Mf[measurable]: set_borel_measurable borel {g a..g b} f
  assumes derivg: ⋀x. x ∈ {a..b} ⟹ (g has_real_derivative g' x) (at x)
  assumes contg': continuous_on {a..b} g'
  assumes derivg_nonneg: ⋀x. x ∈ {a..b} ⟹ g' x ≥ 0
  assumes a ≤ b
  shows (∫⁺x. f x * indicator {g a..g b} x ∂lborel) =
          (∫⁺x. f (g x) * g' x * indicator {a..b} x ∂lborel)
proof (cases a = b)
  assume a ≠ b
  with ‹a ≤ b› have a < b by auto
  let ?f' = λx. f x * indicator {g a..g b} x

  from derivg derivg_nonneg have monog: ⋀x y. a ≤ x ⟹ x ≤ y ⟹ y ≤ b ⟹
g x ≤ g y
    by (rule deriv_nonneg_imp_mono) simp_all
  have bounds: ⋀x. x ≥ a ⟹ x ≤ b ⟹ g x ≥ g a ⋀x. x ≥ a ⟹ x ≤ b ⟹ g x
≤ g b
    by (auto intro: monog)
```

from *derivg_nonneg* **have** *nonneg*:
⋀*f x*. *x* ≥ *a* ⟹ *x* ≤ *b* ⟹ *g*′ *x* ≠ *0* ⟹ *f x* ∗ *ennreal* (*g*′ *x*) ≥ *0* ⟹ *f x* ≥ *0*
by (*force simp*: *field_simps*)
have *nonneg*′: ⋀*x*. *a* ≤ *x* ⟹ *x* ≤ *b* ⟹ ¬ *0* ≤ *f* (*g x*) ⟹ *0* ≤ *f* (*g x*) ∗ *g*′ *x* ⟹ *g*′ *x* = *0*
by (*metis atLeastAtMost_iff derivg_nonneg eq_iff mult_eq_0_iff mult_le_0_iff*)

have (∫ +*x*. *f x* ∗ *indicator* {*g a*..*g b*} *x* ∂*lborel*) =
(∫ +*x*. *ennreal* (*?f*′ *x*) ∗ *indicator* {*g a*..*g b*} *x* ∂*lborel*)
by (*intro nn_integral_cong*)
(*auto split*: *split_indicator split_max simp*: *zero_ennreal.rep_eq ennreal_neg*)
also have ... = ∫ + *x*. *?f*′ (*g x*) ∗ *ennreal* (*g*′ *x*) ∗ *indicator* {*a*..*b*} *x* ∂*lborel*
using *Mf*
by (*subst nn_integral_substitution_aux*[*OF* _ _ *derivg contg*′ *derivg_nonneg* ‹*a* < *b*›])
(*auto simp add*: *mult.commute set_borel_measurable_def*)
also have ... = ∫ + *x*. *f* (*g x*) ∗ *ennreal* (*g*′ *x*) ∗ *indicator* {*a*..*b*} *x* ∂*lborel*
by (*intro nn_integral_cong*) (*auto split*: *split_indicator simp*: *max_def dest*: *bounds*)
also have ... = ∫ +*x*. *ennreal* (*f* (*g x*) ∗ *g*′ *x* ∗ *indicator* {*a*..*b*} *x*) ∂*lborel*
by (*intro nn_integral_cong*) (*auto simp*: *mult.commute derivg_nonneg ennreal_mult*′ *split*: *split_indicator*)
finally show *?thesis* **.**
qed *auto*

theorem *integral_substitution*:
assumes *integrable*: *set_integrable lborel* {*g a*..*g b*} *f*
assumes *derivg*: ⋀*x*. *x* ∈ {*a*..*b*} ⟹ (*g has_real_derivative g*′ *x*) (*at x*)
assumes *contg*′: *continuous_on* {*a*..*b*} *g*′
assumes *derivg_nonneg*: ⋀*x*. *x* ∈ {*a*..*b*} ⟹ *g*′ *x* ≥ *0*
assumes *a* ≤ *b*
shows *set_integrable lborel* {*a*..*b*} (λ*x*. *f* (*g x*) ∗ *g*′ *x*)
and (*LBINT x*. *f x* ∗ *indicator* {*g a*..*g b*} *x*) = (*LBINT x*. *f* (*g x*) ∗ *g*′ *x* ∗ *indicator* {*a*..*b*} *x*)
proof−
from *derivg* **have** *contg*: *continuous_on* {*a*..*b*} *g* **by** (*rule has_real_derivative_imp_continuous_on*)
with *contg*′ **have** *Mg*: *set_borel_measurable borel* {*a*..*b*} *g*
and *Mg*′: *set_borel_measurable borel* {*a*..*b*} *g*′
by (*simp_all only*: *set_measurable_continuous_on_ivl*)
from *derivg derivg_nonneg* **have** *monog*: ⋀*x y*. *a* ≤ *x* ⟹ *x* ≤ *y* ⟹ *y* ≤ *b* ⟹ *g x* ≤ *g y*
by (*rule deriv_nonneg_imp_mono*) *simp_all*

have (λ*x*. *ennreal* (*f x*) ∗ *indicator* {*g a*..*g b*} *x*) =
(λ*x*. *ennreal* (*f x* ∗ *indicator* {*g a*..*g b*} *x*))
by (*intro ext*) (*simp split*: *split_indicator*)
with *integrable* **have** *M1*: (λ*x*. *f x* ∗ *indicator* {*g a*..*g b*} *x*) ∈ *borel_measurable borel*

by (*force simp*: *mult.commute set_integrable_def*)
from *integrable* **have** *M2*: (λ*x*. −*f x* ∗ *indicator* {*g a*..*g b*} *x*) ∈ *borel_measurable borel*
by (*force simp*: *mult.commute set_integrable_def*)

have *LBINT x*. (*f x* :: *real*) ∗ *indicator* {*g a*..*g b*} *x* =
enn2real (∫⁺ *x*. *ennreal* (*f x*) ∗ *indicator* {*g a*..*g b*} *x* ∂*lborel*) −
enn2real (∫⁺ *x*. *ennreal* (− (*f x*)) ∗ *indicator* {*g a*..*g b*} *x* ∂*lborel*) **using** *integrable*
unfolding *set_integrable_def*
by (*subst real_lebesgue_integral_def*) (*simp_all add*: *nn_integral_set_ennreal mult.commute*)
also have ∗: (∫⁺*x*. *ennreal* (*f x*) ∗ *indicator* {*g a*..*g b*} *x* ∂*lborel*) =
(∫⁺*x*. *ennreal* (*f x* ∗ *indicator* {*g a*..*g b*} *x*) ∂*lborel*)
by (*intro nn_integral_cong*) (*simp split*: *split_indicator*)
also from *M1* ∗ **have** *A*: (∫⁺ *x*. *ennreal* (*f x* ∗ *indicator* {*g a*..*g b*} *x*) ∂*lborel*) =
(∫⁺ *x*. *ennreal* (*f* (*g x*) ∗ *g*′ *x* ∗ *indicator* {*a*..*b*} *x*) ∂*lborel*)
by (*subst nn_integral_substitution*[*OF* _ *derivg contg*′ *derivg_nonneg* ‹*a* ≤ *b*›])
(*auto simp*: *nn_integral_set_ennreal mult.commute set_borel_measurable_def*)
also have ∗∗: (∫⁺ *x*. *ennreal* (− (*f x*)) ∗ *indicator* {*g a*..*g b*} *x* ∂*lborel*) =
(∫⁺ *x*. *ennreal* (− (*f x*) ∗ *indicator* {*g a*..*g b*} *x*) ∂*lborel*)
by (*intro nn_integral_cong*) (*simp split*: *split_indicator*)
also from *M2* ∗∗ **have** *B*: (∫⁺ *x*. *ennreal* (− (*f x*) ∗ *indicator* {*g a*..*g b*} *x*) ∂*lborel*) =
(∫⁺ *x*. *ennreal* (− (*f* (*g x*)) ∗ *g*′ *x* ∗ *indicator* {*a*..*b*} *x*) ∂*lborel*)
by (*subst nn_integral_substitution*[*OF* _ *derivg contg*′ *derivg_nonneg* ‹*a* ≤ *b*›])
(*auto simp*: *nn_integral_set_ennreal mult.commute set_borel_measurable_def*)

also {
from *integrable* **have** *Mf*: *set_borel_measurable borel* {*g a*..*g b*} *f*
unfolding *set_borel_measurable_def set_integrable_def* **by** *simp*
from *measurable_compose Mg Mf Mg*′ *borel_measurable_times*
have (λ*x*. *f* (*g x* ∗ *indicator* {*a*..*b*} *x*) ∗ *indicator* {*g a*..*g b*} (*g x* ∗ *indicator* {*a*..*b*} *x*) ∗
(*g*′ *x* ∗ *indicator* {*a*..*b*} *x*)) ∈ *borel_measurable borel* (**is** *?f* ∈ _)
by (*simp add*: *mult.commute set_borel_measurable_def*)
also have *?f* = (λ*x*. *f* (*g x*) ∗ *g*′ *x* ∗ *indicator* {*a*..*b*} *x*)
using *monog* **by** (*intro ext*) (*auto split*: *split_indicator*)
finally show *set_integrable lborel* {*a*..*b*} (λ*x*. *f* (*g x*) ∗ *g*′ *x*)
using *A B integrable* **unfolding** *real_integrable_def set_integrable_def*
by (*simp_all add*: *nn_integral_set_ennreal mult.commute*)
} note *integrable*′ = *this*

have *enn2real* (∫⁺ *x*. *ennreal* (*f* (*g x*) ∗ *g*′ *x* ∗ *indicator* {*a*..*b*} *x*) ∂*lborel*) −
enn2real (∫⁺ *x*. *ennreal* (−*f* (*g x*) ∗ *g*′ *x* ∗ *indicator* {*a*..*b*} *x*) ∂*lborel*) =
(*LBINT x*. *f* (*g x*) ∗ *g*′ *x* ∗ *indicator* {*a*..*b*} *x*)
using *integrable*′ **unfolding** *set_integrable_def*

by (*subst real_lebesgue_integral_def*) (*simp_all add*: *field_simps*)
finally show (*LBINT x. f x * indicator {g a..g b} x*) =
(*LBINT x. f (g x) * g' x * indicator {a..b} x*) .
qed

theorem *interval_integral_substitution*:
assumes *integrable*: *set_integrable lborel {g a..g b} f*
assumes *derivg*: $\bigwedge x.\ x \in \{a..b\} \Longrightarrow$ (*g has_real_derivative g' x*) (*at x*)
assumes *contg'*: *continuous_on {a..b} g'*
assumes *derivg_nonneg*: $\bigwedge x.\ x \in \{a..b\} \Longrightarrow g'\ x \geq 0$
assumes $a \leq b$
shows *set_integrable lborel {a..b}* ($\lambda x.$ *f (g x) * g' x*)
and (*LBINT x=g a..g b. f x*) = (*LBINT x=a..b. f (g x) * g' x*)
apply (*rule integral_substitution*[*OF assms*], *simp*, *simp*)
apply (*subst* (*1 2*) *interval_integral_Icc*, *fact*)
apply (*rule deriv_nonneg_imp_mono*[*OF derivg derivg_nonneg*], *simp*, *simp*, *fact*)
using *integral_substitution*(*2*)[*OF assms*]
apply (*simp add*: *mult.commute set_lebesgue_integral_def*)
done

lemma *set_borel_integrable_singleton*[*simp*]: *set_integrable lborel {x}* (*f* :: *real* ⇒ *real*)
unfolding *set_integrable_def*
by (*subst integrable_discrete_difference*[**where** *X*={*x*} **and** *g*=λ_. *0*]) *auto*

end

6.22 The Volume of an n-Dimensional Ball

theory *Ball_Volume*
imports *Gamma_Function Lebesgue_Integral_Substitution*
begin

We define the volume of the unit ball in terms of the Gamma function. Note that the dimension need not be an integer; we also allow fractional dimensions, although we do not use this case or prove anything about it for now.

definition *unit_ball_vol* :: *real* ⇒ *real* **where**
unit_ball_vol n = pi powr (n / 2) / Gamma (n / 2 + 1)

lemma *unit_ball_vol_pos* [*simp*]: $n \geq 0 \Longrightarrow$ *unit_ball_vol n > 0*
by (*force simp*: *unit_ball_vol_def intro*: *divide_nonneg_pos*)

lemma *unit_ball_vol_nonneg* [*simp*]: $n \geq 0 \Longrightarrow$ *unit_ball_vol n* ≥ 0
by (*simp add*: *dual_order.strict_implies_order*)

We first need the value of the following integral, which is at the core of

computing the measure of an $n + 1$-dimensional ball in terms of the measure of an n-dimensional one.

```
lemma emeasure_cball_aux_integral:
  (∫⁺x. indicator {-1..1} x * sqrt (1 - x²) ^ n ∂lborel) =
     ennreal (Beta (1 / 2) (real n / 2 + 1))
proof -
  have ((λt. t powr (-1 / 2) * (1 - t) powr (real n / 2)) has_integral
        Beta (1 / 2) (real n / 2 + 1)) {0..1}
    using has_integral_Beta_real[of 1/2 n / 2 + 1] by simp
  from nn_integral_has_integral_lebesgue[OF _ this] have
     ennreal (Beta (1 / 2) (real n / 2 + 1)) =
        nn_integral lborel (λt. ennreal (t powr (-1 / 2) * (1 - t) powr (real n /
2) *
                                    indicator {0^2..1^2} t))
    by (simp add: mult_ac ennreal_mult' ennreal_indicator)
  also have ... = (∫⁺ x. ennreal (x² powr - (1 / 2) * (1 - x²) powr (real n /
2) * (2 * x) *
                          indicator {0..1} x) ∂lborel)
    by (subst nn_integral_substitution[where g = λx. x ^ 2 and g' = λx. 2 * x])
     (auto intro!: derivative_eq_intros continuous_intros simp: set_borel_measurable_def)
  also have ... = (∫⁺ x. 2 * ennreal ((1 - x²) powr (real n / 2) * indicator
{0..1} x) ∂lborel)
    by (intro nn_integral_cong_AE AE_I[of _ _ {0}])
       (auto simp: indicator_def powr_minus powr_half_sqrt field_split_simps
ennreal_mult')
  also have ... = (∫⁺ x. ennreal ((1 - x²) powr (real n / 2) * indicator {0..1}
x) ∂lborel) +
                  (∫⁺ x. ennreal ((1 - x²) powr (real n / 2) * indicator {0..1} x)
∂lborel)
    (is _ = ?I + _) by (simp add: mult_2 nn_integral_add)
  also have ?I = (∫⁺ x. ennreal ((1 - x²) powr (real n / 2) * indicator {-1..0}
x) ∂lborel)
    by (subst nn_integral_real_affine[of _ -1 0])
      (auto simp: indicator_def intro!: nn_integral_cong)
  hence ?I + ?I = ... + ?I by simp
  also have ... = (∫⁺ x. ennreal ((1 - x²) powr (real n / 2) *
                  (indicator {-1..0} x + indicator{0..1} x)) ∂lborel)
    by (subst nn_integral_add [symmetric]) (auto simp: algebra_simps)
  also have ... = (∫⁺ x. ennreal ((1 - x²) powr (real n / 2) * indicator {-1..1}
x) ∂lborel)
   by (intro nn_integral_cong_AE AE_I[of _ _ {0}]) (auto simp: indicator_def)
  also have ... = (∫⁺ x. ennreal (indicator {-1..1} x * sqrt (1 - x²) ^ n)
∂lborel)
    by (intro nn_integral_cong_AE AE_I[of _ _ {1, -1}])
      (auto simp: powr_half_sqrt [symmetric] indicator_def abs_square_le_1
       abs_square_eq_1 powr_def exp_of_nat_mult [symmetric] emeasure_lborel_countable)
  finally show ?thesis ..
qed
```

lemma *real_sqrt_le_iff'*: $x \geq 0 \Longrightarrow y \geq 0 \Longrightarrow sqrt\ x \leq y \longleftrightarrow x \leq y \mathbin{\hat{}} 2$
using *real_le_lsqrt sqrt_le_D* **by** *blast*

Isabelle's type system makes it very difficult to do an induction over the dimension of a Euclidean space type, because the type would change in the inductive step. To avoid this problem, we instead formulate the problem in a more concrete way by unfolding the definition of the Euclidean norm.

lemma *emeasure_cball_aux*:
assumes *finite A r > 0*
shows *emeasure* $(Pi_M\ A\ (\lambda\_.\ lborel))$
$(\{f.\ sqrt\ (\sum i{\in}A.\ (f\ i)^2) \leq r\} \cap space\ (Pi_M\ A\ (\lambda\_.\ lborel))) =$
ennreal $(unit\_ball\_vol\ (real\ (card\ A)) * r \mathbin{\hat{}} card\ A)$
using *assms*
proof (*induction arbitrary*: *r*)
case (*empty r*)
thus *?case*
by (*simp add*: *unit_ball_vol_def space_PiM*)
next
case (*insert i A r*)
interpret *product_sigma_finite* $\lambda\_.\ lborel$
by *standard*
have *emeasure* $(Pi_M\ (insert\ i\ A)\ (\lambda\_.\ lborel))$
$(\{f.\ sqrt\ (\sum i{\in}insert\ i\ A.\ (f\ i)^2) \leq r\} \cap space\ (Pi_M\ (insert\ i\ A)\ (\lambda\_.\ lborel))) =$
$nn\_integral\ (Pi_M\ (insert\ i\ A)\ (\lambda\_.\ lborel))$
$(indicator\ (\{f.\ sqrt\ (\sum i{\in}insert\ i\ A.\ (f\ i)^2) \leq r\} \cap space\ (Pi_M\ (insert\ i\ A)\ (\lambda\_.\ lborel))))$
by (*subst nn_integral_indicator*) *auto*
also have $\ldots = (\int^+ y.\ \int^+ x.\ indicator\ (\{f.\ sqrt\ ((f\ i)^2 + (\sum i{\in}A.\ (f\ i)^2)) \leq r\} \cap space\ (Pi_M\ (insert\ i\ A)\ (\lambda\_.\ lborel)))\ (x(i := y))\ \partial Pi_M\ A\ (\lambda\_.\ lborel)\ \partial lborel)$
using *insert.prems insert.hyps* **by** (*subst product_nn_integral_insert_rev*) *auto*
also have $\ldots = (\int^+ (y{::}real).\ \int^+ x.\ indicator\ \{-r..r\}\ y * indicator\ (\{f.\ sqrt\ ((\sum i{\in}A.\ (f\ i)^2)) \leq sqrt\ (r \mathbin{\hat{}} 2 - y \mathbin{\hat{}} 2)\} \cap space\ (Pi_M\ A\ (\lambda\_.\ lborel)))\ x\ \partial Pi_M\ A\ (\lambda\_.\ lborel)\ \partial lborel)$
proof (*intro nn_integral_cong*, *goal_cases*)
case (*1 y f*)
have $*$: $y \in \{-r..r\}$ **if** $y \mathbin{\hat{}} 2 + c \leq r \mathbin{\hat{}} 2\ c \geq 0$ **for** *c*
proof −
have $y \mathbin{\hat{}} 2 \leq y \mathbin{\hat{}} 2 + c$ **using** *that* **by** *simp*
also have $\ldots \leq r \mathbin{\hat{}} 2$ **by** *fact*
finally show *?thesis*
using ‹$r > 0$› **by** (*simp add*: *power2_le_iff_abs_le abs_if split*: *if_splits*)
qed
have $(\sum x{\in}A.\ (if\ x = i\ then\ y\ else\ f\ x)^2) = (\sum x{\in}A.\ (f\ x)^2)$
using *insert.hyps* **by** (*intro sum.cong*) *auto*
thus *?case* **using** *1* ‹$r > 0$›

by (*auto simp*: *sum_nonneg real_sqrt_le_iff' indicator_def PiE_def space_PiM dest*!: ∗)
qed
also have ... = (∫⁺ (*y*::*real*). *indicator* {−*r*..*r*} *y* ∗ (∫⁺ *x*. *indicator* ({*f*. *sqrt* ((∑ *i*∈*A*. (*f i*)²))
≤ *sqrt* (*r* ^ 2 − *y* ^ 2)} ∩ *space* (*Pi*$_M$ *A* (λ_. *lborel*))) *x*
∂*Pi*$_M$ *A* (λ_. *lborel*)) ∂*lborel*) **by** (*subst nn_integral_cmult*) *auto*
also have ... = (∫⁺ (*y*::*real*). *indicator* {−*r*..*r*} *y* ∗ *emeasure* (*PiM A* (λ_. *lborel*))
({*f*. *sqrt* ((∑ *i*∈*A*. (*f i*)²)) ≤ *sqrt* (*r* ^ 2 − *y* ^ 2)} ∩ *space* (*Pi*$_M$ *A* (λ_. *lborel*))) ∂*lborel*)
using ‹*finite A*› **by** (*intro nn_integral_cong*, *subst nn_integral_indicator*) *auto*
also have ... = (∫⁺ (*y*::*real*). *indicator* {−*r*..*r*} *y* ∗ *ennreal* (*unit_ball_vol* (*real* (*card A*)) ∗
(*sqrt* (*r* ^ 2 − *y* ^ 2)) ^ *card A*) ∂*lborel*)
proof (*intro nn_integral_cong_AE*, *goal_cases*)
case *1*
have *AE y in lborel*. *y* ∉ {−*r*,*r*}
by (*intro AE_not_in countable_imp_null_set_lborel*) *auto*
thus *?case*
proof *eventually_elim*
case (*elim y*)
show *?case*
proof (*cases y* ∈ {−*r*<..<*r*})
case *True*
hence *y*² < *r*² **by** (*subst real_sqrt_less_iff* [*symmetric*]) *auto*
thus *?thesis* **by** (*subst insert.IH*) (*auto*)
qed (*insert elim*, *auto*)
qed
qed
also have ... = *ennreal* (*unit_ball_vol* (*real* (*card A*))) ∗
(∫⁺ (*y*::*real*). *indicator* {−*r*..*r*} *y* ∗ (*sqrt* (*r* ^ 2 − *y* ^ 2)) ^ *card A* ∂*lborel*)
by (*subst nn_integral_cmult* [*symmetric*])
(*auto simp*: *mult_ac ennreal_mult'* [*symmetric*] *indicator_def intro*!: *nn_integral_cong*)
also have (∫⁺ (*y*::*real*). *indicator* {−*r*..*r*} *y* ∗ (*sqrt* (*r* ^ 2 − *y* ^ 2)) ^ *card A* ∂*lborel*) =
(∫⁺ (*y*::*real*). *r* ^ *card A* ∗ *indicator* {−1..1} *y* ∗ (*sqrt* (1 − *y* ^ 2)) ^ *card A*
∂(*distr lborel borel* ((∗) (1/*r*)))) **using** ‹*r* > *0*›
by (*subst nn_integral_distr*)
(*auto simp*: *indicator_def field_simps real_sqrt_divide intro*!: *nn_integral_cong*)
also have ... = (∫⁺ *x*. *ennreal* (*r* ^ *Suc* (*card A*)) ∗
(*indicator* {− 1..1} *x* ∗ *sqrt* (1 − *x*²) ^ *card A*) ∂*lborel*) **using** ‹*r* > *0*›
by (*subst lborel_distr_mult*) (*auto simp*: *nn_integral_density ennreal_mult'* [*symmetric*] *mult_ac*)
also have ... = *ennreal* (*r* ^ *Suc* (*card A*)) ∗ (∫⁺ *x*. *indicator* {− 1..1} *x* ∗
sqrt (1 − *x*²) ^ *card A* ∂*lborel*)

```
    by (subst nn_integral_cmult) auto
  also note emeasure_cball_aux_integral
  also have ennreal (unit_ball_vol (real (card A))) * (ennreal (r ^ Suc (card A))
*
                ennreal (Beta (1/2) (card A / 2 + 1))) =
              ennreal (unit_ball_vol (card A) * Beta (1/2) (card A / 2 + 1) * r ^
Suc (card A))
    using ‹r > 0› by (simp add: ennreal_mult' [symmetric] mult_ac)
  also have unit_ball_vol (card A) * Beta (1/2) (card A / 2 + 1) = unit_ball_vol
(Suc (card A))
    by (auto simp: unit_ball_vol_def Beta_def Gamma_eq_zero_iff field_simps
        Gamma_one_half_real powr_half_sqrt [symmetric] powr_add [symmetric])
  also have Suc (card A) = card (insert i A) using insert.hyps by simp
  finally show ?case .
qed
```

We now get the main theorem very easily by just applying the above lemma.

```
context
  fixes c :: 'a :: euclidean_space and r :: real
  assumes r: r ≥ 0
begin

theorem emeasure_cball:
  emeasure lborel (cball c r) = ennreal (unit_ball_vol (DIM('a)) * r ^ DIM('a))
proof (cases r = 0)
  case False
  with r have r: r > 0 by simp
  have (lborel :: 'a measure) =
        distr (Pi_M Basis (λ_. lborel)) borel (λf. ∑ b∈Basis. f b *_R b)
    by (rule lborel_eq)
  also have emeasure ... (cball 0 r) =
              emeasure (Pi_M Basis (λ_. lborel))
              ({y. dist 0 (∑ b∈Basis. y b *_R b :: 'a) ≤ r} ∩ space (Pi_M Basis (λ_.
lborel)))
    by (subst emeasure_distr) (auto simp: cball_def)
  also have {f. dist 0 (∑ b∈Basis. f b *_R b :: 'a) ≤ r} = {f. sqrt (∑ i∈Basis. (f
i)^2) ≤ r}
    by (subst euclidean_dist_l2) (auto simp: L2_set_def)
  also have emeasure (Pi_M Basis (λ_. lborel)) (... ∩ space (Pi_M Basis (λ_.
lborel))) =
              ennreal (unit_ball_vol (real DIM('a)) * r ^ DIM('a))
    using r by (subst emeasure_cball_aux) simp_all
  also have emeasure lborel (cball 0 r :: 'a set) =
              emeasure (distr lborel borel (λx. c + x)) (cball c r)
   by (subst emeasure_distr) (auto simp: cball_def dist_norm norm_minus_commute)
  also have distr lborel borel (λx. c + x) = lborel
    using lborel_affine[of 1 c] by (simp add: density_1)
  finally show ?thesis .
qed auto
```

corollary *content_cball*:
content (*cball c r*) = *unit_ball_vol* (*DIM*('*a*)) * *r* ^ *DIM*('*a*)
by (*simp add*: *measure_def emeasure_cball r*)

corollary *emeasure_ball*:
emeasure lborel (*ball c r*) = *ennreal* (*unit_ball_vol* (*DIM*('*a*)) * *r* ^ *DIM*('*a*))
proof −
from *negligible_sphere*[*of c r*] **have** *sphere c r* ∈ *null_sets lborel*
by (*auto simp*: *null_sets_completion_iff negligible_iff_null_sets negligible_convex_frontier*)
hence *emeasure lborel* (*ball c r* ∪ *sphere c r* :: '*a set*) = *emeasure lborel* (*ball c r* :: '*a set*)
by (*intro emeasure_Un_null_set*) *auto*
also have *ball c r* ∪ *sphere c r* = (*cball c r* :: '*a set*) **by** *auto*
also have *emeasure lborel* ... = *ennreal* (*unit_ball_vol* (*real DIM*('*a*)) * *r* ^ *DIM*('*a*))
by (*rule emeasure_cball*)
finally show *?thesis* **..**
qed

corollary *content_ball*:
content (*ball c r*) = *unit_ball_vol* (*DIM*('*a*)) * *r* ^ *DIM*('*a*)
by (*simp add*: *measure_def r emeasure_ball*)

end

Lastly, we now prove some nicer explicit formulas for the volume of the unit balls in the cases of even and odd integer dimensions.

lemma *unit_ball_vol_even*:
unit_ball_vol (*real* (2 * *n*)) = *pi* ^ *n* / *fact n*
by (*simp add*: *unit_ball_vol_def add_ac powr_realpow Gamma_fact*)

lemma *unit_ball_vol_odd'*:
unit_ball_vol (*real* (2 * *n* + 1)) = *pi* ^ *n* / *pochhammer* (1 / 2) (*Suc n*)
and *unit_ball_vol_odd*:
unit_ball_vol (*real* (2 * *n* + 1)) =
(2 ^ (2 * *Suc n*) * *fact* (*Suc n*)) / *fact* (2 * *Suc n*) * *pi* ^ *n*
proof −
have *unit_ball_vol* (*real* (2 * *n* + 1)) =
pi powr (*real n* + 1 / 2) / *Gamma* (1 / 2 + *real* (*Suc n*))
by (*simp add*: *unit_ball_vol_def field_simps*)
also have *pochhammer* (1 / 2) (*Suc n*) = *Gamma* (1 / 2 + *real* (*Suc n*)) / *Gamma* (1 / 2)
by (*intro pochhammer_Gamma*) *auto*
hence *Gamma* (1 / 2 + *real* (*Suc n*)) = *sqrt pi* * *pochhammer* (1 / 2) (*Suc n*)
by (*simp add*: *Gamma_one_half_real*)
also have *pi powr* (*real n* + 1 / 2) / ... = *pi* ^ *n* / *pochhammer* (1 / 2) (*Suc n*)
by (*simp add*: *powr_add powr_half_sqrt powr_realpow*)

```
  finally show unit_ball_vol (real (2 * n + 1)) = ... .
  also have pochhammer (1 / 2 :: real) (Suc n) =
              fact (2 * Suc n) / (2 ^ (2 * Suc n) * fact (Suc n))
    using fact_double[of Suc n, where ?'a = real] by (simp add: divide_simps
mult_ac)
  also have pi ^ n / ... = (2 ^ (2 * Suc n) * fact (Suc n)) / fact (2 * Suc n) *
pi ^ n
    by simp
  finally show unit_ball_vol (real (2 * n + 1)) = ... .
qed

lemma unit_ball_vol_numeral:
  unit_ball_vol (numeral (Num.Bit0 n)) = pi ^ numeral n / fact (numeral n) (is
?th1)
  unit_ball_vol (numeral (Num.Bit1 n)) = 2 ^ (2 * Suc (numeral n)) * fact (Suc
(numeral n)) /
    fact (2 * Suc (numeral n)) * pi ^ numeral n (is ?th2)
proof -
  have numeral (Num.Bit0 n) = (2 * numeral n :: nat)
    by (simp only: numeral_Bit0 mult_2 ring_distribs)
  also have unit_ball_vol ... = pi ^ numeral n / fact (numeral n)
    by (rule unit_ball_vol_even)
  finally show ?th1 by simp
next
  have numeral (Num.Bit1 n) = (2 * numeral n + 1 :: nat)
    by (simp only: numeral_Bit1 mult_2)
  also have unit_ball_vol ... = 2 ^ (2 * Suc (numeral n)) * fact (Suc (numeral
n)) /
                                fact (2 * Suc (numeral n)) * pi ^ numeral n
    by (rule unit_ball_vol_odd)
  finally show ?th2 by simp
qed

lemmas eval_unit_ball_vol = unit_ball_vol_numeral fact_numeral
```

Just for fun, we compute the volume of unit balls for a few dimensions.

```
lemma unit_ball_vol_0 [simp]: unit_ball_vol 0 = 1
  using unit_ball_vol_even[of 0] by simp

lemma unit_ball_vol_1 [simp]: unit_ball_vol 1 = 2
  using unit_ball_vol_odd[of 0] by simp

corollary
        unit_ball_vol_2: unit_ball_vol 2 = pi
    and unit_ball_vol_3: unit_ball_vol 3 = 4 / 3 * pi
    and unit_ball_vol_4: unit_ball_vol 4 = pi^2 / 2
    and unit_ball_vol_5: unit_ball_vol 5 = 8 / 15 * pi^2
  by (simp_all add: eval_unit_ball_vol)
```

corollary *circle_area*:
  $r \geq 0 \Longrightarrow$ *content* (*ball c r* :: (*real* ^ *2*) *set*) = *r* ^ *2* * *pi*
  by (*simp add*: *content_ball unit_ball_vol_2*)

corollary *sphere_volume*:
  $r \geq 0 \Longrightarrow$ *content* (*ball c r* :: (*real* ^ *3*) *set*) = *4* / *3* * *r* ^ *3* * *pi*
  by (*simp add*: *content_ball unit_ball_vol_3*)

Useful equivalent forms

corollary *content_ball_eq_0_iff* [*simp*]: *content* (*ball c r*) = *0* $\longleftrightarrow$ *r* $\leq$ *0*
proof −
  have *r* > *0* $\Longrightarrow$ *content* (*ball c r*) > *0*
    by (*simp add*: *content_ball unit_ball_vol_def*)
  then show *?thesis*
    by (*fastforce simp*: *ball_empty*)
qed

corollary *content_ball_gt_0_iff* [*simp*]: *0* < *content* (*ball z r*) $\longleftrightarrow$ *0* < *r*
  by (*auto simp*: *zero_less_measure_iff*)

corollary *content_cball_eq_0_iff* [*simp*]: *content* (*cball c r*) = *0* $\longleftrightarrow$ *r* $\leq$ *0*
proof (*cases r* = *0*)
  case *False*
  moreover have *r* > *0* $\Longrightarrow$ *content* (*cball c r*) > *0*
    by (*simp add*: *content_cball unit_ball_vol_def*)
  ultimately show *?thesis*
    by *fastforce*
qed *auto*

corollary *content_cball_gt_0_iff* [*simp*]: *0* < *content* (*cball z r*) $\longleftrightarrow$ *0* < *r*
  by (*auto simp*: *zero_less_measure_iff*)

end

6.23 Integral Test for Summability

theory *Integral_Test*
imports *Henstock_Kurzweil_Integration*
begin

The integral test for summability. We show here that for a decreasing non-negative function, the infinite sum over that function evaluated at the natural numbers converges iff the corresponding integral converges.

As a useful side result, we also provide some results on the difference between the integral and the partial sum. (This is useful e.g. for the definition of the Euler-Mascheroni constant)

locale *antimono_fun_sum_integral_diff* =
  fixes *f* :: *real* ⇒ *real*

```
  assumes dec: ⋀x y. x ≥ 0 ⟹ x ≤ y ⟹ f x ≥ f y
  assumes nonneg: ⋀x. x ≥ 0 ⟹ f x ≥ 0
  assumes cont: continuous_on {0..} f
begin

definition sum_integral_diff_series n = (∑ k≤n. f (of_nat k)) − (integral {0..of_nat
n} f)

lemma sum_integral_diff_series_nonneg:
  sum_integral_diff_series n ≥ 0
proof −
  note int = integrable_continuous_real[OF continuous_on_subset[OF cont]]
  let ?int = λa b. integral {of_nat a..of_nat b} f
  have −sum_integral_diff_series n = ?int 0 n − (∑ k≤n. f (of_nat k))
    by (simp add: sum_integral_diff_series_def)
  also have ?int 0 n = (∑ k<n. ?int k (Suc k))
  proof (induction n)
    case (Suc n)
    have ?int 0 (Suc n) = ?int 0 n + ?int n (Suc n)
      by (intro integral_combine[symmetric] int) simp_all
    with Suc show ?case by simp
  qed simp_all
  also have ... ≤ (∑ k<n. integral {of_nat k..of_nat (Suc k)} (λ_::real. f (of_nat
k)))
    by (intro sum_mono integral_le int) (auto intro: dec)
  also have ... = (∑ k<n. f (of_nat k)) by simp
  also have ... − (∑ k≤n. f (of_nat k)) = −(∑ k∈{..n} − {..<n}. f (of_nat k))
    by (subst sum_diff) auto
  also have ... ≤ 0 by (auto intro!: sum_nonneg nonneg)
  finally show sum_integral_diff_series n ≥ 0 by simp
qed

lemma sum_integral_diff_series_antimono:
  assumes m ≤ n
  shows   sum_integral_diff_series m ≥ sum_integral_diff_series n
proof −
  let ?int = λa b. integral {of_nat a..of_nat b} f
  note int = integrable_continuous_real[OF continuous_on_subset[OF cont]]
  have d_mono: sum_integral_diff_series (Suc n) ≤ sum_integral_diff_series n
for n
  proof −
    fix n :: nat
    have sum_integral_diff_series (Suc n) − sum_integral_diff_series n =
          f (of_nat (Suc n)) + (?int 0 n − ?int 0 (Suc n))
      unfolding sum_integral_diff_series_def by (simp add: algebra_simps)
    also have ?int 0 n − ?int 0 (Suc n) = −?int n (Suc n)
      by (subst integral_combine [symmetric, of of_nat 0 of_nat n of_nat (Suc
n)])
         (auto intro!: int simp: algebra_simps)
```

```
    also have ?int n (Suc n) ≥ integral {of_nat n..of_nat (Suc n)} (λ_::real. f
(of_nat (Suc n)))
      by (intro integral_le int) (auto intro: dec)
   hence f (of_nat (Suc n)) + − ?int n (Suc n) ≤ 0 by (simp add: algebra_simps)
    finally show sum_integral_diff_series (Suc n) ≤ sum_integral_diff_series n
by simp
  qed
  with assms show ?thesis
    by (induction rule: inc_induct) (auto intro: order.trans[OF _ d_mono])
qed

lemma sum_integral_diff_series_Bseq: Bseq sum_integral_diff_series
proof −
 from sum_integral_diff_series_nonneg and sum_integral_diff_series_antimono
    have norm (sum_integral_diff_series n) ≤ sum_integral_diff_series 0 for n
by simp
  thus Bseq sum_integral_diff_series by (rule BseqI′)
qed

lemma sum_integral_diff_series_monoseq: monoseq sum_integral_diff_series
  using sum_integral_diff_series_antimono unfolding monoseq_def by blast

lemma sum_integral_diff_series_convergent: convergent sum_integral_diff_series
  using sum_integral_diff_series_Bseq sum_integral_diff_series_monoseq
  by (blast intro!: Bseq_monoseq_convergent)

theorem integral_test:
  summable (λn. f (of_nat n)) ⟷ convergent (λn. integral {0..of_nat n} f)
proof −
  have summable (λn. f (of_nat n)) ⟷ convergent (λn. ∑ k≤n. f (of_nat k))
    by (simp add: summable_iff_convergent′)
  also have ... ⟷ convergent (λn. integral {0..of_nat n} f)
  proof
    assume convergent (λn. ∑ k≤n. f (of_nat k))
    from convergent_diff[OF this sum_integral_diff_series_convergent]
      show convergent (λn. integral {0..of_nat n} f)
        unfolding sum_integral_diff_series_def by simp
  next
    assume convergent (λn. integral {0..of_nat n} f)
    from convergent_add[OF this sum_integral_diff_series_convergent]
    show convergent (λn. ∑ k≤n. f (of_nat k)) unfolding sum_integral_diff_series_def
by simp
  qed
  finally show ?thesis by simp
qed

end

end
```

6.24 Continuity of the indefinite integral; improper integral theorem

theory *Improper_Integral*
imports *Equivalence_Lebesgue_Henstock_Integration*
begin

6.24.1 Equiintegrability

The definition here only really makes sense for an elementary set. We just use compact intervals in applications below.

definition *equiintegrable_on* (**infixr** *equiintegrable'_on 46*)
where *F equiintegrable_on I* ≡
(∀*f* ∈ *F*. *f integrable_on I*) ∧
(∀*e* > *0*. ∃*γ*. *gauge γ* ∧
(∀*f* 𝒟. *f* ∈ *F* ∧ 𝒟 *tagged_division_of I* ∧ *γ fine* 𝒟
⟶ *norm* ((∑(*x*,*K*) ∈ 𝒟. *content K* ∗$_R$ *f x*) − *integral I f*)
< *e*))

lemma *equiintegrable_on_integrable*:
⟦*F equiintegrable_on I*; *f* ∈ *F*⟧ ⟹ *f integrable_on I*
using *equiintegrable_on_def* **by** *metis*

lemma *equiintegrable_on_sing* [*simp*]:
{*f*} *equiintegrable_on cbox a b* ⟷ *f integrable_on cbox a b*
by (*simp add*: *equiintegrable_on_def has_integral_integral has_integral integrable_on_def*)

lemma *equiintegrable_on_subset*: ⟦*F equiintegrable_on I*; *G* ⊆ *F*⟧ ⟹ *G equiintegrable_on I*
unfolding *equiintegrable_on_def Ball_def*
by (*erule conj_forward imp_forward all_forward ex_forward* | *blast*)+

lemma *equiintegrable_on_Un*:
assumes *F equiintegrable_on I G equiintegrable_on I*
shows (*F* ∪ *G*) *equiintegrable_on I*
unfolding *equiintegrable_on_def*
proof (*intro conjI impI allI*)
show ∀*f*∈*F* ∪ *G*. *f integrable_on I*
using *assms* **unfolding** *equiintegrable_on_def* **by** *blast*
show ∃*γ*. *gauge γ* ∧
(∀*f* 𝒟. *f* ∈ *F* ∪ *G* ∧
𝒟 *tagged_division_of I* ∧ *γ fine* 𝒟 ⟶
norm ((∑(*x*,*K*) ∈ 𝒟. *content K* ∗$_R$ *f x*) − *integral I f*) < *ε*)
if *ε* > *0* **for** *ε*
proof −
obtain *γ1* **where** *gauge γ1*
and *γ1*: ⋀*f* 𝒟. *f* ∈ *F* ∧ 𝒟 *tagged_division_of I* ∧ *γ1 fine* 𝒟

$\Longrightarrow$ *norm* $((\sum (x,K) \in \mathcal{D}.\ \textit{content}\ K *_R f\ x) - \textit{integral}\ I\ f) < \varepsilon$
using *assms* ‹$\varepsilon > 0$› **unfolding** *equiintegrable_on_def* **by** *auto*
obtain $\gamma 2$ **where** *gauge* $\gamma 2$
and $\gamma 2$: $\bigwedge f\ \mathcal{D}.\ f \in G \wedge \mathcal{D}$ *tagged_division_of* $I \wedge \gamma 2$ *fine* $\mathcal{D}$
$\Longrightarrow$ *norm* $((\sum (x,K) \in \mathcal{D}.\ \textit{content}\ K *_R f\ x) - \textit{integral}\ I\ f) < \varepsilon$
using *assms* ‹$\varepsilon > 0$› **unfolding** *equiintegrable_on_def* **by** *auto*
have *gauge* $(\lambda x.\ \gamma 1\ x \cap \gamma 2\ x)$
using ‹*gauge* $\gamma 1$› ‹*gauge* $\gamma 2$› **by** *blast*
moreover have $\forall f\ \mathcal{D}.\ f \in F \cup G \wedge \mathcal{D}$ *tagged_division_of* $I \wedge (\lambda x.\ \gamma 1\ x \cap \gamma 2\ x)$ *fine* $\mathcal{D} \longrightarrow$
norm $((\sum (x,K) \in \mathcal{D}.\ \textit{content}\ K *_R f\ x) - \textit{integral}\ I\ f) < \varepsilon$
using $\gamma 1$ $\gamma 2$ **by** (*auto simp*: *fine_Int*)
ultimately show *?thesis*
by (*intro exI conjI*) *assumption+*
qed
qed

lemma *equiintegrable_on_insert*:
assumes *f integrable_on cbox a b F equiintegrable_on cbox a b*
shows (*insert f F*) *equiintegrable_on cbox a b*
by (*metis assms equiintegrable_on_Un equiintegrable_on_sing insert_is_Un*)

lemma *equiintegrable_cmul*:
assumes *F*: *F equiintegrable_on I*
shows $(\bigcup c \in \{-k..k\}.\ \bigcup f \in F.\ \{(\lambda x.\ c *_R f\ x)\})$ *equiintegrable_on I*
unfolding *equiintegrable_on_def*
proof (*intro conjI impI allI ballI*)
show *f integrable_on I*
if $f \in (\bigcup c \in \{-\ k..k\}.\ \bigcup f \in F.\ \{\lambda x.\ c *_R f\ x\})$
for $f :: 'a \Rightarrow 'b$
using *that assms equiintegrable_on_integrable integrable_cmul* **by** *blast*
show $\exists \gamma.$ *gauge* $\gamma \wedge (\forall f\ \mathcal{D}.\ f \in (\bigcup c \in \{-\ k..k\}.\ \bigcup f \in F.\ \{\lambda x.\ c *_R f\ x\}) \wedge \mathcal{D}$ *tagged_division_of I*
$\wedge\ \gamma$ *fine* $\mathcal{D} \longrightarrow$ *norm* $((\sum (x,\ K) \in \mathcal{D}.\ \textit{content}\ K *_R f\ x) - \textit{integral}\ I\ f) < \varepsilon)$
if $\varepsilon > 0$ **for** ε
proof −
obtain γ **where** *gauge* γ
and γ: $\bigwedge f\ \mathcal{D}.\ [\![f \in F;\ \mathcal{D}$ *tagged_division_of* $I;\ \gamma$ *fine* $\mathcal{D}]\!]$
$\Longrightarrow$ *norm* $((\sum (x,K) \in \mathcal{D}.\ \textit{content}\ K *_R f\ x) - \textit{integral}\ I\ f) < \varepsilon\ /\ (|k| + 1)$
using *assms* ‹$\varepsilon > 0$› **unfolding** *equiintegrable_on_def*
by (*metis add.commute add.right_neutral add_strict_mono divide_pos_pos norm_eq_zero real_norm_def zero_less_norm_iff zero_less_one*)
moreover have *norm* $((\sum (x,\ K) \in \mathcal{D}.\ \textit{content}\ K *_R c *_R (f\ x)) - \textit{integral}\ I\ (\lambda x.\ c *_R f\ x)) < \varepsilon$
if *c*: $c \in \{-\ k..k\}$

```
      and f ∈ F 𝒟 tagged_division_of I γ fine 𝒟
    for 𝒟 c f
  proof −
    have norm ((∑ x∈𝒟. case x of (x, K) ⇒ content K *R c *R f x) − integral
I (λx. c *R f x))
        = |c| * norm ((∑ x∈𝒟. case x of (x, K) ⇒ content K *R f x) − integral
I f)
          by (simp add: algebra_simps scale_sum_right case_prod_unfold flip:
norm_scaleR)
    also have ... ≤ (|k| + 1) * norm ((∑ x∈𝒟. case x of (x, K) ⇒ content K
*R f x) − integral I f)
      using c by (auto simp: mult_right_mono)
    also have ... < (|k| + 1) * (ε / (|k| + 1))
      by (rule mult_strict_left_mono) (use γ less_eq_real_def that in auto)
    also have ... = ε
      by auto
    finally show ?thesis .
  qed
  ultimately show ?thesis
    by (rule_tac x=γ in exI) auto
 qed
qed


lemma equiintegrable_add:
 assumes F: F equiintegrable_on I and G: G equiintegrable_on I
 shows (⋃f ∈ F. ⋃g ∈ G. {(λx. f x + g x)}) equiintegrable_on I
 unfolding equiintegrable_on_def
proof (intro conjI impI allI ballI)
 show f integrable_on I
   if f ∈ (⋃f∈F. ⋃g∈G. {λx. f x + g x}) for f
   using that equiintegrable_on_integrable assms by (auto intro: integrable_add)
  show ∃γ. gauge γ ∧ (∀f 𝒟. f ∈ (⋃f∈F. ⋃g∈G. {λx. f x + g x}) ∧ 𝒟
tagged_division_of I
        ∧ γ fine 𝒟 ⟶ norm ((∑ (x, K)∈𝒟. content K *R f x) − integral I f) <
ε)
   if ε > 0 for ε
 proof −
   obtain γ1 where gauge γ1
     and γ1: ⋀f 𝒟. ⟦f ∈ F; 𝒟 tagged_division_of I; γ1 fine 𝒟⟧
                ⟹ norm ((∑ (x,K) ∈ 𝒟. content K *R f x) − integral I f) < ε/2
   using assms ‹ε > 0› unfolding equiintegrable_on_def by (meson half_gt_zero_iff)
   obtain γ2 where gauge γ2
     and γ2: ⋀g 𝒟. ⟦g ∈ G; 𝒟 tagged_division_of I; γ2 fine 𝒟⟧
                ⟹ norm ((∑ (x,K) ∈ 𝒟. content K *R g x) − integral I g) < ε/2
   using assms ‹ε > 0› unfolding equiintegrable_on_def by (meson half_gt_zero_iff)
   have gauge (λx. γ1 x ∩ γ2 x)
     using ‹gauge γ1› ‹gauge γ2› by blast
   moreover have norm ((∑ (x,K) ∈ 𝒟. content K *R h x) − integral I h) < ε
```

if h: $h \in (\bigcup f{\in}F.\ \bigcup g{\in}G.\ \{\lambda x.\ f\ x + g\ x\})$
and $\mathcal{D}$: $\mathcal{D}$ *tagged_division_of* I **and** *fine*: $(\lambda x.\ \gamma 1\ x \cap \gamma 2\ x)$ *fine* $\mathcal{D}$
for h $\mathcal{D}$
proof −
obtain $f\ g$ **where** $f \in F\ g \in G$ **and** *heq*: $h = (\lambda x.\ f\ x + g\ x)$
using h **by** *blast*
then have *int*: f *integrable_on* I g *integrable_on* I
using F G *equiintegrable_on_def* **by** *blast+*
have *norm* $((\sum (x,K) \in \mathcal{D}.\ \mathit{content}\ K *_R h\ x) - \mathit{integral}\ I\ h)$
$= \mathit{norm}\ ((\sum (x,K) \in \mathcal{D}.\ \mathit{content}\ K *_R f\ x + \mathit{content}\ K *_R g\ x) - (\mathit{integral}\ I\ f + \mathit{integral}\ I\ g))$
by (*simp add*: *heq algebra_simps integral_add int*)
also have $\ldots = \mathit{norm}\ (((\sum (x,K) \in \mathcal{D}.\ \mathit{content}\ K *_R f\ x) - \mathit{integral}\ I\ f + (\sum (x,K) \in \mathcal{D}.\ \mathit{content}\ K *_R g\ x) - \mathit{integral}\ I\ g))$
by (*simp add*: *sum.distrib algebra_simps case_prod_unfold*)
also have $\ldots \le \mathit{norm}\ ((\sum (x,K) \in \mathcal{D}.\ \mathit{content}\ K *_R f\ x) - \mathit{integral}\ I\ f) + \mathit{norm}\ ((\sum (x,K) \in \mathcal{D}.\ \mathit{content}\ K *_R g\ x) - \mathit{integral}\ I\ g)$
by (*metis* (*mono_tags*) *add_diff_eq norm_triangle_ineq*)
also have $\ldots < \varepsilon/2 + \varepsilon/2$
using $\gamma 1$ [*OF* ‹$f \in F$› $\mathcal{D}$] $\gamma 2$ [*OF* ‹$g \in G$› $\mathcal{D}$] *fine* **by** (*simp add*: *fine_Int*)
finally show *?thesis* **by** *simp*
qed
ultimately show *?thesis*
by *meson*
qed
qed

lemma *equiintegrable_minus*:
assumes F *equiintegrable_on* I
shows $(\bigcup f \in F.\ \{(\lambda x.\ -f\ x)\})$ *equiintegrable_on* I
by (*force intro*: *equiintegrable_on_subset* [*OF equiintegrable_cmul* [*OF assms*, *of 1*]])

lemma *equiintegrable_diff*:
assumes F: F *equiintegrable_on* I **and** G: G *equiintegrable_on* I
shows $(\bigcup f \in F.\ \bigcup g \in G.\ \{(\lambda x.\ f\ x - g\ x)\})$ *equiintegrable_on* I
by (*rule equiintegrable_on_subset* [*OF equiintegrable_add* [*OF F equiintegrable_minus* [*OF G*]]]) *auto*

lemma *equiintegrable_sum*:
fixes $F :: ('a{::}\mathit{euclidean\_space} \Rightarrow 'b{::}\mathit{euclidean\_space})\ \mathit{set}$
assumes F *equiintegrable_on cbox a b*
shows $(\bigcup I \in \mathit{Collect\ finite}.\ \bigcup c \in \{c.\ (\forall i \in I.\ c\ i \ge 0) \wedge \mathit{sum}\ c\ I = 1\}.$
$\bigcup f \in I \to F.\ \{(\lambda x.\ \mathit{sum}\ (\lambda i{::}'j.\ c\ i *_R f\ i\ x)\ I)\})$ *equiintegrable_on cbox a b*
(**is** *?G equiintegrable_on _*)
unfolding *equiintegrable_on_def*
proof (*intro conjI impI allI ballI*)

```
  show f integrable_on cbox a b if f ∈ ?G for f
    using that assms by (auto simp: equiintegrable_on_def intro!: integrable_sum
integrable_cmul)
  show ∃γ. gauge γ
         ∧ (∀g D. g ∈ ?G ∧ D tagged_division_of cbox a b ∧ γ fine D
              ⟶ norm ((∑(x,K) ∈ D. content K *R g x) − integral (cbox a b) g)
< ε)
    if ε > 0 for ε
  proof −
    obtain γ where gauge γ
      and γ: ⋀f D. ⟦f ∈ F; D tagged_division_of cbox a b; γ fine D⟧
                ⟹ norm ((∑(x,K) ∈ D. content K *R f x) − integral (cbox a b)
f) < ε / 2
    using assms ‹ε > 0› unfolding equiintegrable_on_def by (meson half_gt_zero_iff)
    moreover have norm ((∑(x,K) ∈ D. content K *R g x) − integral (cbox a
b) g) < ε
      if g: g ∈ ?G
        and D: D tagged_division_of cbox a b
        and fine: γ fine D
      for D g
    proof −
      obtain I c f where finite I and 0: ⋀i::'j. i ∈ I ⟹ 0 ≤ c i
        and 1: sum c I = 1 and f: f ∈ I → F and geq: g = (λx. ∑i∈I. c i *R f
i x)
        using g by auto
      have fi_int: f i integrable_on cbox a b if i ∈ I for i
        by (metis Pi_iff assms equiintegrable_on_def f that)
      have *: integral (cbox a b) (λx. c i *R f i x) = (∑(x, K)∈D. integral K (λx.
c i *R f i x))
        if i ∈ I for i
      proof −
        have f i integrable_on cbox a b
          by (metis Pi_iff assms equiintegrable_on_def f that)
        then show ?thesis
          by (intro D integrable_cmul integral_combine_tagged_division_topdown)
      qed
      have finite D
        using D by blast
      have swap: (∑(x,K)∈D. content K *R (∑i∈I. c i *R f i x))
            = (∑i∈I. c i *R (∑(x,K)∈D. content K *R f i x))
          by (simp add: scale_sum_right case_prod_unfold algebra_simps) (rule
sum.swap)
      have norm ((∑(x, K)∈D. content K *R g x) − integral (cbox a b) g)
          = norm ((∑i∈I. c i *R ((∑(x,K)∈D. content K *R f i x) − integral
(cbox a b) (f i))))
        unfolding geq swap
        by (simp add: scaleR_right.sum algebra_simps integral_sum fi_int inte-
grable_cmul ‹finite I› sum_subtractf flip: sum_diff)
      also have ... ≤ (∑i∈I. c i * ε / 2)
```

```
      proof (rule sum_norm_le)
        show norm (c i *R ((∑(xa, K)∈𝒟. content K *R f i xa) − integral (cbox
a b) (f i))) ≤ c i * ε / 2
          if i ∈ I for i
        proof −
          have norm ((∑(x, K)∈𝒟. content K *R f i x) − integral (cbox a b) (f i))
≤ ε/2
            using γ [OF _ 𝒟 fine, of f i] funcset_mem [OF f] that  by auto
          then show ?thesis
            using that by (auto simp: 0 mult.assoc intro: mult_left_mono)
        qed
      qed
      also have ... < ε
       using 1 ‹ε > 0› by (simp add: flip: sum_divide_distrib sum_distrib_right)
      finally show ?thesis .
    qed
    ultimately show ?thesis
      by (rule_tac x=γ in exI) auto
  qed
qed

corollary equiintegrable_sum_real:
  fixes F :: (real ⇒ 'b::euclidean_space) set
  assumes F equiintegrable_on {a..b}
  shows (⋃I ∈ Collect finite. ⋃c ∈ {c. (∀i ∈ I. c i ≥ 0) ∧ sum c I = 1}.
         ⋃f ∈ I → F. {(λx. sum (λi. c i *R f i x) I)})
        equiintegrable_on {a..b}
  using equiintegrable_sum [of F a b] assms by auto
```

Basic combining theorems for the interval of integration.

```
lemma equiintegrable_on_null [simp]:
   content(cbox a b) = 0 ⟹ F equiintegrable_on cbox a b
  unfolding equiintegrable_on_def
  by (metis diff_zero gauge_trivial integrable_on_null integral_null norm_zero
sum_content_null)
```

Main limit theorem for an equiintegrable sequence.

```
theorem equiintegrable_limit:
  fixes g :: 'a :: euclidean_space ⇒ 'b :: banach
  assumes feq: range f equiintegrable_on cbox a b
      and to_g: ⋀x. x ∈ cbox a b ⟹ (λn. f n x) ⟶ g x
    shows g integrable_on cbox a b ∧ (λn. integral (cbox a b) (f n)) ⟶ integral
(cbox a b) g
proof −
  have Cauchy (λn. integral(cbox a b) (f n))
  proof (clarsimp simp add: Cauchy_def)
    fix e::real
    assume 0 < e
    then have e3: 0 < e/3
```

by *simp*
then obtain γ **where** *gauge* γ
and γ: $\bigwedge n\ \mathcal{D}$. $[\![\mathcal{D}$ *tagged_division_of cbox a b*; γ *fine* $\mathcal{D}]\!] \Longrightarrow$ *norm*$((\sum (x,K) \in \mathcal{D}.$ *content* $K *_R f\ n\ x) -$ *integral* (*cbox a b*) (*f n*)) $<$ *e/3*
using *feq* **unfolding** *equiintegrable_on_def*
by (*meson image_eqI iso_tuple_UNIV_I*)
obtain $\mathcal{D}$ **where** $\mathcal{D}$: $\mathcal{D}$ *tagged_division_of* (*cbox a b*) **and** γ *fine* $\mathcal{D}$ *finite* $\mathcal{D}$
by (*meson* ‹*gauge* γ› *fine_division_exists tagged_division_of_finite*)
with γ **have** δT: $\bigwedge n$. *dist* $((\sum (x,K) \in \mathcal{D}.$ *content* $K *_R f\ n\ x))$ (*integral* (*cbox a b*) (*f n*)) $<$ *e/3*
by (*force simp*: *dist_norm*)
have $(\lambda n. \sum (x,K) \in \mathcal{D}.$ *content* $K *_R f\ n\ x) \longrightarrow (\sum (x,K) \in \mathcal{D}.$ *content* $K *_R g\ x)$
using $\mathcal{D}$ *to_g* **by** (*auto intro*!: *tendsto_sum tendsto_scaleR*)
then have *Cauchy* $(\lambda n. \sum (x,K) \in \mathcal{D}.$ *content* $K *_R f\ n\ x)$
by (*meson convergent_eq_Cauchy*)
with *e3* **obtain** M **where**
M: $\bigwedge m\ n.\ [\![m \geq M;\ n \geq M]\!] \Longrightarrow$ *dist* $(\sum (x,K) \in \mathcal{D}.$ *content* $K *_R f\ m\ x)$ $(\sum (x,K) \in \mathcal{D}.$ *content* $K *_R f\ n\ x)$ $<$ *e/3*
unfolding *Cauchy_def* **by** *blast*
have $\bigwedge m\ n.\ [\![m \geq M;\ n \geq M;$ *dist* $(\sum (x,K) \in \mathcal{D}.$ *content* $K *_R f\ m\ x)$ $(\sum (x,K) \in \mathcal{D}.$ *content* $K *_R f\ n\ x) <$ *e/3*$]\!] \Longrightarrow$ *dist* (*integral* (*cbox a b*) (*f m*)) (*integral* (*cbox a b*) (*f n*)) $<$ *e*
by (*metis* δT *dist_commute dist_triangle_third* [*OF* _ _ δT])
then show $\exists M. \forall m \geq M. \forall n \geq M.$ *dist* (*integral* (*cbox a b*) (*f m*)) (*integral* (*cbox a b*) (*f n*)) $<$ *e*
using *M* **by** *auto*
qed
then obtain *L* **where** *L*: $(\lambda n.$ *integral* (*cbox a b*) (*f n*)) $\longrightarrow$ *L*
by (*meson convergent_eq_Cauchy*)
have (*g has_integral L*) (*cbox a b*)
proof (*clarsimp simp*: *has_integral*)
fix *e*::*real* **assume** *0 < e*
then have *e2*: *0 < e/2*
by *simp*
then obtain γ **where** *gauge* γ
and γ: $\bigwedge n\ \mathcal{D}$. $[\![\mathcal{D}$ *tagged_division_of cbox a b*; γ *fine* $\mathcal{D}]\!] \Longrightarrow$ *norm*$((\sum (x,K) \in \mathcal{D}.$ *content* $K *_R f\ n\ x) -$ *integral* (*cbox a b*) (*f n*)) $<$ *e/2*
using *feq* **unfolding** *equiintegrable_on_def*
by (*meson image_eqI iso_tuple_UNIV_I*)
moreover
have *norm* $((\sum (x,K) \in \mathcal{D}.$ *content* $K *_R g\ x) - L) <$ *e*
if $\mathcal{D}$ *tagged_division_of cbox a b* γ *fine* $\mathcal{D}$ **for** $\mathcal{D}$
proof −
have *norm* $((\sum (x,K) \in \mathcal{D}.$ *content* $K *_R g\ x) - L) \leq$ *e/2*

```
    proof (rule Lim_norm_ubound)
      show (λn. (∑(x,K)∈𝒟. content K *R f n x) - integral (cbox a b) (f n))
⟶ (∑(x,K)∈𝒟. content K *R g x) - L
       using to_g that L
        by (intro tendsto_diff tendsto_sum) (auto simp: tag_in_interval tend-
sto_scaleR)
      show ∀F n in sequentially.
             norm ((∑(x,K) ∈ 𝒟. content K *R f n x) - integral (cbox a b) (f
n)) ≤ e/2
       by (intro eventuallyI less_imp_le γ that)
    qed auto
    with ‹0 < e› show ?thesis
     by linarith
  qed
  ultimately
  show ∃γ. gauge γ ∧
        (∀𝒟. 𝒟 tagged_division_of cbox a b ∧ γ fine 𝒟 ⟶
          norm ((∑(x,K)∈𝒟. content K *R g x) - L) < e)
    by meson
 qed
 with L show ?thesis
  by (simp add: ‹(λn. integral (cbox a b) (f n)) ⟶ L› has_integral_integrable_integral)
qed


lemma equiintegrable_reflect:
 assumes F equiintegrable_on cbox a b
 shows (λf. f ∘ uminus) ` F equiintegrable_on cbox (-b) (-a)
proof -
 have §: ∃γ. gauge γ ∧
        (∀f 𝒟. f ∈ (λf. f ∘ uminus) ` F ∧ 𝒟 tagged_division_of cbox (- b) (-
a) ∧ γ fine 𝒟 ⟶
             norm ((∑(x,K) ∈ 𝒟. content K *R f x) - integral (cbox (- b) (-
a)) f) < e)
      if gauge γ and
        γ: ⋀f 𝒟. ⟦f ∈ F; 𝒟 tagged_division_of cbox a b; γ fine 𝒟⟧ ⟹
             norm ((∑(x,K) ∈ 𝒟. content K *R f x) - integral (cbox a b) f)
< e for e γ
 proof (intro exI, safe)
  show gauge (λx. uminus ` γ (-x))
   by (metis ‹gauge γ› gauge_reflect)
  show norm ((∑(x,K) ∈ 𝒟. content K *R (f ∘ uminus) x) - integral (cbox (-
b) (- a)) (f ∘ uminus)) < e
    if f ∈ F and tag: 𝒟 tagged_division_of cbox (- b) (- a)
     and fine: (λx. uminus ` γ (- x)) fine 𝒟 for f 𝒟
  proof -
   have 1: (λ(x,K). (- x, uminus ` K)) ` 𝒟 tagged_partial_division_of cbox a b
     if 𝒟 tagged_partial_division_of cbox (- b) (- a)
    proof -
```

```
      have − y ∈ cbox a b
        if ⋀x K. (x,K) ∈ 𝒟 ⟹ x ∈ K ∧ K ⊆ cbox (− b) (− a) ∧ (∃ a b. K =
cbox a b)
            (x, Y) ∈ 𝒟 y ∈ Y for x Y y
      proof −
        have y ∈ uminus ‘ cbox a b
          using that by auto
        then show − y ∈ cbox a b
          by force
      qed
      with that show ?thesis
          by (fastforce simp: tagged_partial_division_of_def interior_negations
image_iff)
    qed
    have 2: ∃ K. (∃ x. (x,K) ∈ (λ(x,K). (− x, uminus ‘ K)) ‘ 𝒟) ∧ x ∈ K
          if ⋃{K. ∃ x. (x,K) ∈ 𝒟} = cbox (− b) (− a) x ∈ cbox a b for x
    proof −
      have xm: x ∈ uminus ‘ ⋃{A. ∃ a. (a, A) ∈ 𝒟}
        by (simp add: that)
      then obtain a X where −x ∈ X (a, X) ∈ 𝒟
        by auto
      then show ?thesis
       by (metis (no_types, lifting) add.inverse_inverse image_iff pair_imageI)
    qed
    have 3: ⋀x X y. ⟦𝒟 tagged_partial_division_of cbox (− b) (− a); (x, X) ∈
𝒟; y ∈ X⟧ ⟹ − y ∈ cbox a b
    by (metis (no_types, lifting) equation_minus_iff imageE subsetD tagged_partial_division_ofD(3)
uminus_interval_vector)
    have tag′: (λ(x,K). (− x, uminus ‘ K)) ‘ 𝒟 tagged_division_of cbox a b
      using tag  by (auto simp: tagged_division_of_def dest: 1 2 3)
    have fine′: γ fine (λ(x,K). (− x, uminus ‘ K)) ‘ 𝒟
      using fine by (fastforce simp: fine_def)
    have inj: inj_on (λ(x,K). (− x, uminus ‘ K)) 𝒟
      unfolding inj_on_def by force
    have eq: content (uminus ‘ I) = content I
          if I: (x, I) ∈ 𝒟 and fnz: f (− x) ≠ 0 for x I
    proof −
      obtain a b where I = cbox a b
     using tag I that by (force simp: tagged_division_of_def tagged_partial_division_of_def)
      then show ?thesis
        using content_image_affinity_cbox [of −1 0] by auto
    qed
    have (∑(x,K) ∈ (λ(x,K). (− x, uminus ‘ K)) ‘ 𝒟.  content K ∗R f x) =
          (∑(x,K) ∈ 𝒟. content K ∗R f (− x))
      by (auto simp add: eq sum.reindex [OF inj] intro!: sum.cong)
    then show ?thesis
      using γ [OF ‹f ∈ F› tag′ fine′] integral_reflect
    by (metis (mono_tags, lifting) Henstock_Kurzweil_Integration.integral_cong
comp_apply split_def sum.cong)
```

```
    qed
  qed
   show ?thesis
    using assms
    apply (auto simp: equiintegrable_on_def)
    subgoal for f
      by (metis (mono_tags, lifting) comp_apply integrable_eq integrable_reflect)
    using § by fastforce
qed
```

6.24.2 Subinterval restrictions for equiintegrable families

First, some technical lemmas about minimizing a "flat" part of a sum over a division.

```
lemma lemma0:
  assumes i ∈ Basis
      shows content (cbox u v) / (interval_upperbound (cbox u v) · i − interval_lowerbound (cbox u v) · i) =
          (if content (cbox u v) = 0 then 0
             else ∏j ∈ Basis − {i}. interval_upperbound (cbox u v) · j − interval_lowerbound (cbox u v) · j)
proof (cases content (cbox u v) = 0)
  case True
  then show ?thesis by simp
next
  case False
  then show ?thesis
    using prod.subset_diff [of {i} Basis] assms
      by (force simp: content_cbox_if divide_simps  split: if_split_asm)
qed


lemma content_division_lemma1:
  assumes div: 𝒟 division_of S and S: S ⊆ cbox a b and i: i ∈ Basis
      and mt: ⋀K. K ∈ 𝒟 ⟹ content K ≠ 0
      and disj: (∀K ∈ 𝒟. K ∩ {x. x · i = a · i} ≠ {}) ∨ (∀K ∈ 𝒟. K ∩ {x. x · i = b · i} ≠ {})
    shows (b · i − a · i) * (∑K∈𝒟. content K / (interval_upperbound K · i − interval_lowerbound K · i))
          ≤ content(cbox a b)   (is ?lhs ≤ ?rhs)
proof −
  have finite 𝒟
    using div by blast
  define extend where
   extend ≡ λK. cbox (∑j ∈ Basis. if j = i then (a · i) *R i else (interval_lowerbound K · j) *R j)
                     (∑j ∈ Basis. if j = i then (b · i) *R i else (interval_upperbound K · j) *R j)
  have div_subset_cbox: ⋀K. K ∈ 𝒟 ⟹ K ⊆ cbox a b
```

```
    using S div by auto
  have ⋀K. K ∈ 𝒟 ⟹ K ≠ {}
    using div by blast
  have extend_cbox: ⋀K.  K ∈ 𝒟 ⟹ ∃ a b. extend K = cbox a b
    using extend_def by blast
  have extend: extend K ≠ {} extend K ⊆ cbox a b if K: K ∈ 𝒟 for K
  proof −
    obtain u v where K: K = cbox u v K ≠ {} K ⊆ cbox a b
      using K cbox_division_memE [OF _ div] by (meson div_subset_cbox)
    with i show extend K ⊆ cbox a b
      by (auto simp: extend_def subset_box box_ne_empty)
    have a · i ≤ b · i
      using K by (metis bot.extremum_uniqueI box_ne_empty(1) i)
    with K show extend K ≠ {}
      by (simp add: extend_def i box_ne_empty)
  qed
  have int_extend_disjoint:
      interior(extend K1) ∩ interior(extend K2) = {} if K: K1 ∈ 𝒟 K2 ∈ 𝒟 K1
≠ K2 for K1 K2
  proof −
    obtain u v where K1: K1 = cbox u v K1 ≠ {} K1 ⊆ cbox a b
      using K cbox_division_memE [OF _ div] by (meson div_subset_cbox)
    obtain w z where K2: K2 = cbox w z K2 ≠ {} K2 ⊆ cbox a b
      using K cbox_division_memE [OF _ div] by (meson div_subset_cbox)
    have cboxes: cbox u v ∈ 𝒟 cbox w z ∈ 𝒟 cbox u v ≠ cbox w z
      using K1 K2 that by auto
    with div have interior (cbox u v) ∩ interior (cbox w z) = {}
      by blast
    moreover
    have ∃ x. x ∈ box u v ∧ x ∈ box w z
        if x ∈ interior (extend K1) x ∈ interior (extend K2) for x
    proof −
      have a · i < x · i x · i < b · i
       and ux: ⋀k. k ∈ Basis − {i} ⟹ u · k < x · k
       and xv: ⋀k. k ∈ Basis − {i} ⟹ x · k < v · k
       and wx: ⋀k. k ∈ Basis − {i} ⟹ w · k < x · k
       and xz: ⋀k. k ∈ Basis − {i} ⟹ x · k < z · k
        using that K1 K2 i by (auto simp: extend_def box_ne_empty mem_box)
      have box u v ≠ {} box w z ≠ {}
       using cboxes interior_cbox by (auto simp: content_eq_0_interior dest: mt)
      then obtain q s
        where q: ⋀k. k ∈ Basis ⟹ w · k < q · k ∧ q · k < z · k
          and s: ⋀k. k ∈ Basis ⟹ u · k < s · k ∧ s · k < v · k
        by (meson all_not_in_conv mem_box(1))
      show ?thesis  using disj
      proof
        assume ∀ K∈𝒟. K ∩ {x. x · i = a · i} ≠ {}
        then have uva: (cbox u v) ∩ {x. x · i = a · i} ≠ {}
            and  wza: (cbox w z) ∩ {x. x · i = a · i} ≠ {}
```

using *cboxes* **by** (*auto simp*: *content_eq_0_interior*)
then obtain $r\ t$ **where** $r \cdot i = a \cdot i$ **and** r: $\bigwedge k.\ k \in Basis \Longrightarrow w \cdot k \le r \cdot k \wedge r \cdot k \le z \cdot k$
and $t \cdot i = a \cdot i$ **and** t: $\bigwedge k.\ k \in Basis \Longrightarrow u \cdot k \le t \cdot k \wedge t \cdot k \le v \cdot k$
by (*fastforce simp*: *mem_box*)
have u: $u \cdot i < q \cdot i$
using *i K2(1) K2(3)* ‹$t \cdot i = a \cdot i$› *q s t* [*OF i*] **by** (*force simp*: *subset_box*)
have w: $w \cdot i < s \cdot i$
using *i K1(1) K1(3)* ‹$r \cdot i = a \cdot i$› *s r* [*OF i*] **by** (*force simp*: *subset_box*)
define ξ **where** $\xi \equiv (\sum j \in Basis.\ \textit{if } j = i \textit{ then } min\ (q \cdot i)\ (s \cdot i) *_R i \textit{ else } (x \cdot j) *_R j)$
have [*simp*]: $\xi \cdot j = (\textit{if } j = i \textit{ then } min\ (q \cdot j)\ (s \cdot j) \textit{ else } x \cdot j)$ **if** $j \in Basis$ **for** j
unfolding ξ_*def*
by (*intro sum_if_inner that* ‹$i \in Basis$›)
show *?thesis*
proof (*intro exI conjI*)
have $min\ (q \cdot i)\ (s \cdot i) < v \cdot i$
using *i s* **by** *fastforce*
with ‹$i \in Basis$› *s u ux xv*
show $\xi \in box\ u\ v$
by (*force simp*: *mem_box*)
have $min\ (q \cdot i)\ (s \cdot i) < z \cdot i$
using *i q* **by** *force*
with ‹$i \in Basis$› *q w wx xz*
show $\xi \in box\ w\ z$
by (*force simp*: *mem_box*)
qed
next
assume $\forall K \in \mathcal{D}.\ K \cap \{x.\ x \cdot i = b \cdot i\} \neq \{\}$
then have *uva*: $(cbox\ u\ v) \cap \{x.\ x \cdot i = b \cdot i\} \neq \{\}$
and *wza*: $(cbox\ w\ z) \cap \{x.\ x \cdot i = b \cdot i\} \neq \{\}$
using *cboxes* **by** (*auto simp*: *content_eq_0_interior*)
then obtain $r\ t$ **where** $r \cdot i = b \cdot i$ **and** r: $\bigwedge k.\ k \in Basis \Longrightarrow w \cdot k \le r \cdot k \wedge r \cdot k \le z \cdot k$
and $t \cdot i = b \cdot i$ **and** t: $\bigwedge k.\ k \in Basis \Longrightarrow u \cdot k \le t \cdot k \wedge t \cdot k \le v \cdot k$
by (*fastforce simp*: *mem_box*)
have z: $s \cdot i < z \cdot i$
using *K1(1) K1(3)* ‹$r \cdot i = b \cdot i$› *r* [*OF i*] *i s* **by** (*force simp*: *subset_box*)
have v: $q \cdot i < v \cdot i$
using *K2(1) K2(3)* ‹$t \cdot i = b \cdot i$› *t* [*OF i*] *i q* **by** (*force simp*: *subset_box*)
define ξ **where** $\xi \equiv (\sum j \in Basis.\ \textit{if } j = i \textit{ then } max\ (q \cdot i)\ (s \cdot i) *_R i \textit{ else } (x \cdot j) *_R j)$
have [*simp*]: $\xi \cdot j = (\textit{if } j = i \textit{ then } max\ (q \cdot j)\ (s \cdot j) \textit{ else } x \cdot j)$ **if** $j \in Basis$ **for** j
unfolding ξ_*def*
by (*intro sum_if_inner that* ‹$i \in Basis$›)

```
      show ?thesis
      proof (intro exI conjI)
        show ξ ∈ box u v
          using ‹i ∈ Basis› s by (force simp: mem_box ux v xv)
        show ξ ∈ box w z
          using ‹i ∈ Basis› q by (force simp: mem_box wx xz z)
      qed
    qed
  qed
  ultimately show ?thesis by auto
 qed
 define interv_diff where interv_diff ≡ λK. λi::'a. interval_upperbound K · i
− interval_lowerbound K · i
 have ?lhs = (∑ K∈𝒟. (b · i − a · i) ∗ content K / (interv_diff K i))
  by (simp add: sum_distrib_left interv_diff_def)
 also have ... = sum (content ∘ extend) 𝒟
 proof (rule sum.cong [OF refl])
  fix K assume K ∈ 𝒟
  then obtain u v where K: K = cbox u v cbox u v ≠ {} K ⊆ cbox a b
   using cbox_division_memE [OF _ div] div_subset_cbox by metis
  then have uv: u · i < v · i
   using mt [OF ‹K ∈ 𝒟›] ‹i ∈ Basis› content_eq_0 by fastforce
  have insert i (Basis ∩ −{i}) = Basis
   using ‹i ∈ Basis› by auto
  then have (b · i − a · i) ∗ content K / (interv_diff K i)
          = (b · i − a · i) ∗ (∏ i ∈ insert i (Basis ∩ −{i}). v · i − u · i) /
(interv_diff (cbox u v) i)
   using K box_ne_empty(1) content_cbox by fastforce
  also have ... = (∏ x∈Basis. if x = i then b · x − a · x
                 else (interval_upperbound (cbox u v) − interval_lowerbound (cbox
u v)) · x)
   using ‹i ∈ Basis› K uv by (simp add: prod.If_cases interv_diff_def) (simp
add: algebra_simps)
  also have ... = (∏ k∈Basis.
                   (∑ j∈Basis. if j = i then (b · i − a · i) ∗R i
                                        else ((interval_upperbound (cbox u v) −
interval_lowerbound (cbox u v)) · j) ∗R j) · k)
   using ‹i ∈ Basis› by (subst prod.cong [OF refl sum_if_inner]; simp)
  also have ... = (∏ k∈Basis.
                   (∑ j∈Basis. if j = i then (b · i) ∗R i else (interval_upperbound
(cbox u v) · j) ∗R j) · k −
                   (∑ j∈Basis. if j = i then (a · i) ∗R i else (interval_lowerbound
(cbox u v) · j) ∗R j) · k)
   using ‹i ∈ Basis›
  by (intro prod.cong [OF refl]) (subst sum_if_inner; simp add: algebra_simps)+
  also have ... = (content ∘ extend) K
   using ‹i ∈ Basis› K box_ne_empty ‹K ∈ 𝒟› extend(1)
   by (auto simp add: extend_def content_cbox_if)
  finally show (b · i − a · i) ∗ content K / (interv_diff K i) = (content ∘
```

```
extend) K .
  qed
  also have ... = sum content (extend ` 𝒟)
  proof −
    have ⟦K1 ∈ 𝒟; K2 ∈ 𝒟; K1 ≠ K2; extend K1 = extend K2⟧ ⟹ content
(extend K1) = 0 for K1 K2
        using int_extend_disjoint [of K1 K2] extend_def by (simp add: con-
tent_eq_0_interior)
    then show ?thesis
      by (simp add: comm_monoid_add_class.sum.reindex_nontrivial [OF ‹finite
𝒟›])
  qed
  also have ... ≤ ?rhs
  proof (rule subadditive_content_division)
    show extend ` 𝒟 division_of ⋃ (extend ` 𝒟)
      using int_extend_disjoint by (auto simp: division_of_def ‹finite 𝒟› extend
extend_cbox)
    show ⋃ (extend ` 𝒟) ⊆ cbox a b
      using extend by fastforce
  qed
  finally show ?thesis .
qed


proposition sum_content_area_over_thin_division:
  assumes div: 𝒟 division_of S and S: S ⊆ cbox a b and i: i ∈ Basis
    and a · i ≤ c c ≤ b · i
    and nonmt: ⋀K. K ∈ 𝒟 ⟹ K ∩ {x. x · i = c} ≠ {}
  shows (b · i − a · i) ∗ (∑K∈𝒟. content K / (interval_upperbound K · i −
interval_lowerbound K · i))
        ≤ 2 ∗ content(cbox a b)
proof (cases content(cbox a b) = 0)
  case True
  have (∑K∈𝒟. content K / (interval_upperbound K · i − interval_lowerbound
K · i)) = 0
    using S div by (force intro!: sum.neutral content_0_subset [OF True])
  then show ?thesis
    by (auto simp: True)
next
  case False
  then have content(cbox a b) > 0
    using zero_less_measure_iff by blast
  then have a · i < b · i if i ∈ Basis for i
    using content_pos_lt_eq that by blast
  have finite 𝒟
    using div by blast
  define Dlec where Dlec ≡ {L ∈ (λL. L ∩ {x. x · i ≤ c}) ` 𝒟. content L ≠ 0}
  define Dgec where Dgec ≡ {L ∈ (λL. L ∩ {x. x · i ≥ c}) ` 𝒟. content L ≠ 0}
  define a′ where a′ ≡ (∑j∈Basis. (if j = i then c else a · j) ∗R j)
```

define b' **where** $b' \equiv (\sum j{\in}Basis.\ (if\ j = i\ then\ c\ else\ b \cdot j) *_R j)$
define *interv_diff* **where** *interv_diff* $\equiv \lambda K.\ \lambda i{::}'a.\ interval\_upperbound\ K \cdot i - interval\_lowerbound\ K \cdot i$
have *Dlec_cbox*: $\bigwedge K.\ K \in Dlec \Longrightarrow \exists\, a\ b.\ K = cbox\ a\ b$
using *interval_split* [*OF i*] *div* **by** (*fastforce simp*: *Dlec_def division_of_def*)
then have *lec_is_cbox*: $[\![content\ (L \cap \{x.\ x \cdot i \le c\}) \neq 0;\ L \in \mathcal{D}]\!] \Longrightarrow \exists\, a\ b.\ L \cap \{x.\ x \cdot i \le c\} = cbox\ a\ b$ **for** L
using *Dlec_def* **by** *blast*
have *Dgec_cbox*: $\bigwedge K.\ K \in Dgec \Longrightarrow \exists\, a\ b.\ K = cbox\ a\ b$
using *interval_split* [*OF i*] *div* **by** (*fastforce simp*: *Dgec_def division_of_def*)
then have *gec_is_cbox*: $[\![content\ (L \cap \{x.\ x \cdot i \ge c\}) \neq 0;\ L \in \mathcal{D}]\!] \Longrightarrow \exists\, a\ b.\ L \cap \{x.\ x \cdot i \ge c\} = cbox\ a\ b$ **for** L
using *Dgec_def* **by** *blast*

have *zero_left*: $\bigwedge x\ y.\ [\![x \in \mathcal{D};\ y \in \mathcal{D};\ x \neq y;\ x \cap \{x.\ x \cdot i \le c\} = y \cap \{x.\ x \cdot i \le c\}]\!]$
$\Longrightarrow content\ (y \cap \{x.\ x \cdot i \le c\}) = 0$
by (*metis division_split_left_inj* [*OF div*] *lec_is_cbox content_eq_0_interior*)
have *zero_right*: $\bigwedge x\ y.\ [\![x \in \mathcal{D};\ y \in \mathcal{D};\ x \neq y;\ x \cap \{x.\ c \le x \cdot i\} = y \cap \{x.\ c \le x \cdot i\}]\!]$
$\Longrightarrow content\ (y \cap \{x.\ c \le x \cdot i\}) = 0$
by (*metis division_split_right_inj* [*OF div*] *gec_is_cbox content_eq_0_interior*)

have $(b' \cdot i - a \cdot i) * (\sum K{\in}Dlec.\ content\ K\ /\ interv\_diff\ K\ i) \le content(cbox\ a\ b')$
unfolding *interv_diff_def*
proof (*rule content_division_lemma1*)
show *Dlec division_of* $\bigcup Dlec$
unfolding *division_of_def*
proof (*intro conjI ballI Dlec_cbox*)
show $\bigwedge K1\ K2.\ [\![K1 \in Dlec;\ K2 \in Dlec]\!] \Longrightarrow K1 \neq K2 \longrightarrow interior\ K1 \cap interior\ K2 = \{\}$
by (*clarsimp simp*: *Dlec_def*) (*use div* **in** *auto*)
qed (*use* ‹*finite* $\mathcal{D}$› *Dlec_def* **in** *auto*)
show $\bigcup Dlec \subseteq cbox\ a\ b'$
using *Dlec_def div S* **by** (*auto simp*: b'*_def division_of_def mem_box*)
show $(\forall K{\in}Dlec.\ K \cap \{x.\ x \cdot i = a \cdot i\} \neq \{\}) \vee (\forall K{\in}Dlec.\ K \cap \{x.\ x \cdot i = b' \cdot i\} \neq \{\})$
using *nonmt* **by** (*fastforce simp*: *Dlec_def* b'*_def i*)
qed (*use i Dlec_def* **in** *auto*)
moreover
have $(\sum K{\in}Dlec.\ content\ K\ /\ (interv\_diff\ K\ i)) = (\sum K{\in}(\lambda K.\ K \cap \{x.\ x \cdot i \le c\})\ `\ \mathcal{D}.\ content\ K\ /\ interv\_diff\ K\ i)$
unfolding *Dlec_def* **using** ‹*finite* $\mathcal{D}$› **by** (*auto simp*: *sum.mono_neutral_left*)
moreover have ... $=$
$(\sum K{\in}\mathcal{D}.\ ((\lambda K.\ content\ K\ /\ (interv\_diff\ K\ i)) \circ ((\lambda K.\ K \cap \{x.\ x \cdot i \le c\}))\ K)$
by (*simp add*: *zero_left sum.reindex_nontrivial* [*OF* ‹*finite* $\mathcal{D}$›])
moreover have $(b' \cdot i - a \cdot i) = (c - a \cdot i)$

by (*simp add*: $b'\_def\ i$)
ultimately
have *lec*: $(c - a \cdot i) * (\sum K{\in}\mathcal{D}.\ ((\lambda K.\ content\ K\ /\ (interv\_diff\ K\ i)) \circ ((\lambda K.\ K \cap \{x.\ x \cdot i \le c\}))\ K)$
$\le content(cbox\ a\ b')$
by *simp*

have $(b \cdot i - a' \cdot i) * (\sum K{\in}Dgec.\ content\ K\ /\ (interv\_diff\ K\ i)) \le content(cbox\ a'\ b)$
unfolding *interv_diff_def*
proof (*rule content_division_lemma1*)
show *Dgec division_of* $\bigcup Dgec$
unfolding *division_of_def*
proof (*intro conjI ballI Dgec_cbox*)
show $\bigwedge K1\ K2.\ [\![K1 \in Dgec; K2 \in Dgec]\!] \Longrightarrow K1 \neq K2 \longrightarrow interior\ K1 \cap interior\ K2 = \{\}$
by (*clarsimp simp*: *Dgec_def*) (*use div* **in** *auto*)
qed (*use* ‹*finite* $\mathcal{D}$› *Dgec_def* **in** *auto*)
show $\bigcup Dgec \subseteq cbox\ a'\ b$
using *Dgec_def div S* **by** (*auto simp*: a'_def *division_of_def mem_box*)
show $(\forall K{\in}Dgec.\ K \cap \{x.\ x \cdot i = a' \cdot i\} \neq \{\}) \vee (\forall K{\in}Dgec.\ K \cap \{x.\ x \cdot i = b \cdot i\} \neq \{\})$
using *nonmt* **by** (*fastforce simp*: *Dgec_def* $a'\_def\ i$)
qed (*use i Dgec_def* **in** *auto*)
moreover
have $(\sum K{\in}Dgec.\ content\ K\ /\ (interv\_diff\ K\ i)) = (\sum K{\in}(\lambda K.\ K \cap \{x.\ c \le x \cdot i\})\ `\mathcal{D}.$
$content\ K\ /\ interv\_diff\ K\ i)$
unfolding *Dgec_def* **using** ‹*finite* $\mathcal{D}$› **by** (*auto simp*: *sum.mono_neutral_left*)
moreover have $\ldots =$
$(\sum K{\in}\mathcal{D}.\ ((\lambda K.\ content\ K\ /\ (interv\_diff\ K\ i)) \circ ((\lambda K.\ K \cap \{x.\ x \cdot i \ge c\}))\ K)$
by (*simp add*: *zero_right sum.reindex_nontrivial* [*OF* ‹*finite* $\mathcal{D}$›])
moreover have $(b \cdot i - a' \cdot i) = (b \cdot i - c)$
by (*simp add*: $a'\_def\ i$)
ultimately
have *gec*: $(b \cdot i - c) * (\sum K{\in}\mathcal{D}.\ ((\lambda K.\ content\ K\ /\ (interv\_diff\ K\ i)) \circ ((\lambda K.\ K \cap \{x.\ x \cdot i \ge c\}))\ K)$
$\le content(cbox\ a'\ b)$
by *simp*

show *?thesis*
proof (*cases* $c = a \cdot i \vee c = b \cdot i$)
case *True*
then show *?thesis*
proof
assume *c*: $c = a \cdot i$
moreover
have $(\sum j{\in}Basis.\ (if\ j = i\ then\ a \cdot i\ else\ a \cdot j) *_R j) = a$

```
      using euclidean_representation [of a] sum.cong [OF refl, of Basis λi. (a ·
i) *R i] by presburger
    ultimately have a' = a
      by (simp add: i a'_def cong: if_cong)
    then have content (cbox a' b) ≤ 2 * content (cbox a b)  by simp
    moreover
    have eq: (∑ K∈𝒟. content (K ∩ {x. a · i ≤ x · i}) / interv_diff (K ∩ {x.
a · i ≤ x · i}) i)
          = (∑ K∈𝒟. content K / interv_diff K i)
      (is sum ?f _ = sum ?g _)
    proof (rule sum.cong [OF refl])
      fix K assume K ∈ 𝒟
      then have a · i ≤ x · i if x ∈ K for x
        by (metis S UnionI div division_ofD(6) i mem_box(2) subsetCE that)
      then have K ∩ {x. a · i ≤ x · i} = K
        by blast
      then show ?f K = ?g K
        by simp
    qed
    ultimately show ?thesis
      using gec c eq interv_diff_def by auto
  next
    assume c: c = b · i
    moreover have (∑ j∈Basis. (if j = i then b · i else b · j) *R j) = b
      using euclidean_representation [of b] sum.cong [OF refl, of Basis λi. (b ·
i) *R i] by presburger
    ultimately have b' = b
      by (simp add: i b'_def cong: if_cong)
    then have content (cbox a b') ≤ 2 * content (cbox a b)  by simp
    moreover
    have eq: (∑ K∈𝒟. content (K ∩ {x. x · i ≤ b · i}) / interv_diff (K ∩ {x. x
· i ≤ b · i}) i)
          = (∑ K∈𝒟. content K / interv_diff K i)
       (is sum ?f _ = sum ?g _)
    proof (rule sum.cong [OF refl])
      fix K assume K ∈ 𝒟
      then have x · i ≤ b · i if x ∈ K for x
        by (metis S UnionI div division_ofD(6) i mem_box(2) subsetCE that)
      then have K ∩ {x. x · i ≤ b · i} = K
        by blast
      then show ?f K = ?g K
        by simp
    qed
    ultimately show ?thesis
      using lec c eq interv_diff_def by auto
  qed
 next
  case False
   have prod_if: (∏ k∈Basis ∩ − {i}. f k) = (∏ k∈Basis. f k) / f i if f i ≠
```

```
(0::real) for f
  proof −
    have f i ∗ prod f (Basis ∩ − {i}) = prod f Basis
      using that mk_disjoint_insert [OF i]
     by (metis Int_insert_left_if0 finite_Basis finite_insert le_iff_inf order_refl
prod.insert subset_Compl_singleton)
    then show ?thesis
      by (metis nonzero_mult_div_cancel_left that)
  qed
  have abc: a · i < c c < b · i
    using False assms by auto
  then have (∑K∈𝒟. ((λK. content K / (interv_diff K i)) ∘ ((λK. K ∩ {x. x
· i ≤ c}))) K)
              ≤ content(cbox a b′) / (c − a · i)
            (∑K∈𝒟. ((λK. content K / (interv_diff K i)) ∘ ((λK. K ∩ {x. x · i
≥ c}))) K)
              ≤ content(cbox a′ b) / (b · i − c)
    using lec gec by (simp_all add: field_split_simps)
  moreover
  have (∑K∈𝒟. content K / (interv_diff K i))
       ≤ (∑K∈𝒟. ((λK. content K / (interv_diff K i)) ∘ ((λK. K ∩ {x. x · i
≤ c}))) K) +
         (∑K∈𝒟. ((λK. content K / (interv_diff K i)) ∘ ((λK. K ∩ {x. x · i ≥
c}))) K)
        (is ?lhs ≤ ?rhs)
  proof −
    have ?lhs ≤
         (∑K∈𝒟. ((λK. content K / (interv_diff K i)) ∘ ((λK. K ∩ {x. x · i ≤
c}))) K +
                  ((λK. content K / (interv_diff K i)) ∘ ((λK. K ∩ {x. x · i ≥
c}))) K)
         (is sum ?f _ ≤ sum ?g _)
    proof (rule sum_mono)
      fix K assume K ∈ 𝒟
      then obtain u v where uv: K = cbox u v
        using div by blast
      obtain u′ v′ where uv′: cbox u v ∩ {x. x · i ≤ c} = cbox u v′
                            cbox u v ∩ {x. c ≤ x · i} = cbox u′ v
                            ⋀k. k ∈ Basis ⟹ u′ · k = (if k = i then max (u · i) c
else u · k)
                            ⋀k. k ∈ Basis ⟹ v′ · k = (if k = i then min (v · i) c
else v · k)
        using i by (auto simp: interval_split)
      have ∗: ⟦content (cbox u v′) = 0; content (cbox u′ v) = 0⟧ ⟹ content (cbox
u v) = 0
              content (cbox u′ v) ≠ 0 ⟹ content (cbox u v) ≠ 0
              content (cbox u v′) ≠ 0 ⟹ content (cbox u v) ≠ 0
      using i uv uv′ by (auto simp: content_eq_0 le_max_iff_disj min_le_iff_disj
split: if_split_asm intro: order_trans)
```

```
      have uniq: ⋀j. ⟦j ∈ Basis; ¬ u * j ≤ v * j⟧ ⟹ j = i
        by (metis ‹K ∈ 𝒟› box_ne_empty(1) div division_of_def uv)
      show ?f K ≤ ?g K
        using i uv uv' by (auto simp add: interv_diff_def lemma0 dest: uniq *
intro!: prod_nonneg)
    qed
    also have ... = ?rhs
      by (simp add: sum.distrib)
    finally show ?thesis .
  qed
  moreover have content (cbox a b') / (c - a * i) = content (cbox a b) / (b * i
- a * i)
    using i abc
     apply (simp add: field_simps a'_def b'_def measure_lborel_cbox_eq in-
ner_diff)
     apply (auto simp: if_distrib if_distrib [of λf. f x for x] prod.If_cases [of
Basis λx. x = i, simplified] prod_if field_simps)
    done
  moreover have content (cbox a' b) / (b * i - c) = content (cbox a b) / (b * i
- a * i)
    using i abc
     apply (simp add: field_simps a'_def b'_def measure_lborel_cbox_eq in-
ner_diff)
     apply (auto simp: if_distrib prod.If_cases [of Basis λx. x = i, simplified]
prod_if field_simps)
    done
  ultimately
  have (∑ K∈𝒟. content K / (interv_diff K i)) ≤ 2 * content (cbox a b) / (b *
i - a * i)
    by linarith
  then show ?thesis
    using abc interv_diff_def by (simp add: field_split_simps)
 qed
qed


proposition bounded_equiintegral_over_thin_tagged_partial_division:
 fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
 assumes F: F equiintegrable_on cbox a b and f: f ∈ F and 0 < ε
    and norm_f: ⋀h x. ⟦h ∈ F; x ∈ cbox a b⟧ ⟹ norm(h x) ≤ norm(f x)
 obtains γ where gauge γ
          ⋀c i S h. ⟦c ∈ cbox a b; i ∈ Basis; S tagged_partial_division_of cbox a
b;
                    γ fine S; h ∈ F; ⋀x K. (x,K) ∈ S ⟹ (K ∩ {x. x * i = c * i}
≠ {})⟧
                    ⟹ (∑(x,K) ∈ S. norm (integral K h)) < ε
proof (cases content(cbox a b) = 0)
 case True
 show ?thesis
```

```
  proof
    show gauge (λx. ball x 1)
      by (simp add: gauge_trivial)
    show (∑(x,K) ∈ S. norm (integral K h)) < ε
        if S tagged_partial_division_of cbox a b (λx. ball x 1) fine S for S and
h:: 'a ⇒ 'b
    proof −
      have (∑(x,K) ∈ S. norm (integral K h)) = 0
        using that True content_0_subset
        by (fastforce simp: tagged_partial_division_of_def intro: sum.neutral)
      with ‹0 < ε› show ?thesis
        by simp
    qed
  qed
next
  case False
  then have contab_gt0:  content(cbox a b) > 0
    by (simp add: zero_less_measure_iff)
  then have a_less_b: ⋀i. i ∈ Basis ⟹ a•i < b•i
    by (auto simp: content_pos_lt_eq)
  obtain γ0 where gauge γ0
        and γ0: ⋀S h. ⟦S tagged_partial_division_of cbox a b; γ0 fine S; h ∈ F⟧
                      ⟹ (∑(x,K) ∈ S. norm (content K *R h x − integral K
h)) < ε/2
  proof −
    obtain γ where gauge γ
          and γ: ⋀f D. ⟦f ∈ F; D tagged_division_of cbox a b; γ fine D⟧
                       ⟹ norm ((∑(x,K) ∈ D. content K *R f x) − integral
(cbox a b) f)
                          < ε/(5 * (Suc DIM('b)))
    proof −
      have e5: ε/(5 * (Suc DIM('b))) > 0
        using ‹ε > 0› by auto
      then show ?thesis
        using F that by (auto simp: equiintegrable_on_def)
    qed
    show ?thesis
    proof
      show gauge γ
        by (rule ‹gauge γ›)
      show (∑(x,K) ∈ S. norm (content K *R h x − integral K h)) < ε/2
          if S tagged_partial_division_of cbox a b γ fine S h ∈ F for S h
      proof −
        have (∑(x,K) ∈ S. norm (content K *R h x − integral K h)) ≤ 2 * real
DIM('b) * (ε/(5 * Suc DIM('b)))
        proof (rule Henstock_lemma_part2 [of h a b])
          show h integrable_on cbox a b
            using that F equiintegrable_on_def by metis
          show gauge γ
```

```
        by (rule ‹gauge γ›)
      qed (use that ‹ε > 0› γ in auto)
      also have ... < ε/2
        using ‹ε > 0› by (simp add: divide_simps)
      finally show ?thesis .
    qed
  qed
 qed
 define γ where γ ≡ λx. γ0 x ∩
                      ball x ((ε/8 / (norm(f x) + 1)) * (INF m∈Basis. b · m − a
· m) / content(cbox a b))
 define interv_diff where interv_diff ≡ λK. λi::'a. interval_upperbound K · i
− interval_lowerbound K · i
 have 8 * content (cbox a b) + norm (f x) * (8 * content (cbox a b)) > 0 for x
  by (metis add.right_neutral add_pos_pos contab_gt0 mult_pos_pos mult_zero_left
norm_eq_zero zero_less_norm_iff zero_less_numeral)
 then have gauge (λx. ball x
                  (ε * (INF m∈Basis. b · m − a · m) / ((8 * norm (f x) + 8) *
content (cbox a b))))
   using ‹0 < content (cbox a b)› ‹0 < ε› a_less_b
    by (auto simp add: gauge_def field_split_simps add_nonneg_eq_0_iff fi-
nite_less_Inf_iff)
 then have gauge γ
   unfolding γ_def using ‹gauge γ0› gauge_Int by auto
 moreover
 have (∑(x,K) ∈ S. norm (integral K h)) < ε
     if c ∈ cbox a b i ∈ Basis and S: S tagged_partial_division_of cbox a b
       and γ fine S h ∈ F and ne: ⋀x K. (x,K) ∈ S ⟹ K ∩ {x. x · i = c · i}
≠ {} for c i S h
 proof −
   have cbox c b ⊆ cbox a b
    by (meson mem_box(2) order_refl subset_box(1) that(1))
   have finite S
    using S unfolding tagged_partial_division_of_def by blast
   have γ0 fine S and fineS:
       (λx. ball x (ε * (INF m∈Basis. b · m − a · m) / ((8 * norm (f x) + 8) *
content (cbox a b)))) fine S
    using ‹γ fine S› by (auto simp: γ_def fine_Int)
   then have (∑(x,K) ∈ S. norm (content K *R h x − integral K h)) < ε/2
    by (intro γ0 that fineS)
   moreover have (∑(x,K) ∈ S. norm (integral K h) − norm (content K *R h
x − integral K h)) ≤ ε/2
   proof −
    have (∑(x,K) ∈ S. norm (integral K h) − norm (content K *R h x − integral
K h))
          ≤ (∑(x,K) ∈ S. norm (content K *R h x))
     proof (clarify intro!: sum_mono)
      fix x K
      assume xK: (x,K) ∈ S
```

```
      have norm (integral K h) − norm (content K *R h x − integral K h) ≤
norm (integral K h − (integral K h − content K *R h x))
        by (metis norm_minus_commute norm_triangle_ineq2)
      also have ... ≤ norm (content K *R h x)
        by simp
      finally show norm (integral K h) − norm (content K *R h x − integral K
h) ≤ norm (content K *R h x) .
    qed
    also have ... ≤ (∑(x,K) ∈ S. ε/4 * (b · i − a · i) / content (cbox a b) *
content K / interv_diff K i)
    proof (clarify intro!: sum_mono)
      fix x K
      assume xK: (x,K) ∈ S
      then have x: x ∈ cbox a b
        using S unfolding tagged_partial_division_of_def by (meson subset_iff)
      show norm (content K *R h x) ≤ ε/4 * (b · i − a · i) / content (cbox a b)
* content K / interv_diff K i
      proof (cases content K = 0)
        case True
        then show ?thesis by simp
      next
        case False
        then have Kgt0: content K > 0
          using zero_less_measure_iff by blast
        moreover
        obtain u v where uv: K = cbox u v
          using S ‹(x,K) ∈ S› unfolding tagged_partial_division_of_def by
blast
        then have u_less_v: ⋀i. i ∈ Basis ⟹ u · i < v · i
          using content_pos_lt_eq uv Kgt0 by blast
        then have dist_uv: dist u v > 0
          using that by auto
        ultimately have norm (h x) ≤ (ε * (b · i − a · i)) / (4 * content (cbox
a b) * interv_diff K i)
        proof −
          have dist x u < ε * (INF m∈Basis. b · m − a · m) / (4 * (norm (f x)
+ 1) * content (cbox a b)) / 2
               dist x v < ε * (INF m∈Basis. b · m − a · m) / (4 * (norm (f x) +
1) * content (cbox a b)) / 2
            using fineS u_less_v uv xK
            by (force simp: fine_def mem_box field_simps dest!: bspec)+
          moreover have ε * (INF m∈Basis. b · m − a · m) / (4 * (norm (f x)
+ 1) * content (cbox a b)) / 2
                ≤ ε * (b · i − a · i) / (4 * (norm (f x) + 1) * content (cbox a b))
/ 2
          proof (intro mult_left_mono divide_right_mono)
            show (INF m∈Basis. b · m − a · m) ≤ b · i − a · i
              using ‹i ∈ Basis› by (auto intro!: cInf_le_finite)
          qed (use ‹0 < ε› in auto)
```

```
          ultimately
           have dist x u < ε * (b • i − a • i) / (4 * (norm (f x) + 1) * content
(cbox a b)) / 2
              dist x v < ε * (b • i − a • i) / (4 * (norm (f x) + 1) * content (cbox
a b)) / 2
            by linarith+
          then have duv: dist u v < ε * (b • i − a • i) / (4 * (norm (f x) + 1) *
content (cbox a b))
            using dist_triangle_half_r by blast
          have uvi: |v • i − u • i| ≤ norm (v − u)
          by (metis inner_commute inner_diff_right ‹i ∈ Basis› Basis_le_norm)
          have norm (h x) ≤ norm (f x)
            using x that by (auto simp: norm_f)
          also have ... < (norm (f x) + 1)
            by simp
          also have ... < ε * (b • i − a • i) / dist u v / (4 * content (cbox a b))
          proof −
            have 0 < norm (f x) + 1
              by (simp add: add.commute add_pos_nonneg)
            then show ?thesis
              using duv dist_uv contab_gt0
              by (simp only: mult_ac divide_simps) auto
          qed
          also have ... = ε * (b • i − a • i) / norm (v − u) / (4 * content (cbox
a b))
            by (simp add: dist_norm norm_minus_commute)
          also have ... ≤ ε * (b • i − a • i) / |v • i − u • i| / (4 * content (cbox
a b))
            proof (intro mult_right_mono divide_left_mono divide_right_mono
uvi)
            show norm (v − u) * |v • i − u • i| > 0
              using u_less_v [OF ‹i ∈ Basis›]
              by (auto simp: less_eq_real_def zero_less_mult_iff that)
            show ε * (b • i − a • i) ≥ 0
              using a_less_b ‹0 < ε› ‹i ∈ Basis› by force
          qed auto
         also have ... = ε * (b • i − a • i) / (4 * content (cbox a b) * interv_diff
K i)
            using uv False that(2) u_less_v interv_diff_def by fastforce
          finally show ?thesis by simp
        qed
         with Kgt0 have norm (content K *R h x) ≤ content K * ((ε/4 * (b • i
− a • i) / content (cbox a b)) / interv_diff K i)
          using mult_left_mono by fastforce
         also have ... = ε/4 * (b • i − a • i) / content (cbox a b) * content K /
interv_diff K i
          by (simp add: field_split_simps)
        finally show ?thesis .
      qed
```

```
    qed
    also have ... = (∑K∈snd ` S. ε/4 * (b · i − a · i) / content (cbox a b) *
content K / interv_diff K i)
      unfolding interv_diff_def
    apply (rule sum.over_tagged_division_lemma [OF tagged_partial_division_of_Union_self
[OF S]])
      apply (simp add: box_eq_empty(1) content_eq_0)
      done
    also have ... = ε/2 * ((b · i − a · i) / (2 * content (cbox a b)) * (∑K∈snd
` S. content K / interv_diff K i))
      by (simp add: interv_diff_def sum_distrib_left mult.assoc)
    also have ... ≤ (ε/2) * 1
    proof (rule mult_left_mono)
      have (b · i − a · i) * (∑K∈snd ` S. content K / interv_diff K i) ≤ 2 *
content (cbox a b)
        unfolding interv_diff_def
      proof (rule sum_content_area_over_thin_division)
        show snd ` S division_of ⋃(snd ` S)
       by (auto intro: S tagged_partial_division_of_Union_self division_of_tagged_division)
        show ⋃(snd ` S) ⊆ cbox a b
          using S unfolding tagged_partial_division_of_def by force
        show a · i ≤ c · i c · i ≤ b · i
          using mem_box(2) that by blast+
      qed (use that in auto)
       then show (b · i − a · i) / (2 * content (cbox a b)) * (∑K∈snd ` S.
content K / interv_diff K i) ≤ 1
        by (simp add: contab_gt0)
    qed (use ‹0 < ε› in auto)
    finally show ?thesis by simp
  qed
  then have (∑(x,K) ∈ S. norm (integral K h)) − (∑(x,K) ∈ S. norm (content
K *R h x − integral K h)) ≤ ε/2
    by (simp add: Groups_Big.sum_subtractf [symmetric])
  ultimately show (∑(x,K) ∈ S. norm (integral K h)) < ε
    by linarith
 qed
 ultimately show ?thesis using that by auto
qed



proposition equiintegrable_halfspace_restrictions_le:
 fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
 assumes F: F equiintegrable_on cbox a b and f: f ∈ F
   and norm_f: ⋀h x. ⟦h ∈ F; x ∈ cbox a b⟧ ⟹ norm(h x) ≤ norm(f x)
 shows (⋃i ∈ Basis. ⋃c. ⋃h ∈ F. {(λx. if x · i ≤ c then h x else 0)})
       equiintegrable_on cbox a b
proof (cases content(cbox a b) = 0)
 case True
```

```
  then show ?thesis by simp
next
  case False
  then have content(cbox a b) > 0
    using zero_less_measure_iff by blast
  then have a · i < b · i if i ∈ Basis for i
    using content_pos_lt_eq that by blast
  have int_F: f integrable_on cbox a b if f ∈ F for f
    using F that by (simp add: equiintegrable_on_def)
  let ?CI = λK h x. content K *R h x − integral K h
  show ?thesis
    unfolding equiintegrable_on_def
  proof (intro conjI; clarify)
    show int_lec: ⟦i ∈ Basis; h ∈ F⟧ ⟹ (λx. if x · i ≤ c then h x else 0)
integrable_on cbox a b for i c h
      using integrable_restrict_Int [of {x. x · i ≤ c} h]
      by (simp add: inf_commute int_F integrable_split(1))
    show ∃γ. gauge γ ∧
            (∀f T. f ∈ (⋃i∈Basis. ⋃c. ⋃h∈F. {λx. if x · i ≤ c then h x else 0})
∧
                    T tagged_division_of cbox a b ∧ γ fine T ⟶
                    norm ((∑(x,K) ∈ T. content K *R f x) − integral (cbox a b) f)
< ε)
      if ε > 0 for ε
    proof −
      obtain γ0 where gauge γ0 and γ0:
        ⋀c i S h. ⟦c ∈ cbox a b; i ∈ Basis; S tagged_partial_division_of cbox a b;
                      γ0 fine S; h ∈ F; ⋀x K. (x,K) ∈ S ⟹ (K ∩ {x. x · i = c ·
i} ≠ {})⟧
                     ⟹ (∑(x,K) ∈ S. norm (integral K h)) < ε/12
      proof (rule bounded_equiintegral_over_thin_tagged_partial_division [OF F
f, of ‹ε/12›])
        show ⋀h x. ⟦h ∈ F; x ∈ cbox a b⟧ ⟹ norm (h x) ≤ norm (f x)
          by (auto simp: norm_f)
      qed (use ‹ε > 0› in auto)
      obtain γ1 where gauge γ1
        and γ1: ⋀h T. ⟦h ∈ F; T tagged_division_of cbox a b; γ1 fine T⟧
                              ⟹ norm ((∑(x,K) ∈ T. content K *R h x) − integral
(cbox a b) h)
                                 < ε/(7 * (Suc DIM('b)))
      proof −
        have e5: ε/(7 * (Suc DIM('b))) > 0
          using ‹ε > 0› by auto
        then show ?thesis
          using F that by (auto simp: equiintegrable_on_def)
      qed
      have h_less3: (∑(x,K) ∈ T. norm (?CI K h x)) < ε/3
        if T tagged_partial_division_of cbox a b γ1 fine T h ∈ F for T h
      proof −
```

```
      have (∑(x,K) ∈ T. norm (?CI K h x)) ≤ 2 * real DIM('b) * (ε/(7 * Suc
DIM('b)))
      proof (rule Henstock_lemma_part2 [of h a b])
        show h integrable_on cbox a b
          using that F equiintegrable_on_def by metis
      qed (use that ‹ε > 0› ‹gauge γ1› γ1 in auto)
      also have ... < ε/3
        using ‹ε > 0› by (simp add: divide_simps)
      finally show ?thesis .
    qed
    have *: norm ((∑(x,K) ∈ T. content K *R f x) − integral (cbox a b) f) < ε
           if f: f = (λx. if x · i ≤ c then h x else 0)
           and T: T tagged_division_of cbox a b
          and fine: (λx. γ0 x ∩ γ1 x) fine T and i ∈ Basis h ∈ F for f T i c h
    proof (cases a · i ≤ c ∧ c ≤ b · i)
      case True
      have finite T
        using T by blast
      define T' where T' ≡ {(x,K) ∈ T. K ∩ {x. x · i ≤ c} ≠ {}}
      then have T' ⊆ T
        by auto
      then have finite T'
        using ‹finite T› infinite_super by blast
      have T'_tagged: T' tagged_partial_division_of cbox a b
      by (meson T ‹T' ⊆ T› tagged_division_of_def tagged_partial_division_subset)
      have fine': γ0 fine T' γ1 fine T'
        using ‹T' ⊆ T› fine_Int fine_subset fine by blast+
      have int_KK': (∑(x,K) ∈ T. integral K f) = (∑(x,K) ∈ T'. integral K f)
      proof (rule sum.mono_neutral_right [OF ‹finite T› ‹T' ⊆ T›])
        show ∀i ∈ T − T'. (case i of (x, K) ⇒ integral K f) = 0
          using f ‹finite T› ‹T' ⊆ T› integral_restrict_Int [of _ {x. x · i ≤ c} h]
          by (auto simp: T'_def Int_commute)
      qed
      have (∑(x,K) ∈ T. content K *R f x) = (∑(x,K) ∈ T'. content K *R f
x)
      proof (rule sum.mono_neutral_right [OF ‹finite T› ‹T' ⊆ T›])
        show ∀i ∈ T − T'. (case i of (x, K) ⇒ content K *R f x) = 0
          using T f ‹finite T› ‹T' ⊆ T› by (force simp: T'_def)
      qed
      moreover have norm ((∑(x,K) ∈ T'. content K *R f x) − integral (cbox
a b) f) < ε
      proof −
        have *: norm y < ε if norm x < ε/3 norm(x − y) ≤ 2 * ε/3 for x y::'b
        proof −
          have norm y ≤ norm x + norm(x − y)
            by (metis norm_minus_commute norm_triangle_sub)
          also have ... < ε/3 + 2*ε/3
            using that by linarith
          also have ... = ε
```

```
      by simp
    finally show ?thesis .
  qed
  have norm (∑(x,K) ∈ T'. ?CI K h x)
      ≤ (∑(x,K) ∈ T'. norm (?CI K h x))
    by (simp add: norm_sum split_def)
  also have ... < ε/3
    by (intro h_less3 T'_tagged fine' that)
  finally have norm (∑(x,K) ∈ T'. ?CI K h x) < ε/3 .
  moreover have integral (cbox a b) f = (∑(x,K) ∈ T. integral K f)
using int_lec that by (auto simp: integral_combine_tagged_division_topdown)
  moreover have norm (∑(x,K) ∈ T'. ?CI K h x - ?CI K f x)
      ≤ 2*ε/3
  proof -
    define T'' where T'' ≡ {(x,K) ∈ T'. ¬ (K ⊆ {x. x · i ≤ c})}
    then have T'' ⊆ T'
      by auto
    then have finite T''
      using ‹finite T'› infinite_super by blast
    have T''_tagged: T'' tagged_partial_division_of cbox a b
      using T'_tagged ‹T'' ⊆ T'› tagged_partial_division_subset by blast
    have fine'': γ0 fine T'' γ1 fine T''
      using ‹T'' ⊆ T'› fine' by (blast intro: fine_subset)+
    have (∑(x,K) ∈ T'. ?CI K h x - ?CI K f x)
       = (∑(x,K) ∈ T''. ?CI K h x - ?CI K f x)
     proof (clarify intro!: sum.mono_neutral_right [OF ‹finite T'› ‹T'' ⊆
T'›])
      fix x K
      assume (x,K) ∈ T' (x,K) ∉ T''
      then have x ∈ K x · i ≤ c {x. x · i ≤ c} ∩ K = K
        using T''_def T'_tagged tagged_partial_division_of_def by blast+
      then show ?CI K h x - ?CI K f x = 0
        using integral_restrict_Int [of _ {x. x · i ≤ c} h] by (auto simp: f)
    qed
   moreover have norm (∑(x,K) ∈ T''. ?CI K h x - ?CI K f x) ≤ 2*ε/3
    proof -
      define A where A ≡ {(x,K) ∈ T''. x · i ≤ c}
      define B where B ≡ {(x,K) ∈ T''. x · i > c}
       then have A ⊆ T'' B ⊆ T'' and disj: A ∩ B = {} and T''_eq: T''
= A ∪ B
        by (auto simp: A_def B_def)
      then have finite A finite B
        using ‹finite T''›  by (auto intro: finite_subset)
      have A_tagged: A tagged_partial_division_of cbox a b
       using T''_tagged ‹A ⊆ T''› tagged_partial_division_subset by blast
      have fineA: γ0 fine A γ1 fine A
        using ‹A ⊆ T''› fine'' by (blast intro: fine_subset)+
      have B_tagged: B tagged_partial_division_of cbox a b
       using T''_tagged ‹B ⊆ T''› tagged_partial_division_subset by blast
```

```
          have fineB: γ0 fine B γ1 fine B
            using ‹B ⊆ T''› fine'' by (blast intro: fine_subset)+
          have norm (∑(x,K) ∈ T''. ?CI K h x − ?CI K f x)
                    ≤ (∑(x,K) ∈ T''. norm (?CI K h x − ?CI K f x))
            by (simp add: norm_sum split_def)
          also have ... = (∑(x,K) ∈ A. norm (?CI K h x − ?CI K f x)) +
                          (∑(x,K) ∈ B. norm (?CI K h x − ?CI K f x))
            by (simp add: sum.union_disjoint T''_eq disj ‹finite A› ‹finite B›)
          also have ... = (∑(x,K) ∈ A. norm (integral K h − integral K f)) +
                          (∑(x,K) ∈ B. norm (?CI K h x + integral K f))
          by (auto simp: A_def B_def f norm_minus_commute intro!: sum.cong
arg_cong2 [where f= (+)])
          also have ... ≤ (∑(x,K)∈A. norm (integral K h)) +
                            (∑(x,K)∈(λ(x,K). (x,K ∩ {x. x • i ≤ c})) ‘ A. norm
(integral K h))
                        + ((∑(x,K)∈B. norm (?CI K h x)) +
                           (∑(x,K)∈B. norm (integral K h)) +
                            (∑(x,K)∈(λ(x,K). (x,K ∩ {x. c ≤ x • i})) ‘ B. norm
(integral K h)))
          proof (rule add_mono)
            show (∑(x,K)∈A. norm (integral K h − integral K f))
                 ≤ (∑(x,K)∈A. norm (integral K h)) +
                   (∑(x,K)∈(λ(x,K). (x,K ∩ {x. x • i ≤ c})) ‘ A.
                     norm (integral K h))
            proof (subst sum.reindex_nontrivial [OF ‹finite A›], clarsimp)
              fix x K L
              assume (x,K) ∈ A (x,L) ∈ A
                and int_ne0: integral (L ∩ {x. x • i ≤ c}) h ≠ 0
                and eq: K ∩ {x. x • i ≤ c} = L ∩ {x. x • i ≤ c}
              have False if K ≠ L
              proof −
                obtain u v where uv: L = cbox u v
                  using T'_tagged ‹(x, L) ∈ A› ‹A ⊆ T''› ‹T'' ⊆ T'› by (blast
dest: tagged_partial_division_ofD)
                have interior (K ∩ {x. x • i ≤ c}) = {}
                  proof (rule tagged_division_split_left_inj [OF _ ‹(x,K) ∈ A›
‹(x,L) ∈ A›])
                  show A tagged_division_of ⋃(snd ‘ A)
                    using A_tagged tagged_partial_division_of_Union_self by
auto
                  show K ∩ {x. x • i ≤ c} = L ∩ {x. x • i ≤ c}
                    using eq ‹i ∈ Basis› by auto
                qed (use that in auto)
                then show False
              using interval_split [OF ‹i ∈ Basis›] int_ne0 content_eq_0_interior
eq uv by fastforce
              qed
              then show K = L by blast
            next
```

```
                show (∑(x,K) ∈ A. norm (integral K h − integral K f))
                      ≤ (∑(x,K) ∈ A. norm (integral K h)) +
                        sum ((λ(x,K). norm (integral K h)) ∘ (λ(x,K). (x,K ∩ {x.
x · i ≤ c}))) A
                  using integral_restrict_Int [of _ {x. x · i ≤ c} h] f
                    by (auto simp: Int_commute A_def [symmetric] sum.distrib
[symmetric] intro!: sum_mono norm_triangle_ineq4)
                qed
              next
                show (∑(x,K)∈B. norm (?CI K h x + integral K f))
                  ≤ (∑(x,K)∈B. norm (?CI K h x)) + (∑(x,K)∈B. norm (integral
K h)) +
                      (∑(x,K)∈(λ(x,K). (x,K ∩ {x. c ≤ x · i})) ‘ B. norm (integral
K h))
                proof (subst sum.reindex_nontrivial [OF ‹finite B›], clarsimp)
                  fix x K L
                  assume (x,K) ∈ B (x,L) ∈ B
                    and int_ne0: integral (L ∩ {x. c ≤ x · i}) h ≠ 0
                    and eq: K ∩ {x. c ≤ x · i} = L ∩ {x. c ≤ x · i}
                  have False if K ≠ L
                  proof −
                    obtain u v where uv: L = cbox u v
                      using T'_tagged ‹(x, L) ∈ B› ‹B ⊆ T''› ‹T'' ⊆ T'› by (blast
dest: tagged_partial_division_ofD)
                    have interior (K ∩ {x. c ≤ x · i}) = {}
                    proof (rule tagged_division_split_right_inj [OF _ ‹(x,K) ∈ B›
‹(x,L) ∈ B›])
                      show B tagged_division_of ⋃(snd ‘ B)
                        using B_tagged tagged_partial_division_of_Union_self by
auto
                      show K ∩ {x. c ≤ x · i} = L ∩ {x. c ≤ x · i}
                        using eq ‹i ∈ Basis› by auto
                    qed (use that in auto)
                    then show False
                      using interval_split [OF ‹i ∈ Basis›] int_ne0
                        content_eq_0_interior eq uv by fastforce
                  qed
                  then show K = L by blast
                next
                  show (∑(x,K) ∈ B. norm (?CI K h x + integral K f))
                      ≤ (∑(x,K) ∈ B. norm (?CI K h x)) +
                        (∑(x,K) ∈ B. norm (integral K h)) + sum ((λ(x,K). norm
(integral K h)) ∘ (λ(x,K). (x,K ∩ {x. c ≤ x · i}))) B
                    proof (clarsimp simp: B_def [symmetric] sum.distrib [symmetric]
intro!: sum_mono)
                      fix x K
                      assume (x,K) ∈ B
                        have ∗: i = i1 + i2 ⟹ norm(c + i1) ≤ norm c + norm i +
norm(i2)
```

```
                for i::'b and c i1 i2
              by (metis add.commute add.left_commute add_diff_cancel_right'
dual_order.refl norm_add_rule_thm norm_triangle_ineq4)
              obtain u v where uv: K = cbox u v
                using T'_tagged ‹(x,K) ∈ B› ‹B ⊆ T''› ‹T'' ⊆ T'› by (blast
dest: tagged_partial_division_ofD)
              have huv: h integrable_on cbox u v
              proof (rule integrable_on_subcbox)
                show cbox u v ⊆ cbox a b
            using B_tagged ‹(x,K) ∈ B› uv by (blast dest: tagged_partial_division_ofD)
                show h integrable_on cbox a b
                  by (simp add: int_F ‹h ∈ F›)
              qed
             have integral K h = integral K f + integral (K ∩ {x. c ≤ x • i}) h
                  using integral_restrict_Int [of _ {x. x • i ≤ c} h] f uv ‹i ∈
Basis›
               by (simp add: Int_commute integral_split [OF huv ‹i ∈ Basis›])
            then show norm (?CI K h x + integral K f)
                    ≤ norm (?CI K h x) + norm (integral K h) + norm (integral
(K ∩ {x. c ≤ x • i}) h)
              by (rule *)
          qed
        qed
      qed
      also have ... ≤ 2*ε/3
      proof −
        have overlap: K ∩ {x. x • i = c} ≠ {} if (x,K) ∈ T'' for x K
        proof −
          obtain y y' where y: y' ∈ K c < y' • i y ∈ K y • i ≤ c
            using that T''_def T'_def ‹(x,K) ∈ T''› by fastforce
          obtain u v where uv: K = cbox u v
       using T''_tagged ‹(x,K) ∈ T''› by (blast dest: tagged_partial_division_ofD)
          then have connected K
            by (simp add: is_interval_connected)
          then have (∃ z ∈ K. z • i = c)
            using y connected_ivt_component by fastforce
          then show ?thesis
            by fastforce
        qed
       have **: ⟦x < ε/12; y < ε/12; z ≤ ε/2⟧ ⟹ x + y + z ≤ 2 * ε/3 for
x y z
          by auto
        show ?thesis
        proof (rule **)
          have cb_ab: (∑ j ∈ Basis. if j = i then c *R i else (a • j) *R j) ∈
cbox a b
            using ‹i ∈ Basis› True ‹⋀i. i ∈ Basis ⟹ a • i < b • i›
            by (force simp add: mem_box sum_if_inner [where f = λj. c])
          show (∑ (x,K) ∈ A. norm (integral K h)) < ε/12
```

using ‹$i \in Basis$› ‹$A \subseteq T''$› *overlap*
by (*force simp add*: *sum_if_inner* [**where** $f = \lambda j.\ c$]
intro!: *γ0* [*OF cb_ab* ‹$i \in Basis$› *A_tagged fineA*(*1*) ‹$h \in F$›])
let *?F* = $\lambda(x,K).\ (x,\ K \cap \{x.\ x \cdot i \le c\})$
have *1*: *?F* ‘ *A tagged_partial_division_of cbox a b*
unfolding *tagged_partial_division_of_def*
proof (*intro conjI strip*)
show $\bigwedge x\ K.\ (x,\ K) \in$ *?F* ‘ $A \Longrightarrow \exists\, a\ b.\ K = cbox\ a\ b$
using *A_tagged interval_split*(*1*) [*OF* ‹$i \in Basis$›, *of* _ _ *c*]
by (*force dest*: *tagged_partial_division_ofD*(*4*))
show $\bigwedge x\ K.\ (x,\ K) \in$ *?F* ‘ $A \Longrightarrow x \in K$
using *A_def A_tagged* **by** (*fastforce dest*: *tagged_partial_division_ofD*)
qed (*use A_tagged* **in** ‹*fastforce dest*: *tagged_partial_division_ofD*›)+
have *2*: *γ0 fine* $(\lambda(x,K).\ (x,K \cap \{x.\ x \cdot i \le c\}))$ ‘ *A*
using *fineA*(*1*) *fine_def* **by** *fastforce*
show $(\sum (x,K) \in (\lambda(x,K).\ (x,K \cap \{x.\ x \cdot i \le c\}))$ ‘ *A*. *norm* (*integral K h*)) $< \varepsilon/12$
using ‹$i \in Basis$› ‹$A \subseteq T''$› *overlap*
by (*force simp add*: *sum_if_inner* [**where** $f = \lambda j.\ c$]
intro!: *γ0* [*OF cb_ab* ‹$i \in Basis$› *1 2* ‹$h \in F$›])
have $*$: $[\![x < \varepsilon/3;\ y < \varepsilon/12;\ z < \varepsilon/12]\!] \Longrightarrow x + y + z \le \varepsilon/2$ **for** *x y z*
by *auto*
show $(\sum (x,K) \in B.\ norm\ (?CI\ K\ h\ x)) +$
$(\sum (x,K) \in B.\ norm\ (integral\ K\ h)) +$
$(\sum (x,K) \in (\lambda(x,K).\ (x,K \cap \{x.\ c \le x \cdot i\}))$ ‘ *B*. *norm* (*integral K h*))
$\le \varepsilon/2$
proof (*rule* $*$)
show $(\sum (x,K) \in B.\ norm\ (?CI\ K\ h\ x)) < \varepsilon/3$
by (*intro h_less3 B_tagged fineB that*)
show $(\sum (x,K) \in B.\ norm\ (integral\ K\ h)) < \varepsilon/12$
using ‹$i \in Basis$› ‹$B \subseteq T''$› *overlap*
by (*force simp add*: *sum_if_inner* [**where** $f = \lambda j.\ c$]
intro!: *γ0* [*OF cb_ab* ‹$i \in Basis$› *B_tagged fineB*(*1*) ‹$h \in F$›])
let *?F* = $\lambda(x,K).\ (x,\ K \cap \{x.\ c \le x \cdot i\})$
have *1*: *?F* ‘ *B tagged_partial_division_of cbox a b*
unfolding *tagged_partial_division_of_def*
proof (*intro conjI strip*)
show $\bigwedge x\ K.\ (x,\ K) \in$ *?F* ‘ $B \Longrightarrow \exists\, a\ b.\ K = cbox\ a\ b$
using *B_tagged interval_split*(*2*) [*OF* ‹$i \in Basis$›, *of* _ _ *c*]
by (*force dest*: *tagged_partial_division_ofD*(*4*))
show $\bigwedge x\ K.\ (x,\ K) \in$ *?F* ‘ $B \Longrightarrow x \in K$
using *B_def B_tagged* **by** (*fastforce dest*: *tagged_partial_division_ofD*)
qed (*use B_tagged* **in** ‹*fastforce dest*: *tagged_partial_division_ofD*›)+
have *2*: *γ0 fine* $(\lambda(x,K).\ (x,K \cap \{x.\ c \le x \cdot i\}))$ ‘ *B*
using *fineB*(*1*) *fine_def* **by** *fastforce*
show $(\sum (x,K) \in (\lambda(x,K).\ (x,K \cap \{x.\ c \le x \cdot i\}))$ ‘ *B*. *norm* (*integral K h*)) $< \varepsilon/12$

```
            using ‹i ∈ Basis› ‹A ⊆ T''› overlap
            by (force simp add: B_def sum_if_inner [where f = λj. c]
                intro!: γ0 [OF cb_ab ‹i ∈ Basis› 1 2 ‹h ∈ F›])
          qed
        qed
      qed
      finally show ?thesis .
    qed
    ultimately show ?thesis by metis
  qed
  ultimately show ?thesis
    by (simp add: sum_subtractf [symmetric] int_KK' *)
qed
  ultimately show ?thesis by metis
next
  case False
  then consider c < a · i | b · i < c
    by auto
  then show ?thesis
  proof cases
    case 1
    then have f0: f x = 0 if x ∈ cbox a b for x
      using that f ‹i ∈ Basis› mem_box(2) by force
    then have int_f0: integral (cbox a b) f = 0
      by (simp add: integral_cong)
    have f0_tag: f x = 0 if (x,K) ∈ T for x K
      using T f0 that by (meson tag_in_interval)
    then have (∑(x,K) ∈ T. content K *R f x) = 0
      by (metis (mono_tags, lifting) real_vector.scale_eq_0_iff split_conv
sum.neutral surj_pair)
    then show ?thesis
      using ‹0 < ε› by (simp add: int_f0)
  next
    case 2
    then have fh: f x = h x if x ∈ cbox a b for x
      using that f ‹i ∈ Basis› mem_box(2) by force
    then have int_f: integral (cbox a b) f = integral (cbox a b) h
      using integral_cong by blast
    have fh_tag: f x = h x if (x,K) ∈ T for x K
      using T fh that by (meson tag_in_interval)
   then have fh: (∑(x,K) ∈ T. content K *R f x) = (∑(x,K) ∈ T. content
K *R h x)
      by (metis (mono_tags, lifting) split_cong sum.cong)
    show ?thesis
      unfolding fh int_f
    proof (rule less_trans [OF γ1])
      show γ1 fine T
        by (meson fine fine_Int)
      show ε / (7 * Suc DIM('b)) < ε
```

```
          using ‹0 < ε› by (force simp: divide_simps)+
        qed (use that in auto)
      qed
    qed
    have  gauge (λx. γ0 x ∩ γ1 x)
      by (simp add: ‹gauge γ0› ‹gauge γ1› gauge_Int)
    then show ?thesis
      by (auto intro: ∗)
  qed
 qed
qed


corollary equiintegrable_halfspace_restrictions_ge:
 fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
 assumes F: F equiintegrable_on cbox a b and f: f ∈ F
   and norm_f: ⋀h x. ⟦h ∈ F; x ∈ cbox a b⟧ ⟹ norm(h x) ≤ norm(f x)
 shows (⋃i ∈ Basis. ⋃c. ⋃h ∈ F. {(λx. if x • i ≥ c then h x else 0)})
       equiintegrable_on cbox a b
proof −
 have ∗: (⋃i∈Basis. ⋃c. ⋃h∈(λf. f ∘ uminus) ` F. {λx. if x • i ≤ c then h x
else 0})
         equiintegrable_on  cbox (− b) (− a)
 proof (rule equiintegrable_halfspace_restrictions_le)
   show (λf. f ∘ uminus) ` F equiintegrable_on cbox (− b) (− a)
     using F equiintegrable_reflect by blast
   show f ∘ uminus ∈ (λf. f ∘ uminus) ` F
     using f by auto
   show ⋀h x. ⟦h ∈ (λf. f ∘ uminus) ` F; x ∈ cbox (− b) (− a)⟧ ⟹ norm (h x)
≤ norm ((f ∘ uminus) x)
     using f unfolding comp_def image_iff
       by (metis (no_types, lifting) equation_minus_iff imageE norm_f umi-
nus_interval_vector)
 qed
 have eq: (λf. f ∘ uminus) `
         (⋃i∈Basis. ⋃c. ⋃h∈F. {λx. if x • i ≤ c then (h ∘ uminus) x else 0}) =
          (⋃i∈Basis. ⋃c. ⋃h∈F. {λx. if c ≤ x • i then h x else 0})   (is ?lhs =
?rhs)
 proof
   show ?lhs ⊆ ?rhs
     using minus_le_iff by fastforce
   show ?rhs ⊆ ?lhs
   apply clarsimp
   apply (rule_tac x=λx. if c ≤ (−x) • i then h(−x) else 0 in image_eqI)
     using le_minus_iff by fastforce+
 qed
 show ?thesis
   using equiintegrable_reflect [OF ∗] by (auto simp: eq)
qed
```

corollary *equiintegrable_halfspace_restrictions_lt*:
fixes f :: *'a::euclidean_space* ⇒ *'b::euclidean_space*
assumes F: F *equiintegrable_on cbox a b* **and** f: $f \in F$
and *norm_f*: $\bigwedge h\ x.\ [\![h \in F;\ x \in cbox\ a\ b]\!] \Longrightarrow norm(h\ x) \le norm(f\ x)$
shows $(\bigcup i \in Basis.\ \bigcup c.\ \bigcup h \in F.\ \{(\lambda x.\ if\ x \cdot i < c\ then\ h\ x\ else\ 0)\})$ *equiintegrable_on cbox a b*
(**is** *?G equiintegrable_on cbox a b*)
proof −
have ∗: $(\bigcup i{\in}Basis.\ \bigcup c.\ \bigcup h{\in}F.\ \{\lambda x.\ if\ c \le x \cdot i\ then\ h\ x\ else\ 0\})$ *equiintegrable_on cbox a b*
using *equiintegrable_halfspace_restrictions_ge* [*OF F f*] *norm_f* **by** *auto*
have $(\lambda x.\ if\ x \cdot i < c\ then\ h\ x\ else\ 0) = (\lambda x.\ h\ x - (if\ c \le x \cdot i\ then\ h\ x\ else\ 0))$
if $i \in Basis$ $h \in F$ **for** i c h
using *that* **by** *force*
then show *?thesis*
by (*blast intro*: *equiintegrable_on_subset* [*OF equiintegrable_diff* [*OF F* ∗]])
qed

corollary *equiintegrable_halfspace_restrictions_gt*:
fixes f :: *'a::euclidean_space* ⇒ *'b::euclidean_space*
assumes F: F *equiintegrable_on cbox a b* **and** f: $f \in F$
and *norm_f*: $\bigwedge h\ x.\ [\![h \in F;\ x \in cbox\ a\ b]\!] \Longrightarrow norm(h\ x) \le norm(f\ x)$
shows $(\bigcup i \in Basis.\ \bigcup c.\ \bigcup h \in F.\ \{(\lambda x.\ if\ x \cdot i > c\ then\ h\ x\ else\ 0)\})$ *equiintegrable_on cbox a b*
(**is** *?G equiintegrable_on cbox a b*)
proof −
have ∗: $(\bigcup i{\in}Basis.\ \bigcup c.\ \bigcup h{\in}F.\ \{\lambda x.\ if\ c \ge x \cdot i\ then\ h\ x\ else\ 0\})$ *equiintegrable_on cbox a b*
using *equiintegrable_halfspace_restrictions_le* [*OF F f*] *norm_f* **by** *auto*
have $(\lambda x.\ if\ x \cdot i > c\ then\ h\ x\ else\ 0) = (\lambda x.\ h\ x - (if\ c \ge x \cdot i\ then\ h\ x\ else\ 0))$
if $i \in Basis$ $h \in F$ **for** i c h
using *that* **by** *force*
then show *?thesis*
by (*blast intro*: *equiintegrable_on_subset* [*OF equiintegrable_diff* [*OF F* ∗]])
qed

proposition *equiintegrable_closed_interval_restrictions*:
fixes f :: *'a::euclidean_space* ⇒ *'b::euclidean_space*
assumes f: f *integrable_on cbox a b*
shows $(\bigcup c\ d.\ \{(\lambda x.\ if\ x \in cbox\ c\ d\ then\ f\ x\ else\ 0)\})$ *equiintegrable_on cbox a b*
proof −
let $?g = \lambda B\ c\ d\ x.\ if\ \forall i{\in}B.\ c \cdot i \le x \cdot i \wedge x \cdot i \le d \cdot i\ then\ f\ x\ else\ 0$
have ∗: *insert f* $(\bigcup c\ d.\ \{?g\ B\ c\ d\})$ *equiintegrable_on cbox a b* **if** $B \subseteq Basis$ **for** B
proof −
have *finite B*
using *finite_Basis finite_subset* ‹$B \subseteq Basis$› **by** *blast*
then show *?thesis* **using** ‹$B \subseteq Basis$›

```
  proof (induction B)
    case empty
    with f show ?case by auto
  next
    case (insert i B)
    then have i ∈ Basis B ⊆ Basis
      by auto
    have ∗: norm (h x) ≤ norm (f x)
      if h ∈ insert f (⋃ c d. {?g B c d}) x ∈ cbox a b for h x
      using that by auto
    define F where F ≡ (⋃ i∈Basis.
          ⋃ξ. ⋃ h∈insert f (⋃ i∈Basis. ⋃ψ. ⋃ h∈insert f (⋃ c d. {?g B c d}).
{λx. if x · i ≤ ψ then h x else 0}).
           {λx. if ξ ≤ x · i then h x else 0})
    show ?case
    proof (rule equiintegrable_on_subset)
      have F equiintegrable_on cbox a b
        unfolding F_def
      proof (rule equiintegrable_halfspace_restrictions_ge)
        show insert f (⋃ i∈Basis. ⋃ξ. ⋃ h∈insert f (⋃ c d. {?g B c d}).
            {λx. if x · i ≤ ξ then h x else 0}) equiintegrable_on cbox a b
       by (intro ∗ f equiintegrable_on_insert equiintegrable_halfspace_restrictions_le
[OF insert.IH insertI1] ‹B ⊆ Basis›)
        show norm(h x) ≤ norm(f x)
          if h ∈ insert f (⋃ i∈Basis. ⋃ξ. ⋃ h∈insert f (⋃ c d. {?g B c d}). {λx.
if x · i ≤ ξ then h x else 0})
           x ∈ cbox a b for h x
          using that by auto
      qed auto
      then show insert f F
         equiintegrable_on cbox a b
        by (blast intro: f equiintegrable_on_insert)
       show insert f (⋃ c d. {λx. if ∀ j∈insert i B. c · j ≤ x · j ∧ x · j ≤ d · j
then f x else 0})
          ⊆ insert f F
        using ‹i ∈ Basis›
        apply clarify
        apply (simp add: F_def)
        apply (drule_tac x=i in bspec, assumption)
        apply (drule_tac x=c · i in spec, clarify)
        apply (drule_tac x=i in bspec, assumption)
        apply (drule_tac x=d · i in spec)
        apply (clarsimp simp: fun_eq_iff)
        apply (drule_tac x=c in spec)
        apply (drule_tac x=d in spec)
        apply (simp split: if_split_asm)
        done
    qed
  qed
```

qed
show *?thesis*
by (*rule equiintegrable_on_subset* [*OF* ∗ [*OF subset_refl*]]) (*auto simp*: *mem_box*)
qed

6.24.3 Continuity of the indefinite integral

proposition *indefinite_integral_continuous*:
fixes $f :: 'a :: euclidean\_space \Rightarrow 'b :: euclidean\_space$
assumes *int_f*: *f integrable_on cbox a b*
and *c*: $c \in cbox\ a\ b$ **and** *d*: $d \in cbox\ a\ b\ 0 < \varepsilon$
obtains δ **where** $0 < \delta$
$\bigwedge c'\ d'.\ [\![c' \in cbox\ a\ b;\ d' \in cbox\ a\ b;\ norm(c' - c) \le \delta;\ norm(d' - d) \le \delta]\!]$
$\Longrightarrow norm(integral(cbox\ c'\ d')\ f - integral(cbox\ c\ d)\ f) < \varepsilon$
proof −
{ **assume** $\exists c'\ d'.\ c' \in cbox\ a\ b \wedge d' \in cbox\ a\ b \wedge norm(c' - c) \le \delta \wedge norm(d' - d) \le \delta \wedge$
$norm(integral(cbox\ c'\ d')\ f - integral(cbox\ c\ d)\ f) \ge \varepsilon$
(**is** $\exists c'\ d'.\ ?\Phi\ c'\ d'\ \delta$) **if** $0 < \delta$ **for** δ
then have $\exists c'\ d'.\ ?\Phi\ c'\ d'\ (1\ /\ Suc\ n)$ **for** *n*
by *simp*
then obtain *u v* **where** $\bigwedge n.\ ?\Phi\ (u\ n)\ (v\ n)\ (1\ /\ Suc\ n)$
by *metis*
then have *u*: $u\ n \in cbox\ a\ b$ **and** *norm_u*: $norm(u\ n - c) \le 1\ /\ Suc\ n$
and *v*: $v\ n \in cbox\ a\ b$ **and** *norm_v*: $norm(v\ n - d) \le 1\ /\ Suc\ n$
and ε: $\varepsilon \le norm\ (integral\ (cbox\ (u\ n)\ (v\ n))\ f - integral\ (cbox\ c\ d)\ f)$ **for** *n*
by *blast*+
then have *False*
proof −
have *uvn*: $cbox\ (u\ n)\ (v\ n) \subseteq cbox\ a\ b$ **for** *n*
by (*meson u v mem_box*(2) *subset_box*(1))
define *S* **where** $S \equiv \bigcup i \in Basis.\ \{x.\ x \cdot i = c \cdot i\} \cup \{x.\ x \cdot i = d \cdot i\}$
have *negligible S*
unfolding *S_def* **by** *force*
then have *int_f'*: $(\lambda x.\ if\ x \in S\ then\ 0\ else\ f\ x)\ integrable\_on\ cbox\ a\ b$
by (*force intro*: *integrable_spike assms*)
have *get_n*: $\exists n.\ \forall m \ge n.\ x \in cbox\ (u\ m)\ (v\ m) \longleftrightarrow x \in cbox\ c\ d$ **if** *x*: $x \notin S$ **for** *x*
proof −
define ε **where** $\varepsilon \equiv Min\ ((\lambda i.\ min\ |x \cdot i - c \cdot i|\ |x \cdot i - d \cdot i|)\ `\ Basis)$
have $\varepsilon > 0$
using ‹$x \notin S$› **by** (*auto simp*: *S_def* ε*_def*)
then obtain *n* **where** $n \ne 0$ **and** *n*: $1\ /\ (real\ n) < \varepsilon$
by (*metis inverse_eq_divide real_arch_inverse*)
have *emin*: $\varepsilon \le min\ |x \cdot i - c \cdot i|\ |x \cdot i - d \cdot i|$ **if** $i \in Basis$ **for** *i*
unfolding ε*_def*
by (*meson Min.coboundedI euclidean_space_class.finite_Basis finite_imageI*

```
image_iff that)
      have 1 / real (Suc n) < ε
        using n ‹n ≠ 0› ‹ε > 0› by (simp add: field_simps)
      have x ∈ cbox (u m) (v m) ⟷ x ∈ cbox c d if m ≥ n for m
      proof −
        have ∗: ⟦|u − c| ≤ n; |v − d| ≤ n; N < |x − c|; N < |x − d|; n ≤ N⟧
              ⟹ u ≤ x ∧ x ≤ v ⟷ c ≤ x ∧ x ≤ d for N n u v c d and x::real
          by linarith
        have (u m · i ≤ x · i ∧ x · i ≤ v m · i) = (c · i ≤ x · i ∧ x · i ≤ d · i)
          if i ∈ Basis for i
        proof (rule ∗)
          show |u m · i − c · i| ≤ 1 / Suc m
            using norm_u [of m]
              by (metis (full_types) order_trans Basis_le_norm inner_commute
inner_diff_right that)
          show |v m · i − d · i| ≤ 1 / real (Suc m)
            using norm_v [of m]
              by (metis (full_types) order_trans Basis_le_norm inner_commute
inner_diff_right that)
          show 1/n < |x · i − c · i| 1/n < |x · i − d · i|
            using n ‹n ≠ 0› emin [OF ‹i ∈ Basis›]
            by (simp_all add: inverse_eq_divide)
          show 1 / real (Suc m) ≤ 1 / real n
            using ‹n ≠ 0› ‹m ≥ n› by (simp add: field_split_simps)
        qed
        then show ?thesis by (simp add: mem_box)
      qed
      then show ?thesis by blast
    qed
    have 1: range (λn x. if x ∈ cbox (u n) (v n) then if x ∈ S then 0 else f x else
0) equiintegrable_on cbox a b
    by (blast intro: equiintegrable_on_subset [OF equiintegrable_closed_interval_restrictions
[OF int_f']])
    have 2: (λn. if x ∈ cbox (u n) (v n) then if x ∈ S then 0 else f x else 0)
          ⟶ (if x ∈ cbox c d then if x ∈ S then 0 else f x else 0) for x
    by (fastforce simp: dest: get_n intro: tendsto_eventually eventually_sequentiallyI)
    have [simp]: cbox c d ∩ cbox a b = cbox c d
      using c d by (force simp: mem_box)
    have [simp]: cbox (u n) (v n) ∩ cbox a b = cbox (u n) (v n) for n
      using u v by (fastforce simp: mem_box intro: order.trans)
    have ⋀y A. y ∈ A − S ⟹ f y = (λx. if x ∈ S then 0 else f x) y
      by simp
    then have ⋀A. integral A (λx. if x ∈ S then 0 else f (x)) = integral A (λx.
f (x))
      by (blast intro: integral_spike [OF ‹negligible S›])
    moreover
    obtain N where dist (integral (cbox (u N) (v N)) (λx. if x ∈ S then 0 else f
x))
                   (integral (cbox c d) (λx. if x ∈ S then 0 else f x)) < ε
```

```
    using equiintegrable_limit [OF 1 2] ‹0 < ε› by (force simp: integral_restrict_Int
lim_sequentially)
    ultimately have dist (integral (cbox (u N) (v N)) f) (integral (cbox c d) f)
< ε
      by simp
    then show False
      by (metis dist_norm not_le ε)
   qed
  }
  then show ?thesis
   by (meson not_le that)
qed

corollary indefinite_integral_uniformly_continuous:
  fixes f :: 'a :: euclidean_space ⇒ 'b :: euclidean_space
  assumes f integrable_on cbox a b
  shows uniformly_continuous_on (cbox (Pair a a) (Pair b b)) (λy. integral (cbox
(fst y) (snd y)) f)
proof −
  show ?thesis
  proof (rule compact_uniformly_continuous, clarsimp simp add: continuous_on_iff)
    fix c d and ε::real
    assume c: c ∈ cbox a b and d: d ∈ cbox a b and 0 < ε
    obtain δ where 0 < δ and δ:
            ⋀c' d'. ⟦c' ∈ cbox a b; d' ∈ cbox a b; norm(c' − c) ≤ δ; norm(d' − d)
≤ δ⟧
                                ⟹ norm(integral(cbox c' d') f −
                                       integral(cbox c d) f) < ε
      using indefinite_integral_continuous ‹0 < ε› assms c d by blast
    show ∃δ > 0. ∀x' ∈ cbox (a, a) (b, b).
                 dist x' (c, d) < δ ⟶
                 dist (integral (cbox (fst x') (snd x')) f)
                      (integral (cbox c d) f)
                 < ε
      using ‹0 < δ›
     by (force simp: dist_norm intro: δ order_trans [OF norm_fst_le] order_trans
[OF norm_snd_le] less_imp_le)
  qed auto
qed


corollary bounded_integrals_over_subintervals:
  fixes f :: 'a :: euclidean_space ⇒ 'b :: euclidean_space
  assumes f integrable_on cbox a b
  shows bounded {integral (cbox c d) f |c d. cbox c d ⊆ cbox a b}
proof −
  have bounded ((λy. integral (cbox (fst y) (snd y)) f) ‘ cbox (a, a) (b, b))
       (is bounded ?I)
    by (blast intro: bounded_cbox bounded_uniformly_continuous_image indefi-
```

```
nite_integral_uniformly_continuous [OF assms])
  then obtain B where B > 0 and B: ⋀x. x ∈ ?I ⟹ norm x ≤ B
    by (auto simp: bounded_pos)
  have norm x ≤ B if x = integral (cbox c d) f cbox c d ⊆ cbox a b for x c d
  proof (cases cbox c d = {})
    case True
    with ‹0 < B› that show ?thesis by auto
  next
    case False
    then have ∃ x ∈ cbox (a,a) (b,b). integral (cbox c d) f = integral (cbox (fst x)
(snd x)) f
     using that by (metis cbox_Pair_iff interval_subset_is_interval is_interval_cbox
prod.sel)
    then show ?thesis
      using B that(1) by blast
  qed
  then show ?thesis
    by (blast intro: boundedI)
qed
```

An existence theorem for "improper" integrals. Hake's theorem implies that if the integrals over subintervals have a limit, the integral exists. We only need to assume that the integrals are bounded, and we get absolute integrability, but we also need a (rather weak) bound assumption on the function.

```
theorem absolutely_integrable_improper:
  fixes f :: 'M::euclidean_space ⇒ 'N::euclidean_space
  assumes int_f: ⋀c d. cbox c d ⊆ box a b ⟹ f integrable_on cbox c d
      and bo: bounded {integral (cbox c d) f |c d. cbox c d ⊆ box a b}
      and absi: ⋀i. i ∈ Basis
          ⟹ ∃ g. g absolutely_integrable_on cbox a b ∧
                  ((∀ x ∈ cbox a b. f x • i ≤ g x) ∨ (∀ x ∈ cbox a b. f x • i ≥ g x))
      shows f absolutely_integrable_on cbox a b
proof (cases content(cbox a b) = 0)
  case True
  then show ?thesis
    by auto
next
  case False
  then have pos: content(cbox a b) > 0
    using zero_less_measure_iff by blast
  show ?thesis
    unfolding absolutely_integrable_componentwise_iff [where f = f]
  proof
    fix j::'N
    assume j ∈ Basis
    then obtain g where absint_g: g absolutely_integrable_on cbox a b
                  and g: (∀ x ∈ cbox a b. f x • j ≤ g x) ∨ (∀ x ∈ cbox a b. f x • j ≥
g x)
      using absi by blast
```

```
  have int_gab: g integrable_on cbox a b
    using absint_g set_lebesgue_integral_eq_integral(1) by blast
  define α where α ≡ λk. a + (b − a) /R real k
  define β where β ≡ λk. b − (b − a) /R real k
  define I where I ≡ λk. cbox (α k) (β k)
  have ISuc_box: I (Suc n) ⊆ box a b for n
    using pos unfolding I_def
      by (intro subset_box_imp) (auto simp: α_def β_def content_pos_lt_eq
algebra_simps)
  have ISucSuc: I (Suc n) ⊆ I (Suc (Suc n)) for n
  proof −
    have ⋀i. i ∈ Basis
          ⟹ a · i / Suc n + b · i / (real n + 2) ≤ b · i / Suc n + a · i /
(real n + 2)
      using pos
     by (simp add: content_pos_lt_eq divide_simps) (auto simp: algebra_simps)
    then show ?thesis
      unfolding I_def
       by (intro subset_box_imp) (auto simp: algebra_simps inverse_eq_divide
α_def β_def)
  qed
  have getN: ∃ N::nat. ∀ k. k ≥ N ⟶ x ∈ I k
    if x: x ∈ box a b for x
  proof −
    define Δ where Δ ≡ (⋃ i ∈ Basis. {((x − a) · i) / ((b − a) · i), (b − x) ·
i / ((b − a) · i)})
    obtain N where N: real N > 1 / Inf Δ
      using reals_Archimedean2 by blast
    moreover have Δ: Inf Δ > 0
    using that by (auto simp: Δ_def finite_less_Inf_iff mem_box algebra_simps
divide_simps)
    ultimately have N > 0
      using of_nat_0_less_iff by fastforce
    show ?thesis
    proof (intro exI impI allI)
      fix k assume N ≤ k
      with ‹0 < N› have k > 0
        by linarith
      have xa_gt: (x − a) · i > ((b − a) · i) / (real k) if i ∈ Basis for i
      proof −
        have ∗: Inf Δ ≤ ((x − a) · i) / ((b − a) · i)
          unfolding Δ_def using that by (force intro: cInf_le_finite)
        have 1 / Inf Δ ≥ ((b − a) · i) / ((x − a) · i)
          using le_imp_inverse_le [OF ∗ Δ]
          by (simp add: field_simps)
        with N have k > ((b − a) · i) / ((x − a) · i)
          using ‹N ≤ k› by linarith
        with x that show ?thesis
          by (auto simp: mem_box algebra_simps field_split_simps)
```

```
      qed
      have bx_gt: (b − x) · i > ((b − a) · i) / k if i ∈ Basis for i
      proof −
        have ∗: Inf Δ ≤ ((b − x) · i) / ((b − a) · i)
          using that unfolding Δ_def by (force intro: cInf_le_finite)
        have 1 / Inf Δ ≥ ((b − a) · i) / ((b − x) · i)
          using le_imp_inverse_le [OF ∗ Δ]
          by (simp add: field_simps)
        with N have k > ((b − a) · i) / ((b − x) · i)
          using ‹N ≤ k› by linarith
        with x that show ?thesis
          by (auto simp: mem_box algebra_simps field_split_simps)
      qed
      show x ∈ I k
        using that Δ ‹k > 0› unfolding I_def
       by (auto simp: α_def β_def mem_box algebra_simps divide_inverse dest:
xa_gt bx_gt)
    qed
  qed
  obtain Bf where Bf: ⋀c d. cbox c d ⊆ box a b ⟹ norm (integral (cbox c d)
f) ≤ Bf
    using bo unfolding bounded_iff by blast
  obtain Bg where Bg:⋀c d. cbox c d ⊆ cbox a b ⟹ |integral (cbox c d) g| ≤
Bg
    using bounded_integrals_over_subintervals [OF int_gab] unfolding bounded_iff
real_norm_def by blast
  show (λx. f x · j) absolutely_integrable_on cbox a b
    using g
  proof     — A lot of duplication in the two proofs
    assume fg [rule_format]: ∀ x∈cbox a b. f x · j ≤ g x
    have (λx. (f x · j)) = (λx. g x − (g x − (f x · j)))
      by simp
    moreover have (λx. g x − (g x − (f x · j))) integrable_on cbox a b
    proof (rule Henstock_Kurzweil_Integration.integrable_diff [OF int_gab])
      define φ where φ ≡ λk x. if x ∈ I (Suc k) then g x − f x · j else 0
      have (λx. g x − f x · j) integrable_on box a b
      proof (rule monotone_convergence_increasing [of φ, THEN conjunct1])
        have ∗: I (Suc k) ∩ box a b = I (Suc k) for k
          using box_subset_cbox ISuc_box by fastforce
        show φ k integrable_on box a b for k
        proof −
          have I (Suc k) ⊆ cbox a b
            using ∗ box_subset_cbox by blast
          moreover have (λm. f m · j) integrable_on I (Suc k)
            by (metis ISuc_box I_def int_f integrable_component)
          ultimately have (λm. g m − f m · j) integrable_on I (Suc k)
           by (metis Henstock_Kurzweil_Integration.integrable_diff I_def int_gab
integrable_on_subcbox)
          then show ?thesis
```

```
        by (simp add: * φ_def integrable_restrict_Int)
      qed
      show φ k x ≤ φ (Suc k) x if x ∈ box a b for k x
        using ISucSuc box_subset_cbox that by (force simp: φ_def intro!: fg)
      show (λk. φ k x) ⟶ g x − f x ⋅ j if x: x ∈ box a b for x
      proof (rule tendsto_eventually)
        obtain N::nat where N: ⋀k. k ≥ N ⟹ x ∈ I k
          using getN [OF x] by blast
        show ∀F k in sequentially. φ k x = g x − f x ⋅ j
        proof
          fix k::nat assume N ≤ k
          have x ∈ I (Suc k)
            by (metis ‹N ≤ k› le_Suc_eq N)
          then show φ k x = g x − f x ⋅ j
            by (simp add: φ_def)
        qed
      qed
      have |integral (box a b) (λx. if x ∈ I (Suc k) then g x − f x ⋅ j else 0)| ≤
Bg + Bf for k
      proof −
        have ABK_def [simp]: I (Suc k) ∩ box a b = I (Suc k)
          using ISuc_box by (simp add: Int_absorb2)
        have int_fI: f integrable_on I (Suc k)
          using ISuc_box I_def int_f by auto
        moreover
        have |integral (I (Suc k)) (λx. f x ⋅ j)| ≤ norm (integral (I (Suc k)) f)
          by (simp add: Basis_le_norm int_fI ‹j ∈ Basis›)
        with ISuc_box ABK_def have |integral (I (Suc k)) (λx. f x ⋅ j)| ≤ Bf
            by (metis Bf I_def ‹j ∈ Basis› int_fI integral_component_eq
norm_bound_Basis_le)
        ultimately
        have |integral (I (Suc k)) g − integral (I (Suc k)) (λx. f x ⋅ j)| ≤ Bg
+ Bf
          using * box_subset_cbox unfolding I_def
          by (blast intro: Bg add_mono order_trans [OF abs_triangle_ineq4])
        moreover have g integrable_on I (Suc k)
            by (metis ISuc_box I_def int_gab integrable_on_open_interval
integrable_on_subcbox)
        moreover have (λx. f x ⋅ j) integrable_on I (Suc k)
          using int_fI by (simp add: integrable_component)
        ultimately show ?thesis
          by (simp add: integral_restrict_Int integral_diff)
      qed
      then show bounded (range (λk. integral (box a b) (φ k)))
        by (auto simp add: bounded_iff φ_def)
    qed
    then show (λx. g x − f x ⋅ j) integrable_on cbox a b
      by (simp add: integrable_on_open_interval)
  qed
```

ultimately have $(\lambda x.\ f\ x \cdot j)$ *integrable_on cbox a b*
by *auto*
then show *?thesis*
using *absolutely_integrable_component_ubound* [*OF _ absint_g*] *fg* **by** *force*
next
assume *gf* [*rule_format*]: $\forall x{\in}cbox\ a\ b.\ g\ x \le f\ x \cdot j$
have $(\lambda x.\ (f\ x \cdot j)) = (\lambda x.\ ((f\ x \cdot j) - g\ x) + g\ x)$
by *simp*
moreover have $(\lambda x.\ (f\ x \cdot j - g\ x) + g\ x)$ *integrable_on cbox a b*
proof (*rule Henstock_Kurzweil_Integration.integrable_add* [*OF _ int_gab*])
let $?\varphi = \lambda k\ x.\ if\ x \in I(Suc\ k)\ then\ f\ x \cdot j - g\ x\ else\ 0$
have $(\lambda x.\ f\ x \cdot j - g\ x)$ *integrable_on box a b*
proof (*rule monotone_convergence_increasing* [*of ?φ, THEN conjunct1*])
have $*$: $I\ (Suc\ k) \cap box\ a\ b = I\ (Suc\ k)$ **for** k
using *box_subset_cbox ISuc_box* **by** *fastforce*
show $?\varphi\ k$ *integrable_on box a b* **for** k
proof (*simp add*: *integrable_restrict_Int integral_restrict_Int* $*$)
show $(\lambda x.\ f\ x \cdot j - g\ x)$ *integrable_on* $I\ (Suc\ k)$
by (*metis ISuc_box Henstock_Kurzweil_Integration.integrable_diff I_def int_f int_gab integrable_component integrable_on_open_interval integrable_on_subcbox*)
qed
show $?\varphi\ k\ x \le ?\varphi\ (Suc\ k)\ x$ **if** $x \in box\ a\ b$ **for** $k\ x$
using *ISucSuc box_subset_cbox that* **by** (*force simp*: *I_def intro*!: *gf*)
show $(\lambda k.\ ?\varphi\ k\ x) \longrightarrow f\ x \cdot j - g\ x$ **if** x: $x \in box\ a\ b$ **for** x
proof (*rule tendsto_eventually*)
obtain N::*nat* **where** N: $\bigwedge k.\ k \ge N \Longrightarrow x \in I\ k$
using *getN* [*OF x*] **by** *blast*
then show $\forall_F\ k$ *in sequentially.* $?\varphi\ k\ x = f\ x \cdot j - g\ x$
by (*metis* (*no_types*, *lifting*) *eventually_at_top_linorderI le_Suc_eq*)
qed
have $|integral\ (box\ a\ b)\ (\lambda x.\ if\ x \in I\ (Suc\ k)\ then\ f\ x \cdot j - g\ x\ else\ 0)| \le Bf + Bg$ **for** k
proof −
define *ABK* **where** $ABK \equiv cbox\ (a + (b - a)\ /_R\ (1 + real\ k))\ (b - (b - a)\ /_R\ (1 + real\ k))$
have *ABK_eq* [*simp*]: $ABK \cap box\ a\ b = ABK$
using $*$ *I_def* α*_def* β*_def ABK_def* **by** *auto*
have *int_fI*: *f integrable_on ABK*
unfolding *ABK_def*
using *ISuc_box I_def* α*_def* β*_def int_f* **by** *force*
then have $(\lambda x.\ f\ x \cdot j)$ *integrable_on ABK*
by (*simp add*: *integrable_component*)
moreover have *g integrable_on ABK*
by (*metis ABK_def ABK_eq IntE box_subset_cbox int_gab integrable_on_subcbox subset_eq*)
moreover
have $|integral\ ABK\ (\lambda x.\ f\ x \cdot j)| \le norm\ (integral\ ABK\ f)$
by (*simp add*: *Basis_le_norm int_fI* ‹$j \in Basis$›)
then have $|integral\ ABK\ (\lambda x.\ f\ x \cdot j)| \le Bf$

```
            by (metis ABK_eq  ABK_def Bf IntE dual_order.trans subset_eq)
          ultimately show ?thesis
            using ∗ box_subset_cbox
             apply (simp add: integral_restrict_Int integral_diff ABK_def I_def
α_def β_def)
             by (blast intro: Bg add_mono order_trans [OF abs_triangle_ineq4])
        qed
        then show bounded (range (λk. integral (box a b) (?φ k)))
          by (auto simp add: bounded_iff)
      qed
      then show (λx. f x · j − g x) integrable_on cbox a b
        by (simp add: integrable_on_open_interval)
    qed
    ultimately have (λx. f x · j) integrable_on cbox a b
      by auto
    then show ?thesis
      using absint_g absolutely_integrable_absolutely_integrable_lbound gf by
blast
  qed
 qed
qed
```

6.24.4 Second mean value theorem and corollaries

```
lemma level_approx:
 fixes f :: real ⇒ real and n::nat
 assumes f: ⋀x. x ∈ S ⟹ 0 ≤ f x ∧ f x ≤ 1 and x ∈ S n ≠ 0
 shows |f x − (∑ k = Suc 0..n. if k / n ≤ f x then inverse n else 0)| < inverse n
       (is ?lhs < _)
proof −
 have n ∗ f x ≥ 0
   using assms by auto
 then obtain m::nat where m: floor(n ∗ f x) = int m
   using nonneg_int_cases zero_le_floor by blast
 then have kn: real k / real n ≤ f x ⟷ k ≤ m for k
   using ‹n ≠ 0› by (simp add: field_split_simps) linarith
 then have Suc n / real n ≤ f x ⟷ Suc n ≤ m
   by blast
 have real n ∗ f x ≤ real n
   by (simp add: ‹x ∈ S› f mult_left_le)
 then have m ≤ n
   using m by linarith
 have ?lhs = |f x − (∑ k ∈ {Suc 0..n} ∩ {..m}. inverse n)|
   by (subst sum.inter_restrict) (auto simp: kn)
 also have ... < inverse n
   using ‹m ≤ n› ‹n ≠ 0› m
   by (simp add: min_absorb2 field_split_simps) linarith
 finally show ?thesis .
qed
```

lemma *SMVT_lemma2*:
fixes f :: *real* $\Rightarrow$ *real*
assumes f: f *integrable_on* $\{a..b\}$
and g: $\bigwedge x\ y.\ x \leq y \Longrightarrow g\ x \leq g\ y$
shows $(\bigcup y{::}real.\ \{\lambda x.\ if\ g\ x \geq y\ then\ f\ x\ else\ 0\})$ *equiintegrable_on* $\{a..b\}$
proof −
have *ffab*: $\{f\}$ *equiintegrable_on* $\{a..b\}$
by (*metis equiintegrable_on_sing f interval_cbox*)
then have *ff*: $\{f\}$ *equiintegrable_on* (*cbox a b*)
by *simp*
have *ge*: $(\bigcup c.\ \{\lambda x.\ if\ x \geq c\ then\ f\ x\ else\ 0\})$ *equiintegrable_on* $\{a..b\}$
using *equiintegrable_halfspace_restrictions_ge* [*OF ff*] **by** *auto*
have *gt*: $(\bigcup c.\ \{\lambda x.\ if\ x > c\ then\ f\ x\ else\ 0\})$ *equiintegrable_on* $\{a..b\}$
using *equiintegrable_halfspace_restrictions_gt* [*OF ff*] **by** *auto*
have *0*: $\{(\lambda x.\ 0)\}$ *equiintegrable_on* $\{a..b\}$
by (*metis box_real*(*2*) *equiintegrable_on_sing integrable_0*)
have †: $(\lambda x.\ if\ g\ x \geq y\ then\ f\ x\ else\ 0) \in \{(\lambda x.\ 0), f\} \cup (\bigcup z.\ \{\lambda x.\ if\ z < x\ then\ f\ x\ else\ 0\}) \cup (\bigcup z.\ \{\lambda x.\ if\ z \leq x\ then\ f\ x\ else\ 0\})$
for y
proof (*cases* $(\forall x.\ g\ x \geq y) \vee (\forall x.\ \neg\ (g\ x \geq y))$)
let *?μ* = *Inf* $\{x.\ g\ x \geq y\}$
case *False*
have *lower*: *?μ* $\leq x$ **if** $g\ x \geq y$ **for** x
proof (*rule cInf_lower*)
show $x \in \{x.\ y \leq g\ x\}$
using *False* **by** (*auto simp*: *that*)
show *bdd_below* $\{x.\ y \leq g\ x\}$
by (*metis False bdd_belowI dual_order.trans g linear mem_Collect_eq*)
qed
have *greatest*: *?μ* $\geq z$ **if** $(\bigwedge x.\ g\ x \geq y \Longrightarrow z \leq x)$ **for** z
by (*metis False cInf_greatest empty_iff mem_Collect_eq that*)
show *?thesis*
proof (*cases g ?μ* $\geq y$)
case *True*
then obtain ζ **where** ζ: $\bigwedge x.\ g\ x \geq y \longleftrightarrow x \geq \zeta$
by (*metis g lower order.trans*) — in fact y is *Inf* $\{x.\ y \leq g\ x\}$
then show *?thesis*
by (*force simp*: ζ)
next
case *False*
have $(y \leq g\ x) \longleftrightarrow$ (*?μ* $< x$) **for** x
proof
show *?μ* $< x$ **if** $y \leq g\ x$
using *that False less_eq_real_def lower* **by** *blast*
show $y \leq g\ x$ **if** *?μ* $< x$
by (*metis g greatest le_less_trans that less_le_trans linear not_less*)
qed

```
      then obtain ζ where ζ: ⋀x. g x ≥ y ⟷ x > ζ ..
      then show ?thesis
        by (force simp: ζ)
    qed
  qed auto
  show ?thesis
     using † by (simp add: UN_subset_iff equiintegrable_on_subset [OF equiin-
tegrable_on_Un [OF gt equiintegrable_on_Un [OF ge equiintegrable_on_Un [OF
ffab 0]]]])
qed


lemma SMVT_lemma4:
  fixes f :: real ⇒ real
  assumes f: f integrable_on {a..b}
    and a ≤ b
    and g: ⋀x y. x ≤ y ⟹ g x ≤ g y
    and 01: ⋀x. ⟦a ≤ x; x ≤ b⟧ ⟹ 0 ≤ g x ∧ g x ≤ 1
  obtains c where a ≤ c c ≤ b ((λx. g x *R f x) has_integral integral {c..b} f)
{a..b}
proof −
  have connected ((λx. integral {x..b} f) ‘ {a..b})
   by (simp add: f indefinite_integral_continuous_1′ connected_continuous_image)
  moreover have compact ((λx. integral {x..b} f) ‘ {a..b})
   by (simp add: compact_continuous_image f indefinite_integral_continuous_1′)
   ultimately obtain m M where int_fab: (λx. integral {x..b} f) ‘ {a..b} =
{m..M}
    using connected_compact_interval_1 by meson
  have ∃c. c ∈ {a..b} ∧
            integral {c..b} f =
            integral {a..b} (λx. (∑k = 1..n. if g x ≥ real k / real n then inverse n
*R f x else 0)) for n
  proof (cases n=0)
    case True
    then show ?thesis
      using ‹a ≤ b› by auto
  next
    case False
    have (⋃c::real. {λx. if g x ≥ c then f x else 0}) equiintegrable_on {a..b}
      using SMVT_lemma2 [OF f g] .
    then have int: (λx. if g x ≥ c then f x else 0) integrable_on {a..b} for c
      by (simp add: equiintegrable_on_def)
    have int′: (λx. if g x ≥ c then u * f x else 0) integrable_on {a..b} for c u
    proof −
      have (λx. if g x ≥ c then u * f x else 0) = (λx. u * (if g x ≥ c then f x else
0))
        by (force simp: if_distrib)
      then show ?thesis
        using integrable_on_cmult_left [OF int] by simp
```

```
  qed
  have ∃ d. d ∈ {a..b} ∧ integral {a..b} (λx. if g x ≥ y then f x else 0) = integral
{d..b} f for y
  proof −
    let ?X = {x. g x ≥ y}
    have ∗: ∃ a. ?X = {a..} ∨ ?X = {a<..}
      if 1: ?X ≠ {} and 2: ?X ≠ UNIV
    proof −
      let ?μ = Inf{x. g x ≥ y}
      have lower: ?μ ≤ x if g x ≥ y for x
      proof (rule cInf_lower)
        show x ∈ {x. y ≤ g x}
          using 1 2 by (auto simp: that)
        show bdd_below {x. y ≤ g x}
          unfolding bdd_below_def
       by (metis 2 UNIV_eq_I dual_order.trans g less_eq_real_def mem_Collect_eq
not_le)
      qed
      have greatest: ?μ ≥ z if ⋀x. g x ≥ y ⟹ z ≤ x for z
        by (metis cInf_greatest mem_Collect_eq that 1)
      show ?thesis
      proof (cases g ?μ ≥ y)
        case True
        then obtain ζ where ζ: ⋀x. g x ≥ y ⟷ x ≥ ζ
          by (metis g lower order.trans)  — in fact y is Inf {x. y ≤ g x}
        then show ?thesis
          by (force simp: ζ)
      next
        case False
        have (y ≤ g x) = (?μ < x) for x
        proof
          show ?μ < x if y ≤ g x
            using that False less_eq_real_def lower by blast
          show y ≤ g x if ?μ < x
            by (metis g greatest le_less_trans that less_le_trans linear not_less)
        qed
        then obtain ζ where ζ: ⋀x. g x ≥ y ⟷ x > ζ ..
        then show ?thesis
          by (force simp: ζ)
      qed
    qed
    then consider ?X = {} | ?X = UNIV | (intv) d where ?X = {d..} ∨ ?X
= {d<..}
      by metis
    then have ∃ d. d ∈ {a..b} ∧ integral {a..b} (λx. if x ∈ ?X then f x else 0) =
integral {d..b} f
    proof cases
      case (intv d)
      show ?thesis
```

```
      proof (cases d < a)
        case True
        with intv have integral {a..b} (λx. if y ≤ g x then f x else 0) = integral
{a..b} f
          by (intro Henstock_Kurzweil_Integration.integral_cong) force
        then show ?thesis
          by (rule_tac x=a in exI) (simp add: ‹a ≤ b›)
      next
        case False
        show ?thesis
        proof (cases b < d)
          case True
          have integral {a..b} (λx. if x ∈ {x. y ≤ g x} then f x else 0) = integral
{a..b} (λx. 0)
            by (rule Henstock_Kurzweil_Integration.integral_cong) (use intv True
in fastforce)
          then show ?thesis
            using ‹a ≤ b› by auto
        next
          case False
            with ‹¬ d < a› have eq: {d..} ∩ {a..b} = {d..b} {d<..} ∩ {a..b} =
{d<..b}
              by force+
          moreover have integral {d<..b} f = integral {d..b} f
          by (rule integral_spike_set [OF empty_imp_negligible negligible_subset
[OF negligible_sing [of d]]]) auto
          ultimately
          have integral {a..b} (λx. if x ∈ {x. y ≤ g x} then f x else 0) =  integral
{d..b} f
            unfolding integral_restrict_Int using intv by presburger
          moreover have d ∈ {a..b}
            using ‹¬ d < a› ‹a ≤ b› False by auto
          ultimately show ?thesis
            by auto
        qed
      qed
    qed (use ‹a ≤ b› in auto)
    then show ?thesis
      by auto
  qed
   then have ∀k. ∃d. d ∈ {a..b} ∧ integral {a..b} (λx. if real k / real n ≤ g x
then f x else 0) = integral {d..b} f
    by meson
  then obtain d where dab: ⋀k. d k ∈ {a..b}
    and deq: ⋀k::nat. integral {a..b} (λx. if k/n ≤ g x then f x else 0) = integral
{d k..b} f
    by metis
  have (∑k = 1..n. integral {a..b} (λx. if real k / real n ≤ g x then f x else 0))
/R n ∈ {m..M}
```

```
    unfolding scaleR_right.sum
   proof (intro conjI allI impI convex [THEN iffD1, rule_format])
    show integral {a..b} (λxa. if real k / real n ≤ g xa then f xa else 0) ∈ {m..M}
for k
      by (metis (no_types, lifting) deq image_eqI int_fab dab)
   qed (use ‹n ≠ 0› in auto)
   then have ∃c. c ∈ {a..b} ∧
           integral {c..b} f = inverse n *R (∑k = 1..n. integral {a..b} (λx. if g
x ≥ real k / real n then f x else 0))
    by (metis (no_types, lifting) int_fab imageE)
   then show ?thesis
   by (simp add: sum_distrib_left if_distrib integral_sum int' flip: integral_mult_right
cong: if_cong)
 qed
 then obtain c where cab: ⋀n. c n ∈ {a..b}
   and c: ⋀n. integral {c n..b} f = integral {a..b} (λx. (∑k = 1..n. if g x ≥ real
k / real n then f x /R n else 0))
   by metis
 obtain d and σ :: nat⇒nat
   where d ∈ {a..b} and σ: strict_mono σ and d: (c ∘ σ) ⟶ d and non0:
⋀n. σ n ≥ Suc 0
 proof −
   have compact{a..b}
    by auto
   with cab obtain d and s0
    where d ∈ {a..b} and s0: strict_mono s0 and tends: (c ∘ s0) ⟶ d
    unfolding compact_def
    using that by blast
   show thesis
   proof
    show d ∈ {a..b}
     by fact
    show strict_mono (s0 ∘ Suc)
     using s0 by (auto simp: strict_mono_def)
    show (c ∘ (s0 ∘ Suc)) ⟶ d
        by (metis tends LIMSEQ_subseq_LIMSEQ Suc_less_eq comp_assoc
strict_mono_def)
    show ⋀n. (s0 ∘ Suc) n ≥ Suc 0
     by (metis comp_apply le0 not_less_eq_eq old.nat.exhaust s0 seq_suble)
   qed
 qed
 define φ where φ ≡ λn x. ∑k = Suc 0..σ n. if k/(σ n) ≤ g x then f x /R (σ
n) else 0
 define ψ where ψ ≡ λn x. ∑k = Suc 0..σ n. if k/(σ n) ≤ g x then inverse (σ
n) else 0
 have ∗∗: (λx. g x *R f x) integrable_on cbox a b ∧
       (λn. integral (cbox a b) (φ n)) ⟶ integral (cbox a b) (λx. g x *R f x)
 proof (rule equiintegrable_limit)
   have †: ((λn. λx. (∑k = Suc 0..n. if k / n ≤ g x then inverse n *R f x else
```

```
0)) ‘ {Suc 0..}) equiintegrable_on {a..b}
  proof −
    have ∗: (⋃ c::real. {λx. if g x ≥ c then f x else 0}) equiintegrable_on {a..b}
      using SMVT_lemma2 [OF f g] .
    show ?thesis
     apply (rule equiintegrable_on_subset [OF equiintegrable_sum_real [OF ∗]],
clarify)
      apply (rule_tac a={Suc 0..n} in UN_I, force)
      apply (rule_tac a=λk. inverse n in UN_I, auto)
      apply (rule_tac x=λk x. if real k / real n ≤ g x then f x else 0 in bexI)
       apply (force intro: sum.cong)+
      done
  qed
  show range φ equiintegrable_on cbox a b
    unfolding φ_def
    by (auto simp: non0 intro: equiintegrable_on_subset [OF †])
  show (λn. φ n x) ⟶ g x ∗R f x
    if x: x ∈ cbox a b for x
  proof −
    have eq: φ n x = ψ n x ∗R f x  for n
      by (auto simp: φ_def ψ_def sum_distrib_right if_distrib intro: sum.cong)
    show ?thesis
      unfolding eq
    proof (rule tendsto_scaleR [OF _ tendsto_const])
      show (λn. ψ n x) ⟶ g x
        unfolding lim_sequentially dist_real_def
      proof (intro allI impI)
        fix e :: real
        assume e > 0
        then obtain N where N ≠ 0 0 < inverse (real N) and N: inverse (real
N) < e
          using real_arch_inverse by metis
        moreover have |ψ n x − g x| < inverse (real N) if n≥N for n
        proof −
          have |g x − ψ n x| < inverse (real (σ n))
            unfolding ψ_def
          proof (rule level_approx [of {a..b} g])
            show σ n ≠ 0
              by (metis Suc_n_not_le_n non0)
          qed (use x 01 non0 in auto)
          also have ... ≤ inverse N
           using seq_suble [OF σ] ‹N ≠ 0› non0 that by (auto intro: order_trans
simp: field_split_simps)
          finally show ?thesis
            by linarith
        qed
        ultimately show ∃ N. ∀ n≥N. |ψ n x − g x| < e
          using less_trans by blast
      qed
```

```
    qed
   qed
  qed
  show thesis
  proof
   show a ≤ d d ≤ b
    using ‹d ∈ {a..b}› atLeastAtMost_iff by blast+
   show ((λx. g x *R f x) has_integral integral {d..b} f) {a..b}
    unfolding has_integral_iff
   proof
    show (λx. g x *R f x) integrable_on {a..b}
     using ** by simp
    show integral {a..b} (λx. g x *R f x) = integral {d..b} f
    proof (rule tendsto_unique)
     show (λn. integral {c(σ n)..b} f) ⟶ integral {a..b} (λx. g x *R f x)
      using ** by (simp add: c φ_def)
     have continuous (at d within {a..b}) (λx. integral {x..b} f)
      using indefinite_integral_continuous_1' [OF f] ‹d ∈ {a..b}›
      by (simp add: continuous_on_eq_continuous_within)
     then show (λn. integral {c(σ n)..b} f) ⟶ integral {d..b} f
      using d cab unfolding o_def
      by (simp add: continuous_within_sequentially o_def)
    qed auto
   qed
  qed
qed


theorem second_mean_value_theorem_full:
  fixes f :: real ⇒ real
  assumes f: f integrable_on {a..b} and a ≤ b
    and g: ⋀x y. ⟦a ≤ x; x ≤ y; y ≤ b⟧ ⟹ g x ≤ g y
  obtains c where c ∈ {a..b}
    and ((λx. g x * f x) has_integral (g a * integral {a..c} f + g b * integral {c..b}
f)) {a..b}
proof −
  have gab: g a ≤ g b
    using ‹a ≤ b› g by blast
  then consider g a < g b | g a = g b
    by linarith
  then show thesis
  proof cases
    case 1
    define h where h ≡ λx. if x < a then 0 else if b < x then 1
                                 else (g x − g a) / (g b − g a)
    obtain c where a ≤ c c ≤ b and c: ((λx. h x *R f x) has_integral integral
{c..b} f) {a..b}
    proof (rule SMVT_lemma4 [OF f ‹a ≤ b›, of h])
     show h x ≤ h y 0 ≤ h x ∧ h x ≤ 1 if x ≤ y for x y
```

```
      using that gab by (auto simp: divide_simps g h_def)
    qed
    show ?thesis
    proof
      show c ∈ {a..b}
        using ‹a ≤ c› ‹c ≤ b› by auto
      have I: ((λx. g x * f x − g a * f x) has_integral (g b − g a) * integral {c..b}
f) {a..b}
      proof (subst has_integral_cong)
        show g x * f x − g a * f x = (g b − g a) * h x *R f x
          if x ∈ {a..b} for x
          using 1 that by (simp add: h_def field_split_simps)
       show ((λx. (g b − g a) * h x *R f x) has_integral (g b − g a) * integral
{c..b} f) {a..b}
          using has_integral_mult_right [OF c, of g b − g a] .
      qed
      have II: ((λx. g a * f x) has_integral g a * integral {a..b} f) {a..b}
       using has_integral_mult_right [where c = g a, OF integrable_integral [OF
f]] .
       have ((λx. g x * f x) has_integral (g b − g a) * integral {c..b} f + g a *
integral {a..b} f) {a..b}
        using has_integral_add [OF I II] by simp
     then show ((λx. g x * f x) has_integral g a * integral {a..c} f + g b * integral
{c..b} f) {a..b}
       by (simp add: algebra_simps flip: integral_combine [OF ‹a ≤ c› ‹c ≤ b› f])
    qed
  next
    case 2
    show ?thesis
    proof
      show a ∈ {a..b}
        by (simp add: ‹a ≤ b›)
      have ((λx. g x * f x) has_integral g a * integral {a..b} f) {a..b}
      proof (rule has_integral_eq)
        show ((λx. g a * f x) has_integral g a * integral {a..b} f) {a..b}
          using f has_integral_mult_right by blast
        show g a * f x = g x * f x
          if x ∈ {a..b} for x
          by (metis atLeastAtMost_iff g less_eq_real_def not_le that 2)
      qed
     then show ((λx. g x * f x) has_integral g a * integral {a..a} f + g b * integral
{a..b} f) {a..b}
        by (simp add: 2)
    qed
  qed
qed


corollary second_mean_value_theorem:
```

fixes $f :: real \Rightarrow real$
assumes f: f *integrable_on* $\{a..b\}$ **and** $a \le b$
and g: $\bigwedge x\ y.\ [\![a \le x;\ x \le y;\ y \le b]\!] \Longrightarrow g\ x \le g\ y$
obtains c **where** $c \in \{a..b\}$
integral $\{a..b\}$ $(\lambda x.\ g\ x * f\ x) = g\ a * integral\ \{a..c\}\ f + g\ b * integral\ \{c..b\}\ f$
using *second_mean_value_theorem_full* [**where** g=g, *OF assms*]
by (*metis* (*full_types*) *integral_unique*)

end

6.25 Continuous Extensions of Functions

theory *Continuous_Extension*
imports *Starlike*
begin

6.25.1 Partitions of unity subordinate to locally finite open coverings

A difference from HOL Light: all summations over infinite sets equal zero, so the "support" must be made explicit in the summation below!

proposition *subordinate_partition_of_unity*:
fixes S :: *'a::metric_space set*
assumes $S \subseteq \bigcup \mathcal{C}$ **and** *opC*: $\bigwedge T.\ T \in \mathcal{C} \Longrightarrow open\ T$
and *fin*: $\bigwedge x.\ x \in S \Longrightarrow \exists\, V.\ open\ V \wedge x \in V \wedge finite\ \{U \in \mathcal{C}.\ U \cap V \neq \{\}\}$
obtains F :: [*'a set*, *'a*] $\Rightarrow$ *real*
where $\bigwedge U.\ U \in \mathcal{C} \Longrightarrow continuous\_on\ S\ (F\ U) \wedge (\forall x \in S.\ 0 \le F\ U\ x)$
and $\bigwedge x\ U.\ [\![U \in \mathcal{C};\ x \in S;\ x \notin U]\!] \Longrightarrow F\ U\ x = 0$
and $\bigwedge x.\ x \in S \Longrightarrow supp\_sum\ (\lambda W.\ F\ W\ x)\ \mathcal{C} = 1$
and $\bigwedge x.\ x \in S \Longrightarrow \exists\, V.\ open\ V \wedge x \in V \wedge finite\ \{U \in \mathcal{C}.\ \exists x \in V.\ F\ U\ x \neq 0\}$
proof (*cases* $\exists\, W.\ W \in \mathcal{C} \wedge S \subseteq W$)
case *True*
then obtain W **where** $W \in \mathcal{C}\ S \subseteq W$ **by** *metis*
then show *?thesis*
by (*rule_tac* $F = \lambda V\ x.\ if\ V = W\ then\ 1\ else\ 0$ **in** *that*) (*auto simp*: *supp_sum_def support_on_def*)
next
case *False*
have *nonneg*: $0 \le supp\_sum\ (\lambda V.\ setdist\ \{x\}\ (S - V))\ \mathcal{C}$ **for** x
by (*simp add*: *supp_sum_def sum_nonneg*)
have *sd_pos*: $0 < setdist\ \{x\}\ (S - V)$ **if** $V \in \mathcal{C}\ x \in S\ x \in V$ **for** $V\ x$
proof −
have *closedin* (*top_of_set S*) $(S - V)$
by (*simp add*: *Diff_Diff_Int closedin_def opC openin_open_Int* ‹$V \in \mathcal{C}$›)
with *that False setdist_pos_le* [*of* $\{x\}$ $S - V$]
show *?thesis*

```
      using setdist_gt_0_closedin by fastforce
    qed
    have ss_pos: 0 < supp_sum (λV. setdist {x} (S − V)) C if x ∈ S for x
    proof −
      obtain U where U ∈ C x ∈ U using ‹x ∈ S› ‹S ⊆ ⋃C›
        by blast
      obtain V where open V x ∈ V finite {U ∈ C. U ∩ V ≠ {}}
        using ‹x ∈ S› fin by blast
      then have ∗: finite {A ∈ C. ¬ S ⊆ A ∧ x ∉ closure (S − A)}
        using closure_def that by (blast intro: rev_finite_subset)
      have x ∉ closure (S − U)
        using ‹U ∈ C› ‹x ∈ U› opC open_Int_closure_eq_empty by fastforce
      then show ?thesis
        apply (simp add: setdist_eq_0_sing_1 supp_sum_def support_on_def)
        apply (rule ordered_comm_monoid_add_class.sum_pos2 [OF ∗, of U])
        using ‹U ∈ C› ‹x ∈ U› False
        apply (auto simp: sd_pos that)
        done
    qed
    define F where
      F ≡ λW x. if x ∈ S then setdist {x} (S − W) / supp_sum (λV. setdist {x}
(S − V)) C else 0
    show ?thesis
    proof (rule_tac F = F in that)
      have continuous_on S (F U) if U ∈ C for U
      proof −
        have ∗: continuous_on S (λx. supp_sum (λV. setdist {x} (S − V)) C)
        proof (clarsimp simp add: continuous_on_eq_continuous_within)
          fix x assume x ∈ S
          then obtain X where open X and x: x ∈ S ∩ X and finX: finite {U ∈
C. U ∩ X ≠ {}}
            using assms by blast
          then have OSX: openin (top_of_set S) (S ∩ X) by blast
          have sumeq: ⋀x. x ∈ S ∩ X ⟹
                  (∑ V | V ∈ C ∧ V ∩ X ≠ {}. setdist {x} (S − V))
                  = supp_sum (λV. setdist {x} (S − V)) C
            apply (simp add: supp_sum_def)
            apply (rule sum.mono_neutral_right [OF finX])
            apply (auto simp: setdist_eq_0_sing_1 support_on_def subset_iff)
            apply (meson DiffI closure_subset disjoint_iff_not_equal subsetCE)
            done
          show continuous (at x within S) (λx. supp_sum (λV. setdist {x} (S −
V)) C)
            apply (rule continuous_transform_within_openin
                [where f = λx. (sum (λV. setdist {x} (S − V)) {V ∈ C. V ∩ X
≠ {}})
                    and S =S ∩ X])
        apply (rule continuous_intros continuous_at_setdist continuous_at_imp_continuous_at_within
OSX x)+
```

```
        apply (simp add: sumeq)
        done
    qed
    show ?thesis
      apply (simp add: F_def)
      apply (rule continuous_intros *)+
      using ss_pos apply force
      done
  qed
  moreover have ⟦U ∈ 𝒞; x ∈ S⟧ ⟹ 0 ≤ F U x for U x
    using nonneg [of x] by (simp add: F_def field_split_simps)
  ultimately show ⋀U. U ∈ 𝒞 ⟹ continuous_on S (F U) ∧ (∀x∈S. 0 ≤ F
U x)
    by metis
 next
  show ⋀x U. ⟦U ∈ 𝒞; x ∈ S; x ∉ U⟧ ⟹ F U x = 0
    by (simp add: setdist_eq_0_sing_1 closure_def F_def)
 next
  show supp_sum (λW. F W x) 𝒞 = 1 if x ∈ S for x
    using that ss_pos [OF that]
  by (simp add: F_def field_split_simps supp_sum_divide_distrib [symmetric])
 next
   show ∃V. open V ∧ x ∈ V ∧ finite {U ∈ 𝒞. ∃x∈V. F U x ≠ 0} if x ∈ S
for x
    using fin [OF that] that
  by (fastforce simp: setdist_eq_0_sing_1 closure_def F_def elim!: rev_finite_subset)
 qed
qed
```

6.25.2 Urysohn's Lemma for Euclidean Spaces

For Euclidean spaces the proof is easy using distances.

```
lemma Urysohn_both_ne:
  assumes US: closedin (top_of_set U) S
      and UT: closedin (top_of_set U) T
      and S ∩ T = {} S ≠ {} T ≠ {} a ≠ b
  obtains f :: 'a::euclidean_space ⇒ 'b::real_normed_vector
    where continuous_on U f
          ⋀x. x ∈ U ⟹ f x ∈ closed_segment a b
          ⋀x. x ∈ U ⟹ (f x = a ⟷ x ∈ S)
          ⋀x. x ∈ U ⟹ (f x = b ⟷ x ∈ T)
proof −
  have S0: ⋀x. x ∈ U ⟹ setdist {x} S = 0 ⟷ x ∈ S
    using ‹S ≠ {}› US setdist_eq_0_closedin by auto
  have T0: ⋀x. x ∈ U ⟹ setdist {x} T = 0 ⟷ x ∈ T
    using ‹T ≠ {}› UT setdist_eq_0_closedin by auto
  have sdpos: 0 < setdist {x} S + setdist {x} T if x ∈ U for x
  proof −
    have ¬ (setdist {x} S = 0 ∧ setdist {x} T = 0)
```

```
    using assms by (metis IntI empty_iff setdist_eq_0_closedin that)
   then show ?thesis
   by (metis add.left_neutral add.right_neutral add_pos_pos linorder_neqE_linordered_idom
not_le setdist_pos_le)
 qed
 define f where f ≡ λx. a + (setdist {x} S / (setdist {x} S + setdist {x} T))
*R (b − a)
 show ?thesis
 proof (rule_tac f = f in that)
  show continuous_on U f
   using sdpos unfolding f_def
   by (intro continuous_intros | force)+
  show f x ∈ closed_segment a b if x ∈ U for x
    unfolding f_def
   apply (simp add: closed_segment_def)
   apply (rule_tac x=(setdist {x} S / (setdist {x} S + setdist {x} T)) in exI)
   using sdpos that apply (simp add: algebra_simps)
   done
  show ⋀x. x ∈ U ⟹ (f x = a ⟷ x ∈ S)
   using S0 ‹a ≠ b› f_def sdpos by force
  show (f x = b ⟷ x ∈ T) if x ∈ U for x
  proof −
   have f x = b ⟷ (setdist {x} S / (setdist {x} S + setdist {x} T)) = 1
    unfolding f_def
    by (metis add_diff_cancel_left' ‹a ≠ b› diff_add_cancel eq_iff_diff_eq_0
scaleR_cancel_right scaleR_one)
   also have ... ⟷ setdist {x} T = 0 ∧ setdist {x} S ≠ 0
    using sdpos that
    by (simp add: field_split_simps) linarith
   also have ... ⟷ x ∈ T
    using ‹S ≠ {}› ‹T ≠ {}› ‹S ∩ T = {}› that
    by (force simp: S0 T0)
   finally show ?thesis .
  qed
 qed
qed

lemma Urysohn_local_strong_aux:
 assumes US: closedin (top_of_set U) S
    and UT: closedin (top_of_set U) T
    and S ∩ T = {} a ≠ b S ≠ {}
 obtains f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  where continuous_on U f
       ⋀x. x ∈ U ⟹ f x ∈ closed_segment a b
       ⋀x. x ∈ U ⟹ (f x = a ⟷ x ∈ S)
       ⋀x. x ∈ U ⟹ (f x = b ⟷ x ∈ T)
proof (cases T = {})
 case True show ?thesis
 proof (cases S = U)
```

```
    case True with ‹T = {}› ‹a ≠ b› show ?thesis
      by (rule_tac f = λx. a in that) (auto)
  next
    case False
    with US closedin_subset obtain c where c: c ∈ U c ∉ S
      by fastforce
    obtain f where f: continuous_on U f
      ⋀x. x ∈ U ⟹ f x ∈ closed_segment a (midpoint a b)
      ⋀x. x ∈ U ⟹ (f x = midpoint a b ⟷ x = c)
      ⋀x. x ∈ U ⟹ (f x = a ⟷ x ∈ S)
      apply (rule Urysohn_both_ne [of U S {c} a midpoint a b])
      using c ‹S ≠ {}› assms apply simp_all
      apply (metis midpoint_eq_endpoint)
      done
    show ?thesis
      apply (rule_tac f=f in that)
      using ‹S ≠ {}› ‹T = {}› f  ‹a ≠ b›
        apply simp_all
     apply (metis (no_types) closed_segment_commute csegment_midpoint_subset
midpoint_sym subset_iff)
    apply (metis closed_segment_commute midpoint_sym notin_segment_midpoint)
      done
  qed
next
  case False
  show ?thesis
    using Urysohn_both_ne [OF US UT ‹S ∩ T = {}› ‹S ≠ {}› ‹T ≠ {}› ‹a ≠
b›] that
    by blast
qed

proposition Urysohn_local_strong:
  assumes US: closedin (top_of_set U) S
      and UT: closedin (top_of_set U) T
      and S ∩ T = {} a ≠ b
  obtains f :: 'a::euclidean_space ⇒ 'b::euclidean_space
    where continuous_on U f
          ⋀x. x ∈ U ⟹ f x ∈ closed_segment a b
          ⋀x. x ∈ U ⟹ (f x = a ⟷ x ∈ S)
          ⋀x. x ∈ U ⟹ (f x = b ⟷ x ∈ T)
proof (cases S = {})
  case True show ?thesis
  proof (cases T = {})
    case True show ?thesis
    proof (rule_tac f = λx. midpoint a b in that)
      show continuous_on U (λx. midpoint a b)
        by (intro continuous_intros)
      show midpoint a b ∈ closed_segment a b
        using csegment_midpoint_subset by blast
```

```
      show (midpoint a b = a) = (x ∈ S) for x
        using ‹S = {}› ‹a ≠ b› by simp
      show (midpoint a b = b) = (x ∈ T) for x
        using ‹T = {}› ‹a ≠ b› by simp
    qed
  next
    case False
    with Urysohn_local_strong_aux [OF UT US] assms show ?thesis
      by (smt (verit) True closed_segment_commute inf_bot_right that)
  qed
next
  case False
  with Urysohn_local_strong_aux [OF assms] show ?thesis
    using that by blast
qed

lemma Urysohn_local:
  assumes US: closedin (top_of_set U) S
      and UT: closedin (top_of_set U) T
      and S ∩ T = {}
  obtains f :: 'a::euclidean_space ⇒ 'b::euclidean_space
    where continuous_on U f
          ⋀x. x ∈ U ⟹ f x ∈ closed_segment a b
          ⋀x. x ∈ S ⟹ f x = a
          ⋀x. x ∈ T ⟹ f x = b
proof (cases a = b)
  case True then show ?thesis
    by (rule_tac f = λx. b in that) (auto)
next
  case False
  with Urysohn_local_strong [OF assms] show ?thesis
    by (smt (verit) US UT closedin_imp_subset subset_eq that)
qed

lemma Urysohn_strong:
  assumes US: closed S
      and UT: closed T
      and S ∩ T = {} a ≠ b
  obtains f :: 'a::euclidean_space ⇒ 'b::euclidean_space
    where continuous_on UNIV f
          ⋀x. f x ∈ closed_segment a b
          ⋀x. f x = a ⟷ x ∈ S
          ⋀x. f x = b ⟷ x ∈ T
using assms by (auto intro: Urysohn_local_strong [of UNIV S T])

proposition Urysohn:
  assumes US: closed S
      and UT: closed T
      and S ∩ T = {}
```

obtains $f :: 'a{::}euclidean\_space \Rightarrow 'b{::}euclidean\_space$
where *continuous_on UNIV f*
$\bigwedge x.\ f\ x \in closed\_segment\ a\ b$
$\bigwedge x.\ x \in S \Longrightarrow f\ x = a$
$\bigwedge x.\ x \in T \Longrightarrow f\ x = b$
using *assms* **by** (*auto intro*: *Urysohn_local* [*of UNIV S T a b*])

6.25.3 Dugundji's Extension Theorem and Tietze Variants

See [3].

theorem *Dugundji*:
fixes $f :: 'a{::}\{metric\_space,second\_countable\_topology\} \Rightarrow 'b{::}real\_inner$
assumes *convex C* $C \neq \{\}$
and *cloin*: *closedin* (*top_of_set U*) *S*
and *contf*: *continuous_on S f* **and** $f\ `\ S \subseteq C$
obtains *g* **where** *continuous_on U g* $g\ `\ U \subseteq C$
$\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
proof (*cases* $S = \{\}$)
case *True* **show** *thesis*
proof
show *continuous_on U* ($\lambda x.$ *SOME y*. $y \in C$)
by (*rule continuous_intros*)
show ($\lambda x.$ *SOME y*. $y \in C$) $`\ U \subseteq C$
by (*simp add*: ‹$C \neq \{\}$› *image_subsetI some_in_eq*)
qed (*use True* **in** *auto*)
next
case *False*
then have *sd_pos*: $\bigwedge x.\ [\![x \in U;\ x \notin S]\!] \Longrightarrow 0 < setdist\ \{x\}\ S$
using *setdist_eq_0_closedin* [*OF cloin*] *le_less setdist_pos_le* **by** *fastforce*
define $\mathcal{B}$ **where** $\mathcal{B} = \{ball\ x\ (setdist\ \{x\}\ S\ /\ 2)\ |x.\ x \in U - S\}$
have [*simp*]: $\bigwedge T.\ T \in \mathcal{B} \Longrightarrow open\ T$
by (*auto simp*: $\mathcal{B}$*_def*)
have *USS*: $U - S \subseteq \bigcup \mathcal{B}$
by (*auto simp*: *sd_pos* $\mathcal{B}$*_def*)
obtain $\mathcal{C}$ **where** *USsub*: $U - S \subseteq \bigcup \mathcal{C}$
and *nbrhd*: $\bigwedge U.\ U \in \mathcal{C} \Longrightarrow open\ U \wedge (\exists T.\ T \in \mathcal{B} \wedge U \subseteq T)$
and *fin*: $\bigwedge x.\ x \in U - S \Longrightarrow \exists V.\ open\ V \wedge x \in V \wedge finite\ \{U.\ U \in \mathcal{C} \wedge U \cap V \neq \{\}\}$
by (*rule paracompact* [*OF USS*]) *auto*
have $\exists v\ a.\ v \in U \wedge v \notin S \wedge a \in S \wedge$
$T \subseteq ball\ v\ (setdist\ \{v\}\ S\ /\ 2) \wedge$
$dist\ v\ a \leq 2 * setdist\ \{v\}\ S$ **if** $T \in \mathcal{C}$ **for** *T*
proof −
obtain *v* **where** *v*: $T \subseteq ball\ v\ (setdist\ \{v\}\ S\ /\ 2)\ v \in U\ v \notin S$
using ‹$T \in \mathcal{C}$› *nbrhd* **by** (*force simp*: $\mathcal{B}$*_def*)
then obtain *a* **where** $a \in S\ dist\ v\ a < 2 * setdist\ \{v\}\ S$
using *setdist_ltE* [*of* $\{v\}$ *S* 2 * *setdist* $\{v\}$ *S*]
using *False sd_pos* **by** *force*
with *v* **show** *?thesis*

```
    by force
  qed
  then obtain 𝒱 𝒜 where
    VA: ⋀T. T ∈ 𝒞 ⟹ 𝒱 T ∈ U ∧ 𝒱 T ∉ S ∧ 𝒜 T ∈ S ∧
          T ⊆ ball (𝒱 T) (setdist {𝒱 T} S / 2) ∧
          dist (𝒱 T) (𝒜 T) ≤ 2 * setdist {𝒱 T} S
    by metis
  have sdle: setdist {𝒱 T} S ≤ 2 * setdist {v} S if T ∈ 𝒞 v ∈ T for T v
    using setdist_Lipschitz [of 𝒱 T S v] VA [OF ‹T ∈ 𝒞›] ‹v ∈ T› by auto
  have d6: dist a (𝒜 T) ≤ 6 * dist a v if T ∈ 𝒞 v ∈ T a ∈ S for T v a
  proof −
    have dist (𝒱 T) v < setdist {𝒱 T} S / 2
      using that VA mem_ball by blast
    also have ... ≤ setdist {v} S
      using sdle [OF ‹T ∈ 𝒞› ‹v ∈ T›] by simp
    also have vS: setdist {v} S ≤ dist a v
      by (simp add: setdist_le_dist setdist_sym ‹a ∈ S›)
    finally have VTV: dist (𝒱 T) v < dist a v .
    have VTS: setdist {𝒱 T} S ≤ 2 * dist a v
      using sdle that vS by force
    have dist a (𝒜 T) ≤ dist a v + dist v (𝒱 T) + dist (𝒱 T) (𝒜 T)
      by (metis add.commute add_le_cancel_left dist_commute dist_triangle2
dist_triangle_le)
    also have ... ≤ dist a v + dist a v + dist (𝒱 T) (𝒜 T)
      using VTV by (simp add: dist_commute)
    also have ... ≤ 2 * dist a v + 2 * setdist {𝒱 T} S
      using VA [OF ‹T ∈ 𝒞›] by auto
    finally show ?thesis
      using VTS by linarith
  qed
  obtain H :: ['a set, 'a] ⇒ real
    where Hcont: ⋀Z. Z ∈ 𝒞 ⟹ continuous_on (U−S) (H Z)
      and Hge0: ⋀Z x. ⟦Z ∈ 𝒞; x ∈ U−S⟧ ⟹ 0 ≤ H Z x
      and Heq0: ⋀x Z. ⟦Z ∈ 𝒞; x ∈ U−S; x ∉ Z⟧ ⟹ H Z x = 0
      and H1: ⋀x. x ∈ U−S ⟹ supp_sum (λW. H W x) 𝒞 = 1
      and Hfin: ⋀x. x ∈ U−S ⟹ ∃ V. open V ∧ x ∈ V ∧ finite {U ∈ 𝒞. ∃x∈V.
H U x ≠ 0}
    apply (rule subordinate_partition_of_unity [OF USsub _ fin])
    using nbrhd by auto
  define g where g ≡ λx. if x ∈ S then f x else supp_sum (λT. H T x *R f(𝒜
T)) 𝒞
  show ?thesis
  proof (rule that)
    show continuous_on U g
    proof (clarsimp simp: continuous_on_eq_continuous_within)
      fix a assume a ∈ U
      show continuous (at a within U) g
      proof (cases a ∈ S)
        case True show ?thesis
```

```
      proof (clarsimp simp add: continuous_within_topological)
        fix W
        assume open W g a ∈ W
        then obtain e where 0 < e and e: ball (f a) e ⊆ W
          using openE True g_def by auto
        have continuous (at a within S) f
          using True contf continuous_on_eq_continuous_within by blast
        then obtain d where 0 < d
                    and d: ⋀x. ⟦x ∈ S; dist x a < d⟧ ⟹ dist (f x) (f a) < e
          using continuous_within_eps_delta ‹0 < e› by force
        have g y ∈ ball (f a) e if y ∈ U and y: y ∈ ball a (d / 6) for y
        proof (cases y ∈ S)
          case True
          then have dist (f a) (f y) < e
         by (metis ball_divide_subset_numeral dist_commute in_mono mem_ball
y d)
          then show ?thesis
            by (simp add: True g_def)
        next
          case False
          have ∗: dist (f (𝒜 T)) (f a) < e if T ∈ 𝒞 H T y ≠ 0 for T
          proof −
            have y ∈ T
              using Heq0 that False ‹y ∈ U› by blast
            have dist (𝒜 T) a < d
              using d6 [OF ‹T ∈ 𝒞› ‹y ∈ T› ‹a ∈ S›] y
              by (simp add: dist_commute mult.commute)
            then show ?thesis
              using VA [OF ‹T ∈ 𝒞›] by (auto simp: d)
          qed
          have supp_sum (λT. H T y ∗R f (𝒜 T)) 𝒞 ∈ ball (f a) e
            apply (rule convex_supp_sum [OF convex_ball])
            apply (simp_all add: False H1 Hge0 ‹y ∈ U›)
            by (metis dist_commute ∗)
          then show ?thesis
            by (simp add: False g_def)
        qed
        then show ∃A. open A ∧ a ∈ A ∧ (∀y∈U. y ∈ A ⟶ g y ∈ W)
          apply (rule_tac x = ball a (d / 6) in exI)
          using e ‹0 < d› by fastforce
      qed
    next
      case False
      obtain N where N: open N a ∈ N
                and finN: finite {U ∈ 𝒞. ∃a∈N. H U a ≠ 0}
        using Hfin False ‹a ∈ U› by auto
      have oUS: openin (top_of_set U) (U − S)
        using cloin by (simp add: openin_diff)
      have HcontU: continuous (at a within U) (H T) if T ∈ 𝒞 for T
```

```
        using Hcont [OF ‹T ∈ 𝒞›] False  ‹a ∈ U› ‹T ∈ 𝒞›
      apply (simp add: continuous_on_eq_continuous_within continuous_within)
        apply (rule Lim_transform_within_set)
        using oUS
          apply (force simp: eventually_at openin_contains_ball dist_commute
dest!: bspec)+
        done
      show ?thesis
      proof (rule continuous_transform_within_openin [OF _ oUS])
       show continuous (at a within U) (λx. supp_sum (λT. H T x *R f (𝒜 T))
𝒞)
        proof (rule continuous_transform_within_openin)
          show continuous (at a within U)
                 (λx. ∑ T∈{U ∈ 𝒞. ∃x∈N. H U x ≠ 0}. H T x *R f (𝒜 T))
           by (force intro: continuous_intros HcontU)+
        next
          show openin (top_of_set U) ((U − S) ∩ N)
           using N oUS openin_trans by blast
        next
          show a ∈ (U − S) ∩ N using False ‹a ∈ U› N by blast
        next
          show ⋀x. x ∈ (U − S) ∩ N ⟹
                       (∑ T ∈ {U ∈ 𝒞. ∃x∈N. H U x ≠ 0}. H T x *R f (𝒜 T))
                       = supp_sum (λT. H T x *R f (𝒜 T)) 𝒞
           by (auto simp: supp_sum_def support_on_def
                    intro: sum.mono_neutral_right [OF finN])
          qed
        next
          show a ∈ U − S using False ‹a ∈ U› by blast
        next
          show ⋀x. x ∈ U − S ⟹ supp_sum (λT. H T x *R f (𝒜 T)) 𝒞 = g x
           by (simp add: g_def)
        qed
      qed
    qed
    show g ‘ U ⊆ C
     using ‹f ‘ S ⊆ C› VA
      by (fastforce simp: g_def Hge0 intro!: convex_supp_sum [OF ‹convex C›]
H1)
    show ⋀x. x ∈ S ⟹ g x = f x
     by (simp add: g_def)
  qed
qed


corollary Tietze:
  fixes f :: 'a::{metric_space,second_countable_topology} ⇒ 'b::real_inner
  assumes continuous_on S f
    and closedin (top_of_set U) S
```

and $0 \leq B$
and $\bigwedge x.\ x \in S \Longrightarrow norm(f\ x) \leq B$
obtains g **where** *continuous_on* $U\ g$ $\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
$\bigwedge x.\ x \in U \Longrightarrow norm(g\ x) \leq B$
using *assms* **by** (*auto simp*: *image_subset_iff intro*: *Dugundji* [*of cball 0 B U S f*])

corollary *Tietze_closed_interval*:
fixes f :: *'a*::{*metric_space*,*second_countable_topology*} ⇒ *'b*::*euclidean_space*
assumes *continuous_on* $S\ f$
and *closedin* (*top_of_set* U) S
and $cbox\ a\ b \neq \{\}$
and $\bigwedge x.\ x \in S \Longrightarrow f\ x \in cbox\ a\ b$
obtains g **where** *continuous_on* $U\ g$ $\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
$\bigwedge x.\ x \in U \Longrightarrow g\ x \in cbox\ a\ b$
apply (*rule Dugundji* [*of cbox a b U S f*])
using *assms* **by** *auto*

corollary *Tietze_closed_interval_1*:
fixes f :: *'a*::{*metric_space*,*second_countable_topology*} ⇒ *real*
assumes *continuous_on* $S\ f$
and *closedin* (*top_of_set* U) S
and $a \leq b$
and $\bigwedge x.\ x \in S \Longrightarrow f\ x \in cbox\ a\ b$
obtains g **where** *continuous_on* $U\ g$ $\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
$\bigwedge x.\ x \in U \Longrightarrow g\ x \in cbox\ a\ b$
apply (*rule Dugundji* [*of cbox a b U S f*])
using *assms* **by** (*auto simp*: *image_subset_iff*)

corollary *Tietze_open_interval*:
fixes f :: *'a*::{*metric_space*,*second_countable_topology*} ⇒ *'b*::*euclidean_space*
assumes *continuous_on* $S\ f$
and *closedin* (*top_of_set* U) S
and $box\ a\ b \neq \{\}$
and $\bigwedge x.\ x \in S \Longrightarrow f\ x \in box\ a\ b$
obtains g **where** *continuous_on* $U\ g$ $\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
$\bigwedge x.\ x \in U \Longrightarrow g\ x \in box\ a\ b$
apply (*rule Dugundji* [*of box a b U S f*])
using *assms* **by** *auto*

corollary *Tietze_open_interval_1*:
fixes f :: *'a*::{*metric_space*,*second_countable_topology*} ⇒ *real*
assumes *continuous_on* $S\ f$
and *closedin* (*top_of_set* U) S
and $a < b$
and *no*: $\bigwedge x.\ x \in S \Longrightarrow f\ x \in box\ a\ b$
obtains g **where** *continuous_on* $U\ g$ $\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
$\bigwedge x.\ x \in U \Longrightarrow g\ x \in box\ a\ b$
apply (*rule Dugundji* [*of box a b U S f*])

using *assms* **by** (*auto simp*: *image_subset_iff*)

corollary *Tietze_unbounded*:
fixes f :: *'a*::{*metric_space*,*second_countable_topology*} ⇒ *'b*::*real_inner*
assumes *continuous_on S f*
and *closedin* (*top_of_set U*) *S*
obtains g **where** *continuous_on U g* $\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
apply (*rule Dugundji* [*of UNIV U S f*])
using *assms* **by** *auto*

end

6.26 Equivalence Between Classical Borel Measurability and HOL Light's

theory *Equivalence_Measurable_On_Borel*
imports *Equivalence_Lebesgue_Henstock_Integration Improper_Integral Continuous_Extension*
begin

6.26.1 Austin's Lemma

lemma *Austin_Lemma*:
fixes $\mathcal{D}$ :: *'a*::*euclidean_space set set*
assumes *finite* $\mathcal{D}$ **and** $\mathcal{D}$: $\bigwedge D.\ D \in \mathcal{D} \Longrightarrow \exists k\ a\ b.\ D = cbox\ a\ b \wedge (\forall i \in Basis.\ b{\cdot}i - a{\cdot}i = k)$
obtains $\mathcal{C}$ **where** $\mathcal{C} \subseteq \mathcal{D}$ *pairwise disjnt* $\mathcal{C}$
measure lebesgue $(\bigcup \mathcal{C}) \geq$ *measure lebesgue* $(\bigcup \mathcal{D})$ / *3* ^ (*DIM*(*'a*))
using *assms*
proof (*induction card* $\mathcal{D}$ *arbitrary*: $\mathcal{D}$ *thesis rule*: *less_induct*)
case *less*
show *?case*
proof (*cases* $\mathcal{D} = \{\}$)
case *True*
then show *thesis*
using *less* **by** *auto*
next
case *False*
then have *Max* (*Sigma_Algebra.measure lebesgue* ' $\mathcal{D}$) ∈ *Sigma_Algebra.measure lebesgue* ' $\mathcal{D}$
using *Max_in finite_imageI* ‹*finite* $\mathcal{D}$› **by** *blast*
then obtain D **where** $D \in \mathcal{D}$ **and** *measure lebesgue* D = *Max* (*measure lebesgue* ' $\mathcal{D}$)
by *auto*
then have D: $\bigwedge C.\ C \in \mathcal{D} \Longrightarrow$ *measure lebesgue* $C \leq$ *measure lebesgue* D
by (*simp add*: ‹*finite* $\mathcal{D}$›)
let $?\mathcal{E} = \{C.\ C \in \mathcal{D} - \{D\} \wedge$ *disjnt* $C\ D\}$
obtain $\mathcal{D}'$ **where** $\mathcal{D}'$*sub*: $\mathcal{D}' \subseteq ?\mathcal{E}$ **and** $\mathcal{D}'$*dis*: *pairwise disjnt* $\mathcal{D}'$

```
    and 𝒟'm: measure lebesgue (⋃𝒟') ≥ measure lebesgue (⋃ ?ℰ) / 3 ^ (DIM('a))
   proof (rule less.hyps)
    have ∗: ?ℰ ⊂ 𝒟
     using ‹D ∈ 𝒟› by auto
    then show card ?ℰ < card 𝒟 finite ?ℰ
     by (auto simp: ‹finite 𝒟› psubset_card_mono)
    show ∃k a b. D = cbox a b ∧ (∀i∈Basis. b • i − a • i = k) if D ∈ ?ℰ for D
     using less.prems(3) that by auto
   qed
   then have [simp]: ⋃𝒟' − D = ⋃𝒟'
    by (auto simp: disjnt_iff)
   show ?thesis
   proof (rule less.prems)
    show insert D 𝒟' ⊆ 𝒟
     using 𝒟'sub ‹D ∈ 𝒟› by blast
    show disjoint (insert D 𝒟')
     using 𝒟'dis 𝒟'sub by (fastforce simp add: pairwise_def disjnt_sym)
    obtain a3 b3 where m3: content (cbox a3 b3) = 3 ^ DIM('a) ∗ measure
lebesgue D
     and sub3: ⋀C. ⟦C ∈ 𝒟; ¬ disjnt C D⟧ ⟹ C ⊆ cbox a3 b3
    proof −
     obtain k a b where ab: D = cbox a b and k: ⋀i. i ∈ Basis ⟹ b•i − a•i
= k
      using less.prems ‹D ∈ 𝒟› by meson
     then have eqk: ⋀i. i ∈ Basis ⟹ a • i ≤ b • i ⟷ k ≥ 0
      by force
     show thesis
     proof
      let ?a = (a + b) /ℝ 2 − (3/2) ∗ℝ (b − a)
      let ?b = (a + b) /ℝ 2 + (3/2) ∗ℝ (b − a)
      have eq: (∏i∈Basis. b • i ∗ 3 − a • i ∗ 3) = (∏i∈Basis. b • i − a • i) ∗
3 ^ DIM('a)
       by (simp add: comm_monoid_mult_class.prod.distrib flip: left_diff_distrib
inner_diff_left)
      show content (cbox ?a ?b) = 3 ^ DIM('a) ∗ measure lebesgue D
       by (simp add: content_cbox_if box_eq_empty algebra_simps eq ab k)
      show C ⊆ cbox ?a ?b if C ∈ 𝒟 and CD: ¬ disjnt C D for C
      proof −
       obtain k' a' b' where ab': C = cbox a' b' and k': ⋀i. i ∈ Basis ⟹
b'•i − a'•i = k'
        using less.prems ‹C ∈ 𝒟› by meson
       then have eqk': ⋀i. i ∈ Basis ⟹ a' • i ≤ b' • i ⟷ k' ≥ 0
        by force
       show ?thesis
        proof (clarsimp simp add: disjoint_interval disjnt_def ab ab' not_less
subset_box algebra_simps)
         show a • i ∗ 2 ≤ a' • i + b • i ∧ a • i + b' • i ≤ b • i ∗ 2
          if ∗ [rule_format]: ∀j∈Basis. a' • j ≤ b' • j and i ∈ Basis for i
         proof −
```

```
            have a' · i ≤ b' · i ∧ a · i ≤ b · i ∧ a · i ≤ b' · i ∧ a' · i ≤ b · i
            using ‹i ∈ Basis› CD by (simp_all add: disjoint_interval disjnt_def
ab ab' not_less)
            then show ?thesis
              using D [OF ‹C ∈ 𝒟›] ‹i ∈ Basis›
              apply (simp add: ab ab' k k' eqk eqk' content_cbox_cases)
              using k k' by fastforce
          qed
        qed
      qed
    qed
    have 𝒟lm: ⋀D. D ∈ 𝒟 ⟹ D ∈ lmeasurable
      using less.prems(3) by blast
    have measure lebesgue (⋃𝒟) ≤ measure lebesgue (cbox a3 b3 ∪ (⋃𝒟 − cbox
a3 b3))
    proof (rule measure_mono_fmeasurable)
      show ⋃𝒟 ∈ sets lebesgue
        using 𝒟lm ‹finite 𝒟› by blast
      show cbox a3 b3 ∪ (⋃𝒟 − cbox a3 b3) ∈ lmeasurable
       by (simp add: 𝒟lm fmeasurable.Un fmeasurable.finite_Union less.prems(2)
subset_eq)
    qed auto
     also have ... = content (cbox a3 b3) + measure lebesgue (⋃𝒟 − cbox a3
b3)
      by (simp add: 𝒟lm fmeasurable.finite_Union less.prems(2) measure_Un2
subsetI)
     also have ... ≤ (measure lebesgue D + measure lebesgue (⋃𝒟')) * 3 ^
DIM('a)
    proof −
      have (⋃𝒟 − cbox a3 b3) ⊆ ⋃ ?ℰ
        using sub3 by fastforce
      then have measure lebesgue (⋃𝒟 − cbox a3 b3) ≤ measure lebesgue (⋃ ?ℰ)
       proof (rule measure_mono_fmeasurable)
        show ⋃ 𝒟 − cbox a3 b3 ∈ sets lebesgue
        by (simp add: 𝒟lm fmeasurableD less.prems(2) sets.Diff sets.finite_Union
subsetI)
        show ⋃ {C ∈ 𝒟 − {D}. disjnt C D} ∈ lmeasurable
          using 𝒟lm less.prems(2) by auto
       qed
       then have measure lebesgue (⋃𝒟 − cbox a3 b3) / 3 ^ DIM('a) ≤ measure
lebesgue (⋃ 𝒟')
        using 𝒟'm by (simp add: field_split_simps)
       then show ?thesis
        by (simp add: m3 field_simps)
    qed
    also have ... ≤ measure lebesgue (⋃(insert D 𝒟')) * 3 ^ DIM('a)
    proof (simp add: 𝒟lm ‹D ∈ 𝒟›)
     show measure lebesgue D + measure lebesgue (⋃𝒟') ≤ measure lebesgue (D
```

$\cup \bigcup \mathcal{D}'$)
proof (*subst measure_Un2*)
show $\bigcup \mathcal{D}' \in$ *lmeasurable*
by (*meson* $\mathcal{D}$*lm* ‹*insert D* $\mathcal{D}' \subseteq \mathcal{D}$› *fmeasurable.finite_Union less.prems*(*2*) *finite_subset subset_eq subset_insertI*)
show *measure lebesgue D + measure lebesgue* $(\bigcup \mathcal{D}') \leq$ *measure lebesgue D + measure lebesgue* $(\bigcup \mathcal{D}' - D)$
using ‹*insert D* $\mathcal{D}' \subseteq \mathcal{D}$› *infinite_super less.prems*(*2*) **by** *force*
qed (*simp add*: $\mathcal{D}$*lm* ‹$D \in \mathcal{D}$›)
qed
finally show *measure lebesgue* $(\bigcup \mathcal{D})$ / *3* ^ *DIM*('*a*) $\leq$ *measure lebesgue* $(\bigcup$(*insert D* $\mathcal{D}'$))
by (*simp add*: *field_split_simps*)
qed
qed
qed

6.26.2 A differentiability-like property of the indefinite integral.

proposition *integrable_ccontinuous_explicit*:
fixes *f* :: '*a*::*euclidean_space* ⇒ '*b*::*euclidean_space*
assumes $\bigwedge$*a b*::'*a*. *f integrable_on cbox a b*
obtains *N* **where**
negligible N
$\bigwedge x\ e.\ [\![x \notin N;\ 0 < e]\!] \Longrightarrow$
$\exists\, d{>}0.\ \forall h.\ 0 < h \wedge h < d \longrightarrow$
norm(*integral* (*cbox x* (*x* + *h* *$_R$ *One*)) *f* /$_R$ *h* ^ *DIM*('*a*) − *f x*) < *e*
proof −
define *BOX* **where** *BOX* ≡ λ*h*. λ*x*::'*a*. *cbox x* (*x* + *h* *$_R$ *One*)
define *BOX2* **where** *BOX2* ≡ λ*h*. λ*x*::'*a*. *cbox* (*x* − *h* *$_R$ *One*) (*x* + *h* *$_R$ *One*)
define *i* **where** *i* ≡ λ*h x*. *integral* (*BOX h x*) *f* /$_R$ *h* ^ *DIM*('*a*)
define Ψ **where** Ψ ≡ λ*x r*. ∀ *d*>*0*. ∃ *h*. *0* < *h* ∧ *h* < *d* ∧ *r* ≤ *norm*(*i h x* − *f x*)
let *?N* = {*x*. ∃ *e*>*0*. Ψ *x e*}
have ∃ *N*. *negligible N* ∧ (∀ *x e*. *x* ∉ *N* ∧ *0* < *e* ⟶ ¬ Ψ *x e*)
proof (*rule exI* ; *intro conjI allI impI*)
let *?M* = $\bigcup n.$ {*x*. Ψ *x* (*inverse*(*real n* + *1*))}
have *negligible* ({*x*. Ψ *x* μ} ∩ *cbox a b*)
if μ > *0* **for** *a b* μ
proof (*cases negligible*(*cbox a b*))
case *True*
then show *?thesis*
by (*simp add*: *negligible_Int*)
next
case *False*
then have *box a b* ≠ {}
by (*simp add*: *negligible_interval*)
then have *ab*: $\bigwedge i.\ i \in$ *Basis* ⟹ *a*•*i* < *b*•*i*

```
      by (simp add: box_ne_empty)
    show ?thesis
      unfolding negligible_outer_le
    proof (intro allI impI)
      fix e::real
      let ?ee = (e * μ) / 2 / 6 ^ (DIM('a))
      assume e > 0
      then have gt0: ?ee > 0
        using ‹μ > 0› by auto
      have f': f integrable_on cbox (a - One) (b + One)
        using assms by blast
      obtain γ where gauge γ
        and γ: ⋀p. ⟦p tagged_partial_division_of (cbox (a - One) (b + One));
γ fine p⟧
                  ⟹ (∑(x, k)∈p. norm (content k *R f x - integral k f)) < ?ee
        using Henstock_lemma [OF f' gt0] that by auto
      let ?E = {x. x ∈ cbox a b ∧ Ψ x μ}
      have ∃h>0. BOX h x ⊆ γ x ∧
                BOX h x ⊆ cbox (a - One) (b + One) ∧ μ ≤ norm (i h x - f x)
        if x ∈ cbox a b Ψ x μ for x
      proof -
        obtain d where d > 0 and d: ball x d ⊆ γ x
          using gaugeD [OF ‹gauge γ›, of x] openE by blast
        then obtain h where 0 < h h < 1 and hless: h < d / real DIM('a)
                        and mule: μ ≤ norm (i h x - f x)
          using ‹Ψ x μ› [unfolded Ψ_def, rule_format, of min 1 (d / DIM('a))]
          by auto
        show ?thesis
        proof (intro exI conjI)
          show 0 < h μ ≤ norm (i h x - f x) by fact+
          have BOX h x ⊆ ball x d
          proof (clarsimp simp: BOX_def mem_box dist_norm algebra_simps)
            fix y
            assume ∀i∈Basis. x · i ≤ y · i ∧ y · i ≤ h + x · i
            then have lt: |(x - y) · i| < d / real DIM('a) if i ∈ Basis for i
              using hless that by (force simp: inner_diff_left)
            have norm (x - y) ≤ (∑i∈Basis. |(x - y) · i|)
              using norm_le_l1 by blast
            also have ... < d
                using sum_bounded_above_strict [of Basis λi. |(x - y) · i| d /
DIM('a), OF lt]
              by auto
            finally show norm (x - y) < d .
          qed
          with d show BOX h x ⊆ γ x
            by blast
          show BOX h x ⊆ cbox (a - One) (b + One)
            using that ‹h < 1›
        qed
      by (force simp: BOX_def mem_box algebra_simps intro: subset_box_imp)
```

```
      qed
    qed
    then obtain η where h0: ⋀x. x ∈ ?E ⟹ η x > 0
      and BOX_γ: ⋀x. x ∈ ?E ⟹ BOX (η x) x ⊆ γ x
      and ⋀x. x ∈ ?E ⟹ BOX (η x) x ⊆ cbox (a − One) (b + One) ∧ μ ≤
norm (i (η x) x − f x)
      by simp metis
    then have BOX_cbox: ⋀x. x ∈ ?E ⟹ BOX (η x) x ⊆ cbox (a − One) (b
+ One)
        and μ_le: ⋀x. x ∈ ?E ⟹ μ ≤ norm (i (η x) x − f x)
      by blast+
    define γ' where γ' ≡ λx. if x ∈ cbox a b ∧ Ψ x μ then ball x (η x) else γ x
    have gauge γ'
      using ‹gauge γ› by (auto simp: h0 gauge_def γ'_def)
    obtain 𝒟 where countable 𝒟
      and 𝒟: ⋃𝒟 ⊆ cbox a b
      ⋀K. K ∈ 𝒟 ⟹ interior K ≠ {} ∧ (∃ c d. K = cbox c d)
      and Dcovered: ⋀K. K ∈ 𝒟 ⟹ ∃x. x ∈ cbox a b ∧ Ψ x μ ∧ x ∈ K ∧ K
⊆ γ' x
      and subUD: ?E ⊆ ⋃𝒟
      by (rule covering_lemma [of ?E a b γ']) (simp_all add: Bex_def ‹box a b
≠ {}› ‹gauge γ'›)
    then have 𝒟 ⊆ sets lebesgue
      by fastforce
  show ∃ T. {x. Ψ x μ} ∩ cbox a b ⊆ T ∧ T ∈ lmeasurable ∧ measure lebesgue
T ≤ e
    proof (intro exI conjI)
      show {x. Ψ x μ} ∩ cbox a b ⊆ ⋃𝒟
        apply auto
        using subUD by auto
      have mUE: measure lebesgue (⋃ ℰ) ≤ measure lebesgue (cbox a b)
        if ℰ ⊆ 𝒟 finite ℰ for ℰ
      proof (rule measure_mono_fmeasurable)
        show ⋃ ℰ ⊆ cbox a b
          using 𝒟(1) that(1) by blast
        show ⋃ ℰ ∈ sets lebesgue
       by (metis 𝒟(2) fmeasurable.finite_Union fmeasurableD lmeasurable_cbox
subset_eq that)
      qed auto
      then show ⋃𝒟 ∈ lmeasurable
          by (metis 𝒟(2) ‹countable 𝒟› fmeasurable_Union_bound lmeasur-
able_cbox)
      then have leab: measure lebesgue (⋃𝒟) ≤ measure lebesgue (cbox a b)
     by (meson 𝒟(1) fmeasurableD lmeasurable_cbox measure_mono_fmeasurable)
      obtain ℱ where ℱ ⊆ 𝒟 finite ℱ
        and ℱ: measure lebesgue (⋃𝒟) ≤ 2 * measure lebesgue (⋃ℱ)
      proof (cases measure lebesgue (⋃𝒟) = 0)
        case True
        then show ?thesis
```

```
        by (force intro: that [where ℱ = {}])
      next
        case False
        obtain ℱ where ℱ ⊆ 𝒟 finite ℱ
          and ℱ: measure lebesgue (⋃𝒟)/2 < measure lebesgue (⋃ℱ)
          proof (rule measure_countable_Union_approachable [of 𝒟 measure
lebesgue (⋃𝒟) / 2 content (cbox a b)])
          show countable 𝒟
            by fact
          show 0 < measure lebesgue (⋃ 𝒟) / 2
            using False by (simp add: zero_less_measure_iff)
          show Dlm: D ∈ lmeasurable if D ∈ 𝒟 for D
            using 𝒟(2) that by blast
          show measure lebesgue (⋃ ℱ) ≤ content (cbox a b)
            if ℱ ⊆ 𝒟 finite ℱ for ℱ
          proof −
            have measure lebesgue (⋃ ℱ) ≤ measure lebesgue (⋃𝒟)
            proof (rule measure_mono_fmeasurable)
              show ⋃ ℱ ⊆ ⋃ 𝒟
                by (simp add: Sup_subset_mono ‹ℱ ⊆ 𝒟›)
              show ⋃ ℱ ∈ sets lebesgue
                by (meson Dlm fmeasurableD sets.finite_Union subset_eq that)
              show ⋃ 𝒟 ∈ lmeasurable
                by fact
            qed
            also have ... ≤ measure lebesgue (cbox a b)
            proof (rule measure_mono_fmeasurable)
              show ⋃ 𝒟 ∈ sets lebesgue
                by (simp add: ‹⋃ 𝒟 ∈ lmeasurable› fmeasurableD)
            qed (auto simp:𝒟(1))
            finally show ?thesis
              by simp
          qed
        qed auto
        then show ?thesis
          using that by auto
      qed
      obtain tag where tag_in_E: ⋀D. D ∈ 𝒟 ⟹ tag D ∈ ?E
        and tag_in_self: ⋀D. D ∈ 𝒟 ⟹ tag D ∈ D
        and tag_sub: ⋀D. D ∈ 𝒟 ⟹ D ⊆ γ' (tag D)
        using Dcovered by simp metis
      then have sub_ball_tag: ⋀D. D ∈ 𝒟 ⟹ D ⊆ ball (tag D) (η (tag D))
        by (simp add: γ'_def)
      define Φ where Φ ≡ λD. BOX (η(tag D)) (tag D)
      define Φ2 where Φ2 ≡ λD. BOX2 (η(tag D)) (tag D)
      obtain 𝒞 where 𝒞 ⊆ Φ2 ‘ ℱ pairwise disjnt 𝒞
        measure lebesgue (⋃𝒞) ≥ measure lebesgue (⋃(Φ2‘ℱ)) / 3 ^ (DIM('a))
      proof (rule Austin_Lemma)
        show finite (Φ2‘ℱ)
```

```
      using ‹finite ℱ› by blast
    have ∃k a b. Φ2 D = cbox a b ∧ (∀i∈Basis. b • i − a • i = k) if D ∈ ℱ
for D
      apply (rule_tac x=2 * η(tag D) in exI)
      apply (rule_tac x=tag D − η(tag D) *R One in exI)
      apply (rule_tac x=tag D + η(tag D) *R One in exI)
      using that
      apply (auto simp: Φ2_def BOX2_def algebra_simps)
      done
    then show ⋀D. D ∈ Φ2 ‘ ℱ ⟹ ∃k a b. D = cbox a b ∧ (∀i∈Basis. b
• i − a • i = k)
      by blast
  qed auto
  then obtain 𝒢 where 𝒢 ⊆ ℱ and disj: pairwise disjnt (Φ2 ‘ 𝒢)
    and measure lebesgue (⋃(Φ2 ‘ 𝒢)) ≥ measure lebesgue (⋃(Φ2‘ℱ)) / 3
^ (DIM('a))
    unfolding Φ2_def subset_image_iff
    by (meson empty_subsetI equals0D pairwise_imageI)
  moreover
  have measure lebesgue (⋃(Φ2 ‘ 𝒢)) * 3 ^ DIM('a) ≤ e/2
  proof −
    have finite 𝒢
      using ‹finite ℱ› ‹𝒢 ⊆ ℱ› infinite_super by blast
    have BOX2_m: ⋀x. x ∈ tag ‘ 𝒢 ⟹ BOX2 (η x) x ∈ lmeasurable
      by (auto simp: BOX2_def)
    have BOX_m: ⋀x. x ∈ tag ‘ 𝒢 ⟹ BOX (η x) x ∈ lmeasurable
      by (auto simp: BOX_def)
    have BOX_sub: BOX (η x) x ⊆ BOX2 (η x) x for x
      by (auto simp: BOX_def BOX2_def subset_box algebra_simps)
    have DISJ2: BOX2 (η (tag X)) (tag X) ∩ BOX2 (η (tag Y)) (tag Y)
= {}
      if X ∈ 𝒢 Y ∈ 𝒢 tag X ≠ tag Y for X Y
    proof −
      obtain i where i: i ∈ Basis tag X • i ≠ tag Y • i
        using ‹tag X ≠ tag Y› by (auto simp: euclidean_eq_iff [of tag X])
      have XY: X ∈ 𝒟 Y ∈ 𝒟
        using ‹ℱ ⊆ 𝒟› ‹𝒢 ⊆ ℱ› that by auto
      then have 0 ≤ η (tag X) 0 ≤ η (tag Y)
        by (meson h0 le_cases not_le tag_in_E)+
      with XY i have BOX2 (η (tag X)) (tag X) ≠ BOX2 (η (tag Y)) (tag
Y)
        unfolding eq_iff
        by (fastforce simp add: BOX2_def subset_box algebra_simps)
      then show ?thesis
        using disj that by (auto simp: pairwise_def disjnt_def Φ2_def)
    qed
  then have BOX2_disj: pairwise (λx y. negligible (BOX2 (η x) x ∩ BOX2
(η y) y)) (tag ‘ 𝒢)
    by (simp add: pairwise_imageI)
```

then have *BOX_disj*: *pairwise* (λ*x y*. *negligible* (*BOX* (η *x*) *x* ∩ *BOX* (η *y*) *y*)) (*tag* ‘ *G*)
proof (*rule pairwise_mono*)
show *negligible* (*BOX* (η *x*) *x* ∩ *BOX* (η *y*) *y*)
if *negligible* (*BOX2* (η *x*) *x* ∩ *BOX2* (η *y*) *y*) **for** *x y*
by (*metis* (*no_types*, *opaque_lifting*) *that Int_mono negligible_subset BOX_sub*)
qed *auto*

have *eq*: ⋀*box*. (λ*D*. *box* (η (*tag D*)) (*tag D*)) ‘ *G* = (λ*t*. *box* (η *t*) *t*) ‘ *tag* ‘ *G*
by (*simp add*: *image_comp*)
have *measure lebesgue* (*BOX2* (η *t*) *t*) ∗ *3* ^ *DIM*(′*a*)
= *measure lebesgue* (*BOX* (η *t*) *t*) ∗ (*2*∗*3*) ^ *DIM*(′*a*)
if *t* ∈ *tag* ‘ *G* **for** *t*
proof −
have *content* (*cbox* (*t* − η *t* ∗*R One*) (*t* + η *t* ∗*R One*))
= *content* (*cbox t* (*t* + η *t* ∗*R One*)) ∗ *2* ^ *DIM*(′*a*)
using *that* **by** (*simp add*: *algebra_simps content_cbox_if box_eq_empty*)
then show *?thesis*
by (*simp add*: *BOX2_def BOX_def flip*: *power_mult_distrib*)
qed
then have *measure lebesgue* (⋃(Φ*2* ‘ *G*)) ∗ *3* ^ *DIM*(′*a*) = *measure lebesgue* (⋃(Φ ‘ *G*)) ∗ *6* ^ *DIM*(′*a*)
unfolding Φ*_def* Φ*2_def eq*
by (*simp add*: *measure_negligible_finite_Union_image*
‹*finite G*› *BOX2_m BOX_m BOX2_disj BOX_disj sum_distrib_right*
del: *UN_simps*)
also have ... ≤ *e*/*2*
proof −
have μ ∗ *measure lebesgue* (⋃*D*∈*G*. Φ *D*) ≤ μ ∗ (∑ *D* ∈ Φ ‘ *G*. *measure lebesgue D*)
using ‹μ > *0*› ‹*finite G*› **by** (*force simp*: *BOX_m* Φ*_def fmeasurableD intro*: *measure_Union_le*)
also have ... = (∑ *D* ∈ Φ ‘ *G*. *measure lebesgue D* ∗ μ)
by (*metis mult.commute sum_distrib_right*)
also have ... ≤ (∑ (*x*, *K*) ∈ (λ*D*. (*tag D*, Φ *D*)) ‘ *G*. *norm* (*content K* ∗*R f x* − *integral K f*))
proof (*rule sum_le_included*; *clarify?*)
fix *D*
assume *D* ∈ *G*
then have η (*tag D*) > *0*
using ‹*F* ⊆ *D*› ‹*G* ⊆ *F*› *h0 tag_in_E* **by** *auto*
then have *m_*Φ: *measure lebesgue* (Φ *D*) > *0*
by (*simp add*: Φ*_def BOX_def algebra_simps*)
have μ ≤ *norm* (*i* (η(*tag D*)) (*tag D*) − *f*(*tag D*))
using μ*_le* ‹*D* ∈ *G*› ‹*F* ⊆ *D*› ‹*G* ⊆ *F*› *tag_in_E* **by** *auto*
also have ... = *norm* ((*content* (Φ *D*) ∗*R f*(*tag D*) − *integral* (Φ *D*) *f*) /*R measure lebesgue* (Φ *D*))

```
            using m_Φ
            unfolding i_def Φ_def BOX_def
          by (simp add: algebra_simps content_cbox_plus norm_minus_commute)
           finally have measure lebesgue (Φ D) * μ ≤ norm (content (Φ D) *R
f(tag D) − integral (Φ D) f)
            using m_Φ by simp (simp add: field_simps)
           then show ∃ y∈(λD. (tag D, Φ D)) ‘ G.
                snd y = Φ D ∧ measure lebesgue (Φ D) * μ ≤ (case y of (x, k)
⇒ norm (content k *R f x − integral k f))
            using ‹D ∈ G› by auto
          qed (use ‹finite G› in auto)
          also have ... < ?ee
          proof (rule γ)
           show (λD. (tag D, Φ D)) ‘ G tagged_partial_division_of cbox (a −
One) (b + One)
            unfolding tagged_partial_division_of_def
           proof (intro conjI allI impI ; clarify ?)
            show tag D ∈ Φ D
             if D ∈ G for D
             using that ‹F ⊆ D› ‹G ⊆ F› h0 tag_in_E
                  by (auto simp: Φ_def BOX_def mem_box algebra_simps
eucl_less_le_not_le in_mono)
            show y ∈ cbox (a − One) (b + One) if D ∈ G y ∈ Φ D for D y
              using that BOX_cbox Φ_def ‹F ⊆ D› ‹G ⊆ F› tag_in_E by
blast
            show tag D = tag E ∧ Φ D = Φ E
              if D ∈ G E ∈ G and ne: interior (Φ D) ∩ interior (Φ E) ≠ {}
for D E
            proof −
             have BOX2 (η (tag D)) (tag D) ∩ BOX2 (η (tag E)) (tag E) =
{} ∨ tag E = tag D
              using DISJ2 ‹D ∈ G› ‹E ∈ G› by force
             then have BOX (η (tag D)) (tag D) ∩ BOX (η (tag E)) (tag E)
= {} ∨ tag E = tag D
              using BOX_sub by blast
             then show tag D = tag E ∧ Φ D = Φ E
              by (metis Φ_def interior_Int interior_empty ne)
            qed
           qed (use ‹finite G› Φ_def BOX_def in auto)
           show γ fine (λD. (tag D, Φ D)) ‘ G
              unfolding fine_def Φ_def using BOX_γ ‹F ⊆ D› ‹G ⊆ F›
tag_in_E by blast
          qed
          finally show ?thesis
           using ‹μ > 0› by (auto simp: field_split_simps)
        qed
         finally show ?thesis .
        qed
        moreover
```

```
        have measure lebesgue (⋃ℱ) ≤ measure lebesgue (⋃(Φ2`ℱ))
        proof (rule measure_mono_fmeasurable)
          have D ⊆ ball (tag D) (η(tag D)) if D ∈ ℱ for D
            using ‹ℱ ⊆ 𝒟› sub_ball_tag that by blast
          moreover have ball (tag D) (η(tag D)) ⊆ BOX2 (η (tag D)) (tag D) if
D ∈ ℱ for D
            proof (clarsimp simp: Φ2_def BOX2_def mem_box algebra_simps
dist_norm)
              fix x and i::'a
              assume norm (tag D − x) < η (tag D) and i ∈ Basis
              then have |tag D • i − x • i| ≤ η (tag D)
                  by (metis eucl_less_le_not_le inner_commute inner_diff_right
norm_bound_Basis_le)
              then show tag D • i ≤ x • i + η (tag D) ∧ x • i ≤ η (tag D) + tag D
• i
                by (simp add: abs_diff_le_iff)
            qed
            ultimately show ⋃ℱ ⊆ ⋃(Φ2`ℱ)
              by (force simp: Φ2_def)
            show ⋃ℱ ∈ sets lebesgue
              using ‹finite ℱ› ‹𝒟 ⊆ sets lebesgue› ‹ℱ ⊆ 𝒟› by blast
            show ⋃(Φ2`ℱ) ∈ lmeasurable
              unfolding Φ2_def BOX2_def using ‹finite ℱ› by blast
          qed
          ultimately
          have measure lebesgue (⋃ℱ) ≤ e/2
            by (auto simp: field_split_simps)
          then show measure lebesgue (⋃𝒟) ≤ e
            using ℱ by linarith
        qed
      qed
    qed
    then have ⋀j. negligible {x. Ψ x (inverse(real j + 1))}
      using negligible_on_intervals
       by (metis (full_types) inverse_positive_iff_positive le_add_same_cancel1
linorder_not_le nat_le_real_less not_add_less1 of_nat_0)
    then have negligible ?M
      by auto
    moreover have ?N ⊆ ?M
    proof (clarsimp simp: dist_norm)
      fix y e
      assume 0 < e
        and ye [rule_format]: Ψ y e
      then obtain k where k: 0 < k inverse (real k + 1) < e
       by (metis One_nat_def add.commute less_add_same_cancel2 less_imp_inverse_less
less_trans neq0_conv of_nat_1 of_nat_Suc reals_Archimedean zero_less_one)
      with ye show ∃n. Ψ y (inverse (real n + 1))
        apply (rule_tac x=k in exI)
        unfolding Ψ_def
```

```
      by (force intro: less_le_trans)
    qed
    ultimately show negligible ?N
      by (blast intro: negligible_subset)
    show ¬ Ψ x e if x ∉ ?N ∧ 0 < e for x e
      using that by blast
  qed
  with that show ?thesis
    unfolding i_def BOX_def Ψ_def by (fastforce simp add: not_le)
qed
```

6.26.3 HOL Light measurability

```
definition measurable_on :: ('a::euclidean_space ⇒ 'b::real_normed_vector) ⇒
'a set ⇒ bool
  (infixr measurable'_on 46)
  where f measurable_on S ≡
        ∃N g. negligible N ∧
              (∀n. continuous_on UNIV (g n)) ∧
              (∀x. x ∉ N ⟶ (λn. g n x) ⟶⟶ (if x ∈ S then f x else 0))

lemma measurable_on_UNIV:
  (λx.  if x ∈ S then f x else 0) measurable_on UNIV ⟷ f measurable_on S
  by (auto simp: measurable_on_def)

lemma measurable_on_spike_set:
  assumes f: f measurable_on S and neg: negligible ((S − T) ∪ (T − S))
  shows f measurable_on T
proof −
  obtain N and F
    where N: negligible N
      and conF: ⋀n. continuous_on UNIV (F n)
      and tendsF: ⋀x. x ∉ N ⟹ (λn. F n x) ⟶⟶ (if x ∈ S then f x else 0)
    using f by (auto simp: measurable_on_def)
  show ?thesis
    unfolding measurable_on_def
  proof (intro exI conjI allI impI)
    show continuous_on UNIV (λx. F n x) for n
      by (intro conF continuous_intros)
    show negligible (N ∪ (S − T) ∪ (T − S))
      by (metis (full_types) N neg negligible_Un_eq)
    show (λn. F n x) ⟶⟶ (if x ∈ T then f x else 0)
      if x ∉ (N ∪ (S − T) ∪ (T − S)) for x
      using that tendsF [of x] by auto
  qed
qed
```

Various common equivalent forms of function measurability.

```
lemma measurable_on_0 [simp]: (λx. 0) measurable_on S
```

```
  unfolding measurable_on_def
proof (intro exI conjI allI impI)
  show (λn. 0) ⟶ (if x ∈ S then 0::'b else 0) for x
    by force
qed auto

lemma measurable_on_scaleR_const:
  assumes f: f measurable_on S
  shows (λx. c *R f x) measurable_on S
proof −
  obtain NF and F
    where NF: negligible NF
      and conF: ⋀n. continuous_on UNIV (F n)
      and tendsF: ⋀x. x ∉ NF ⟹ (λn. F n x) ⟶ (if x ∈ S then f x else 0)
    using f by (auto simp: measurable_on_def)
  show ?thesis
    unfolding measurable_on_def
  proof (intro exI conjI allI impI)
    show continuous_on UNIV (λx. c *R F n x) for n
      by (intro conF continuous_intros)
    show (λn. c *R F n x) ⟶ (if x ∈ S then c *R f x else 0)
      if x ∉ NF for x
      using tendsto_scaleR [OF tendsto_const tendsF, of x] that by auto
  qed (auto simp: NF)
qed


lemma measurable_on_cmul:
  fixes c :: real
  assumes f measurable_on S
  shows (λx. c * f x) measurable_on S
  using measurable_on_scaleR_const [OF assms] by simp

lemma measurable_on_cdivide:
  fixes c :: real
  assumes f measurable_on S
  shows (λx. f x / c) measurable_on S
proof (cases c=0)
  case False
  then show ?thesis
    using measurable_on_cmul [of f S 1/c]
    by (simp add: assms)
qed auto


lemma measurable_on_minus:
   f measurable_on S ⟹ (λx. −(f x)) measurable_on S
  using measurable_on_scaleR_const [of f S −1] by auto
```

lemma *continuous_imp_measurable_on*:
  continuous_on UNIV f $\Longrightarrow$ *f measurable_on UNIV*
 unfolding *measurable_on_def*
 apply (*rule_tac x*={} **in** *exI*)
 apply (*rule_tac x*=$\lambda n.$ *f* **in** *exI*, *auto*)
 done

proposition *integrable_subintervals_imp_measurable*:
 fixes *f* :: *'a*::*euclidean_space* $\Rightarrow$ *'b*::*euclidean_space*
 assumes $\bigwedge a\ b.$ *f integrable_on cbox a b*
 shows *f measurable_on UNIV*
proof −
 define *BOX* **where** *BOX* $\equiv \lambda h.\ \lambda x$::*'a. cbox x* (*x* + *h* $*_R$ *One*)
 define *i* **where** *i* $\equiv \lambda h\ x.$ *integral* (*BOX h x*) *f* $/_R$ *h* $\hat{}$ *DIM*(*'a*)
 obtain *N* **where** *negligible N*
   and *k*: $\bigwedge x\ e.$ $[\![x \notin N;\ 0 < e]\!]$
        $\Longrightarrow \exists\, d{>}0.\ \forall h.\ 0 < h \wedge h < d \longrightarrow$
            norm (*integral* (*cbox x* (*x* + *h* $*_R$ *One*)) *f* $/_R$ *h* $\hat{}$ *DIM*(*'a*) − *f x*) < *e*
   using *integrable_ccontinuous_explicit assms* **by** *blast*
 show *?thesis*
   unfolding *measurable_on_def*
 proof (*intro exI conjI allI impI*)
   show *continuous_on UNIV* ((*λn x. i* (*inverse*(*Suc n*)) *x*) *n*) **for** *n*
   proof (*clarsimp simp*: *continuous_on_iff*)
     show $\exists\, d{>}0.\ \forall x'.$ *dist x' x* < *d* $\longrightarrow$ *dist* (*i* (*inverse* (*1* + *real n*)) *x'*) (*i* (*inverse* (*1* + *real n*)) *x*) < *e*
      if *0* < *e*
      for *x e*
     proof −
      let *?e* = *e* / (*1* + *real n*) $\hat{}$ *DIM*(*'a*)
      have *?e* > *0*
       using ‹*e* > *0*› **by** *auto*
      moreover have *x* ∈ *cbox* (*x* − *2* $*_R$ *One*) (*x* + *2* $*_R$ *One*)
       by (*simp add*: *mem_box inner_diff_left inner_left_distrib*)
      moreover have *x* + *One* $/_R$ *real* (*Suc n*) ∈ *cbox* (*x* − *2* $*_R$ *One*) (*x* + *2* $*_R$ *One*)
       by (*auto simp*: *mem_box inner_diff_left inner_left_distrib field_simps*)
      ultimately obtain *δ* **where** *δ* > *0*
       and *δ*: $\bigwedge c'\ d'.$ $[\![c' \in$ *cbox* (*x* − *2* $*_R$ *One*) (*x* + *2* $*_R$ *One*);
                  d' ∈ *cbox* (*x* − *2* $*_R$ *One*) (*x* + *2* $*_R$ *One*);
                  norm(*c'* − *x*) ≤ *δ*; *norm*(*d'* − (*x* + *One* $/_R$ *Suc n*)) ≤ *δ*$]\!]$
                  $\Longrightarrow$ *norm*(*integral*(*cbox c' d'*) *f* − *integral*(*cbox x* (*x* + *One* $/_R$ *Suc n*)) *f*) < *?e*
       by (*blast intro*: *indefinite_integral_continuous* [*of f _ _ x*] *assms*)
      show *?thesis*
      proof (*intro exI impI conjI allI*)
       show *min δ 1* > *0*

```
          using ‹δ > 0› by auto
        show dist (i (inverse (1 + real n)) y) (i (inverse (1 + real n)) x) < e
          if dist y x < min δ 1 for y
        proof −
          have no: norm (y − x) < 1
            using that by (auto simp: dist_norm)
          have le1: inverse (1 + real n) ≤ 1
            by (auto simp: field_split_simps)
          have norm (integral (cbox y (y + One /R real (Suc n))) f
              − integral (cbox x (x + One /R real (Suc n))) f)
              < e / (1 + real n) ^ DIM('a)
          proof (rule δ)
            show y ∈ cbox (x − 2 *R One) (x + 2 *R One)
           using no by (auto simp: mem_box algebra_simps dest: Basis_le_norm
[of _ y−x])
            show y + One /R real (Suc n) ∈ cbox (x − 2 *R One) (x + 2 *R One)
            proof (simp add: dist_norm mem_box algebra_simps, intro ballI conjI)
              fix i::'a
              assume i ∈ Basis
              then have 1: |y · i − x · i| < 1
             by (metis inner_commute inner_diff_right no norm_bound_Basis_lt)
               moreover have ... < (2 + inverse (1 + real n)) 1 ≤ 2 − inverse
(1 + real n)
                 by (auto simp: field_simps)
              ultimately show x · i ≤ y · i + (2 + inverse (1 + real n))
                                y · i + inverse (1 + real n) ≤ x · i + 2
                by linarith+
            qed
            show norm (y − x) ≤ δ norm (y + One /R real (Suc n) − (x + One
/R real (Suc n))) ≤ δ
              using that by (auto simp: dist_norm)
          qed
          then show ?thesis
        using that by (simp add: dist_norm i_def BOX_def flip: scaleR_diff_right)
(simp add: field_simps)
        qed
      qed
    qed
  qed
  show negligible N
    by (simp add: ‹negligible N›)
  show (λn. i (inverse (Suc n)) x) ⟶ (if x ∈ UNIV then f x else 0)
    if x ∉ N for x
    unfolding lim_sequentially
  proof clarsimp
    show ∃no. ∀n≥no. dist (i (inverse (1 + real n)) x) (f x) < e
      if 0 < e for e
    proof −
      obtain d where d > 0
```

```
        and d: ⋀h. ⟦0 < h; h < d⟧ ⟹
            norm (integral (cbox x (x + h *R One)) f /R h ^ DIM('a) - f x) < e
        using k [of x e] ‹x ∉ N› ‹0 < e› by blast
      then obtain M where M: M ≠ 0 0 < inverse (real M) inverse (real M)
< d
        using real_arch_invD by auto
      show ?thesis
      proof (intro exI allI impI)
        show dist (i (inverse (1 + real n)) x) (f x) < e
          if M ≤ n for n
        proof -
          have *: 0 < inverse (1 + real n) inverse (1 + real n) ≤ inverse M
            using that ‹M ≠ 0› by auto
          show ?thesis
            using that M
            apply (simp add: i_def BOX_def dist_norm)
            apply (blast intro: le_less_trans * d)
            done
        qed
      qed
    qed
  qed
qed
```

6.26.4 Composing continuous and measurable functions; a few variants

```
lemma measurable_on_compose_continuous:
  assumes f: f measurable_on UNIV and g: continuous_on UNIV g
  shows (g ∘ f) measurable_on UNIV
proof -
  obtain N and F
    where negligible N
      and conF: ⋀n. continuous_on UNIV (F n)
      and tendsF: ⋀x. x ∉ N ⟹ (λn. F n x) ⟶ f x
    using f by (auto simp: measurable_on_def)
  show ?thesis
    unfolding measurable_on_def
  proof (intro exI conjI allI impI)
    show negligible N
      by fact
    show continuous_on UNIV (g ∘ (F n)) for n
      using conF continuous_on_compose continuous_on_subset g by blast
    show (λn. (g ∘ F n) x) ⟶ (if x ∈ UNIV then (g ∘ f) x else 0)
      if x ∉ N for x :: 'a
    using that g tendsF by (auto simp: continuous_on_def intro: tendsto_compose)
  qed
qed
```

lemma *measurable_on_compose_continuous_0*:
assumes *f*: *f measurable_on S* **and** *g*: *continuous_on UNIV g* **and** *g 0 = 0*
shows *(g ∘ f) measurable_on S*
proof −
have *f'*: *(λx. if x ∈ S then f x else 0) measurable_on UNIV*
using *f measurable_on_UNIV* **by** *blast*
show *?thesis*
using *measurable_on_compose_continuous* [*OF f' g*]
by (*simp add*: *measurable_on_UNIV o_def if_distrib* ‹*g 0 = 0*› *cong*: *if_cong*)
qed

lemma *measurable_on_compose_continuous_box*:
assumes *fm*: *f measurable_on UNIV* **and** *fab*: ⋀*x*. *f x ∈ box a b*
and *contg*: *continuous_on (box a b) g*
shows *(g ∘ f) measurable_on UNIV*
proof −
have ∃*γ*. (∀ *n*. *continuous_on UNIV* (*γ n*)) ∧ (∀ *x*. *x ∉ N* ⟶ (*λn*. *γ n x*) ⟶ *g* (*f x*))
if *negligible N*
and *conth* [*rule_format*]: ∀ *n*. *continuous_on UNIV* (*λx*. *h n x*)
and *tends* [*rule_format*]: ∀ *x*. *x ∉ N* ⟶ (*λn*. *h n x*) ⟶ *f x*
for *N* **and** *h* :: *nat* ⇒ *'a* ⇒ *'b*
proof −
define *ϑ* **where** *ϑ* ≡ *λn x*. (∑ *i*∈*Basis*. (*max* (*a*•*i* + (*b*•*i* − *a*•*i*) / *real* (*n*+2))
(*min* ((*h n x*)•*i*)
(*b*•*i* − (*b*•*i* − *a*•*i*) / *real* (*n*+2)))) ∗$_R$ *i*)
have *aibi*: ⋀*i*. *i* ∈ *Basis* ⟹ *a* • *i* < *b* • *i*
using *box_ne_empty*(2) *fab* **by** *auto*
then have ∗: ⋀*i n*. *i* ∈ *Basis* ⟹ *a* • *i* + *real n* ∗ (*a* • *i*) < *b* • *i* + *real n* ∗ (*b* • *i*)
by (*meson add_mono_thms_linordered_field*(3) *less_eq_real_def mult_left_mono of_nat_0_le_iff*)
show *?thesis*
proof (*intro exI conjI allI impI*)
show *continuous_on UNIV* (*g* ∘ (*ϑ n*)) **for** *n* :: *nat*
unfolding *ϑ_def*
apply (*intro continuous_on_compose2* [*OF contg*] *continuous_intros conth*)
apply (*auto simp*: *aibi* ∗ *mem_box less_max_iff_disj min_less_iff_disj field_split_simps*)
done
show (*λn*. (*g* ∘ *ϑ n*) *x*) ⟶ *g* (*f x*)
if *x ∉ N* **for** *x*
unfolding *o_def*
proof (*rule isCont_tendsto_compose* [**where** *g*=*g*])
show *isCont g* (*f x*)
using *contg fab continuous_on_eq_continuous_at* **by** *blast*
have (*λn*. *ϑ n x*) ⟶ (∑ *i*∈*Basis*. *max* (*a* • *i*) (*min* (*f x* • *i*) (*b* • *i*)) ∗$_R$

```
i)
        unfolding ϑ_def
      proof (intro tendsto_intros ‹x ∉ N› tends)
        fix i::'b
        assume i ∈ Basis
        have a: (λn. a • i + (b • i − a • i) / real n) ⟶ a•i + 0
          by (intro tendsto_add lim_const_over_n tendsto_const)
        show (λn. a • i + (b • i − a • i) / real (n + 2)) ⟶ a • i
          using LIMSEQ_ignore_initial_segment [where k=2, OF a] by simp
        have b: (λn. b•i − (b • i − a • i) / (real n)) ⟶ b•i − 0
          by (intro tendsto_diff lim_const_over_n tendsto_const)
        show (λn. b • i − (b • i − a • i) / real (n + 2)) ⟶ b • i
          using LIMSEQ_ignore_initial_segment [where k=2, OF b] by simp
      qed
    also have (∑ i∈Basis. max (a • i) (min (f x • i) (b • i)) *R i) = (∑ i∈Basis.
(f x • i) *R i)
        using fab by (auto simp add: mem_box intro: sum.cong)
      also have ... = f x
        using euclidean_representation by blast
      finally show (λn. ϑ n x) ⟶ f x .
    qed
  qed
  then show ?thesis
    using fm by (auto simp: measurable_on_def)
qed

lemma measurable_on_Pair:
  assumes f: f measurable_on S and g: g measurable_on S
  shows (λx. (f x, g x)) measurable_on S
proof −
  obtain NF and F
    where NF: negligible NF
      and conF: ⋀n. continuous_on UNIV (F n)
      and tendsF: ⋀x. x ∉ NF ⟹ (λn. F n x) ⟶ (if x ∈ S then f x else 0)
    using f by (auto simp: measurable_on_def)
  obtain NG and G
    where NG: negligible NG
      and conG: ⋀n. continuous_on UNIV (G n)
      and tendsG: ⋀x. x ∉ NG ⟹ (λn. G n x) ⟶ (if x ∈ S then g x else 0)
    using g by (auto simp: measurable_on_def)
  show ?thesis
    unfolding measurable_on_def
  proof (intro exI conjI allI impI)
    show negligible (NF ∪ NG)
      by (simp add: NF NG)
    show continuous_on UNIV (λx. (F n x, G n x)) for n
      using conF conG continuous_on_Pair by blast
    show (λn. (F n x, G n x)) ⟶ (if x ∈ S then (f x, g x) else 0)
```

```
    if x ∉ NF ∪ NG for x
   using tendsto_Pair [OF tendsF tendsG, of x x] that unfolding zero_prod_def
    by (simp add: split: if_split_asm)
  qed
qed

lemma measurable_on_combine:
  assumes f: f measurable_on S and g: g measurable_on S
    and h: continuous_on UNIV (λx. h (fst x) (snd x)) and h 0 0 = 0
  shows (λx. h (f x) (g x)) measurable_on S
proof −
  have ∗: (λx. h (f x) (g x)) = (λx. h (fst x) (snd x)) ∘ (λx. (f x, g x))
    by auto
  show ?thesis
    unfolding ∗ by (auto simp: measurable_on_compose_continuous_0 measur-
able_on_Pair assms)
qed

lemma measurable_on_add:
  assumes f: f measurable_on S and g: g measurable_on S
  shows (λx. f x + g x) measurable_on S
  by (intro continuous_intros measurable_on_combine [OF assms]) auto

lemma measurable_on_diff:
  assumes f: f measurable_on S and g: g measurable_on S
  shows (λx. f x − g x) measurable_on S
  by (intro continuous_intros measurable_on_combine [OF assms]) auto

lemma measurable_on_scaleR:
  assumes f: f measurable_on S and g: g measurable_on S
  shows (λx. f x ∗R g x) measurable_on S
  by (intro continuous_intros measurable_on_combine [OF assms]) auto

lemma measurable_on_sum:
  assumes finite I ⋀i. i ∈ I ⟹ f i measurable_on S
  shows (λx. sum (λi. f i x) I) measurable_on S
  using assms by (induction I) (auto simp: measurable_on_add)

lemma measurable_on_spike:
  assumes f: f measurable_on T and negligible S and gf: ⋀x. x ∈ T − S ⟹ g
x = f x
  shows g measurable_on T
proof −
  obtain NF and F
    where NF: negligible NF
      and conF: ⋀n. continuous_on UNIV (F n)
      and tendsF: ⋀x. x ∉ NF ⟹ (λn. F n x) ⟶ (if x ∈ T then f x else 0)
    using f by (auto simp: measurable_on_def)
  show ?thesis
```

```
    unfolding measurable_on_def
  proof (intro exI conjI allI impI)
    show negligible (NF ∪ S)
      by (simp add: NF ‹negligible S›)
    show ⋀x. x ∉ NF ∪ S ⟹ (λn. F n x) ⟶ (if x ∈ T then g x else 0)
      by (metis (full_types) Diff_iff Un_iff gf tendsF)
  qed (auto simp: conF)
qed

proposition indicator_measurable_on:
  assumes S ∈ sets lebesgue
  shows indicat_real S measurable_on UNIV
proof −
  { fix n::nat
    let ?ε = (1::real) / (2 * 2^n)
    have ε: ?ε > 0
      by auto
     obtain T where closed T T ⊆ S S−T ∈ lmeasurable and ST: emeasure
lebesgue (S − T) < ?ε
      by (meson ε assms sets_lebesgue_inner_closed)
    obtain U where open U S ⊆ U (U − S) ∈ lmeasurable and US: emeasure
lebesgue (U − S) < ?ε
      by (meson ε assms sets_lebesgue_outer_open)
    have eq: −T ∩ U = (S−T) ∪ (U − S)
      using ‹T ⊆ S› ‹S ⊆ U› by auto
    have emeasure lebesgue ((S−T) ∪ (U − S)) ≤ emeasure lebesgue (S − T) +
emeasure lebesgue (U − S)
      using ‹S − T ∈ lmeasurable› ‹U − S ∈ lmeasurable› emeasure_subadditive
by blast
    also have ... < ?ε + ?ε
      using ST US add_mono_ennreal by metis
    finally have le: emeasure lebesgue (−T ∩ U) < ennreal (1 / 2^n)
      by (simp add: eq)
    have 1: continuous_on (T ∪ −U) (indicat_real S)
      unfolding indicator_def of_bool_def
    proof (rule continuous_on_cases [OF ‹closed T›])
      show closed (− U)
        using ‹open U› by blast
      show continuous_on T (λx. 1::real) continuous_on (− U) (λx. 0::real)
        by (auto simp: continuous_on)
      show ∀x. x ∈ T ∧ x ∉ S ∨ x ∈ − U ∧ x ∈ S ⟶ (1::real) = 0
        using ‹T ⊆ S› ‹S ⊆ U› by auto
    qed
    have 2: closedin (top_of_set UNIV) (T ∪ −U)
      using ‹closed T› ‹open U› by auto
     obtain g where continuous_on UNIV g ⋀x. x ∈ T ∪ −U ⟹ g x = indi-
cat_real S x ⋀x. norm(g x) ≤ 1
      by (rule Tietze [OF 1 2, of 1]) auto
    with le have ∃g E. continuous_on UNIV g ∧ (∀x ∈ −E. g x = indicat_real
```

```
S x) ∧
                    (∀x. norm(g x) ≤ 1) ∧ E ∈ sets lebesgue ∧ emeasure lebesgue
E < ennreal (1 / 2^n)
      apply (rule_tac x=g in exI)
      apply (rule_tac x=-T ∩ U in exI)
      using ‹S - T ∈ lmeasurable› ‹U - S ∈ lmeasurable› eq by auto
  }
  then obtain g E where cont: ⋀n. continuous_on UNIV (g n)
    and geq: ⋀n x. x ∈ - E n ⟹ g n x = indicat_real S x
    and ng1: ⋀n x. norm(g n x) ≤ 1
    and Eset: ⋀n. E n ∈ sets lebesgue
    and Em: ⋀n. emeasure lebesgue (E n) < ennreal (1 / 2^n)
    by metis
  have null: limsup E ∈ null_sets lebesgue
  proof (rule borel_cantelli_limsup1 [OF Eset])
    show emeasure lebesgue (E n) < ∞ for n
      by (metis Em infinity_ennreal_def order.asym top.not_eq_extremum)
    show summable (λn. measure lebesgue (E n))
    proof (rule summable_comparison_test' [OF summable_geometric, of 1/2 0])
      show norm (measure lebesgue (E n)) ≤ (1/2) ^ n  for n
       using Em [of n] by (simp add: measure_def enn2real_leI power_one_over)
    qed auto
  qed
  have tends: (λn. g n x) ⟶ indicat_real S x if x ∉ limsup E for x
  proof -
    have ∀F n in sequentially. x ∈ - E n
      using that by (simp add: mem_limsup_iff not_frequently)
    then show ?thesis
      unfolding tendsto_iff dist_real_def
      by (simp add: eventually_mono geq)
  qed
  show ?thesis
    unfolding measurable_on_def
  proof (intro exI conjI allI impI)
    show negligible (limsup E)
      using negligible_iff_null_sets null by blast
    show continuous_on UNIV (g n) for n
      using cont by blast
  qed (use tends in auto)
qed

lemma measurable_on_restrict:
  assumes f: f measurable_on UNIV and S: S ∈ sets lebesgue
  shows (λx. if x ∈ S then f x else 0) measurable_on UNIV
proof -
  have indicat_real S measurable_on UNIV
    by (simp add: S indicator_measurable_on)
  then show ?thesis
    using measurable_on_scaleR [OF _ f, of indicat_real S]
```

by (*simp add*: *indicator_scaleR_eq_if*)
qed

lemma *measurable_on_const_UNIV*: $(\lambda x.\ k)$ *measurable_on UNIV*
by (*simp add*: *continuous_imp_measurable_on*)

lemma *measurable_on_const* [*simp*]: $S \in$ *sets lebesgue* $\Longrightarrow (\lambda x.\ k)$ *measurable_on* S
using *measurable_on_UNIV measurable_on_const_UNIV measurable_on_restrict*
by *blast*

lemma *simple_function_indicator_representation_real*:
fixes $f :: 'a \Rightarrow$ *real*
assumes *f*: *simple_function M f* **and** *x*: $x \in$ *space M* **and** *nn*: $\bigwedge x.\ f\ x \geq 0$
shows $f\ x = (\sum y \in f\ `$ *space M*$.\ y *$ *indicator* $(f -` \{y\} \cap$ *space M*$)\ x)$
proof −
have *f'*: *simple_function M* (*ennreal* $\circ$ *f*)
by (*simp add*: *f*)
have $*$: $f\ x =$
enn2real
$(\sum y \in$ *ennreal* $`\ f\ `$ *space M*.
$y *$ *indicator* $((ennreal \circ f) -` \{y\} \cap$ *space M*$)\ x)$
using *arg_cong* [*OF simple_function_indicator_representation* [*OF f' x*], *of enn2real*, *simplified nn o_def*] *nn*
unfolding *o_def image_comp*
by (*metis enn2real_ennreal*)
have *enn2real* $(\sum y \in ennreal\ `\ f\ `$ *space M*. *if ennreal* $(f\ x) = y \wedge x \in$ *space M then y else 0*)
$=$ *sum* $(enn2real \circ (\lambda y.$ *if ennreal* $(f\ x) = y \wedge x \in$ *space M then y else 0*$))$
$(ennreal\ `\ f\ `$ *space M*$)$
by (*rule enn2real_sum*) *auto*
also have $\ldots =$ *sum* $(enn2real \circ (\lambda y.$ *if ennreal* $(f\ x) = y \wedge x \in$ *space M then y else 0*$) \circ$ *ennreal*$)$
$(f\ `$ *space M*$)$
by (*rule sum.reindex*) (*use nn* **in** ‹*auto simp*: *inj_on_def intro*: *sum.cong*›)
also have $\ldots = (\sum y \in f\ `$ *space M*. *if* $f\ x = y \wedge x \in$ *space M then y else 0*$)$
using *nn*
by (*auto simp*: *inj_on_def intro*: *sum.cong*)
finally show *?thesis*
by (*subst* $*$) (*simp add*: *enn2real_sum indicator_def of_bool_def if_distrib cong*: *if_cong*)
qed

lemma *simple_function_induct_real*
[*consumes 1*, *case_names cong set mult add*, *induct set*: *simple_function*]:
fixes $u :: 'a \Rightarrow$ *real*
assumes *u*: *simple_function M u*
assumes *cong*: $\bigwedge f\ g.$ *simple_function M f* $\Longrightarrow$ *simple_function M g* $\Longrightarrow$ (*AE x in M*. $f\ x = g\ x) \Longrightarrow P\ f \Longrightarrow P\ g$

```
  assumes set: ⋀A. A ∈ sets M ⟹ P (indicator A)
  assumes mult: ⋀u c. P u ⟹ P (λx. c * u x)
  assumes add: ⋀u v. P u ⟹ P v ⟹ P (λx. u x + v x)
  and nn: ⋀x. u x ≥ 0
  shows P u
proof (rule cong)
  from AE_space show AE x in M. (∑ y∈u ‘ space M. y * indicator (u -‘ {y}
∩ space M) x) = u x
  proof eventually_elim
    fix x assume x: x ∈ space M
    from simple_function_indicator_representation_real[OF u x] nn
    show (∑ y∈u ‘ space M. y * indicator (u -‘ {y} ∩ space M) x) = u x
      by metis
  qed
next
  from u have finite (u ‘ space M)
    unfolding simple_function_def by auto
  then show P (λx. ∑ y∈u ‘ space M. y * indicator (u -‘ {y} ∩ space M) x)
  proof induct
    case empty
    then show ?case
      using set[of {}] by (simp add: indicator_def[abs_def])
  next
    case (insert a F)
    have eq: ∑ {y. u x = y ∧ (y = a ∨ y ∈ F) ∧ x ∈ space M}
          = (if u x = a ∧ x ∈ space M then a else 0) + ∑ {y. u x = y ∧ y ∈ F
∧ x ∈ space M} for x
    proof (cases x ∈ space M)
      case True
      have *: {y. u x = y ∧ (y = a ∨ y ∈ F)} = {y. u x = a ∧ y = a} ∪ {y. u x
= y ∧ y ∈ F}
        by auto
      show ?thesis
        using insert by (simp add: * True)
    qed auto
    have a: P (λx. a * indicator (u -‘ {a} ∩ space M) x)
    proof (intro mult set)
      show u -‘ {a} ∩ space M ∈ sets M
        using u by auto
    qed
    show ?case
      using nn insert a
      by (simp add: eq indicator_times_eq_if [where f = λx. a] add)
  qed
next
  show simple_function M (λx. (∑ y∈u ‘ space M. y * indicator (u -‘ {y} ∩
space M) x))
    apply (subst simple_function_cong)
    apply (rule simple_function_indicator_representation_real[symmetric])
```

```
    apply (auto intro: u nn)
    done
qed fact

proposition simple_function_measurable_on_UNIV:
  fixes f :: 'a::euclidean_space ⇒ real
  assumes f: simple_function lebesgue f and nn: ⋀x. f x ≥ 0
  shows f measurable_on UNIV
  using f
proof (induction f)
  case (cong f g)
  then obtain N where negligible N {x. g x ≠ f x} ⊆ N
    by (auto simp: eventually_ae_filter_negligible eq_commute)
  then show ?case
    by (blast intro: measurable_on_spike cong)
next
  case (set S)
  then show ?case
    by (simp add: indicator_measurable_on)
next
  case (mult u c)
  then show ?case
    by (simp add: measurable_on_cmul)
  case (add u v)
  then show ?case
    by (simp add: measurable_on_add)
qed (auto simp: nn)

lemma simple_function_lebesgue_if:
  fixes f :: 'a::euclidean_space ⇒ real
  assumes f: simple_function lebesgue f and S: S ∈ sets lebesgue
  shows simple_function lebesgue (λx. if x ∈ S then f x else 0)
proof −
  have ffin: finite (range f) and fsets: ∀ x. f −‘ {f x} ∈ sets lebesgue
    using f by (auto simp: simple_function_def)
  have finite (f ‘ S)
    by (meson finite_subset subset_image_iff ffin top_greatest)
  moreover have finite ((λx. 0::real) ‘ T) for T :: 'a set
    by (auto simp: image_def)
  moreover have if_sets: (λx. if x ∈ S then f x else 0) −‘ {f a} ∈ sets lebesgue
for a
  proof −
    have ∗: (λx. if x ∈ S then f x else 0) −‘ {f a}
          = (if f a = 0 then −S ∪ f −‘ {f a} else (f −‘ {f a}) ∩ S)
      by (auto simp: split: if_split_asm)
    show ?thesis
      unfolding ∗ by (metis Compl_in_sets_lebesgue S sets.Int sets.Un fsets)
  qed
  moreover have (λx. if x ∈ S then f x else 0) −‘ {0} ∈ sets lebesgue
```

proof (*cases* $0 \in$ *range* f)
case *True*
then show *?thesis*
by (*metis* (*no_types*, *lifting*) *if_sets rangeE*)
next
case *False*
then have $(\lambda x.\ \textit{if}\ x \in S\ \textit{then}\ f\ x\ \textit{else}\ 0) -` \{0\} = -S$
by *auto*
then show *?thesis*
by (*simp add*: *Compl_in_sets_lebesgue S*)
qed
ultimately show *?thesis*
by (*auto simp*: *simple_function_def*)
qed

corollary *simple_function_measurable_on*:
fixes $f :: 'a::\textit{euclidean\_space} \Rightarrow \textit{real}$
assumes *f*: *simple_function lebesgue f* **and** *nn*: $\bigwedge x.\ f\ x \geq 0$ **and** *S*: $S \in$ *sets lebesgue*
shows *f measurable_on S*
by (*simp add*: *measurable_on_UNIV* [*symmetric*, *of f*] *S f simple_function_lebesgue_if nn simple_function_measurable_on_UNIV*)

lemma
fixes $f :: 'a::\textit{euclidean\_space} \Rightarrow 'b::\textit{ordered\_euclidean\_space}$
assumes *f*: *f measurable_on S* **and** *g*: *g measurable_on S*
shows *measurable_on_sup*: $(\lambda x.\ \textit{sup}\ (f\ x)\ (g\ x))$ *measurable_on S*
and *measurable_on_inf*: $(\lambda x.\ \textit{inf}\ (f\ x)\ (g\ x))$ *measurable_on S*
proof −
obtain *NF* **and** *F*
where *NF*: *negligible NF*
and *conF*: $\bigwedge n.$ *continuous_on UNIV* $(F\ n)$
and *tendsF*: $\bigwedge x.\ x \notin NF \Longrightarrow (\lambda n.\ F\ n\ x) \longrightarrow (\textit{if}\ x \in S\ \textit{then}\ f\ x\ \textit{else}\ 0)$
using *f* **by** (*auto simp*: *measurable_on_def*)
obtain *NG* **and** *G*
where *NG*: *negligible NG*
and *conG*: $\bigwedge n.$ *continuous_on UNIV* $(G\ n)$
and *tendsG*: $\bigwedge x.\ x \notin NG \Longrightarrow (\lambda n.\ G\ n\ x) \longrightarrow (\textit{if}\ x \in S\ \textit{then}\ g\ x\ \textit{else}\ 0)$
using *g* **by** (*auto simp*: *measurable_on_def*)
show $(\lambda x.\ \textit{sup}\ (f\ x)\ (g\ x))$ *measurable_on S*
unfolding *measurable_on_def*
proof (*intro exI conjI allI impI*)
show *continuous_on UNIV* $(\lambda x.\ \textit{sup}\ (F\ n\ x)\ (G\ n\ x))$ **for** *n*
unfolding *sup_max eucl_sup* **by** (*intro conF conG continuous_intros*)
show $(\lambda n.\ \textit{sup}\ (F\ n\ x)\ (G\ n\ x)) \longrightarrow (\textit{if}\ x \in S\ \textit{then}\ \textit{sup}\ (f\ x)\ (g\ x)\ \textit{else}\ 0)$
if $x \notin NF \cup NG$ **for** *x*
using *tendsto_sup* [*OF tendsF tendsG*, *of x x*] *that* **by** *auto*
qed (*simp add*: *NF NG*)
show $(\lambda x.\ \textit{inf}\ (f\ x)\ (g\ x))$ *measurable_on S*

```
    unfolding measurable_on_def
  proof (intro exI conjI allI impI)
    show continuous_on UNIV (λx. inf (F n x) (G n x)) for n
      unfolding inf_min eucl_inf  by (intro conF conG continuous_intros)
    show (λn. inf (F n x) (G n x)) ⟶ (if x ∈ S then inf (f x) (g x) else 0)
      if x ∉ NF ∪ NG for x
      using tendsto_inf [OF tendsF tendsG, of x x] that by auto
  qed (simp add: NF NG)
qed

proposition measurable_on_componentwise_UNIV:
 f measurable_on UNIV ⟷ (∀ i∈Basis. (λx. (f x ∙ i) *R i) measurable_on UNIV)
 (is ?lhs = ?rhs)
proof
  assume L: ?lhs
  show ?rhs
  proof
    fix i::'b
    assume i ∈ Basis
    have cont: continuous_on UNIV (λx. (x ∙ i) *R i)
      by (intro continuous_intros)
    show (λx. (f x ∙ i) *R i) measurable_on UNIV
      using measurable_on_compose_continuous [OF L cont]
      by (simp add: o_def)
  qed
next
  assume ?rhs
  then have ∃ N g. negligible N ∧
            (∀ n. continuous_on UNIV (g n)) ∧
            (∀ x. x ∉ N ⟶ (λn. g n x) ⟶ (f x ∙ i) *R i)
    if i ∈ Basis for i
    by (simp add: measurable_on_def that)
  then obtain N g where N: ⋀i. i ∈ Basis ⟹ negligible (N i)
       and cont: ⋀i n. i ∈ Basis ⟹ continuous_on UNIV (g i n)
       and tends: ⋀i x. ⟦i ∈ Basis; x ∉ N i⟧ ⟹ (λn. g i n x) ⟶ (f x ∙ i) *R i
    by metis
  show ?lhs
    unfolding measurable_on_def
  proof (intro exI conjI allI impI)
    show negligible (⋃ i ∈ Basis. N i)
      using N eucl.finite_Basis by blast
    show continuous_on UNIV (λx. (∑ i∈Basis. g i n x)) for n
      by (intro continuous_intros cont)
  next
    fix x
    assume x ∉ (⋃ i ∈ Basis. N i)
    then have ⋀i. i ∈ Basis ⟹ x ∉ N i
      by auto
    then have (λn. (∑ i∈Basis. g i n x)) ⟶ (∑ i∈Basis. (f x ∙ i) *R i)
```

```
    by (intro tends tendsto_intros)
  then show (λn. (∑ i∈Basis. g i n x)) ⟶ (if x ∈ UNIV then f x else 0)
    by (simp add: euclidean_representation)
 qed
qed

corollary measurable_on_componentwise:
 f measurable_on S ⟷ (∀ i∈Basis. (λx. (f x • i) *R i) measurable_on S)
 apply (subst measurable_on_UNIV [symmetric])
 apply (subst measurable_on_componentwise_UNIV)
 apply (simp add: measurable_on_UNIV if_distrib [of λx. inner x _] if_distrib
[of λx. scaleR x _] cong: if_cong)
 done



lemma borel_measurable_implies_simple_function_sequence_real:
 fixes u :: 'a ⇒ real
 assumes u[measurable]: u ∈ borel_measurable M and nn: ⋀x. u x ≥ 0
 shows ∃f. incseq f ∧ (∀i. simple_function M (f i)) ∧ (∀x. bdd_above (range
(λi. f i x))) ∧
           (∀i x. 0 ≤ f i x) ∧ u = (SUP i. f i)
proof −
 define f where [abs_def]:
   f i x = real_of_int (floor ((min i (u x)) * 2^i)) / 2^i for i x

 have [simp]: 0 ≤ f i x for i x
   by (auto simp: f_def intro!: divide_nonneg_nonneg mult_nonneg_nonneg nn)

 have *: 2^n * real_of_int x = real_of_int (2^n * x) for n x
   by simp

 have real_of_int ⌊real i * 2 ^ i⌋ = real_of_int ⌊i * 2 ^ i⌋ for i
   by (intro arg_cong[where f=real_of_int]) simp
 then have [simp]: real_of_int ⌊real i * 2 ^ i⌋ = i * 2 ^ i for i
   unfolding floor_of_nat by simp

 have bdd: bdd_above (range (λi. f i x)) for x
  by (rule bdd_aboveI [where M = u x]) (auto simp: f_def field_simps min_def)

 have incseq f
 proof (intro monoI le_funI)
   fix m n :: nat and x assume m ≤ n
   moreover
   { fix d :: nat
     have ⌊2^d::real⌋ * ⌊2^m * (min (of_nat m) (u x))⌋ ≤ ⌊2^d * (2^m * (min
(of_nat m) (u x)))⌋
       by (rule le_mult_floor) (auto simp: nn)
     also have ... ≤ ⌊2^d * (2^m *  (min (of_nat d + of_nat m) (u x)))⌋
```

```
      by (intro floor_mono mult_mono min.mono)
        (auto simp: nn min_less_iff_disj of_nat_less_top)
    finally have f m x ≤ f(m + d) x
      unfolding f_def
      by (auto simp: field_simps power_add * simp del: of_int_mult) }
  ultimately show f m x ≤ f n x
    by (auto simp: le_iff_add)
qed
then have inc_f: incseq (λi. f i x) for x
  by (auto simp: incseq_def le_fun_def)
moreover
have simple_function M (f i) for i
proof (rule simple_function_borel_measurable)
  have ⌊(min (of_nat i) (u x)) * 2 ^ i⌋ ≤ ⌊int i * 2 ^ i⌋ for x
    by (auto split: split_min intro!: floor_mono)
  then have f i ‘ space M ⊆ (λn. real_of_int n / 2^i) ‘ {0 .. of_nat i * 2^i}
    unfolding floor_of_int by (auto simp: f_def nn intro!: imageI)
  then show finite (f i ‘ space M)
    by (rule finite_subset) auto
  show f i ∈ borel_measurable M
    unfolding f_def enn2real_def by measurable
qed
moreover
{ fix x
  have (SUP i. (f i x)) = u x
  proof −
    obtain n where u x ≤ of_nat n using real_arch_simple by auto
    then have min_eq_r: ∀F i in sequentially. min (real i) (u x) = u x
      by (auto simp: eventually_sequentially intro!: exI[of _ n] split: split_min)
    have (λi. real_of_int ⌊min (real i) (u x) * 2^i⌋ / 2^i) ⟶ u x
    proof (rule tendsto_sandwich)
      show (λn. u x − (1/2)^n) ⟶ u x
        by (auto intro!: tendsto_eq_intros LIMSEQ_power_zero)
      show ∀F n in sequentially. real_of_int ⌊min (real n) (u x) * 2 ^ n⌋ / 2 ^
n ≤ u x
        using min_eq_r by eventually_elim (auto simp: field_simps)
      have *: u x * (2 ^ n * 2 ^ n) ≤ 2^n + 2^n * real_of_int ⌊u x * 2 ^ n⌋ for
n
        using real_of_int_floor_ge_diff_one[of u x * 2^n, THEN mult_left_mono,
of 2^n]
        by (auto simp: field_simps)
      show ∀F n in sequentially. u x − (1/2)^n ≤ real_of_int ⌊min (real n) (u
x) * 2 ^ n⌋ / 2 ^ n
        using min_eq_r by eventually_elim (insert *, auto simp: field_simps)
    qed auto
    then have (λi. (f i x)) ⟶ u x
      by (simp add: f_def)
    from LIMSEQ_unique LIMSEQ_incseq_SUP [OF bdd inc_f] this
    show ?thesis
```

```
      by blast
    qed }
  ultimately show ?thesis
    by (intro exI [of _ λi x. f i x]) (auto simp: ‹incseq f› bdd image_comp)
qed


lemma homeomorphic_open_interval_UNIV:
  fixes a b:: real
  assumes a < b
  shows {a<..<b} homeomorphic (UNIV::real set)
proof −
  have {a<..<b} = ball ((b+a) / 2) ((b−a) / 2)
    using assms
    by (auto simp: dist_real_def abs_if field_split_simps split: if_split_asm)
  then show ?thesis
    by (simp add: homeomorphic_ball_UNIV assms)
qed

proposition homeomorphic_box_UNIV:
  fixes a b:: 'a::euclidean_space
  assumes box a b ≠ {}
  shows box a b homeomorphic (UNIV::'a set)
proof −
  have {a · i <..<b · i} homeomorphic (UNIV::real set) if i ∈ Basis for i
   using assms box_ne_empty that by (blast intro: homeomorphic_open_interval_UNIV)
  then have ∃f g. (∀x. a · i < x ∧ x < b · i ⟶ g (f x) = x) ∧
                  (∀y. a · i < g y ∧ g y < b · i ∧ f(g y) = y) ∧
                  continuous_on {a · i<..<b · i} f ∧
                  continuous_on (UNIV::real set) g
    if i ∈ Basis for i
    using that by (auto simp: homeomorphic_minimal mem_box Ball_def)
  then obtain f g where gf: ⋀i x. ⟦i ∈ Basis; a · i < x; x < b · i⟧ ⟹ g i (f i
x) = x
            and fg: ⋀i y. i ∈ Basis ⟹ a · i < g i y ∧ g i y < b · i ∧ f i (g i y)
= y
            and contf: ⋀i. i ∈ Basis ⟹ continuous_on {a · i<..<b · i} (f i)
            and contg: ⋀i. i ∈ Basis ⟹ continuous_on (UNIV::real set) (g i)
    by metis
  define F where F ≡ λx. ∑i∈Basis. (f i (x · i)) *R i
  define G where G ≡ λx. ∑i∈Basis. (g i (x · i)) *R i
  show ?thesis
    unfolding homeomorphic_minimal
  proof (intro exI conjI ballI)
    show G y ∈ box a b for y
      using fg by (simp add: G_def mem_box)
    show G (F x) = x if x ∈ box a b for x
     using that by (simp add: F_def G_def gf mem_box euclidean_representation)
    show F (G y) = y for y
```

```
    by (simp add: F_def G_def fg mem_box euclidean_representation)
   show continuous_on (box a b) F
    unfolding F_def
   proof (intro continuous_intros continuous_on_compose2 [OF contf continuous_on_inner])
    show (λx. x • i) ` box a b ⊆ {a • i<..<b • i} if i ∈ Basis for i
     using that by (auto simp: mem_box)
   qed
   show continuous_on UNIV G
    unfolding G_def
     by (intro continuous_intros continuous_on_compose2 [OF contg continuous_on_inner]) auto
  qed auto
qed



lemma diff_null_sets_lebesgue: ⟦N ∈ null_sets (lebesgue_on S); X−N ∈ sets (lebesgue_on S); N ⊆ X⟧
   ⟹ X ∈ sets (lebesgue_on S)
  by (metis Int_Diff_Un inf.commute inf.orderE null_setsD2 sets.Un)

lemma borel_measurable_diff_null:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes N: N ∈ null_sets (lebesgue_on S) and S: S ∈ sets lebesgue
  shows f ∈ borel_measurable (lebesgue_on (S−N)) ⟷ f ∈ borel_measurable (lebesgue_on S)
  unfolding in_borel_measurable space_lebesgue_on sets_restrict_UNIV
proof (intro ball_cong iffI)
  show f −` T ∩ S ∈ sets (lebesgue_on S)
    if f −` T ∩ (S−N) ∈ sets (lebesgue_on (S−N)) for T
  proof −
    have N ∩ S = N
      by (metis N S inf.orderE null_sets_restrict_space)
    moreover have N ∩ S ∈ sets lebesgue
      by (metis N S inf.orderE null_setsD2 null_sets_restrict_space)
    moreover have f −` T ∩ S ∩ (f −` T ∩ N) ∈ sets lebesgue
     by (metis N S completion.complete inf.absorb2 inf_le2 inf_mono null_sets_restrict_space)
    ultimately show ?thesis
     by (metis Diff_Int_distrib Int_Diff_Un S inf_le2 sets.Diff sets.Un sets_restrict_space_iff space_lebesgue_on space_restrict_space that)
  qed
  show f −` T ∩ (S−N) ∈ sets (lebesgue_on (S−N))
    if f −` T ∩ S ∈ sets (lebesgue_on S) for T
  proof −
    have (S − N) ∩ f −` T = (S − N) ∩ (f −` T ∩ S)
      by blast
    then have (S − N) ∩ f −` T ∈ sets.restricted_space lebesgue (S − N)
      by (metis S image_iff sets.Int_space_eq2 sets_restrict_space_iff that)
```

```
    then show ?thesis
      by (simp add: inf.commute sets_restrict_space)
  qed
qed auto

lemma lebesgue_measurable_diff_null:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes N ∈ null_sets lebesgue
   shows f ∈ borel_measurable (lebesgue_on (−N)) ⟷ f ∈ borel_measurable
lebesgue
  by (simp add: Compl_eq_Diff_UNIV assms borel_measurable_diff_null lebesgue_on_UNIV_eq)



proposition measurable_on_imp_borel_measurable_lebesgue_UNIV:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes f measurable_on UNIV
  shows f ∈ borel_measurable lebesgue
proof −
  obtain N and F
    where NF: negligible N
      and conF: ⋀n. continuous_on UNIV (F n)
      and tendsF: ⋀x. x ∉ N ⟹ (λn. F n x) ⟶ f x
    using assms by (auto simp: measurable_on_def)
   obtain N where N ∈ null_sets lebesgue f ∈ borel_measurable (lebesgue_on
(−N))
  proof
    show f ∈ borel_measurable (lebesgue_on (− N))
    proof (rule borel_measurable_LIMSEQ_metric)
      show F i ∈ borel_measurable (lebesgue_on (− N)) for i
      by (meson Compl_in_sets_lebesgue NF conF continuous_imp_measurable_on_sets_lebesgue
continuous_on_subset negligible_imp_sets subset_UNIV)
      show (λi. F i x) ⟶ f x if x ∈ space (lebesgue_on (− N)) for x
        using that
        by (simp add: tendsF)
    qed
    show N ∈ null_sets lebesgue
      using NF negligible_iff_null_sets by blast
  qed
  then show ?thesis
    using lebesgue_measurable_diff_null by blast
qed

corollary measurable_on_imp_borel_measurable_lebesgue:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes f measurable_on S and S: S ∈ sets lebesgue
  shows f ∈ borel_measurable (lebesgue_on S)
proof −
  have (λx. if x ∈ S then f x else 0) measurable_on UNIV
```

```
    using assms(1) measurable_on_UNIV by blast
  then show ?thesis
   by (simp add: borel_measurable_if_D measurable_on_imp_borel_measurable_lebesgue_UNIV)
qed


proposition measurable_on_limit:
  fixes f :: nat ⇒ 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes f: ⋀n. f n measurable_on S and N: negligible N
    and lim: ⋀x. x ∈ S − N ⟹ (λn. f n x) ⟶ g x
  shows g measurable_on S
proof −
  have box (0::'b) One homeomorphic (UNIV::'b set)
    by (simp add: homeomorphic_box_UNIV)
  then obtain h h'::'b⇒'b where hh': ⋀x. x ∈ box 0 One ⟹ h (h' x) = x
                and h'im:  h' ‘ box 0 One = UNIV
                and conth: continuous_on UNIV h
                and conth': continuous_on (box 0 One) h'
                and h'h:   ⋀y. h' (h y) = y
                and rangeh: range h = box 0 One
    by (auto simp: homeomorphic_def homeomorphism_def)
  have norm y ≤ DIM('b) if y: y ∈ box 0 One for y::'b
  proof −
    have y01: 0 < y • i y • i < 1 if i ∈ Basis for i
      using that y by (auto simp: mem_box)
    have norm y ≤ (∑ i∈Basis. |y • i|)
      using norm_le_l1 by blast
    also have ... ≤ (∑ i::'b∈Basis. 1)
    proof (rule sum_mono)
      show |y • i| ≤ 1 if i ∈ Basis for i
        using y01 that by fastforce
    qed
    also have ... ≤ DIM('b)
      by auto
    finally show ?thesis .
  qed
  then have norm_le: norm(h y) ≤ DIM('b) for y
    by (metis UNIV_I image_eqI rangeh)
  have (h' ∘ (h ∘ (λx. if x ∈ S then g x else 0))) measurable_on UNIV
  proof (rule measurable_on_compose_continuous_box)
    let ?χ =  h ∘ (λx. if x ∈ S then g x else 0)
    let ?f = λn. h ∘ (λx. if x ∈ S then f n x else 0)
    show ?χ measurable_on UNIV
    proof (rule integrable_subintervals_imp_measurable)
      show ?χ integrable_on cbox a b for a b
      proof (rule integrable_spike_set)
        show ?χ integrable_on (cbox a b − N)
        proof (rule dominated_convergence_integrable)
          show const: (λx. DIM('b)) integrable_on cbox a b − N
```

```
          by (simp add: N has_integral_iff integrable_const integrable_negligible
integrable_setdiff negligible_diff)
        show norm ((h ∘ (λx. if x ∈ S then g x else 0)) x) ≤ DIM('b) if x ∈ cbox
a b − N for x
          using that norm_le  by (simp add: o_def)
        show (λk. ?f k x) ⟶ ?χ x if x ∈ cbox a b − N for x
          using that lim [of x] conth
          by (auto simp: continuous_on_def intro: tendsto_compose)
        show (?f n) absolutely_integrable_on cbox a b − N for n
      proof (rule measurable_bounded_by_integrable_imp_absolutely_integrable)
        show ?f n ∈ borel_measurable (lebesgue_on (cbox a b − N))
         proof (rule measurable_on_imp_borel_measurable_lebesgue [OF mea-
surable_on_spike_set])
          show ?f n measurable_on cbox a b
            unfolding measurable_on_UNIV [symmetric, of _ cbox a b]
          proof (rule measurable_on_restrict)
            have f': (λx. if x ∈ S then f n x else 0) measurable_on UNIV
              by (simp add: f measurable_on_UNIV)
            show ?f n measurable_on UNIV
              using measurable_on_compose_continuous [OF f' conth] by auto
          qed auto
          show negligible (sym_diff (cbox a b) (cbox a b − N))
            by (auto intro: negligible_subset [OF N])
          show cbox a b − N ∈ sets lebesgue
            by (simp add: N negligible_imp_sets sets.Diff)
        qed
        show cbox a b − N ∈ sets lebesgue
          by (simp add: N negligible_imp_sets sets.Diff)
        show norm (?f n x) ≤ DIM('b)
          if x ∈ cbox a b − N for x
          using that local.norm_le by simp
      qed (auto simp: const)
    qed
    show negligible {x ∈ cbox a b − N − cbox a b. ?χ x ≠ 0}
      by (auto simp: empty_imp_negligible)
    have {x ∈ cbox a b − (cbox a b − N). ?χ x ≠ 0} ⊆ N
      by auto
    then show negligible {x ∈ cbox a b − (cbox a b − N). ?χ x ≠ 0}
      using N negligible_subset by blast
  qed
 qed
 show ?χ x ∈ box 0 One for x
  using rangeh by auto
 show continuous_on (box 0 One) h'
  by (rule conth')
qed
then show ?thesis
 by (simp add: o_def h'h measurable_on_UNIV)
qed
```

lemma *measurable_on_if_simple_function_limit*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
shows $[\![\bigwedge n.\ g\ n\ measurable\_on\ UNIV;\ \bigwedge n.\ finite\ (range\ (g\ n));\ \bigwedge x.\ (\lambda n.\ g\ n\ x) \longrightarrow f\ x]\!]$
$\Longrightarrow f\ measurable\_on\ UNIV$
by (*force intro*: *measurable_on_limit* [**where** $N=\{\}$])

lemma *lebesgue_measurable_imp_measurable_on_nnreal_UNIV*:
fixes $u :: 'a::euclidean\_space \Rightarrow real$
assumes *u*: $u \in borel\_measurable\ lebesgue$ **and** *nn*: $\bigwedge x.\ u\ x \geq 0$
shows $u\ measurable\_on\ UNIV$
proof −
obtain *f* **where** *incseq f* **and** *f*: $\forall i.\ simple\_function\ lebesgue\ (f\ i)$
and *bdd*: $\bigwedge x.\ bdd\_above\ (range\ (\lambda i.\ f\ i\ x))$
and *nnf*: $\bigwedge i\ x.\ 0 \leq f\ i\ x$ **and** $*$: $u = (SUP\ i.\ f\ i)$
using *borel_measurable_implies_simple_function_sequence_real nn u* **by** *metis*
show *?thesis*
unfolding $*$
proof (*rule measurable_on_if_simple_function_limit* [*of concl*: *Sup* (*range f*)])
show $(f\ i)\ measurable\_on\ UNIV$ **for** *i*
by (*simp add*: *f nnf simple_function_measurable_on_UNIV*)
show $finite\ (range\ (f\ i))$ **for** *i*
by (*metis f simple_function_def space_borel space_completion space_lborel*)
show $(\lambda i.\ f\ i\ x) \longrightarrow Sup\ (range\ f)\ x$ **for** *x*
proof −
have *incseq* $(\lambda i.\ f\ i\ x)$
using ‹*incseq f*› **apply** (*auto simp*: *incseq_def*)
by (*simp add*: *le_funD*)
then show *?thesis*
by (*metis SUP_apply bdd LIMSEQ_incseq_SUP*)
qed
qed
qed

lemma *lebesgue_measurable_imp_measurable_on_nnreal*:
fixes $u :: 'a::euclidean\_space \Rightarrow real$
assumes $u \in borel\_measurable\ lebesgue\ \bigwedge x.\ u\ x \geq 0 S \in sets\ lebesgue$
shows $u\ measurable\_on\ S$
unfolding *measurable_on_UNIV* [*symmetric*, *of u*]
using *assms*
by (*auto intro*: *lebesgue_measurable_imp_measurable_on_nnreal_UNIV*)

lemma *lebesgue_measurable_imp_measurable_on_real*:
fixes $u :: 'a::euclidean\_space \Rightarrow real$
assumes *u*: $u \in borel\_measurable\ lebesgue$ **and** *S*: $S \in sets\ lebesgue$
shows $u\ measurable\_on\ S$

proof −
let *?f* = λ*x*. |*u x*| + *u x*
let *?g* = λ*x*. |*u x*| − *u x*
have *?f measurable_on S ?g measurable_on S*
using *S u* **by** (*auto intro*: *lebesgue_measurable_imp_measurable_on_nnreal*)
then have (λ*x*. (*?f x* − *?g x*) / *2*) *measurable_on S*
using *measurable_on_cdivide measurable_on_diff* **by** *blast*
then show *?thesis*
by *auto*
qed

proposition *lebesgue_measurable_imp_measurable_on*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *f*: *f* ∈ *borel_measurable lebesgue* **and** *S*: *S* ∈ *sets lebesgue*
shows *f measurable_on S*
unfolding *measurable_on_componentwise* [*of f*]
proof
fix *i*::*'b*
assume *i* ∈ *Basis*
have (λ*x*. (*f x* ∙ *i*)) ∈ *borel_measurable lebesgue*
using ‹*i* ∈ *Basis*› *borel_measurable_euclidean_space f* **by** *blast*
then have (λ*x*. (*f x* ∙ *i*)) *measurable_on S*
using *S lebesgue_measurable_imp_measurable_on_real* **by** *blast*
then show (λ*x*. (*f x* ∙ *i*) ∗$_R$ *i*) *measurable_on S*
by (*intro measurable_on_scaleR measurable_on_const S*)
qed

proposition *measurable_on_iff_borel_measurable*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *S* ∈ *sets lebesgue*
shows *f measurable_on S* ⟷ *f* ∈ *borel_measurable* (*lebesgue_on S*) (**is** *?lhs* = *?rhs*)
proof
show *f* ∈ *borel_measurable* (*lebesgue_on S*)
if *f measurable_on S*
using *that* **by** (*simp add*: *assms measurable_on_imp_borel_measurable_lebesgue*)
next
assume *f* ∈ *borel_measurable* (*lebesgue_on S*)
then have (λ*a*. *if a* ∈ *S then f a else 0*) *measurable_on UNIV*
by (*simp add*: *assms borel_measurable_if lebesgue_measurable_imp_measurable_on*)
then show *f measurable_on S*
using *measurable_on_UNIV* **by** *blast*
qed

6.26.5 Monotonic functions are Lebesgue integrable

lemma *integrable_mono_on_nonneg*:
fixes *f* :: *real* ⇒ *real*

```
  assumes mon: mono_on {a..b} f and 0: ⋀x. 0 ≤ f x
  shows integrable (lebesgue_on {a..b}) f
proof −
  have space lborel = space lebesgue sets borel ⊆ sets lebesgue
    by force+
  then have fborel: f ∈ borel_measurable (lebesgue_on {a..b})
    by (metis mon borel_measurable_mono_on_fnc borel_measurable_subalgebra
mono_restrict_space space_lborel space_restrict_space)
  then obtain g where g: incseq g and simple: ⋀i. simple_function (lebesgue_on
{a..b}) (g i)
                  and bdd:  (∀ x. bdd_above (range (λi. g i x))) and nonneg: ∀ i x. 0
≤ g i x
                and fsup: f = (SUP i. g i)
    by (metis borel_measurable_implies_simple_function_sequence_real 0)
  have f ‘ {a..b} ⊆ {f a..f b}
    using assms by (auto simp: mono_on_def)
  have g_le_f: g i x ≤ f x for i x
  proof −
    have bdd_above ((λh. h x) ‘ range g)
      using bdd cSUP_lessD linorder_not_less by fastforce
    then show ?thesis
      by (metis SUP_apply UNIV_I bdd cSUP_upper fsup)
  qed
  then have gfb: g i x ≤ f b if x ∈ {a..b} for i x
    by (smt (verit, best) mon atLeastAtMost_iff mono_on_def that)
  have g_le: g i x ≤ g j x if i≤j  for i j x
    using g by (simp add: incseq_def le_funD that)
  show integrable (lebesgue_on {a..b}) ( f)
  proof (rule integrable_dominated_convergence)
    show f ∈ borel_measurable (lebesgue_on {a..b})
      using fborel by blast
    have ⋀x. (λi. g i x) ⟶ (SUP h ∈ range g. h  x)
    proof (rule order_tendstoI)
      show ∀ F i in sequentially. y < g i x
        if y < (SUP h∈range g. h x) for x y
      proof −
        from that obtain h where h: h ∈ range g y < h x
          using g_le_f by (subst (asm)less_cSUP_iff) fastforce+
        then show ?thesis
          by (smt (verit, ccfv_SIG) eventually_sequentially g_le imageE)
      qed
      show ∀ F i in sequentially. g i x < y
        if (SUP h∈range g. h x) < y for x y
          by (smt (verit, best) that Sup_apply g_le_f always_eventually fsup im-
age_cong)
    qed
    then show AE x in lebesgue_on {a..b}. (λi. g i x) ⟶ f x
      by (simp add: fsup)
    fix i
```

```
    show g i ∈ borel_measurable (lebesgue_on {a..b})
      using borel_measurable_simple_function simple by blast
    show AE x in lebesgue_on {a..b}. norm (g i x) ≤ f b
      by (simp add: gfb nonneg Measure_Space.AE_I' [of {}])
  qed auto
qed

lemma integrable_mono_on:
  fixes f :: real ⇒ real
  assumes mono_on {a..b} f
  shows integrable (lebesgue_on {a..b}) f
proof −
  define f' where f' ≡ λx. if x ∈ {a..b} then f x − f a else 0
  have mono_on {a..b} f'
    by (smt (verit, best) assms f'_def mono_on_def)
  moreover have 0: ⋀x. 0 ≤ f' x
    by (smt (verit, best) assms atLeastAtMost_iff f'_def mono_on_def)
  ultimately have integrable (lebesgue_on {a..b}) f'
    using integrable_mono_on_nonneg by presburger
  then have integrable (lebesgue_on {a..b}) (λx. f' x + f a)
    by force
  moreover have space lborel = space lebesgue sets borel ⊆ sets lebesgue
    by force+
  then have fborel: f ∈ borel_measurable (lebesgue_on {a..b})
   by (metis assms borel_measurable_mono_on_fnc borel_measurable_subalgebra
mono_restrict_space space_lborel space_restrict_space)
  ultimately show ?thesis
    by (rule integrable_cong_AE_imp) (auto simp add: f'_def)
qed

lemma integrable_on_mono_on:
  fixes f :: real ⇒ real
  assumes mono_on {a..b} f
  shows f integrable_on {a..b}
  by (simp add: assms integrable_mono_on integrable_on_lebesgue_on)
```

6.26.6 Measurability on generalisations of the binary product

```
lemma measurable_on_bilinear:
  fixes h :: 'a::euclidean_space ⇒ 'b::euclidean_space ⇒ 'c::euclidean_space
  assumes h: bilinear h and f: f measurable_on S and g: g measurable_on S
  shows (λx. h (f x) (g x)) measurable_on S
proof (rule measurable_on_combine [where h = h])
  show continuous_on UNIV (λx. h (fst x) (snd x))
    by (simp add: bilinear_continuous_on_compose [OF continuous_on_fst con-
tinuous_on_snd h])
  show h 0 0 = 0
  by (simp add: bilinear_lzero h)
```

qed (*auto intro*: *assms*)

lemma *borel_measurable_bilinear*:
fixes h :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space* ⇒ *'c*::*euclidean_space*
assumes *bilinear h f* ∈ *borel_measurable* (*lebesgue_on S*) *g* ∈ *borel_measurable* (*lebesgue_on S*)
and *S*: *S* ∈ *sets lebesgue*
shows (λ*x*. *h* (*f x*) (*g x*)) ∈ *borel_measurable* (*lebesgue_on S*)
using *assms measurable_on_bilinear* [*of h f S g*]
by (*simp flip*: *measurable_on_iff_borel_measurable*)

lemma *absolutely_integrable_bounded_measurable_product*:
fixes *h* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space* ⇒ *'c*::*euclidean_space*
assumes *bilinear h* **and** *f*: *f* ∈ *borel_measurable* (*lebesgue_on S*) *S* ∈ *sets lebesgue*
and *bou*: *bounded* (*f* ‘ *S*) **and** *g*: *g absolutely_integrable_on S*
shows (λ*x*. *h* (*f x*) (*g x*)) *absolutely_integrable_on S*
proof −
obtain *B* **where** *B* > *0* **and** *B*: ⋀*x y*. *norm* (*h x y*) ≤ *B* ∗ *norm x* ∗ *norm y*
using *bilinear_bounded_pos* ‹*bilinear h*› **by** *blast*
obtain *C* **where** *C* > *0* **and** *C*: ⋀*x*. *x* ∈ *S* ⟹ *norm* (*f x*) ≤ *C*
using *bounded_pos* **by** (*metis bou imageI*)
show *?thesis*
proof (*rule measurable_bounded_by_integrable_imp_absolutely_integrable* [*OF* _ ‹*S* ∈ *sets lebesgue*›])
show *norm* (*h* (*f x*) (*g x*)) ≤ *B* ∗ *C* ∗ *norm*(*g x*) **if** *x* ∈ *S* **for** *x*
by (*meson less_le mult_left_mono mult_right_mono norm_ge_zero order_trans that* ‹*B* > *0*› *B C*)
show (λ*x*. *h* (*f x*) (*g x*)) ∈ *borel_measurable* (*lebesgue_on S*)
using ‹*bilinear h*› *f g*
by (*blast intro*: *borel_measurable_bilinear dest*: *absolutely_integrable_measurable*)
show (λ*x*. *B* ∗ *C* ∗ *norm*(*g x*)) *integrable_on S*
using ‹*0* < *B*› ‹*0* < *C*› *absolutely_integrable_on_def g* **by** *auto*
qed
qed

lemma *absolutely_integrable_bounded_measurable_product_real*:
fixes *f* :: *real* ⇒ *real*
assumes *f* ∈ *borel_measurable* (*lebesgue_on S*) *S* ∈ *sets lebesgue*
and *bounded* (*f* ‘ *S*) **and** *g absolutely_integrable_on S*
shows (λ*x*. *f x* ∗ *g x*) *absolutely_integrable_on S*
using *absolutely_integrable_bounded_measurable_product bilinear_times assms*
by *blast*

lemma *borel_measurable_AE*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *f* ∈ *borel_measurable lebesgue* **and** *ae*: *AE x in lebesgue*. *f x* = *g x*
shows *g* ∈ *borel_measurable lebesgue*

proof −
obtain N **where** N: $N \in$ *null_sets lebesgue* $\bigwedge x.\ x \notin N \Longrightarrow f\ x = g\ x$
using *ae* **unfolding** *completion.AE_iff_null_sets* **by** *auto*
have f *measurable_on UNIV*
by (*simp add*: *assms lebesgue_measurable_imp_measurable_on*)
then have g *measurable_on UNIV*
by (*metis Diff_iff N measurable_on_spike negligible_iff_null_sets*)
then show *?thesis*
using *measurable_on_imp_borel_measurable_lebesgue_UNIV* **by** *blast*
qed

lemma *has_bochner_integral_combine*:
fixes f :: *real* ⇒ *'a*::*euclidean_space*
assumes $a \leq c\ c \leq b$
and *ac*: *has_bochner_integral* (*lebesgue_on* {*a*..*c*}) $f\ i$
and *cb*: *has_bochner_integral* (*lebesgue_on* {*c*..*b*}) $f\ j$
shows *has_bochner_integral* (*lebesgue_on* {*a*..*b*}) $f(i + j)$
proof −
have *i*: *has_bochner_integral lebesgue* (λ*x*. *indicator* {*a*..*c*} *x* ∗$_R$ *f x*) *i*
and *j*: *has_bochner_integral lebesgue* (λ*x*. *indicator* {*c*..*b*} *x* ∗$_R$ *f x*) *j*
using *assms* **by** (*auto simp*: *has_bochner_integral_restrict_space*)
have *AE*: *AE x in lebesgue*. *indicat_real* {*a*..*c*} *x* ∗$_R$ *f x* + *indicat_real* {*c*..*b*} *x* ∗$_R$ *f x* = *indicat_real* {*a*..*b*} *x* ∗$_R$ *f x*
proof (*rule AE_I'*)
have *eq*: *indicat_real* {*a*..*c*} *x* ∗$_R$ *f x* + *indicat_real* {*c*..*b*} *x* ∗$_R$ *f x* = *indicat_real* {*a*..*b*} *x* ∗$_R$ *f x* **if** $x \neq c$ **for** *x*
using *assms that* **by** (*auto simp*: *indicator_def*)
then show {*x* ∈ *space lebesgue*. *indicat_real* {*a*..*c*} *x* ∗$_R$ *f x* + *indicat_real* {*c*..*b*} *x* ∗$_R$ *f x* ≠ *indicat_real* {*a*..*b*} *x* ∗$_R$ *f x*} ⊆ {*c*}
by *auto*
qed *auto*
have *has_bochner_integral lebesgue* (λ*x*. *indicator* {*a*..*b*} *x* ∗$_R$ *f x*) (*i* + *j*)
proof (*rule has_bochner_integralI_AE* [*OF has_bochner_integral_add* [*OF i j*] _ *AE*])
have *eq*: *indicat_real* {*a*..*c*} *x* ∗$_R$ *f x* + *indicat_real* {*c*..*b*} *x* ∗$_R$ *f x* = *indicat_real* {*a*..*b*} *x* ∗$_R$ *f x* **if** $x \neq c$ **for** *x*
using *assms that* **by** (*auto simp*: *indicator_def*)
show (λ*x*. *indicat_real* {*a*..*b*} *x* ∗$_R$ *f x*) ∈ *borel_measurable lebesgue*
proof (*rule borel_measurable_AE* [*OF borel_measurable_add AE*])
show (λ*x*. *indicator* {*a*..*c*} *x* ∗$_R$ *f x*) ∈ *borel_measurable lebesgue*
(λ*x*. *indicator* {*c*..*b*} *x* ∗$_R$ *f x*) ∈ *borel_measurable lebesgue*
using *i j* **by** *auto*
qed
qed
then show *?thesis*
by (*simp add*: *has_bochner_integral_restrict_space*)
qed

lemma *integrable_combine*:

fixes $f :: real \Rightarrow 'a::euclidean\_space$
assumes *integrable* (*lebesgue_on* $\{a..c\}$) f *integrable* (*lebesgue_on* $\{c..b\}$) f
and $a \leq c$ $c \leq b$
shows *integrable* (*lebesgue_on* $\{a..b\}$) f
using *assms has_bochner_integral_combine has_bochner_integral_iff* **by** *blast*

lemma *integral_combine*:
fixes $f :: real \Rightarrow 'a::euclidean\_space$
assumes f: *integrable* (*lebesgue_on* $\{a..b\}$) f **and** $a \leq c$ $c \leq b$
shows $integral^L$ (*lebesgue_on* $\{a..b\}$) f = $integral^L$ (*lebesgue_on* $\{a..c\}$) f + $integral^L$ (*lebesgue_on* $\{c..b\}$) f
proof −
have i: *has_bochner_integral* (*lebesgue_on* $\{a..c\}$) f($integral^L$ (*lebesgue_on* $\{a..c\}$) f)
using *integrable_subinterval* ‹$c \leq b$› f *has_bochner_integral_iff* **by** *fastforce*
have j: *has_bochner_integral* (*lebesgue_on* $\{c..b\}$) f($integral^L$ (*lebesgue_on* $\{c..b\}$) f)
using *integrable_subinterval* ‹$a \leq c$› f *has_bochner_integral_iff* **by** *fastforce*
show *?thesis*
by (*meson* ‹$a \leq c$› ‹$c \leq b$› *has_bochner_integral_combine has_bochner_integral_iff* i j)
qed

lemma *has_bochner_integral_null* [*intro*]:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes $N \in null\_sets\ lebesgue$
shows *has_bochner_integral* (*lebesgue_on* N) f 0
unfolding *has_bochner_integral_iff* — strange that the proof's so long
proof
show *integrable* (*lebesgue_on* N) f
proof (*subst integrable_restrict_space*)
show $N \cap space\ lebesgue \in sets\ lebesgue$
using *assms* **by** *force*
show *integrable lebesgue* ($\lambda x.$ *indicat_real* N x $*_R$ f x)
proof (*rule integrable_cong_AE_imp*)
show *integrable lebesgue* ($\lambda x.$ 0)
by *simp*
show $*$: *AE x in lebesgue.* 0 = *indicat_real* N x $*_R$ f x
using *assms*
by (*simp add*: *indicator_def completion.null_sets_iff_AE eventually_mono*)
show ($\lambda x.$ *indicat_real* N x $*_R$ f x) $\in$ *borel_measurable lebesgue*
by (*auto intro*: *borel_measurable_AE* [*OF* _ $*$])
qed
qed
show $integral^L$ (*lebesgue_on* N) f = 0
proof (*rule integral_eq_zero_AE*)
show *AE x in lebesgue_on* N. f x = 0
by (*rule AE_I'* [**where** N=N]) (*auto simp*: *assms null_setsD2 null_sets_restrict_space*)
qed

qed

lemma *has_bochner_integral_null_eq*[*simp*]:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes $N \in null\_sets\ lebesgue$
shows $has\_bochner\_integral\ (lebesgue\_on\ N)\ f\ i \longleftrightarrow i = 0$
using *assms has_bochner_integral_eq* **by** *blast*

end

6.27 Embedding Measure Spaces with a Function

theory *Embed_Measure*
imports *Binary_Product_Measure*
begin

Given a measure space on some carrier set Ω and a function f, we can define a push-forward measure on the carrier set $f(\Omega)$ whose σ-algebra is the one generated by mapping f over the original sigma algebra.

This is useful e. g. when f is injective, i. e. it is some kind of "tagging" function. For instance, suppose we have some algebraaic datatype of values with various constructors, including a constructor *RealVal* for real numbers. Then *embed_measure* allows us to lift a measure on real numbers to the appropriate subset of that algebraic datatype.

definition $embed\_measure :: 'a\ measure \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b\ measure$ **where**
$embed\_measure\ M\ f = measure\_of\ (f\ `\ space\ M)\ \{f\ `\ A\ |A.\ A \in sets\ M\}$
$(\lambda A.\ emeasure\ M\ (f\ -`\ A \cap space\ M))$

lemma *space_embed_measure*: $space\ (embed\_measure\ M\ f) = f\ `\ space\ M$
unfolding *embed_measure_def*
by (*subst space_measure_of*) (*auto dest*: *sets.sets_into_space*)

lemma *sets_embed_measure'*:
assumes *inj*: $inj\_on\ f\ (space\ M)$
shows $sets\ (embed\_measure\ M\ f) = \{f\ `\ A\ |A.\ A \in sets\ M\}$
unfolding *embed_measure_def*
proof (*intro sigma_algebra.sets_measure_of_eq sigma_algebra_iff2*[*THEN iffD2*] *conjI allI ballI impI*)
fix s **assume** $s \in \{f\ `\ A\ |A.\ A \in sets\ M\}$
then obtain s' **where** s'_props: $s = f\ `\ s'\ s' \in sets\ M$ **by** *auto*
hence $f\ `\ space\ M - s = f\ `\ (space\ M - s')$ **using** *inj*
by (*auto dest*: *inj_onD sets.sets_into_space*)
also have $... \in \{f\ `\ A\ |A.\ A \in sets\ M\}$ **using** s'_props **by** *auto*
finally show $f\ `\ space\ M - s \in \{f\ `\ A\ |A.\ A \in sets\ M\}$ **.**
next
fix $A :: nat \Rightarrow \_$ **assume** $range\ A \subseteq \{f\ `\ A\ |A.\ A \in sets\ M\}$
then obtain A' **where** A': $\bigwedge i.\ A\ i = f\ `\ A'\ i\ \bigwedge i.\ A'\ i \in sets\ M$
by (*auto simp*: *subset_eq choice_iff*)

then have $(\bigcup x.\ f\ `\ A'\ x) = f\ `\ (\bigcup x.\ A'\ x)$ **by** *blast*
with A' **show** $(\bigcup i.\ A\ i) \in \{f\ `\ A\ |A.\ A \in sets\ M\}$
by *simp blast*
qed (*auto dest*: *sets.sets_into_space*)

lemma *the_inv_into_vimage*:
$inj\_on\ f\ X \Longrightarrow A \subseteq X \Longrightarrow the\_inv\_into\ X\ f\ -`\ A \cap (f`X) = f\ `\ A$
by (*auto simp*: *the_inv_into_f_f*)

lemma *sets_embed_eq_vimage_algebra*:
assumes $inj\_on\ f\ (space\ M)$
shows $sets\ (embed\_measure\ M\ f) = sets\ (vimage\_algebra\ (f`space\ M)\ (the\_inv\_into\ (space\ M)\ f)\ M)$
by (*auto simp*: *sets_embed_measure'*[*OF assms*] *Pi_iff the_inv_into_f_f assms sets_vimage_algebra2 Setcompr_eq_image*
dest: *sets.sets_into_space*
intro!: *image_cong the_inv_into_vimage*[*symmetric*])

lemma *sets_embed_measure*:
assumes *inj*: *inj f*
shows $sets\ (embed\_measure\ M\ f) = \{f\ `\ A\ |A.\ A \in sets\ M\}$
using *assms* **by** (*subst sets_embed_measure'*) (*auto intro*!: *inj_onI dest*: *injD*)

lemma *in_sets_embed_measure*: $A \in sets\ M \Longrightarrow f\ `\ A \in sets\ (embed\_measure\ M\ f)$
unfolding *embed_measure_def*
by (*intro in_measure_of*) (*auto dest*: *sets.sets_into_space*)

lemma *measurable_embed_measure1*:
assumes *g*: $(\lambda x.\ g\ (f\ x)) \in measurable\ M\ N$
shows $g \in measurable\ (embed\_measure\ M\ f)\ N$
unfolding *measurable_def*
proof *safe*
fix A **assume** $A \in sets\ N$
with *g* **have** $(\lambda x.\ g\ (f\ x))\ -`\ A \cap space\ M \in sets\ M$
by (*rule measurable_sets*)
then have $f\ `\ ((\lambda x.\ g\ (f\ x))\ -`\ A \cap space\ M) \in sets\ (embed\_measure\ M\ f)$
by (*rule in_sets_embed_measure*)
also have $f\ `\ ((\lambda x.\ g\ (f\ x))\ -`\ A \cap space\ M) = g\ -`\ A \cap space\ (embed\_measure\ M\ f)$
by (*auto simp*: *space_embed_measure*)
finally show $g\ -`\ A \cap space\ (embed\_measure\ M\ f) \in sets\ (embed\_measure\ M\ f)$.
qed (*insert measurable_space*[*OF assms*], *auto simp*: *space_embed_measure*)

lemma *measurable_embed_measure2'*:
assumes $inj\_on\ f\ (space\ M)$
shows $f \in measurable\ M\ (embed\_measure\ M\ f)$
proof−

```
  {
    fix A assume A: A ∈ sets M
    also from A have A = A ∩ space M by auto
    also have ... = f −‘ f ‘ A ∩ space M using A assms
      by (auto dest: inj_onD sets.sets_into_space)
    finally have f −‘ f ‘ A ∩ space M ∈ sets M .
  }
  thus ?thesis using assms unfolding embed_measure_def
    by (intro measurable_measure_of) (auto dest: sets.sets_into_space)
qed

lemma measurable_embed_measure2:
  assumes [simp]: inj f shows f ∈ measurable M (embed_measure M f)
  by (auto simp: inj_vimage_image_eq embed_measure_def
          intro!: measurable_measure_of dest: sets.sets_into_space)

lemma embed_measure_eq_distr':
  assumes inj_on f (space M)
  shows embed_measure M f = distr M (embed_measure M f) f
proof−
  have distr M (embed_measure M f) f =
          measure_of (f ‘ space M) {f ‘ A |A. A ∈ sets M}
                     (λA. emeasure M (f −‘ A ∩ space M)) unfolding distr_def
    by (simp add: space_embed_measure sets_embed_measure'[OF assms])
  also have ... = embed_measure M f unfolding embed_measure_def ..
  finally show ?thesis ..
qed

lemma embed_measure_eq_distr:
    inj f ⟹ embed_measure M f = distr M (embed_measure M f) f
  by (rule embed_measure_eq_distr') (auto intro!: inj_onI dest: injD)

lemma nn_integral_embed_measure':
  inj_on f (space M) ⟹ g ∈ borel_measurable (embed_measure M f) ⟹
  nn_integral (embed_measure M f) g = nn_integral M (λx. g (f x))
  apply (subst embed_measure_eq_distr', simp)
  apply (subst nn_integral_distr)
  apply (simp_all add: measurable_embed_measure2')
  done

lemma nn_integral_embed_measure:
  inj f ⟹ g ∈ borel_measurable (embed_measure M f) ⟹
  nn_integral (embed_measure M f) g = nn_integral M (λx. g (f x))
  by(erule nn_integral_embed_measure'[OF subset_inj_on]) simp

lemma emeasure_embed_measure':
    assumes inj_on f (space M) A ∈ sets (embed_measure M f)
    shows emeasure (embed_measure M f) A = emeasure M (f −‘ A ∩ space M)
  by (subst embed_measure_eq_distr'[OF assms(1)])
```

(*simp add*: *emeasure_distr*[*OF measurable_embed_measure2'*[*OF assms*(*1*)] *assms*(*2*)])

lemma *emeasure_embed_measure*:
assumes *inj f A* ∈ *sets* (*embed_measure M f*)
shows *emeasure* (*embed_measure M f*) *A* = *emeasure M* (*f* −‘ *A* ∩ *space M*)
using *assms* **by** (*intro emeasure_embed_measure'*) (*auto intro*!: *inj_onI dest*: *injD*)

lemma *embed_measure_comp*:
assumes [*simp*]: *inj f inj g*
shows *embed_measure* (*embed_measure M f*) *g* = *embed_measure M* (*g* ∘ *f*)
proof−
have [*simp*]: *inj* (λ*x*. *g* (*f x*)) **by** (*subst o_def*[*symmetric*]) (*auto intro*: *inj_compose*)
note *measurable_embed_measure2*[*measurable*]
have *embed_measure* (*embed_measure M f*) *g* =
distr M (*embed_measure* (*embed_measure M f*) *g*) (*g* ∘ *f*)
by (*subst* (*1 2*) *embed_measure_eq_distr*)
(*simp_all add*: *distr_distr sets_embed_measure cong*: *distr_cong*)
also have ... = *embed_measure M* (*g* ∘ *f*)
by (*subst* (*3*) *embed_measure_eq_distr*, *simp add*: *o_def*, *rule distr_cong*)
(*auto simp*: *sets_embed_measure o_def image_image*[*symmetric*]
intro: *inj_compose cong*: *distr_cong*)
finally show *?thesis* **.**
qed

lemma *sigma_finite_embed_measure*:
assumes *sigma_finite_measure M* **and** *inj*: *inj f*
shows *sigma_finite_measure* (*embed_measure M f*)
proof −
from *assms*(*1*) **interpret** *sigma_finite_measure M* **.**
from *sigma_finite_countable* **obtain** *A* **where**
A_props: *countable A A* ⊆ *sets M* ⋃ *A* = *space M* ⋀*X*. *X*∈*A* ⟹ *emeasure M X* ≠ ∞ **by** *blast*
from *A_props* **have** *countable* ((‘) *f*‘*A*) **by** *auto*
moreover
from *inj* **and** *A_props* **have** (‘) *f*‘*A* ⊆ *sets* (*embed_measure M f*)
by (*auto simp*: *sets_embed_measure*)
moreover
from *A_props* **and** *inj* **have** ⋃((‘) *f*‘*A*) = *space* (*embed_measure M f*)
by (*auto simp*: *space_embed_measure intro*!: *imageI*)
moreover
from *A_props* **and** *inj* **have** ∀ *a*∈(‘) *f* ‘ *A*. *emeasure* (*embed_measure M f*) *a* ≠ ∞
by (*intro ballI*, *subst emeasure_embed_measure*)
(*auto simp*: *inj_vimage_image_eq intro*: *in_sets_embed_measure*)
ultimately show *?thesis* **by** − (*standard*, *blast*)
qed

lemma *embed_measure_count_space′*:
inj_on f A ⟹ embed_measure (count_space A) f = count_space (f‘A)
apply (*subst distr_bij_count_space*[*of f A f‘A*, *symmetric*])
apply (*simp add*: *inj_on_def bij_betw_def*)
apply (*subst embed_measure_eq_distr′*)
apply *simp*
apply(*auto 4 3 intro*!: *measure_eqI imageI simp add*: *sets_embed_measure′ subset_image_iff*)
apply (*subst* (*1 2*) *emeasure_distr*)
apply (*auto simp*: *space_embed_measure sets_embed_measure′*)
done

lemma *embed_measure_count_space*:
inj f ⟹ embed_measure (count_space A) f = count_space (f‘A)
by(*rule embed_measure_count_space′*)(*erule subset_inj_on*, *simp*)

lemma *sets_embed_measure_alt*:
inj f ⟹ sets (embed_measure M f) = ((‘) f) ‘ sets M
by (*auto simp*: *sets_embed_measure*)

lemma *emeasure_embed_measure_image′*:
assumes *inj_on f* (*space M*) *X* ∈ *sets M*
shows *emeasure* (*embed_measure M f*) (*f‘X*) = *emeasure M X*
proof−
from *assms* **have** *emeasure* (*embed_measure M f*) (*f‘X*) = *emeasure M* (*f* −‘ *f* ‘ *X* ∩ *space M*)
by (*subst emeasure_embed_measure′*) (*auto simp*: *sets_embed_measure′*)
also from *assms* **have** *f* −‘ *f* ‘ *X* ∩ *space M* = *X* **by** (*auto dest*: *inj_onD sets.sets_into_space*)
finally show *?thesis* **.**
qed

lemma *emeasure_embed_measure_image*:
inj f ⟹ X ∈ *sets M ⟹ emeasure* (*embed_measure M f*) (*f‘X*) = *emeasure M X*
by (*simp_all add*: *emeasure_embed_measure in_sets_embed_measure inj_vimage_image_eq*)

lemma *embed_measure_eq_iff*:
assumes *inj f*
shows *embed_measure A f* = *embed_measure B f* ⟷ *A* = *B* (**is** *?M* = *?N* ⟷ _)
proof
from *assms* **have** *I*: *inj* ((‘) *f*) **by** (*auto intro*: *injI dest*: *injD*)
assume *asm*: *?M* = *?N*
hence *sets* (*embed_measure A f*) = *sets* (*embed_measure B f*) **by** *simp*
with *assms* **have** *sets A* = *sets B* **by** (*simp only*: *I inj_image_eq_iff sets_embed_measure_alt*)
moreover {
fix *X* **assume** *X* ∈ *sets A*
from *asm* **have** *emeasure ?M* (*f‘X*) = *emeasure ?N* (*f‘X*) **by** *simp*

```
    with ‹X ∈ sets A› and ‹sets A = sets B› and assms
     have emeasure A X = emeasure B X by (simp add: emeasure_embed_measure_image)
  }
  ultimately show A = B by (rule measure_eqI)
qed simp

lemma the_inv_into_in_Pi: inj_on f A ⟹ the_inv_into A f ∈ f ‘ A → A
  by (auto simp: the_inv_into_f_f)

lemma map_prod_image: map_prod f g ‘ (A × B) = (f‘A) × (g‘B)
  using map_prod_surj_on[OF refl refl] .

lemma map_prod_vimage: map_prod f g −‘ (A × B) = (f−‘A) × (g−‘B)
  by auto

lemma embed_measure_prod:
 assumes f: inj f and g: inj g and [simp]: sigma_finite_measure M sigma_finite_measure
N
   shows embed_measure M f ⨂M embed_measure N g = embed_measure (M
⨂M N) (λ(x, y). (f x, g y))
    (is ?L = _)
  unfolding map_prod_def[symmetric]
proof (rule pair_measure_eqI)
  have fg[simp]: ⋀A. inj_on (map_prod f g) A ⋀A. inj_on f A ⋀A. inj_on g A
    using f g by (auto simp: inj_on_def)

  note complete_lattice_class.Sup_insert[simp del] ccSup_insert[simp del]
     ccSUP_insert[simp del]
  show sets: sets ?L = sets (embed_measure (M ⨂M N) (map_prod f g))
    unfolding map_prod_def[symmetric]
    apply (simp add: sets_pair_eq_sets_fst_snd sets_embed_eq_vimage_algebra
      cong: vimage_algebra_cong)
    apply (subst sets_vimage_Sup_eq[where Y=space (M ⨂M N)])
    apply (simp_all add: space_pair_measure[symmetric])
    apply (auto simp add: the_inv_into_f_f
                simp del: map_prod_simp
                del: prod_fun_imageE) []
    apply auto []
    apply (subst (1 2 3 4 ) vimage_algebra_vimage_algebra_eq)
   apply (simp_all add: the_inv_into_in_Pi Pi_iff[of snd] Pi_iff[of fst] space_pair_measure)
    apply (simp_all add: Pi_iff[of snd] Pi_iff[of fst] the_inv_into_in_Pi vim-
age_algebra_vimage_algebra_eq
       space_pair_measure[symmetric] map_prod_image[symmetric])
   apply (intro arg_cong[where f=sets] arg_cong[where f=Sup] arg_cong2[where
f=insert] vimage_algebra_cong)
    apply (auto simp: map_prod_image the_inv_into_f_f
                simp del: map_prod_simp del: prod_fun_imageE)
    apply (simp_all add: the_inv_into_f_f space_pair_measure)
    done
```

note *measurable_embed_measure2*[*measurable*]
fix *A B* **assume** *AB*: *A* ∈ *sets* (*embed_measure M f*) *B* ∈ *sets* (*embed_measure N g*)
moreover have *f* −‘ *A* × *g* −‘ *B* ∩ *space* (*M* ⨂$_M$ *N*) = (*f* −‘ *A* ∩ *space M*) × (*g* −‘ *B* ∩ *space N*)
by (*auto simp*: *space_pair_measure*)
ultimately show *emeasure* (*embed_measure M f*) *A* ∗ *emeasure* (*embed_measure N g*) *B* =
emeasure (*embed_measure* (*M* ⨂$_M$ *N*) (*map_prod f g*)) (*A* × *B*)
by (*simp add*: *map_prod_vimage sets*[*symmetric*] *emeasure_embed_measure sigma_finite_measure.emeasure_pair_measure_Times*)
qed (*insert assms*, *simp_all add*: *sigma_finite_embed_measure*)

lemma *mono_embed_measure*:
space M = *space M′* ⟹ *sets M* ⊆ *sets M′* ⟹ *sets* (*embed_measure M f*) ⊆ *sets* (*embed_measure M′ f*)
unfolding *embed_measure_def*
apply (*subst* (*1 2*) *sets_measure_of*)
apply (*blast dest*: *sets.sets_into_space*)
apply (*blast dest*: *sets.sets_into_space*)
apply *simp*
apply (*intro sigma_sets_mono′*)
apply *safe*
apply (*simp add*: *subset_eq*)
apply *metis*
done

lemma *density_embed_measure*:
assumes *inj*: *inj f* **and** *Mg*[*measurable*]: *g* ∈ *borel_measurable* (*embed_measure M f*)
shows *density* (*embed_measure M f*) *g* = *embed_measure* (*density M* (*g* ∘ *f*)) *f* (**is** *?M1* = *?M2*)
proof (*rule measure_eqI*)
fix *X* **assume** *X*: *X* ∈ *sets ?M1*
from *inj* **have** *Mf*[*measurable*]: *f* ∈ *measurable M* (*embed_measure M f*)
by (*rule measurable_embed_measure2*)
from *Mg* **and** *X* **have** *emeasure ?M1 X* = ∫⁺ *x*. *g x* ∗ *indicator X x* ∂*embed_measure M f*
by (*subst emeasure_density*) *simp_all*
also from *X* **have** ... = ∫⁺ *x*. *g* (*f x*) ∗ *indicator X* (*f x*) ∂*M*
by (*subst embed_measure_eq_distr*[*OF inj*], *subst nn_integral_distr*) *auto*
also have ... = ∫⁺ *x*. *g* (*f x*) ∗ *indicator* (*f* −‘ *X* ∩ *space M*) *x* ∂*M*
by (*intro nn_integral_cong*) (*auto split*: *split_indicator*)
also from *X* **have** ... = *emeasure* (*density M* (*g* ∘ *f*)) (*f* −‘ *X* ∩ *space M*)
by (*subst emeasure_density*) (*simp_all add*: *measurable_comp*[*OF Mf Mg*] *measurable_sets*[*OF Mf*])
also from *X* **and** *inj* **have** ... = *emeasure ?M2 X*
by (*subst emeasure_embed_measure*) (*simp_all add*: *sets_embed_measure*)

finally show *emeasure ?M1 X = emeasure ?M2 X* **.**
qed (*simp_all add*: *sets_embed_measure inj*)

lemma *density_embed_measure'*:
assumes *inj*: *inj f* **and** *inv*: $\bigwedge x.\ f'\ (f\ x) = x$ **and** *Mg*[*measurable*]: *g* ∈ *borel_measurable M*
shows *density* (*embed_measure M f*) (*g* ∘ *f'*) = *embed_measure* (*density M g*) *f*
proof−
have *density* (*embed_measure M f*) (*g* ∘ *f'*) = *embed_measure* (*density M* (*g* ∘ *f'* ∘ *f*)) *f*
by (*rule density_embed_measure*[*OF inj*])
(*rule measurable_comp*, *rule measurable_embed_measure1*, *subst measurable_cong*,
rule inv, *rule measurable_ident_sets*, *simp*, *rule Mg*)
also have *density M* (*g* ∘ *f'* ∘ *f*) = *density M g*
by (*intro density_cong*) (*subst measurable_cong*, *simp add*: *o_def inv*, *simp_all add*: *Mg inv*)
finally show *?thesis* **.**
qed

lemma *inj_on_image_subset_iff*:
assumes *inj_on f C A* ⊆ *C B* ⊆ *C*
shows *f* ‘ *A* ⊆ *f* ‘ *B* ⟷ *A* ⊆ *B*
proof (*intro iffI subsetI*)
fix *x* **assume** *A*: *f* ‘ *A* ⊆ *f* ‘ *B* **and** *B*: *x* ∈ *A*
from *B* **have** *f x* ∈ *f* ‘ *A* **by** *blast*
with *A* **have** *f x* ∈ *f* ‘ *B* **by** *blast*
then obtain *y* **where** *f x* = *f y* **and** *y* ∈ *B* **by** *blast*
with *assms* **and** *B* **have** *x* = *y* **by** (*auto dest*: *inj_onD*)
with ‹*y* ∈ *B*› **show** *x* ∈ *B* **by** *simp*
qed *auto*

lemma *AE_embed_measure'*:
assumes *inj*: *inj_on f* (*space M*)
shows (*AE x in embed_measure M f*. *P x*) ⟷ (*AE x in M*. *P* (*f x*))
proof
let *?M* = *embed_measure M f*
assume *AE x in ?M*. *P x*
then obtain *A* **where** *A_props*: *A* ∈ *sets ?M emeasure ?M A* = *0* {*x*∈*space ?M*. ¬*P x*} ⊆ *A*
by (*force elim*: *AE_E*)
then obtain *A'* **where** *A'_props*: *A* = *f* ‘ *A' A'* ∈ *sets M* **by** (*auto simp*: *sets_embed_measure' inj*)
moreover have *B*: {*x*∈*space ?M*. ¬*P x*} = *f* ‘ {*x*∈*space M*. ¬*P* (*f x*)}
by (*auto simp*: *inj space_embed_measure*)
from *A_props*(*3*) **have** {*x*∈*space M*. ¬*P* (*f x*)} ⊆ *A'*
by (*subst* (*asm*) *B*, *subst* (*asm*) *A'_props*, *subst* (*asm*) *inj_on_image_subset_iff*[*OF inj*])

```
      (insert A'_props, auto dest: sets.sets_into_space)
  moreover from A_props A'_props have emeasure M A' = 0
    by (simp add: emeasure_embed_measure_image' inj)
  ultimately show AE x in M. P (f x) by (intro AE_I)
next
  let ?M = embed_measure M f
  assume AE x in M. P (f x)
  then obtain A where A_props: A ∈ sets M emeasure M A = 0 {x∈space M.
¬P (f x)} ⊆ A
    by (force elim: AE_E)
  hence f`A ∈ sets ?M emeasure ?M (f`A) = 0 {x∈space ?M. ¬P x} ⊆ f`A
   by (auto simp: space_embed_measure emeasure_embed_measure_image' sets_embed_measure'
inj)
  thus AE x in ?M. P x by (intro AE_I)
qed

lemma AE_embed_measure:
  assumes inj: inj f
  shows (AE x in embed_measure M f. P x) ⟷ (AE x in M. P (f x))
  using assms by (intro AE_embed_measure') (auto intro!: inj_onI dest: injD)

lemma nn_integral_monotone_convergence_SUP_countable:
  fixes f :: 'a ⇒ 'b ⇒ ennreal
  assumes nonempty: Y ≠ {}
  and chain: Complete_Partial_Order.chain (≤) (f ` Y)
  and countable: countable B
  shows (∫⁺ x. (SUP i∈Y. f i x) ∂count_space B) = (SUP i∈Y. (∫⁺ x. f i x
∂count_space B))
  (is ?lhs = ?rhs)
proof −
  let ?f = (λi x. f i (from_nat_into B x) ∗ indicator (to_nat_on B ` B) x)
   have ?lhs = ∫⁺ x. (SUP i∈Y. f i (from_nat_into B (to_nat_on B x)))
∂count_space B
    by(rule nn_integral_cong)(simp add: countable)
  also have ... = ∫⁺ x. (SUP i∈Y. f i (from_nat_into B x)) ∂count_space
(to_nat_on B ` B)
     by(simp add: embed_measure_count_space'[symmetric] inj_on_to_nat_on
countable nn_integral_embed_measure' measurable_embed_measure1)
  also have ... = ∫⁺ x. (SUP i∈Y. ?f i x) ∂count_space UNIV
   by(simp add: nn_integral_count_space_indicator ennreal_indicator[symmetric]
SUP_mult_right_ennreal nonempty)
  also have ... = (SUP i∈Y. ∫⁺ x. ?f i x ∂count_space UNIV)
  proof(rule nn_integral_monotone_convergence_SUP_nat)
    show Complete_Partial_Order.chain (≤) (?f ` Y)
     by(rule chain_imageI[OF chain, unfolded image_image])(auto intro!: le_funI
split: split_indicator dest: le_funD)
  qed fact
  also have ... = (SUP i∈Y. ∫⁺ x. f i (from_nat_into B x) ∂count_space
(to_nat_on B ` B))
```

by(*simp add*: *nn_integral_count_space_indicator*)
also have ... = (*SUP i∈Y*. ∫⁺ *x*. *f i* (*from_nat_into B* (*to_nat_on B x*)) ∂*count_space B*)
by(*simp add*: *embed_measure_count_space'*[*symmetric*] *inj_on_to_nat_on countable nn_integral_embed_measure' measurable_embed_measure1*)
also have ... = *?rhs*
by(*intro arg_cong2*[**where** *f* = λ*A f*. *Sup* (*f* ‘ *A*)] *ext nn_integral_cong_AE*)(*simp_all add*: *AE_count_space countable*)
finally show *?thesis* **.**
qed

end

6.28 Brouwer's Fixed Point Theorem

theory *Brouwer_Fixpoint*
imports *Homeomorphism Derivative*
begin

6.28.1 Retractions

lemma *retract_of_contractible*:
assumes *contractible T S retract_of T*
shows *contractible S*
using *assms*
apply (*clarsimp simp add*: *retract_of_def contractible_def retraction_def homotopic_with image_subset_iff_funcset*)
apply (*rule_tac x*=*r a* **in** *exI*)
apply (*rule_tac x*=*r* ∘ *h* **in** *exI*)
apply (*intro conjI continuous_intros continuous_on_compose*)
apply (*erule continuous_on_subset* | *force*)+
done

lemma *retract_of_path_connected*:
⟦*path_connected T*; *S retract_of T*⟧ ⟹ *path_connected S*
by (*metis path_connected_continuous_image retract_of_def retraction*)

lemma *retract_of_simply_connected*:
⟦*simply_connected T*; *S retract_of T*⟧ ⟹ *simply_connected S*
apply (*simp add*: *retract_of_def retraction_def Pi_iff*, *clarify*)
apply (*rule simply_connected_retraction_gen*)
apply (*force elim*!: *continuous_on_subset*)+
done

lemma *retract_of_homotopically_trivial*:
assumes *ts*: *T retract_of S*
and *hom*: ⋀*f g*. ⟦*continuous_on U f*; *f* ∈ *U* → *S*;
continuous_on U g; *g* ∈ *U* → *S*⟧
⟹ *homotopic_with_canon* (λ*x*. *True*) *U S f g*

```
      and continuous_on U f f ∈ U → T
      and continuous_on U g g ∈ U → T
    shows homotopic_with_canon (λx. True) U T f g
proof −
  obtain r where r ∈ S → S continuous_on S r ∀x∈S. r (r x) = r x T = r ‘ S
    using ts by (auto simp: retract_of_def retraction)
  then obtain k where Retracts S r T k
    unfolding Retracts_def using continuous_on_id by blast
  then show ?thesis
    apply (rule Retracts.homotopically_trivial_retraction_gen)
    using assms
    apply (force simp: hom)+
    done
qed

lemma retract_of_homotopically_trivial_null:
  assumes ts: T retract_of S
      and hom: ⋀f. ⟦continuous_on U f; f ∈ U → S⟧
                   ⟹ ∃c. homotopic_with_canon (λx. True) U S f (λx. c)
      and continuous_on U f f ∈ U → T
  obtains c where homotopic_with_canon (λx. True) U T f (λx. c)
proof −
  obtain r where r ∈ S → S continuous_on S r ∀x∈S. r (r x) = r x T = r ‘ S
    using ts by (auto simp: retract_of_def retraction)
  then obtain k where Retracts S r T k
    unfolding Retracts_def by fastforce
  then show ?thesis
    apply (rule Retracts.homotopically_trivial_retraction_null_gen)
    apply (rule TrueI refl assms that | assumption)+
    done
qed

lemma retraction_openin_vimage_iff:
  openin (top_of_set S) (S ∩ r −‘ U) ⟷ openin (top_of_set T) U
  if retraction S T r and U ⊆ T
  by (simp add: retraction_openin_vimage_iff that)

lemma retract_of_locally_compact:
    fixes S :: 'a :: {heine_borel,real_normed_vector} set
    shows ⟦locally compact S; T retract_of S⟧ ⟹ locally compact T
  by (metis locally_compact_closedin closedin_retract)

lemma homotopic_into_retract:
   ⟦f ∈ S → T; g ∈ S → T; T retract_of U; homotopic_with_canon (λx. True)
S U f g⟧
       ⟹ homotopic_with_canon (λx. True) S T f g
  apply (subst (asm) homotopic_with_def)
  apply (simp add: homotopic_with retract_of_def retraction_def Pi_iff, clarify)
  apply (rule_tac x=r ∘ h in exI)
```

by (*smt* (*verit*, *ccfv_SIG*) *comp_def continuous_on_compose continuous_on_subset image_subset_iff*)

lemma *retract_of_locally_connected*:
assumes *locally connected T S retract_of T*
shows *locally connected S*
using *assms*
by (*metis Abstract_Topology_2.retraction_openin_vimage_iff idempotent_imp_retraction locally_connected_quotient_image retract_ofE*)

lemma *retract_of_locally_path_connected*:
assumes *locally path_connected T S retract_of T*
shows *locally path_connected S*
using *assms*
by (*metis Abstract_Topology_2.retraction_openin_vimage_iff idempotent_imp_retraction locally_path_connected_quotient_image retract_ofE*)

A few simple lemmas about deformation retracts

lemma *deformation_retract_imp_homotopy_eqv*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *homotopic_with_canon* (*λx. True*) *S S id r* **and** *r*: *retraction S T r*
shows *S homotopy_eqv T*
proof −
have *homotopic_with_canon* (*λx. True*) *S S* (*id* ∘ *r*) *id*
by (*simp add*: *assms*(*1*) *homotopic_with_symD*)
moreover have *homotopic_with_canon* (*λx. True*) *T T* (*r* ∘ *id*) *id*
using *r* **unfolding** *retraction_def*
by (*metis eq_id_iff homotopic_with_id2 topspace_euclidean_subtopology*)
ultimately
show *?thesis*
unfolding *homotopy_equivalent_space_def*
by (*smt* (*verit*, *del_insts*) *continuous_map_id continuous_map_subtopology_eu id_def r retraction retraction_comp subset_refl*)
qed

lemma *deformation_retract*:
fixes *S* :: *'a*::*euclidean_space set*
shows (∃*r*. *homotopic_with_canon* (*λx. True*) *S S id r* ∧ *retraction S T r*)
⟷
T retract_of S ∧ (∃*f*. *homotopic_with_canon* (*λx. True*) *S S id f* ∧ *f* ∈ *S* → *T*)
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (*auto simp*: *retract_of_def retraction_def*)
next
assume *?rhs*
then show *?lhs*

```
    apply (clarsimp simp add: retract_of_def retraction_def)
    apply (rule_tac x=r in exI, simp)
     apply (rule homotopic_with_trans, assumption)
     apply (rule_tac f = r ∘ f and g=r ∘ id in homotopic_with_eq)
       apply (rule_tac Y=S in homotopic_with_compose_continuous_left)
        apply (auto simp: homotopic_with_sym Pi_iff)
    done
qed

lemma deformation_retract_of_contractible_sing:
  fixes S :: 'a::euclidean_space set
  assumes contractible S a ∈ S
  obtains r where homotopic_with_canon (λx. True) S S id r retraction S {a} r
proof −
  have {a} retract_of S
    by (simp add: ‹a ∈ S›)
  moreover have homotopic_with_canon (λx. True) S S id (λx. a)
     using assms
    by (auto simp: contractible_def homotopic_into_contractible image_subset_iff)
  moreover have (λx. a) ∈ S → {a}
    by (simp add: image_subsetI)
  ultimately show ?thesis
    by (metis that deformation_retract)
qed


lemma continuous_on_compact_surface_projection_aux:
  fixes S :: 'a::t2_space set
  assumes compact S S ⊆ T image q T ⊆ S
      and contp: continuous_on T p
      and ⋀x. x ∈ S ⟹ q x = x
      and [simp]: ⋀x. x ∈ T ⟹ q(p x) = q x
      and ⋀x. x ∈ T ⟹ p(q x) = p x
    shows continuous_on T q
proof −
  have ∗: image p T = image p S
    using assms by auto (metis imageI subset_iff)
  have contp': continuous_on S p
    by (rule continuous_on_subset [OF contp ‹S ⊆ T›])
  have continuous_on (p ‘ T) q
   by (simp add: ∗ assms(1) assms(2) assms(5) continuous_on_inv contp' rev_subsetD)
  then have continuous_on T (q ∘ p)
    by (rule continuous_on_compose [OF contp])
  then show ?thesis
    by (rule continuous_on_eq [of _ q ∘ p]) (simp add: o_def)
qed

lemma continuous_on_compact_surface_projection:
  fixes S :: 'a::real_normed_vector set
```

assumes *compact S*
and *S*: $S \subseteq V - \{0\}$ **and** *cone V*
and *iff*: $\bigwedge x\ k.\ x \in V - \{0\} \Longrightarrow 0 < k \wedge (k *_R x) \in S \longleftrightarrow d\ x = k$
shows *continuous_on* $(V - \{0\})$ $(\lambda x.\ d\ x *_R x)$
proof (*rule continuous_on_compact_surface_projection_aux* [*OF* ‹*compact S*› *S*])
show $(\lambda x.\ d\ x *_R x)$ ‘ $(V - \{0\}) \subseteq S$
using *iff* **by** *auto*
show *continuous_on* $(V - \{0\})$ $(\lambda x.\ inverse(norm\ x) *_R x)$
by (*intro continuous_intros*) *force*
show $\bigwedge x.\ x \in S \Longrightarrow d\ x *_R x = x$
by (*metis S zero_less_one local.iff scaleR_one subset_eq*)
show $d\ (x /_R norm\ x) *_R (x /_R norm\ x) = d\ x *_R x$ **if** $x \in V - \{0\}$ **for** x
using *iff* [*of inverse*(*norm x*) $*_R$ *x norm x* $*$ *d x*, *symmetric*] *iff that* ‹*cone V*›
by (*simp add*: *field_simps cone_def zero_less_mult_iff*)
show $d\ x *_R x /_R norm\ (d\ x *_R x) = x /_R norm\ x$ **if** $x \in V - \{0\}$ **for** x
proof −
have $0 < d\ x$
using *local.iff that* **by** *blast*
then show *?thesis*
by *simp*
qed
qed

6.28.2 Kuhn Simplices

lemma *bij_betw_singleton_eq*:
assumes *f*: *bij_betw f A B* **and** *g*: *bij_betw g A B* **and** *a*: $a \in A$
assumes *eq*: $(\bigwedge x.\ x \in A \Longrightarrow x \neq a \Longrightarrow f\ x = g\ x)$
shows $f\ a = g\ a$
proof −
have f ‘ $(A - \{a\}) = g$ ‘ $(A - \{a\})$
by (*intro image_cong*) (*simp_all add*: *eq*)
then have $B - \{f\ a\} = B - \{g\ a\}$
using *f g a* **by** (*auto simp*: *bij_betw_def inj_on_image_set_diff set_eq_iff*)
moreover have $f\ a \in B\ g\ a \in B$
using *f g a* **by** (*auto simp*: *bij_betw_def*)
ultimately show *?thesis*
by *auto*
qed

lemmas *swap_apply1* = *swap_apply*(*1*)
lemmas *swap_apply2* = *swap_apply*(*2*)

lemma *pointwise_minimal_pointwise_maximal*:
fixes *s* :: (*nat* ⇒ *nat*) *set*
assumes *finite s*
and $s \neq \{\}$
and $\forall x{\in}s.\ \forall y{\in}s.\ x \leq y \vee y \leq x$

shows $\exists a{\in}s.\ \forall x{\in}s.\ a \le x$
and $\exists a{\in}s.\ \forall x{\in}s.\ x \le a$
using *assms*
proof (*induct s rule*: *finite_ne_induct*)
case (*insert b s*)
assume $*$: $\forall x{\in}$*insert b s*. $\forall y{\in}$*insert b s*. $x \le y \vee y \le x$
then obtain *u l* **where** $l \in s\ \forall b{\in}s.\ l \le b\ u \in s\ \forall b{\in}s.\ b \le u$
using *insert* **by** *auto*
with $*$ **show** $\exists a{\in}$*insert b s*. $\forall x{\in}$*insert b s*. $a \le x\ \exists a{\in}$*insert b s*. $\forall x{\in}$*insert b s*. $x \le a$
by (*metis insert_iff order.trans*)+
qed *auto*

lemma *kuhn_labelling_lemma*:
fixes *P Q* :: *'a*::*euclidean_space* $\Rightarrow$ *bool*
assumes $\forall x.\ P\ x \longrightarrow P\ (f\ x)$
and $\forall x.\ P\ x \longrightarrow (\forall i{\in}$*Basis*. $Q\ i \longrightarrow 0 \le x{\cdot}i \wedge x{\cdot}i \le 1)$
shows $\exists l.\ (\forall x.\forall i{\in}$*Basis*. $l\ x\ i \le (1{::}nat)) \wedge$
$(\forall x.\forall i{\in}$*Basis*. $P\ x \wedge Q\ i \wedge (x{\cdot}i = 0) \longrightarrow (l\ x\ i = 0)) \wedge$
$(\forall x.\forall i{\in}$*Basis*. $P\ x \wedge Q\ i \wedge (x{\cdot}i = 1) \longrightarrow (l\ x\ i = 1)) \wedge$
$(\forall x.\forall i{\in}$*Basis*. $P\ x \wedge Q\ i \wedge (l\ x\ i = 0) \longrightarrow x{\cdot}i \le f\ x{\cdot}i) \wedge$
$(\forall x.\forall i{\in}$*Basis*. $P\ x \wedge Q\ i \wedge (l\ x\ i = 1) \longrightarrow f\ x{\cdot}i \le x{\cdot}i)$
proof −
{ fix *x i*
let *?R* = $\lambda y.\ (P\ x \wedge Q\ i \wedge x \cdot i = 0 \longrightarrow y = (0{::}nat)) \wedge$
$(P\ x \wedge Q\ i \wedge x \cdot i = 1 \longrightarrow y = 1) \wedge$
$(P\ x \wedge Q\ i \wedge y = 0 \longrightarrow x \cdot i \le f\ x \cdot i) \wedge$
$(P\ x \wedge Q\ i \wedge y = 1 \longrightarrow f\ x \cdot i \le x \cdot i)$
{ assume *P x Q i i* $\in$ *Basis* **with** *assms* **have** $0 \le f\ x \cdot i \wedge f\ x \cdot i \le 1$ **by** *auto* **}**
then have $i \in$ *Basis* $\Longrightarrow$ *?R 0* $\vee$ *?R 1* **by** *auto* **}**
then show *?thesis*
unfolding *all_conj_distrib*[*symmetric*] *Ball_def*
by (*subst choice_iff*[*symmetric*])+ *blast*
qed

The key "counting" observation, somewhat abstracted

lemma *kuhn_counting_lemma*:
fixes *bnd compo compo' face S F*
defines *nF s* == *card* $\{f{\in}F.$ *face f s* $\wedge$ *compo' f*$\}$
assumes [*simp*, *intro*]: *finite F* — faces **and** [*simp*, *intro*]: *finite S* — simplices
and $\bigwedge f.\ f \in F \Longrightarrow$ *bnd f* $\Longrightarrow$ *card* $\{s{\in}S.$ *face f s*$\} = 1$
and $\bigwedge f.\ f \in F \Longrightarrow \neg$ *bnd f* $\Longrightarrow$ *card* $\{s{\in}S.$ *face f s*$\} = 2$
and $\bigwedge s.\ s \in S \Longrightarrow$ *compo s* $\Longrightarrow$ *nF s* $= 1$
and $\bigwedge s.\ s \in S \Longrightarrow \neg$ *compo s* $\Longrightarrow$ *nF s* $= 0 \vee$ *nF s* $= 2$
and *odd* (*card* $\{f{\in}F.$ *compo' f* $\wedge$ *bnd f*$\}$)
shows *odd* (*card* $\{s{\in}S.$ *compo s*$\}$)
proof −

have $(\sum s \mid s \in S \wedge \neg\ compo\ s.\ nF\ s) + (\sum s \mid s \in S \wedge compo\ s.\ nF\ s) = (\sum s{\in}S.\ nF\ s)$
by (*subst sum.union_disjoint*[*symmetric*]) (*auto intro*!: *sum.cong*)
also have $\ldots = (\sum s{\in}S.\ card\ \{f \in \{f{\in}F.\ compo'\ f \wedge bnd\ f\}.\ face\ f\ s\}) +$
$(\sum s{\in}S.\ card\ \{f \in \{f{\in}F.\ compo'\ f \wedge \neg\ bnd\ f\}.\ face\ f\ s\})$
unfolding *sum.distrib*[*symmetric*]
by (*subst card_Un_disjoint*[*symmetric*])
(*auto simp*: *nF_def intro*!: *sum.cong arg_cong*[**where** *f*=*card*])
also have $\ldots = 1 * card\ \{f{\in}F.\ compo'\ f \wedge bnd\ f\} + 2 * card\ \{f{\in}F.\ compo'\ f \wedge \neg\ bnd\ f\}$
using *assms*(*4*,*5*) **by** (*fastforce intro*!: *arg_cong2*[**where** *f*=(+)] *sum_multicount*)
finally have *odd* $((\sum s \mid s \in S \wedge \neg\ compo\ s.\ nF\ s) + card\ \{s{\in}S.\ compo\ s\})$
using *assms*(*6*,*8*) **by** *simp*
moreover have $(\sum s \mid s \in S \wedge \neg\ compo\ s.\ nF\ s) =$
$(\sum s \mid s \in S \wedge \neg\ compo\ s \wedge nF\ s = 0.\ nF\ s) + (\sum s \mid s \in S \wedge \neg\ compo\ s \wedge nF\ s = 2.\ nF\ s)$
using *assms*(*7*) **by** (*subst sum.union_disjoint*[*symmetric*]) (*fastforce intro*!: *sum.cong*)+
ultimately show *?thesis*
by *auto*
qed

The odd/even result for faces of complete vertices, generalized

lemma *kuhn_complete_lemma*:
assumes [*simp*]: *finite simplices*
and *face*: $\bigwedge f\ s.\ face\ f\ s \longleftrightarrow (\exists\, a{\in}s.\ f = s - \{a\})$
and *card_s*[*simp*]: $\bigwedge s.\ s \in simplices \Longrightarrow card\ s = n + 2$
and *rl_bd*: $\bigwedge s.\ s \in simplices \Longrightarrow rl\ `\ s \subseteq \{..Suc\ n\}$
and *bnd*: $\bigwedge f\ s.\ s \in simplices \Longrightarrow face\ f\ s \Longrightarrow bnd\ f \Longrightarrow card\ \{s{\in}simplices.\ face\ f\ s\} = 1$
and *nbnd*: $\bigwedge f\ s.\ s \in simplices \Longrightarrow face\ f\ s \Longrightarrow \neg\ bnd\ f \Longrightarrow card\ \{s{\in}simplices.\ face\ f\ s\} = 2$
and *odd_card*: *odd* $(card\ \{f.\ (\exists\, s{\in}simplices.\ face\ f\ s) \wedge rl\ `\ f = \{..n\} \wedge bnd\ f\})$
shows *odd* $(card\ \{s{\in}simplices.\ (rl\ `\ s = \{..Suc\ n\})\})$
proof (*rule kuhn_counting_lemma*)
have *finite_s*[*simp*]: $\bigwedge s.\ s \in simplices \Longrightarrow finite\ s$
by (*metis add_is_0 zero_neq_numeral card.infinite assms*(*3*))

let *?F* $= \{f.\ \exists\, s{\in}simplices.\ face\ f\ s\}$
have *F_eq*: *?F* $= (\bigcup s{\in}simplices.\ \bigcup a{\in}s.\ \{s - \{a\}\})$
by (*auto simp*: *face*)
show *finite ?F*
using ‹*finite simplices*› **unfolding** *F_eq* **by** *auto*

show $card\ \{s \in simplices.\ face\ f\ s\} = 1$ **if** $f \in ?F\ bnd\ f$ **for** *f*
using *bnd that* **by** *auto*

show $card\ \{s \in simplices.\ face\ f\ s\} = 2$ **if** $f \in ?F \neg\ bnd\ f$ **for** *f*

```
  using nbnd that by auto

show odd (card {f ∈ {f. ∃ s∈simplices. face f s}. rl ‘ f = {..n} ∧ bnd f})
  using odd_card by simp

fix s assume s[simp]: s ∈ simplices
let ?S = {f ∈ {f. ∃ s∈simplices. face f s}. face f s ∧ rl ‘ f = {..n}}
have ?S = (λa. s − {a}) ‘ {a∈s. rl ‘ (s − {a}) = {..n}}
  using s by (fastforce simp: face)
then have card_S: card ?S = card {a∈s. rl ‘ (s − {a}) = {..n}}
  by (auto intro!: card_image inj_onI)

{ assume rl: rl ‘ s = {..Suc n}
  then have inj_rl: inj_on rl s
    by (intro eq_card_imp_inj_on) auto
  moreover obtain a where rl a = Suc n a ∈ s
    by (metis atMost_iff image_iff le_Suc_eq rl)
  ultimately have n: {..n} = rl ‘ (s − {a})
    by (auto simp: inj_on_image_set_diff rl)
  have {a∈s. rl ‘ (s − {a}) = {..n}} = {a}
    using inj_rl ‹a ∈ s› by (auto simp: n inj_on_image_eq_iff[OF inj_rl])
  then show card ?S = 1
    unfolding card_S by simp }

{ assume rl: rl ‘ s ≠ {..Suc n}
  show card ?S = 0 ∨ card ?S = 2
  proof cases
    assume ∗: {..n} ⊆ rl ‘ s
    with rl rl_bd[OF s] have rl_s: rl ‘ s = {..n}
      by (auto simp: atMost_Suc subset_insert_iff split: if_split_asm)
    then have ¬ inj_on rl s
      by (intro pigeonhole) simp
    then obtain a b where ab: a ∈ s b ∈ s rl a = rl b a ≠ b
      by (auto simp: inj_on_def)
    then have eq: rl ‘ (s − {a}) = rl ‘ s
      by auto
    with ab have inj: inj_on rl (s − {a})
      by (intro eq_card_imp_inj_on) (auto simp: rl_s card_Diff_singleton_if)

    { fix x assume x ∈ s x ∉ {a, b}
      then have rl ‘ s − {rl x} = rl ‘ ((s − {a}) − {x})
        by (auto simp: eq inj_on_image_set_diff[OF inj])
      also have ... = rl ‘ (s − {x})
        using ab ‹x ∉ {a, b}› by auto
      also assume ... = rl ‘ s
      finally have False
        using ‹x∈s› by auto }
    moreover
    { fix x assume x ∈ {a, b} with ab have x ∈ s ∧ rl ‘ (s − {x}) = rl ‘ s
```

```
        by (simp add: set_eq_iff image_iff Bex_def) metis }
    ultimately have {a∈s. rl ‘ (s − {a}) = {..n}} = {a, b}
      unfolding rl_s[symmetric] by fastforce
    with ‹a ≠ b› show card ?S = 0 ∨ card ?S = 2
      unfolding card_S by simp
  next
    assume ¬ {..n} ⊆ rl ‘ s
    then have ⋀x. rl ‘ (s − {x}) ≠ {..n}
      by auto
    then show card ?S = 0 ∨ card ?S = 2
      unfolding card_S by simp
  qed }
qed fact

locale kuhn_simplex =
  fixes p n and base upd and S :: (nat ⇒ nat) set
  assumes base: base ∈ {..< n} → {..< p}
  assumes base_out: ⋀i. n ≤ i ⟹ base i = p
  assumes upd: bij_betw upd {..< n} {..< n}
  assumes s_pre: S = (λi j. if j ∈ upd‘{..< i} then Suc (base j) else base j) ‘ {..
n}
begin

definition enum i j = (if j ∈ upd‘{..< i} then Suc (base j) else base j)

lemma s_eq: S = enum ‘ {.. n}
  unfolding s_pre enum_def[abs_def] ..

lemma upd_space: i < n ⟹ upd i < n
  using upd by (auto dest!: bij_betwE)

lemma s_space: S ⊆ {..< n} → {.. p}
proof −
  { fix i assume i ≤ n then have enum i ∈ {..< n} → {.. p}
    proof (induct i)
      case 0 then show ?case
        using base by (auto simp: Pi_iff less_imp_le enum_def)
    next
      case (Suc i) with base show ?case
        by (auto simp: Pi_iff Suc_le_eq less_imp_le enum_def intro: upd_space)
    qed }
  then show ?thesis
    by (auto simp: s_eq)
qed

lemma inj_upd: inj_on upd {..< n}
  using upd by (simp add: bij_betw_def)

lemma inj_enum: inj_on enum {.. n}
```

proof −
{ fix $x\ y$ :: *nat* **assume** $x \neq y\ x \leq n\ y \leq n$
with *upd* **have** $upd\ `\ \{..< x\} \neq upd\ `\ \{..< y\}$
by (*subst inj_on_image_eq_iff*[**where** C=$\{..< n\}$]) (*auto simp*: *bij_betw_def*)
then have $enum\ x \neq enum\ y$
by (*auto simp*: *enum_def fun_eq_iff*) **}**
then show *?thesis*
by (*auto simp*: *inj_on_def*)
qed

lemma *enum_0*: $enum\ 0 = base$
by (*simp add*: *enum_def*[*abs_def*])

lemma *base_in_s*: $base \in S$
unfolding *s_eq* **by** (*subst enum_0*[*symmetric*]) *auto*

lemma *enum_in*: $i \leq n \Longrightarrow enum\ i \in S$
unfolding *s_eq* **by** *auto*

lemma *one_step*:
assumes *a*: $a \in S\ j < n$
assumes ∗: $\bigwedge a'.\ a' \in S \Longrightarrow a' \neq a \Longrightarrow a'\ j = p'$
shows $a\ j \neq p'$
proof
assume $a\ j = p'$
with ∗ a **have** $\bigwedge a'.\ a' \in S \Longrightarrow a'\ j = p'$
by *auto*
then have $\bigwedge i.\ i \leq n \Longrightarrow enum\ i\ j = p'$
unfolding *s_eq* **by** *auto*
from *this*[*of 0*] *this*[*of n*] **have** $j \notin upd\ `\ \{..< n\}$
by (*auto simp*: *enum_def fun_eq_iff split*: *if_split_asm*)
with *upd* ‹$j < n$› **show** *False*
by (*auto simp*: *bij_betw_def*)
qed

lemma *upd_inj*: $i < n \Longrightarrow j < n \Longrightarrow upd\ i = upd\ j \longleftrightarrow i = j$
using *upd* **by** (*auto simp*: *bij_betw_def inj_on_eq_iff*)

lemma *upd_surj*: $upd\ `\ \{..< n\} = \{..< n\}$
using *upd* **by** (*auto simp*: *bij_betw_def*)

lemma *in_upd_image*: $A \subseteq \{..< n\} \Longrightarrow i < n \Longrightarrow upd\ i \in upd\ `\ A \longleftrightarrow i \in A$
using *inj_on_image_mem_iff*[*of upd* $\{..< n\}$] *upd*
by (*auto simp*: *bij_betw_def*)

lemma *enum_inj*: $i \leq n \Longrightarrow j \leq n \Longrightarrow enum\ i = enum\ j \longleftrightarrow i = j$
using *inj_enum* **by** (*auto simp*: *inj_on_eq_iff*)

lemma *in_enum_image*: $A \subseteq \{..\ n\} \Longrightarrow i \leq n \Longrightarrow enum\ i \in enum\ `\ A \longleftrightarrow i \in$

A
using *inj_on_image_mem_iff*[*OF inj_enum*] **by** *auto*

lemma *enum_mono*: $i \leq n \Longrightarrow j \leq n \Longrightarrow enum\ i \leq enum\ j \longleftrightarrow i \leq j$
by (*auto simp*: *enum_def le_fun_def in_upd_image Ball_def*[*symmetric*])

lemma *enum_strict_mono*: $i \leq n \Longrightarrow j \leq n \Longrightarrow enum\ i < enum\ j \longleftrightarrow i < j$
using *enum_mono*[*of i j*] *enum_inj*[*of i j*] **by** (*auto simp*: *le_less*)

lemma *chain*: $a \in S \Longrightarrow b \in S \Longrightarrow a \leq b \vee b \leq a$
by (*auto simp*: *s_eq enum_mono*)

lemma *less*: $a \in S \Longrightarrow b \in S \Longrightarrow a\ i < b\ i \Longrightarrow a < b$
using *chain*[*of a b*] **by** (*auto simp*: *less_fun_def le_fun_def not_le*[*symmetric*])

lemma *enum_0_bot*: $a \in S \Longrightarrow a = enum\ 0 \longleftrightarrow (\forall\, a' \in S.\ a \leq a')$
unfolding *s_eq* **by** (*auto simp*: *enum_mono Ball_def*)

lemma *enum_n_top*: $a \in S \Longrightarrow a = enum\ n \longleftrightarrow (\forall\, a' \in S.\ a' \leq a)$
unfolding *s_eq* **by** (*auto simp*: *enum_mono Ball_def*)

lemma *enum_Suc*: $i < n \Longrightarrow enum\ (Suc\ i) = (enum\ i)(upd\ i := Suc\ (enum\ i\ (upd\ i)))$
by (*auto simp*: *fun_eq_iff enum_def upd_inj*)

lemma *enum_eq_p*: $i \leq n \Longrightarrow n \leq j \Longrightarrow enum\ i\ j = p$
by (*induct i*) (*auto simp*: *enum_Suc enum_0 base_out upd_space not_less*[*symmetric*])

lemma *out_eq_p*: $a \in S \Longrightarrow n \leq j \Longrightarrow a\ j = p$
unfolding *s_eq* **by** (*auto simp*: *enum_eq_p*)

lemma *s_le_p*: $a \in S \Longrightarrow a\ j \leq p$
using *out_eq_p*[*of a j*] *s_space* **by** (*cases j < n*) *auto*

lemma *le_Suc_base*: $a \in S \Longrightarrow a\ j \leq Suc\ (base\ j)$
unfolding *s_eq* **by** (*auto simp*: *enum_def*)

lemma *base_le*: $a \in S \Longrightarrow base\ j \leq a\ j$
unfolding *s_eq* **by** (*auto simp*: *enum_def*)

lemma *enum_le_p*: $i \leq n \Longrightarrow j < n \Longrightarrow enum\ i\ j \leq p$
using *enum_in*[*of i*] *s_space* **by** *auto*

lemma *enum_less*: $a \in S \Longrightarrow i < n \Longrightarrow enum\ i < a \longleftrightarrow enum\ (Suc\ i) \leq a$
unfolding *s_eq* **by** (*auto simp*: *enum_strict_mono enum_mono*)

lemma *ksimplex_0*:
$n = 0 \Longrightarrow S = \{(\lambda x.\ p)\}$
using *s_eq enum_def base_out* **by** *auto*

lemma *replace_0*:
assumes $j < n$ $a \in S$ **and** p: $\forall x{\in}S - \{a\}.\ x\ j = 0$ **and** $x \in S$
shows $x \leq a$
proof *cases*
assume $x \neq a$
have $a\ j \neq 0$
using *assms* **by** (*intro one_step*[**where** $a{=}a$]) *auto*
with *less*[*OF* ‹$x{\in}S$› ‹$a{\in}S$›, *of* j] p[*rule_format*, *of* x] ‹$x \in S$› ‹$x \neq a$›
show *?thesis*
by *auto*
qed *simp*

lemma *replace_1*:
assumes $j < n$ $a \in S$ **and** p: $\forall x{\in}S - \{a\}.\ x\ j = p$ **and** $x \in S$
shows $a \leq x$
proof *cases*
assume $x \neq a$
have $a\ j \neq p$
using *assms* **by** (*intro one_step*[**where** $a{=}a$]) *auto*
with *enum_le_p*[*of* _ j] ‹$j < n$› ‹$a{\in}S$›
have $a\ j < p$
by (*auto simp*: *less_le s_eq*)
with *less*[*OF* ‹$a{\in}S$› ‹$x{\in}S$›, *of* j] p[*rule_format*, *of* x] ‹$x \in S$› ‹$x \neq a$›
show *?thesis*
by *auto*
qed *simp*

end

locale *kuhn_simplex_pair* = s: *kuhn_simplex* $p\ n\ b\_s\ u\_s\ s$ + t: *kuhn_simplex* $p\ n\ b\_t\ u\_t\ t$
for $p\ n\ b\_s\ u\_s\ s\ b\_t\ u\_t\ t$
begin

lemma *enum_eq*:
assumes l: $i \leq l\ l \leq j$ **and** $j + d \leq n$
assumes *eq*: $s.enum\ `\ \{i\ ..\ j\} = t.enum\ `\ \{i + d\ ..\ j + d\}$
shows $s.enum\ l = t.enum\ (l + d)$
using l **proof** (*induct* l *rule*: *dec_induct*)
case *base*
then have s: $s.enum\ i \in t.enum\ `\ \{i + d\ ..\ j + d\}$ **and** $t$: $t.enum\ (i + d) \in s.enum\ `\ \{i\ ..\ j\}$
using *eq* **by** *auto*
from t ‹$i \leq j$› ‹$j + d \leq n$› **have** $s.enum\ i \leq t.enum\ (i + d)$
by (*auto simp*: *s.enum_mono*)
moreover from s ‹$i \leq j$› ‹$j + d \leq n$› **have** $t.enum\ (i + d) \leq s.enum\ i$
by (*auto simp*: *t.enum_mono*)
ultimately show *?case*

```
    by auto
next
  case (step l)
  moreover from step.prems ‹j + d ≤ n› have
      s.enum l < s.enum (Suc l)
      t.enum (l + d) < t.enum (Suc l + d)
    by (simp_all add: s.enum_strict_mono t.enum_strict_mono)
  moreover have
      s.enum (Suc l) ∈ t.enum ‘ {i + d .. j + d}
      t.enum (Suc l + d) ∈ s.enum ‘ {i .. j}
    using step ‹j + d ≤ n› eq by (auto simp: s.enum_inj t.enum_inj)
  ultimately have s.enum (Suc l) = t.enum (Suc (l + d))
    using ‹j + d ≤ n›
    by (intro antisym s.enum_less[THEN iffD1] t.enum_less[THEN iffD1])
       (auto intro!: s.enum_in t.enum_in)
  then show ?case by simp
qed

lemma ksimplex_eq_bot:
  assumes a: a ∈ s ⋀a'. a' ∈ s ⟹ a ≤ a'
  assumes b: b ∈ t ⋀b'. b' ∈ t ⟹ b ≤ b'
  assumes eq: s - {a} = t - {b}
  shows s = t
proof cases
  assume n = 0 with s.ksimplex_0 t.ksimplex_0 show ?thesis by simp
next
  assume n ≠ 0
  have s.enum 0 = (s.enum (Suc 0)) (u_s 0 := s.enum (Suc 0) (u_s 0) - 1)
       t.enum 0 = (t.enum (Suc 0)) (u_t 0 := t.enum (Suc 0) (u_t 0) - 1)
    using ‹n ≠ 0› by (simp_all add: s.enum_Suc t.enum_Suc)
  moreover have e0: a = s.enum 0 b = t.enum 0
    using a b by (simp_all add: s.enum_0_bot t.enum_0_bot)
  moreover
  { fix j assume 0 < j j ≤ n
    moreover have s - {a} = s.enum ‘ {Suc 0 .. n} t - {b} = t.enum ‘ {Suc 0
.. n}
      unfolding s.s_eq t.s_eq e0 by (auto simp: s.enum_inj t.enum_inj)
    ultimately have s.enum j = t.enum j
      using enum_eq[of 1 j n 0] eq by auto }
  note enum_eq = this
  then have s.enum (Suc 0) = t.enum (Suc 0)
    using ‹n ≠ 0› by auto
  moreover
  { fix j assume Suc j < n
    with enum_eq[of Suc j] enum_eq[of Suc (Suc j)]
    have u_s (Suc j) = u_t (Suc j)
      using s.enum_Suc[of Suc j] t.enum_Suc[of Suc j]
      by (auto simp: fun_eq_iff split: if_split_asm) }
  then have ⋀j. 0 < j ⟹ j < n ⟹ u_s j = u_t j
```

```
    by (auto simp: gr0_conv_Suc)
  with ‹n ≠ 0› have u_t 0 = u_s 0
    by (intro bij_betw_singleton_eq[OF t.upd s.upd, of 0]) auto
  ultimately have a = b
    by simp
  with assms show s = t
    by auto
qed

lemma ksimplex_eq_top:
  assumes a: a ∈ s ⋀a'. a' ∈ s ⟹ a' ≤ a
  assumes b: b ∈ t ⋀b'. b' ∈ t ⟹ b' ≤ b
  assumes eq: s - {a} = t - {b}
  shows s = t
proof (cases n)
  assume n = 0 with s.ksimplex_0 t.ksimplex_0 show ?thesis by simp
next
  case (Suc n')
  have s.enum n = (s.enum n') (u_s n' := Suc (s.enum n' (u_s n')))
       t.enum n = (t.enum n') (u_t n' := Suc (t.enum n' (u_t n')))
    using Suc by (simp_all add: s.enum_Suc t.enum_Suc)
  moreover have en: a = s.enum n b = t.enum n
    using a b by (simp_all add: s.enum_n_top t.enum_n_top)
  moreover
  { fix j assume j < n
    moreover have s - {a} = s.enum ‘ {0 .. n'} t - {b} = t.enum ‘ {0 .. n'}
      unfolding s.s_eq t.s_eq en by (auto simp: s.enum_inj t.enum_inj Suc)
    ultimately have s.enum j = t.enum j
      using enum_eq[of 0 j n' 0] eq Suc by auto }
  note enum_eq = this
  then have s.enum n' = t.enum n'
    using Suc by auto
  moreover
  { fix j assume j < n'
    with enum_eq[of j] enum_eq[of Suc j]
    have u_s j = u_t j
      using s.enum_Suc[of j] t.enum_Suc[of j]
      by (auto simp: Suc fun_eq_iff split: if_split_asm) }
  then have ⋀j. j < n' ⟹ u_s j = u_t j
    by (auto simp: gr0_conv_Suc)
  then have u_t n' = u_s n'
    by (intro bij_betw_singleton_eq[OF t.upd s.upd, of n']) (auto simp: Suc)
  ultimately have a = b
    by simp
  with assms show s = t
    by auto
qed

end
```

```
inductive ksimplex for p n :: nat where
  ksimplex: kuhn_simplex p n base upd s ⟹ ksimplex p n s

lemma finite_ksimplexes: finite {s. ksimplex p n s}
proof (rule finite_subset)
  { fix a s assume ksimplex p n s a ∈ s
    then obtain b u where kuhn_simplex p n b u s by (auto elim: ksimplex.cases)
    then interpret kuhn_simplex p n b u s .
    from s_space ‹a ∈ s› out_eq_p[OF ‹a ∈ s›]
    have a ∈ (λf x. if n ≤ x then p else f x) ` ({..< n} →E {.. p})
      by (auto simp: image_iff subset_eq Pi_iff split: if_split_asm
            intro!: bexI[of _ restrict a {..< n}]) }
  then show {s. ksimplex p n s} ⊆ Pow ((λf x. if n ≤ x then p else f x) ` ({..<
n} →E {.. p}))
    by auto
qed (simp add: finite_PiE)

lemma ksimplex_card:
  assumes ksimplex p n s shows card s = Suc n
using assms proof cases
  case (ksimplex u b)
  then interpret kuhn_simplex p n u b s .
  show ?thesis
    by (simp add: card_image s_eq inj_enum)
qed

lemma simplex_top_face:
  assumes 0 < p ∀ x∈s'. x n = p
  shows ksimplex p n s' ⟷ (∃ s a. ksimplex p (Suc n) s ∧ a ∈ s ∧ s' = s − {a})
  using assms
proof safe
  fix s a assume ksimplex p (Suc n) s and a: a ∈ s and na: ∀ x∈s − {a}. x n =
p
  then show ksimplex p n (s − {a})
  proof cases
    case (ksimplex base upd)
    then interpret kuhn_simplex p Suc n base upd s .

    have a n < p
      using one_step[of a n p] na ‹a∈s› s_space by (auto simp: less_le)
    then have a = enum 0
      using ‹a ∈ s› na by (subst enum_0_bot) (auto simp: le_less intro!: less[of a
_ n])
    then have s_eq: s − {a} = enum ` Suc ` {.. n}
    using s_eq by (simp add: atMost_Suc_eq_insert_0 insert_ident in_enum_image
subset_eq)
    then have enum 1 ∈ s − {a}
      by auto
```

```
    then have upd 0 = n
      using ‹a n < p› ‹a = enum 0› na[rule_format, of enum 1]
      by (auto simp: fun_eq_iff enum_Suc split: if_split_asm)
    then have bij_betw upd (Suc ` {..< n}) {..< n}
      using upd
      by (subst notIn_Un_bij_betw3[where b=0])
        (auto simp: lessThan_Suc[symmetric] lessThan_Suc_eq_insert_0)
    then have bij_betw (upd∘Suc) {..<n} {..<n}
      by (rule bij_betw_trans[rotated]) (auto simp: bij_betw_def)

    have a n = p - 1
      using enum_Suc[of 0] na[rule_format, OF ‹enum 1 ∈ s - {a}›] ‹a = enum
0› by (auto simp: ‹upd 0 = n›)

    show ?thesis
    proof (rule ksimplex.intros, standard)
      show bij_betw (upd∘Suc) {..< n} {..< n} by fact
      show base(n := p) ∈ {..<n} → {..<p} ⋀i. n≤i ⟹ (base(n := p)) i = p
        using base base_out by (auto simp: Pi_iff)

      have ⋀i. Suc ` {..< i} = {..< Suc i} - {0}
        by (auto simp: image_iff Ball_def) arith
      then have upd_Suc: ⋀i. i ≤ n ⟹ (upd∘Suc) ` {..< i} = upd ` {..< Suc i}
- {n}
      using ‹upd 0 = n› upd_inj by (auto simp add: image_iff less_Suc_eq_0_disj)
      have n_in_upd: ⋀i. n ∈ upd ` {..< Suc i}
        using ‹upd 0 = n› by auto

      define f' where f' i j =
        (if j ∈ (upd∘Suc)`{..< i} then Suc ((base(n := p)) j) else (base(n := p)) j)
for i j
      { fix x i
        assume i [arith]: i ≤ n
        with upd_Suc have (upd ∘ Suc) ` {..<i} = upd ` {..<Suc i} - {n} .
        with ‹a n < p› ‹a = enum 0› ‹upd 0 = n› ‹a n = p - 1›
        have enum (Suc i) x = f' i x
          by (auto simp add: f'_def enum_def) }
      then show s - {a} = f' ` {.. n}
        unfolding s_eq image_comp by (intro image_cong) auto
    qed
  qed
next
  assume ksimplex p n s' and ∗: ∀x∈s'. x n = p
  then show ∃s a. ksimplex p (Suc n) s ∧ a ∈ s ∧ s' = s - {a}
  proof cases
    case (ksimplex base upd)
    then interpret kuhn_simplex p n base upd s' .
    define b where b = base (n := p - 1)
    define u where u i = (case i of 0 ⇒ n | Suc i ⇒ upd i) for i
```

```
    have ksimplex p (Suc n) (s' ∪ {b})
    proof (rule ksimplex.intros, standard)
      show b ∈ {..<Suc n} → {..<p}
       using base ‹0 < p› unfolding lessThan_Suc b_def by (auto simp: PiE_iff)
      show ⋀i. Suc n ≤ i ⟹ b i = p
        using base_out by (auto simp: b_def)

      have bij_betw u (Suc ` {..< n} ∪ {0}) ({..<n} ∪ {u 0})
        using upd
       by (intro notIn_Un_bij_betw) (auto simp: u_def bij_betw_def image_comp
comp_def inj_on_def)
      then show bij_betw u {..<Suc n} {..<Suc n}
       by (simp add: u_def lessThan_Suc[symmetric] lessThan_Suc_eq_insert_0)

      define f' where f' i j = (if j ∈ u`{..< i} then Suc (b j) else b j) for i j

      have u_eq: ⋀i. i ≤ n ⟹ u ` {..< Suc i} = upd ` {..< i} ∪ { n }
        by (auto simp: u_def image_iff upd_inj Ball_def split: nat.split) arith

      { fix x have x ≤ n ⟹ n ∉ upd ` {..<x}
          using upd_space by (simp add: image_iff neq_iff) }
      note n_not_upd = this

      have *: f' ` {.. Suc n} = f' ` (Suc ` {.. n} ∪ {0})
        unfolding atMost_Suc_eq_insert_0 by simp
      also have ... = (f' ∘ Suc) ` {.. n} ∪ {b}
        by (auto simp: f'_def)
      also have (f' ∘ Suc) ` {.. n} = s'
        using ‹0 < p› base_out[of n]
        unfolding s_eq enum_def[abs_def] f'_def[abs_def] upd_space
        by (intro image_cong) (simp_all add: u_eq b_def fun_eq_iff n_not_upd)
      finally show s' ∪ {b} = f' ` {.. Suc n} ..
    qed
    moreover have b ∉ s'
      using * ‹0 < p› by (auto simp: b_def)
    ultimately show ?thesis by auto
  qed
qed

lemma ksimplex_replace_0:
  assumes s: ksimplex p n s and a: a ∈ s
  assumes j: j < n and p: ∀x∈s - {a}. x j = 0
  shows card {s'. ksimplex p n s' ∧ (∃b∈s'. s' - {b} = s - {a})} = 1
  using s
proof cases
  case (ksimplex b_s u_s)

  { fix t b assume ksimplex p n t
```

```
    then obtain b_t u_t where kuhn_simplex p n b_t u_t t
      by (auto elim: ksimplex.cases)
    interpret kuhn_simplex_pair p n b_s u_s s b_t u_t t
      by intro_locales fact+

    assume b: b ∈ t t − {b} = s − {a}
    with a j p s.replace_0[of _ a] t.replace_0[of _ b] have s = t
      by (intro ksimplex_eq_top[of a b]) auto }
  then have {s'. ksimplex p n s' ∧ (∃ b∈s'. s' − {b} = s − {a})} = {s}
    using s ‹a ∈ s› by auto
  then show ?thesis
    by simp
qed

lemma ksimplex_replace_1:
  assumes s: ksimplex p n s and a: a ∈ s
  assumes j: j < n and p: ∀ x∈s − {a}. x j = p
  shows card {s'. ksimplex p n s' ∧ (∃ b∈s'. s' − {b} = s − {a})} = 1
  using s
proof cases
  case (ksimplex b_s u_s)

  { fix t b assume ksimplex p n t
    then obtain b_t u_t where kuhn_simplex p n b_t u_t t
      by (auto elim: ksimplex.cases)
    interpret kuhn_simplex_pair p n b_s u_s s b_t u_t t
      by intro_locales fact+

    assume b: b ∈ t t − {b} = s − {a}
    with a j p s.replace_1[of _ a] t.replace_1[of _ b] have s = t
      by (intro ksimplex_eq_bot[of a b]) auto }
  then have {s'. ksimplex p n s' ∧ (∃ b∈s'. s' − {b} = s − {a})} = {s}
    using s ‹a ∈ s› by auto
  then show ?thesis
    by simp
qed

lemma ksimplex_replace_2:
  assumes s: ksimplex p n s and a ∈ s and n ≠ 0
    and lb: ∀ j<n. ∃ x∈s − {a}. x j ≠ 0
    and ub: ∀ j<n. ∃ x∈s − {a}. x j ≠ p
  shows card {s'. ksimplex p n s' ∧ (∃ b∈s'. s' − {b} = s − {a})} = 2
  using s
proof cases
  case (ksimplex base upd)
  then interpret kuhn_simplex p n base upd s .

  from ‹a ∈ s› obtain i where i ≤ n a = enum i
    unfolding s_eq by auto
```

```
  from ‹i ≤ n› have i = 0 ∨ i = n ∨ (0 < i ∧ i < n)
    by linarith
  then have ∃!s'. s' ≠ s ∧ ksimplex p n s' ∧ (∃ b∈s'. s − {a} = s'− {b})
  proof (elim disjE conjE)
    assume i = 0
    define rot where [abs_def]: rot i = (if i + 1 = n then 0 else i + 1) for i
    let ?upd = upd ∘ rot

    have rot: bij_betw rot {..< n} {..< n}
      by (auto simp: bij_betw_def inj_on_def image_iff Ball_def rot_def)
         arith+
    from rot upd have bij_betw ?upd {..<n} {..<n}
      by (rule bij_betw_trans)

    define f' where [abs_def]: f' i j =
      (if j ∈ ?upd‘{..< i} then Suc (enum (Suc 0) j) else enum (Suc 0) j) for i j

    interpret b: kuhn_simplex p n enum (Suc 0) upd ∘ rot f' ‘ {.. n}
    proof
      from ‹a = enum i› ub ‹n ≠ 0› ‹i = 0›
      obtain i' where i' ≤ n enum i' ≠ enum 0 enum i' (upd 0) ≠ p
        unfolding s_eq by (auto intro: upd_space simp: enum_inj)
      then have enum 1 ≤ enum i' enum i' (upd 0) < p
          using enum_le_p[of i' upd 0] by (auto simp: enum_inj enum_mono
upd_space)
      then have enum 1 (upd 0) < p
        by (auto simp: le_fun_def intro: le_less_trans)
      then show enum (Suc 0) ∈ {..<n} → {..<p}
          using base ‹n ≠ 0› by (auto simp: enum_0 enum_Suc PiE_iff exten-
sional_def upd_space)

      { fix i assume n ≤ i then show enum (Suc 0) i = p
        using ‹n ≠ 0› by (auto simp: enum_eq_p) }
      show bij_betw ?upd {..<n} {..<n} by fact
    qed (simp add: f'_def)
    have ks_f': ksimplex p n (f' ‘ {.. n})
      by rule unfold_locales

    have b_enum: b.enum = f' unfolding f'_def b.enum_def[abs_def] ..
    with b.inj_enum have inj_f': inj_on f' {.. n} by simp

    have f'_eq_enum: f' j = enum (Suc j) if j < n for j
    proof −
      from that have rot ‘ {..< j} = {0 <..< Suc j}
        by (auto simp: rot_def image_Suc_lessThan cong: image_cong_simp)
      with that ‹n ≠ 0› show ?thesis
        by (simp only: f'_def enum_def fun_eq_iff image_comp [symmetric])
           (auto simp add: upd_inj)
```

```
    qed
    then have enum ‘ Suc ‘ {..< n} = f' ‘ {..< n}
      by (force simp: enum_inj)
    also have Suc ‘ {..< n} = {.. n} − {0}
      by (auto simp: image_iff Ball_def) arith
    also have {..< n} = {.. n} − {n}
      by auto
    finally have eq: s − {a} = f' ‘ {.. n} − {f' n}
      unfolding s_eq ‹a = enum i› ‹i = 0›
     by (simp add: inj_on_image_set_diff[OF inj_enum] inj_on_image_set_diff[OF
inj_f'])

    have enum 0 < f' 0
      using ‹n ≠ 0› by (simp add: enum_strict_mono f'_eq_enum)
    also have ... < f' n
      using ‹n ≠ 0› b.enum_strict_mono[of 0 n] unfolding b_enum by simp
    finally have a ≠ f' n
      using ‹a = enum i› ‹i = 0› by auto

    { fix t c assume ksimplex p n t c ∈ t and eq_sma: s − {a} = t − {c}
      obtain b u where kuhn_simplex p n b u t
        using ‹ksimplex p n t› by (auto elim: ksimplex.cases)
      then interpret t: kuhn_simplex p n b u t .

      { fix x assume x ∈ s x ≠ a
        then have x (upd 0) = enum (Suc 0) (upd 0)
          by (auto simp: ‹a = enum i› ‹i = 0› s_eq enum_def enum_inj) }
      then have eq_upd0: ∀ x∈t−{c}. x (upd 0) = enum (Suc 0) (upd 0)
        unfolding eq_sma[symmetric] by auto
      then have c (upd 0) ≠ enum (Suc 0) (upd 0)
        using ‹n ≠ 0› by (intro t.one_step[OF ‹c∈t› ]) (auto simp: upd_space)
      then have c (upd 0) < enum (Suc 0) (upd 0) ∨ c (upd 0) > enum (Suc 0)
(upd 0)
        by auto
      then have t = s ∨ t = f' ‘ {..n}
      proof (elim disjE conjE)
        assume ∗: c (upd 0) < enum (Suc 0) (upd 0)
        interpret st: kuhn_simplex_pair p n base upd s b u t ..
        { fix x assume x ∈ t with ∗ ‹c∈t› eq_upd0[rule_format, of x] have c ≤ x
            by (auto simp: le_less intro!: t.less[of _ _ upd 0]) }
        note top = this
        have s = t
          using ‹a = enum i› ‹i = 0› ‹c ∈ t›
          by (intro st.ksimplex_eq_bot[OF _ _ _ _ eq_sma])
             (auto simp: s_eq enum_mono t.s_eq t.enum_mono top)
        then show ?thesis by simp
      next
        assume ∗: c (upd 0) > enum (Suc 0) (upd 0)
        interpret st: kuhn_simplex_pair p n enum (Suc 0) upd ∘ rot f' ‘ {.. n} b
```

```
u t ..
      have eq: f' ` {..n} - {f' n} = t - {c}
        using eq_sma eq by simp
      { fix x assume x ∈ t with * ‹c∈t› eq_upd0[rule_format, of x] have x ≤ c
          by (auto simp: le_less intro!: t.less[of _ _ upd 0]) }
      note top = this
      have f' ` {..n} = t
        using ‹a = enum i› ‹i = 0› ‹c ∈ t›
        by (intro st.ksimplex_eq_top[OF _ _ _ _ eq])
       (auto simp: b.s_eq b.enum_mono t.s_eq t.enum_mono b_enum[symmetric]
top)
      then show ?thesis by simp
    qed }
  with ks_f' eq ‹a ≠ f' n› ‹n ≠ 0› show ?thesis
    apply (intro ex1I[of _ f' ` {.. n}])
    apply auto []
    apply metis
    done
 next
  assume i = n
  from ‹n ≠ 0› obtain n' where n': n = Suc n'
    by (cases n) auto

  define rot where rot i = (case i of 0 ⇒ n' | Suc i ⇒ i) for i
  let ?upd = upd ∘ rot

  have rot: bij_betw rot {..< n} {..< n}
    by (auto simp: bij_betw_def inj_on_def image_iff Bex_def rot_def n' split:
nat.splits)
       arith
  from rot upd have bij_betw ?upd {..<n} {..<n}
    by (rule bij_betw_trans)

  define b where b = base (upd n' := base (upd n') - 1)
  define f' where [abs_def]: f' i j = (if j ∈ ?upd`{..< i} then Suc (b j) else b
j) for i j

  interpret b: kuhn_simplex p n b upd ∘ rot f' ` {.. n}
  proof
    { fix i assume n ≤ i then show b i = p
        using base_out[of i] upd_space[of n'] by (auto simp: b_def n') }
    show b ∈ {..<n} → {..<p}
      using base ‹n ≠ 0› upd_space[of n']
      by (auto simp: b_def PiE_def Pi_iff Ball_def upd_space extensional_def
n')

    show bij_betw ?upd {..<n} {..<n} by fact
  qed (simp add: f'_def)
  have f': b.enum = f' unfolding f'_def b.enum_def[abs_def] ..
```

```
    have ks_f': ksimplex p n (b.enum ` {.. n})
      unfolding f' by rule unfold_locales

    have 0 < n
      using ‹n ≠ 0› by auto

    { from ‹a = enum i› ‹n ≠ 0› ‹i = n› lb upd_space[of n']
      obtain i' where i' ≤ n enum i' ≠ enum n 0 < enum i' (upd n')
        unfolding s_eq by (auto simp: enum_inj n')
      moreover have enum i' (upd n') = base (upd n')
        unfolding enum_def using ‹i' ≤ n› ‹enum i' ≠ enum n› by (auto simp:
n' upd_inj enum_inj)
      ultimately have 0 < base (upd n')
        by auto }
    then have benum1: b.enum (Suc 0) = base
     unfolding b.enum_Suc[OF ‹0<n›] b.enum_0 by (auto simp: b_def rot_def)

    have [simp]: ⋀j. Suc j < n ⟹ rot ` {..< Suc j} = {n'} ∪ {..< j}
      by (auto simp: rot_def image_iff Ball_def split: nat.splits)
    have rot_simps: ⋀j. rot (Suc j) = j rot 0 = n'
      by (simp_all add: rot_def)

    { fix j assume j: Suc j ≤ n then have b.enum (Suc j) = enum j
      by (induct j) (auto simp: benum1 enum_0 b.enum_Suc enum_Suc rot_simps)
}
    note b_enum_eq_enum = this
    then have enum ` {..< n} = b.enum ` Suc ` {..< n}
      by (auto simp: image_comp intro!: image_cong)
    also have Suc ` {..< n} = {.. n} - {0}
      by (auto simp: image_iff Ball_def) arith
    also have {..< n} = {.. n} - {n}
      by auto
    finally have eq: s - {a} = b.enum ` {.. n} - {b.enum 0}
      unfolding s_eq ‹a = enum i› ‹i = n›
      using inj_on_image_set_diff[OF inj_enum Diff_subset, of {n}]
            inj_on_image_set_diff[OF b.inj_enum Diff_subset, of {0}]
      by (simp add: comp_def)

    have b.enum 0 ≤ b.enum n
      by (simp add: b.enum_mono)
    also have b.enum n < enum n
      using ‹n ≠ 0› by (simp add: enum_strict_mono b_enum_eq_enum n')
    finally have a ≠ b.enum 0
      using ‹a = enum i› ‹i = n› by auto

    { fix t c assume ksimplex p n t c ∈ t and eq_sma: s - {a} = t - {c}
      obtain b' u where kuhn_simplex p n b' u t
        using ‹ksimplex p n t› by (auto elim: ksimplex.cases)
      then interpret t: kuhn_simplex p n b' u t .
```

```
    { fix x assume x ∈ s x ≠ a
      then have x (upd n') = enum n' (upd n')
          by (auto simp: ‹a = enum i› n' ‹i = n› s_eq enum_def enum_inj
in_upd_image) }
    then have eq_upd0: ∀x∈t−{c}. x (upd n') = enum n' (upd n')
      unfolding eq_sma[symmetric] by auto
    then have c (upd n') ≠ enum n' (upd n')
    using ‹n ≠ 0› by (intro t.one_step[OF ‹c∈t› ]) (auto simp: n' upd_space[unfolded
n'])
    then have c (upd n') < enum n' (upd n') ∨ c (upd n') > enum n' (upd n')
      by auto
    then have t = s ∨ t = b.enum ‘ {..n}
    proof (elim disjE conjE)
      assume ∗: c (upd n') > enum n' (upd n')
      interpret st: kuhn_simplex_pair p n base upd s b' u t ..
      { fix x assume x ∈ t with ∗ ‹c∈t› eq_upd0[rule_format, of x] have x ≤ c
          by (auto simp: le_less intro!: t.less[of _ _ upd n']) }
      note top = this
      have s = t
        using ‹a = enum i› ‹i = n› ‹c ∈ t›
        by (intro st.ksimplex_eq_top[OF _ _ _ _ eq_sma])
           (auto simp: s_eq enum_mono t.s_eq t.enum_mono top)
      then show ?thesis by simp
    next
      assume ∗: c (upd n') < enum n' (upd n')
      interpret st: kuhn_simplex_pair p n b upd ∘ rot f' ‘ {.. n} b' u t ..
      have eq: f' ‘ {..n} − {b.enum 0} = t − {c}
        using eq_sma eq f' by simp
      { fix x assume x ∈ t with ∗ ‹c∈t› eq_upd0[rule_format, of x] have c ≤ x
          by (auto simp: le_less intro!: t.less[of _ _ upd n']) }
      note bot = this
      have f' ‘ {..n} = t
        using ‹a = enum i› ‹i = n› ‹c ∈ t›
        by (intro st.ksimplex_eq_bot[OF _ _ _ _ eq])
           (auto simp: b.s_eq b.enum_mono t.s_eq t.enum_mono bot)
      with f' show ?thesis by simp
    qed }
  with ks_f' eq ‹a ≠ b.enum 0› ‹n ≠ 0› show ?thesis
    apply (intro ex1I[of _ b.enum ‘ {.. n}])
    apply fastforce
    apply metis
    done
next
  assume i: 0 < i i < n
  define i' where i' = i − 1
  with i have Suc i' < n
    by simp
  with i have Suc_i': Suc i' = i
```

```
    by (simp add: i'_def)

  let ?upd = Fun.swap i' i upd
  from i upd have bij_betw ?upd {..< n} {..< n}
    by (subst bij_betw_swap_iff) (auto simp: i'_def)

  define f' where [abs_def]: f' i j = (if j ∈ ?upd‘{..< i} then Suc (base j) else
base j)
    for i j
  interpret b: kuhn_simplex p n base ?upd f' ‘ {.. n}
  proof
    show base ∈ {..<n} → {..<p} by (rule base)
    { fix i assume n ≤ i then show base i = p by (rule base_out) }
    show bij_betw ?upd {..<n} {..<n} by fact
  qed (simp add: f'_def)
  have f': b.enum = f' unfolding f'_def b.enum_def[abs_def] ..
  have ks_f': ksimplex p n (b.enum ‘ {.. n})
    unfolding f' by rule unfold_locales

  have {i} ⊆ {..n}
    using i by auto
  { fix j assume j ≤ n
    with i Suc_i' have enum j = b.enum j ⟷ j ≠ i
      unfolding fun_eq_iff enum_def b.enum_def image_comp [symmetric]
      apply (cases ‹i = j›)
       apply (metis imageI in_upd_image lessI lessThan_iff lessThan_subset_iff
order_less_le transpose_apply_first)
        by (metis lessThan_iff linorder_not_less not_less_eq_eq order_less_le
transpose_image_eq)
    }
  note enum_eq_benum = this
  then have enum ‘ ({.. n} - {i}) = b.enum ‘ ({.. n} - {i})
    by (intro image_cong) auto
  then have eq: s - {a} = b.enum ‘ {.. n} - {b.enum i}
    unfolding s_eq ‹a = enum i›
    using inj_on_image_set_diff[OF inj_enum Diff_subset ‹{i} ⊆ {..n}›]
          inj_on_image_set_diff[OF b.inj_enum Diff_subset ‹{i} ⊆ {..n}›]
    by (simp add: comp_def)

  have a ≠ b.enum i
    using ‹a = enum i› enum_eq_benum i by auto

  { fix t c assume ksimplex p n t c ∈ t and eq_sma: s - {a} = t - {c}
    obtain b' u where kuhn_simplex p n b' u t
      using ‹ksimplex p n t› by (auto elim: ksimplex.cases)
    then interpret t: kuhn_simplex p n b' u t .
    have enum i' ∈ s - {a} enum (i + 1) ∈ s - {a}
      using ‹a = enum i› i enum_in by (auto simp: enum_inj i'_def)
    then obtain l k where
```

```
      l: t.enum l = enum i' l ≤ n t.enum l ≠ c and
      k: t.enum k = enum (i + 1) k ≤ n t.enum k ≠ c
      unfolding eq_sma by (auto simp: t.s_eq)
    with i have t.enum l < t.enum k
      by (simp add: enum_strict_mono i'_def)
    with ‹l ≤ n› ‹k ≤ n› have l < k
      by (simp add: t.enum_strict_mono)
    { assume Suc l = k
      have enum (Suc (Suc i')) = t.enum (Suc l)
        using i by (simp add: k ‹Suc l = k› i'_def)
      then have False
        using ‹l < k› ‹k ≤ n› ‹Suc i' < n›
          by (auto simp: t.enum_Suc enum_Suc l upd_inj fun_eq_iff split:
if_split_asm)
          (metis Suc_lessD n_not_Suc_n upd_inj) }
    with ‹l < k› have Suc l < k
      by arith
    have c_eq: c = t.enum (Suc l)
    proof (rule ccontr)
      assume c ≠ t.enum (Suc l)
      then have t.enum (Suc l) ∈ s - {a}
        using ‹l < k› ‹k ≤ n› by (simp add: t.s_eq eq_sma)
      then obtain j where t.enum (Suc l) = enum j j ≤ n enum j ≠ enum i
        unfolding s_eq ‹a = enum i› by auto
      with i have t.enum (Suc l) ≤ t.enum l ∨ t.enum k ≤ t.enum (Suc l)
        by (auto simp: i'_def enum_mono enum_inj l k)
      with ‹Suc l < k› ‹k ≤ n› show False
        by (simp add: t.enum_mono)
    qed

    { have t.enum (Suc (Suc l)) ∈ s - {a}
       unfolding eq_sma c_eq t.s_eq using ‹Suc l < k› ‹k ≤ n› by (auto simp:
t.enum_inj)
      then obtain j where eq: t.enum (Suc (Suc l)) = enum j and j ≤ n j ≠ i
        by (auto simp: s_eq ‹a = enum i›)
      moreover have enum i' < t.enum (Suc (Suc l))
         unfolding l(1)[symmetric] using ‹Suc l < k› ‹k ≤ n› by (auto simp:
t.enum_strict_mono)
      ultimately have i' < j
        using i by (simp add: enum_strict_mono i'_def)
      with ‹j ≠ i› ‹j ≤ n› have t.enum k ≤ t.enum (Suc (Suc l))
        unfolding i'_def by (simp add: enum_mono k eq)
      then have k ≤ Suc (Suc l)
        using ‹k ≤ n› ‹Suc l < k› by (simp add: t.enum_mono) }
    with ‹Suc l < k› have Suc (Suc l) = k by simp
    then have enum (Suc (Suc i')) = t.enum (Suc (Suc l))
      using i by (simp add: k i'_def)
     also have ... = (enum i') (u l := Suc (enum i' (u l)), u (Suc l) := Suc
(enum i' (u (Suc l))))
```

```
      using ‹Suc l < k› ‹k ≤ n› by (simp add: t.enum_Suc l t.upd_inj)
    finally have (u l = upd i' ∧ u (Suc l) = upd (Suc i')) ∨
      (u l = upd (Suc i') ∧ u (Suc l) = upd i')
     using ‹Suc i' < n› by (auto simp: enum_Suc fun_eq_iff split: if_split_asm)

    then have t = s ∨ t = b.enum ‘ {..n}
    proof (elim disjE conjE)
      assume u: u l = upd i'
      have c = t.enum (Suc l) unfolding c_eq ..
      also have t.enum (Suc l) = enum (Suc i')
       using u ‹l < k› ‹k ≤ n› ‹Suc i' < n› by (simp add: enum_Suc t.enum_Suc
l)
      also have ... = a
        using ‹a = enum i› i by (simp add: i'_def)
      finally show ?thesis
        using eq_sma ‹a ∈ s› ‹c ∈ t› by auto
    next
      assume u: u l = upd (Suc i')
      define B where B = b.enum ‘ {..n}
      have b.enum i' = enum i'
        using enum_eq_benum[of i'] i by (auto simp: i'_def gr0_conv_Suc)
      have c = t.enum (Suc l) unfolding c_eq ..
      also have t.enum (Suc l) = b.enum (Suc i')
        using u ‹l < k› ‹k ≤ n› ‹Suc i' < n›
          by (simp_all add: enum_Suc t.enum_Suc l b.enum_Suc ‹b.enum i' =
enum i'›)
            (simp add: Suc_i')
      also have ... = b.enum i
        using i by (simp add: i'_def)
      finally have c = b.enum i .
      then have t - {c} = B - {c} c ∈ B
        unfolding eq_sma[symmetric] eq B_def using i by auto
      with ‹c ∈ t› have t = B
        by auto
      then show ?thesis
        by (simp add: B_def)
    qed }
   with ks_f' eq ‹a ≠ b.enum i› ‹n ≠ 0› ‹i ≤ n› show ?thesis
    apply (intro ex1I[of _ b.enum ‘ {.. n}])
    apply auto []
    apply metis
    done
  qed
  then show ?thesis
    using s ‹a ∈ s› by (simp add: card_2_iff' Ex1_def) metis
qed
```

Hence another step towards concreteness.

lemma *kuhn_simplex_lemma*:

```
  assumes ∀ s. ksimplex p (Suc n) s ⟶ rl ‘ s ⊆ {.. Suc n}
    and odd (card {f. ∃ s a. ksimplex p (Suc n) s ∧ a ∈ s ∧ (f = s − {a}) ∧
      rl ‘ f = {..n} ∧ ((∃ j≤n. ∀ x∈f. x j = 0) ∨ (∃ j≤n. ∀ x∈f. x j = p))})
  shows odd (card {s. ksimplex p (Suc n) s ∧ rl ‘ s = {..Suc n}})
proof (rule kuhn_complete_lemma[OF finite_ksimplexes refl, unfolded mem_Collect_eq,
    where bnd=λf. (∃ j∈{..n}. ∀ x∈f. x j = 0) ∨ (∃ j∈{..n}. ∀ x∈f. x j = p)],
    safe del: notI)

  have *: ⋀x y. x = y ⟹ odd (card x) ⟹ odd (card y)
    by auto
  show odd (card {f. (∃ s∈{s. ksimplex p (Suc n) s}. ∃ a∈s. f = s − {a}) ∧
    rl ‘ f = {..n} ∧ ((∃ j∈{..n}. ∀ x∈f. x j = 0) ∨ (∃ j∈{..n}. ∀ x∈f. x j = p))})
    apply (rule *[OF _ assms(2)])
    apply (auto simp: atLeast0AtMost)
    done

next

  fix s assume s: ksimplex p (Suc n) s
  then show card s = n + 2
    by (simp add: ksimplex_card)

  fix a assume a: a ∈ s then show rl a ≤ Suc n
    using assms(1) s by (auto simp: subset_eq)

  let ?S = {t. ksimplex p (Suc n) t ∧ (∃ b∈t. s − {a} = t − {b})}
  { fix j assume j: j ≤ n ∀ x∈s − {a}. x j = 0
    with s a show card ?S = 1
      using ksimplex_replace_0[of p n + 1 s a j]
      by (subst eq_commute) simp }

  { fix j assume j: j ≤ n ∀ x∈s − {a}. x j = p
    with s a show card ?S = 1
      using ksimplex_replace_1[of p n + 1 s a j]
      by (subst eq_commute) simp }

  { assume card ?S ≠ 2 ¬ (∃ j∈{..n}. ∀ x∈s − {a}. x j = p)
    with s a show ∃ j∈{..n}. ∀ x∈s − {a}. x j = 0
      using ksimplex_replace_2[of p n + 1 s a]
      by (subst (asm) eq_commute) auto }
qed
```

Reduced labelling

```
definition reduced :: nat ⇒ (nat ⇒ nat) ⇒ nat where reduced n x = (LEAST
k. k = n ∨ x k ≠ 0)

lemma reduced_labelling:
  shows reduced n x ≤ n
```

```
    and ∀ i<reduced n x. x i = 0
    and reduced n x = n ∨ x (reduced n x) ≠ 0
proof −
  show reduced n x ≤ n
    unfolding reduced_def by (rule LeastI2_wellorder[where a=n]) auto
  show ∀ i<reduced n x. x i = 0
    unfolding reduced_def by (rule LeastI2_wellorder[where a=n]) fastforce+
  show reduced n x = n ∨ x (reduced n x) ≠ 0
    unfolding reduced_def by (rule LeastI2_wellorder[where a=n]) fastforce+
qed

lemma reduced_labelling_unique:
  r ≤ n ⟹ ∀ i<r. x i = 0 ⟹ r = n ∨ x r ≠ 0 ⟹ reduced n x = r
  by (metis linorder_less_linear linorder_not_le reduced_labelling)

lemma reduced_labelling_zero: j < n ⟹ x j = 0 ⟹ reduced n x ≠ j
  using reduced_labelling[of n x] by auto

lemma reduce_labelling_zero[simp]: reduced 0 x = 0
  by (rule reduced_labelling_unique) auto

lemma reduced_labelling_nonzero: j < n ⟹ x j ≠ 0 ⟹ reduced n x ≤ j
  using reduced_labelling[of n x] by (elim allE[where x=j]) auto

lemma reduced_labelling_Suc: reduced (Suc n) x ≠ Suc n ⟹ reduced (Suc n) x
= reduced n x
  using reduced_labelling[of Suc n x]
  by (intro reduced_labelling_unique[symmetric]) auto

lemma complete_face_top:
  assumes ∀ x∈f. ∀ j≤n. x j = 0 ⟶ lab x j = 0
    and ∀ x∈f. ∀ j≤n. x j = p ⟶ lab x j = 1
    and eq: (reduced (Suc n) ∘ lab) ‘ f = {..n}
  shows ((∃ j≤n. ∀ x∈f. x j = 0) ∨ (∃ j≤n. ∀ x∈f. x j = p)) ⟷ (∀ x∈f. x n = p)
proof (safe del: disjCI)
  fix x j assume j: j ≤ n ∀ x∈f. x j = 0
  { fix x assume x ∈ f with assms j have reduced (Suc n) (lab x) ≠ j
      by (intro reduced_labelling_zero) auto }
  moreover have j ∈ (reduced (Suc n) ∘ lab) ‘ f
    using j eq by auto
  ultimately show x n = p
    by force
next
  fix x j assume j: j ≤ n ∀ x∈f. x j = p and x: x ∈ f
  have j = n
  proof (rule ccontr)
    assume ¬ ?thesis
    { fix x assume x ∈ f
      with assms j have reduced (Suc n) (lab x) ≤ j
```

```
      by (intro reduced_labelling_nonzero) auto
    then have reduced (Suc n) (lab x) ≠ n
      using ‹j ≠ n› ‹j ≤ n› by simp }
  moreover
  have n ∈ (reduced (Suc n) ∘ lab) ‘ f
    using eq by auto
  ultimately show False
    by force
 qed
 moreover have j ∈ (reduced (Suc n) ∘ lab) ‘ f
  using j eq by auto
 ultimately show x n = p
  using j x by auto
qed auto
```

Hence we get just about the nice induction.

```
lemma kuhn_induction:
 assumes 0 < p
  and lab_0: ∀ x. ∀ j≤n. (∀ j. x j ≤ p) ∧ x j = 0 ⟶ lab x j = 0
  and lab_1: ∀ x. ∀ j≤n. (∀ j. x j ≤ p) ∧ x j = p ⟶ lab x j = 1
  and odd: odd (card {s. ksimplex p n s ∧ (reduced n∘lab) ‘ s = {..n}})
 shows odd (card {s. ksimplex p (Suc n) s ∧ (reduced (Suc n)∘lab) ‘ s = {..Suc
n}})
proof −
 let ?rl = reduced (Suc n) ∘ lab and ?ext = λf v. ∃ j≤n. ∀ x∈f. x j = v
 let ?ext = λs. (∃ j≤n. ∀ x∈s. x j = 0) ∨ (∃ j≤n. ∀ x∈s. x j = p)
 have ∀ s. ksimplex p (Suc n) s ⟶ ?rl ‘ s ⊆ {..Suc n}
  by (simp add: reduced_labelling subset_eq)
 moreover
 have {s. ksimplex p n s ∧ (reduced n ∘ lab) ‘ s = {..n}} =
       {f. ∃ s a. ksimplex p (Suc n) s ∧ a ∈ s ∧ f = s − {a} ∧ ?rl ‘ f = {..n} ∧
?ext f}
 proof (intro set_eqI, safe del: disjCI equalityI disjE)
  fix s assume s: ksimplex p n s and rl: (reduced n ∘ lab) ‘ s = {..n}
  from s obtain u b where kuhn_simplex p n u b s by (auto elim: ksimplex.cases)
  then interpret kuhn_simplex p n u b s .
  have all_eq_p: ∀ x∈s. x n = p
   by (auto simp: out_eq_p)
  moreover
  { fix x assume x ∈ s
   with lab_1[rule_format, of n x] all_eq_p s_le_p[of x]
   have ?rl x ≤ n
    by (auto intro!: reduced_labelling_nonzero)
   then have ?rl x = reduced n (lab x)
    by (auto intro!: reduced_labelling_Suc) }
  then have ?rl ‘ s = {..n}
   using rl by (simp cong: image_cong)
  moreover
  obtain t a where ksimplex p (Suc n) t a ∈ t s = t − {a}
```

```
    using s unfolding simplex_top_face[OF ‹0 < p› all_eq_p] by auto
  ultimately
  show ∃t a. ksimplex p (Suc n) t ∧ a ∈ t ∧ s = t − {a} ∧ ?rl ‘ s = {..n} ∧ ?ext s
    by auto
 next
  fix x s a assume s: ksimplex p (Suc n) s and rl: ?rl ‘ (s − {a}) = {.. n}
    and a: a ∈ s and ?ext (s − {a})
   from s obtain u b where kuhn_simplex p (Suc n) u b s by (auto elim: ksimplex.cases)
  then interpret kuhn_simplex p Suc n u b s .
  have all_eq_p: ∀x∈s. x (Suc n) = p
    by (auto simp: out_eq_p)

  { fix x assume x ∈ s − {a}
    then have ?rl x ∈ ?rl ‘ (s − {a})
      by auto
    then have ?rl x ≤ n
      unfolding rl by auto
    then have ?rl x = reduced n (lab x)
      by (auto intro!: reduced_labelling_Suc) }
  then show rl': (reduced n∘lab) ‘ (s − {a}) = {..n}
    unfolding rl[symmetric] by (intro image_cong) auto

  from ‹?ext (s − {a})›
  have all_eq_p: ∀x∈s − {a}. x n = p
  proof (elim disjE exE conjE)
    fix j assume j ≤ n ∀x∈s − {a}. x j = 0
    with lab_0[rule_format, of j] all_eq_p s_le_p
    have ⋀x. x ∈ s − {a} ⟹ reduced (Suc n) (lab x) ≠ j
      by (intro reduced_labelling_zero) auto
    moreover have j ∈ ?rl ‘ (s − {a})
      using ‹j ≤ n› unfolding rl by auto
    ultimately show ?thesis
      by force
  next
    fix j assume j ≤ n and eq_p: ∀x∈s − {a}. x j = p
    show ?thesis
    proof cases
      assume j = n with eq_p show ?thesis by simp
    next
      assume j ≠ n
      { fix x assume x: x ∈ s − {a}
        have reduced n (lab x) ≤ j
        proof (rule reduced_labelling_nonzero)
          show lab x j ≠ 0
            using lab_1[rule_format, of j x] x s_le_p[of x] eq_p ‹j ≤ n› by auto
          show j < n
            using ‹j ≤ n› ‹j ≠ n› by simp
```

```
        qed
        then have reduced n (lab x) ≠ n
          using ‹j ≤ n› ‹j ≠ n› by simp }
       moreover have n ∈ (reduced n∘lab) ‘ (s − {a})
        unfolding rl′ by auto
       ultimately show ?thesis
        by force
     qed
   qed
   show ksimplex p n (s − {a})
    unfolding simplex_top_face[OF ‹0 < p› all_eq_p] using s a by auto
 qed
 ultimately show ?thesis
   using assms by (intro kuhn_simplex_lemma) auto
qed
```

And so we get the final combinatorial result.

```
lemma ksimplex_0: ksimplex p 0 s ⟷ s = {(λx. p)}
proof
 assume ksimplex p 0 s then show s = {(λx. p)}
   by (blast dest: kuhn_simplex.ksimplex_0 elim: ksimplex.cases)
next
 assume s: s = {(λx. p)}
 show ksimplex p 0 s
 proof (intro ksimplex, unfold_locales)
   show (λ_. p) ∈ {..<0::nat} → {..<p} by auto
   show bij_betw id {..<0} {..<0}
     by simp
 qed (auto simp: s)
qed

lemma kuhn_combinatorial:
 assumes 0 < p
   and ∀ x j. (∀ j. x j ≤ p) ∧ j < n ∧ x j = 0 ⟶ lab x j = 0
   and ∀ x j. (∀ j. x j ≤ p) ∧ j < n  ∧ x j = p ⟶ lab x j = 1
 shows odd (card {s. ksimplex p n s ∧ (reduced n∘lab) ‘ s = {..n}})
   (is odd (card (?M n)))
 using assms
proof (induct n)
 case 0 then show ?case
   by (simp add: ksimplex_0 cong: conj_cong)
next
 case (Suc n)
 then have odd (card (?M n))
   by force
 with Suc show ?case
   using kuhn_induction[of p n] by (auto simp: comp_def)
qed
```

lemma *kuhn_lemma*:
  fixes *n p* :: *nat*
  assumes *0 < p*
    and ∀ *x*. (∀ *i*<*n*. *x i* ≤ *p*) ⟶ (∀ *i*<*n*. *label x i* = (*0*::*nat*) ∨ *label x i* = *1*)
    and ∀ *x*. (∀ *i*<*n*. *x i* ≤ *p*) ⟶ (∀ *i*<*n*. *x i* = *0* ⟶ *label x i* = *0*)
    and ∀ *x*. (∀ *i*<*n*. *x i* ≤ *p*) ⟶ (∀ *i*<*n*. *x i* = *p* ⟶ *label x i* = *1*)
  obtains *q* **where** ∀ *i*<*n*. *q i* < *p*
    and ∀ *i*<*n*. ∃ *r s*. (∀ *j*<*n*. *q j* ≤ *r j* ∧ *r j* ≤ *q j* + *1*) ∧ (∀ *j*<*n*. *q j* ≤ *s j* ∧ *s j* ≤ *q j* + *1*) ∧ *label r i* ≠ *label s i*
proof −
  let *?rl* = *reduced n* ∘ *label*
  let *?A* = {*s*. *ksimplex p n s* ∧ *?rl* ‘ *s* = {..*n*}}
  have *odd* (*card ?A*)
    using *assms* **by** (*intro kuhn_combinatorial*[*of p n label*]) *auto*
  then have *?A* ≠ {}
    by (*rule odd_card_imp_not_empty*)
  then obtain *s b u* **where** *kuhn_simplex p n b u s* **and** *rl*: *?rl* ‘ *s* = {..*n*}
    by (*auto elim*: *ksimplex.cases*)
  interpret *kuhn_simplex p n b u s* **by** *fact*

  show *?thesis*
  proof (*intro that*[*of b*] *allI impI*)
    fix *i*
    assume *i* < *n*
    then show *b i* < *p*
      using *base* **by** *auto*
  next
    fix *i*
    assume *i* < *n*
    then have *i* ∈ {.. *n*} *Suc i* ∈ {.. *n*}
      by *auto*
    then obtain *u v* **where** *u*: *u* ∈ *s Suc i* = *?rl u* **and** *v*: *v* ∈ *s i* = *?rl v*
      unfolding *rl*[*symmetric*] **by** *blast*

    have *label u i* ≠ *label v i*
      using *reduced_labelling* [*of n label u*] *reduced_labelling* [*of n label v*]
        u(*2*)[*symmetric*] *v*(*2*)[*symmetric*] ‹*i* < *n*›
      by *auto*
    moreover
    have *b j* ≤ *u j u j* ≤ *b j* + *1 b j* ≤ *v j v j* ≤ *b j* + *1* **if** *j* < *n* **for** *j*
      using *that base_le*[*OF* ‹*u*∈*s*›] *le_Suc_base*[*OF* ‹*u*∈*s*›] *base_le*[*OF* ‹*v*∈*s*›] *le_Suc_base*[*OF* ‹*v*∈*s*›]
      by *auto*
    ultimately show ∃ *r s*. (∀ *j*<*n*. *b j* ≤ *r j* ∧ *r j* ≤ *b j* + *1*) ∧
      (∀ *j*<*n*. *b j* ≤ *s j* ∧ *s j* ≤ *b j* + *1*) ∧ *label r i* ≠ *label s i*
      by *blast*
  qed
qed

Main result for the unit cube

lemma *kuhn_labelling_lemma'*:
  assumes $(\forall x{::}nat \Rightarrow real.\ P\ x \longrightarrow P\ (f\ x))$
    and $\forall x.\ P\ x \longrightarrow (\forall i{::}nat.\ Q\ i \longrightarrow 0 \le x\ i \wedge x\ i \le 1)$
  shows $\exists l.\ (\forall x\ i.\ l\ x\ i \le (1{::}nat)) \wedge$
            $(\forall x\ i.\ P\ x \wedge Q\ i \wedge x\ i = 0 \longrightarrow l\ x\ i = 0) \wedge$
            $(\forall x\ i.\ P\ x \wedge Q\ i \wedge x\ i = 1 \longrightarrow l\ x\ i = 1) \wedge$
            $(\forall x\ i.\ P\ x \wedge Q\ i \wedge l\ x\ i = 0 \longrightarrow x\ i \le f\ x\ i) \wedge$
            $(\forall x\ i.\ P\ x \wedge Q\ i \wedge l\ x\ i = 1 \longrightarrow f\ x\ i \le x\ i)$
  unfolding *all_conj_distrib* [*symmetric*]
  apply (*subst choice_iff*[*symmetric*])+
  by (*metis assms bot_nat_0.extremum nle_le zero_neq_one*)

6.28.3 Brouwer's fixed point theorem

We start proving Brouwer's fixed point theorem for the unit cube = *cbox 0 One*.

lemma *brouwer_cube*:
  fixes $f :: 'a{::}euclidean\_space \Rightarrow 'a$
  assumes *continuous_on* (*cbox 0 One*) *f*
    and *f* ‘ *cbox 0 One* $\subseteq$ *cbox 0 One*
  shows $\exists x{\in}cbox\ 0\ One.\ f\ x = x$
proof (*rule ccontr*)
  define *n* **where** $n = DIM('a)$
  have *n*: $1 \le n\ 0 < n\ n \neq 0$
    unfolding *n_def* **by** (*auto simp*: *Suc_le_eq*)
  assume $\neg$ *?thesis*
  then have $*$: $\neg\ (\exists x{\in}cbox\ 0\ One.\ f\ x - x = 0)$
    by *auto*
  obtain *d* **where**
      d: $d > 0\ \bigwedge x.\ x \in cbox\ 0\ One \Longrightarrow d \le norm\ (f\ x - x)$
    apply (*rule brouwer_compactness_lemma*[*OF compact_cbox _* $*$])
    apply (*rule continuous_intros assms*)+
    apply *blast*
    done
  have $*$: $\forall x.\ x \in cbox\ 0\ One \longrightarrow f\ x \in cbox\ 0\ One$
    $\forall x.\ x \in (cbox\ 0\ One{::}'a\ set) \longrightarrow (\forall i{\in}Basis.\ True \longrightarrow 0 \le x \cdot i \wedge x \cdot i \le 1)$
    using *assms*(*2*)[*unfolded image_subset_iff Ball_def*]
    unfolding *cbox_def*
    by *auto*
  obtain *label* $:: 'a \Rightarrow 'a \Rightarrow nat$ **where** *label* [*rule_format*]:
    $\forall x.\ \forall i{\in}Basis.\ label\ x\ i \le 1$
    $\forall x.\ \forall i{\in}Basis.\ x \in cbox\ 0\ One \wedge x \cdot i = 0 \longrightarrow label\ x\ i = 0$
    $\forall x.\ \forall i{\in}Basis.\ x \in cbox\ 0\ One \wedge x \cdot i = 1 \longrightarrow label\ x\ i = 1$
    $\forall x.\ \forall i{\in}Basis.\ x \in cbox\ 0\ One \wedge label\ x\ i = 0 \longrightarrow x \cdot i \le f\ x \cdot i$
    $\forall x.\ \forall i{\in}Basis.\ x \in cbox\ 0\ One \wedge label\ x\ i = 1 \longrightarrow f\ x \cdot i \le x \cdot i$
    using *kuhn_labelling_lemma*[*OF* $*$] **by** *auto*
  note *label* = *this* [*rule_format*]

have *lem1*: $\forall x{\in}cbox\ 0\ One.\ \forall y{\in}cbox\ 0\ One.\ \forall i{\in}Basis.\ label\ x\ i \neq label\ y\ i \longrightarrow$
$|f\ x \cdot i - x \cdot i| \leq norm\ (f\ y - f\ x) + norm\ (y - x)$
proof *safe*
fix $x\ y :: 'a$
assume x: $x \in cbox\ 0\ One$ **and** y: $y \in cbox\ 0\ One$
fix i
assume i: $label\ x\ i \neq label\ y\ i\ i \in Basis$
have $*$: $\bigwedge x\ y\ fx\ fy :: real.\ x \leq fx \wedge fy \leq y \vee fx \leq x \wedge y \leq fy \Longrightarrow$
$|fx - x| \leq |fy - fx| + |y - x|$ **by** *auto*
have $|(f\ x - x) \cdot i| \leq |(f\ y - f\ x){\cdot}i| + |(y - x){\cdot}i|$
proof (*cases label x i = 0*)
case *True*
then have *fxy*: $\neg\ f\ y \cdot i \leq y \cdot i \Longrightarrow f\ x \cdot i \leq x \cdot i$
by (*metis True i label(1) label(5) le_antisym less_one not_le_imp_less y*)
show *?thesis*
unfolding *inner_simps*
by (*rule* $*$) (*auto simp*: *True i label x y fxy*)
next
case *False*
then show *?thesis*
using *label* [*OF* ‹$i \in Basis$›] *i(1) x y*
apply (*auto simp*: *inner_diff_left le_Suc_eq*)
by (*metis* $*$)
qed
also have $\ldots \leq norm\ (f\ y - f\ x) + norm\ (y - x)$
by (*simp add*: *add_mono i(2) norm_bound_Basis_le*)
finally show $|f\ x \cdot i - x \cdot i| \leq norm\ (f\ y - f\ x) + norm\ (y - x)$
unfolding *inner_simps* **.**
qed
have $\exists e{>}0.\ \forall x{\in}cbox\ 0\ One.\ \forall y{\in}cbox\ 0\ One.\ \forall z{\in}cbox\ 0\ One.\ \forall i{\in}Basis.$
$norm\ (x - z) < e \longrightarrow norm\ (y - z) < e \longrightarrow label\ x\ i \neq label\ y\ i \longrightarrow$
$|(f(z) - z){\cdot}i| < d\ /\ (real\ n)$
proof −
have d': $d\ /\ real\ n\ /\ 8 > 0$
using $d(1)$ **by** (*simp add*: *n_def*)
have $*$: *uniformly_continuous_on* (*cbox 0 One*) *f*
by (*rule compact_uniformly_continuous*[*OF assms(1) compact_cbox*])
obtain e **where** e:
$e > 0$
$\bigwedge x\ x'.\ x \in cbox\ 0\ One \Longrightarrow$
$x' \in cbox\ 0\ One \Longrightarrow$
$norm\ (x' - x) < e \Longrightarrow$
$norm\ (f\ x' - f\ x) < d\ /\ real\ n\ /\ 8$
using $*$[*unfolded uniformly_continuous_on_def*,*rule_format*,*OF d'*]
unfolding *dist_norm*
by *blast*
show *?thesis*
proof (*intro exI conjI ballI impI*)
show $0 < min\ (e\ /\ 2)\ (d\ /\ real\ n\ /\ 8)$

```
      using d' e by auto
    fix x y z i
    assume as:
      x ∈ cbox 0 One y ∈ cbox 0 One z ∈ cbox 0 One
      norm (x − z) < min (e / 2) (d / real n / 8)
      norm (y − z) < min (e / 2) (d / real n / 8)
      label x i ≠ label y i
    assume i: i ∈ Basis
    have ∗: ⋀z fz x fx n1 n2 n3 n4 d4 d :: real. |fx − x| ≤ n1 + n2 ⟹
      |fx − fz| ≤ n3 ⟹ |x − z| ≤ n4 ⟹
      n1 < d4 ⟹ n2 < 2 ∗ d4 ⟹ n3 < d4 ⟹ n4 < d4 ⟹
      (8 ∗ d4 = d) ⟹ |fz − z| < d
      by auto
    show |(f z − z) · i| < d / real n
      unfolding inner_simps
    proof (rule ∗)
      show |f x · i − x · i| ≤ norm (f y −f x) + norm (y − x)
        using as(1) as(2) as(6) i lem1 by blast
      show norm (f x − f z) < d / real n / 8
        using d' e as by auto
      show |f x · i − f z · i| ≤ norm (f x − f z) |x · i − z · i| ≤ norm (x − z)
        unfolding inner_diff_left[symmetric]
        by (rule Basis_le_norm[OF i])+
      have tria: norm (y − x) ≤ norm (y − z) + norm (x − z)
        using dist_triangle[of y x z, unfolded dist_norm]
        unfolding norm_minus_commute
        by auto
      also have ... < e / 2 + e / 2
        using as(4) as(5) by auto
      finally show norm (f y − f x) < d / real n / 8
        using as(1) as(2) e(2) by auto
      have norm (y − z) + norm (x − z) < d / real n / 8 + d / real n / 8
        using as(4) as(5) by auto
      with tria show norm (y − x) < 2 ∗ (d / real n / 8)
        by auto
    qed (use as in auto)
  qed
qed
then
obtain e where e:
  e > 0
  ⋀x y z i. x ∈ cbox 0 One ⟹
    y ∈ cbox 0 One ⟹
    z ∈ cbox 0 One ⟹
    i ∈ Basis ⟹
    norm (x − z) < e ∧ norm (y − z) < e ∧ label x i ≠ label y i ⟹
    |(f z − z) · i| < d / real n
  by blast
obtain p :: nat where p: 1 + real n / e ≤ real p
```

```
  using real_arch_simple ..
 have 1 + real n / e > 0
  using e(1) n by (simp add: add_pos_pos)
 then have p > 0
  using p by auto

 obtain b :: nat ⇒ 'a where b: bij_betw b {..< n} Basis
  by atomize_elim (auto simp: n_def intro!: finite_same_card_bij)
 define b' where b' = inv_into {..< n} b
 then have b': bij_betw b' Basis {..< n}
  using bij_betw_inv_into[OF b] by auto
 then have b'_Basis: ⋀i. i ∈ Basis ⟹ b' i ∈ {..< n}
  unfolding bij_betw_def by (auto simp: set_eq_iff)
 have bb'[simp]:⋀i. i ∈ Basis ⟹ b (b' i) = i
  unfolding b'_def
  using b
  by (auto simp: f_inv_into_f bij_betw_def)
 have b'b[simp]:⋀i. i < n ⟹ b' (b i) = i
  unfolding b'_def
  using b
  by (auto simp: inv_into_f_eq bij_betw_def)
 have *: ⋀x :: nat. x = 0 ∨ x = 1 ⟷ x ≤ 1
  by auto
 have b'': ⋀j. j < n ⟹ b j ∈ Basis
  using b unfolding bij_betw_def by auto
 have q1: 0 < p ∀x. (∀i<n. x i ≤ p) ⟶
  (∀i<n. (label (∑i∈Basis. (real (x (b' i)) / real p) *R i) ∘ b) i = 0 ∨
        (label (∑i∈Basis. (real (x (b' i)) / real p) *R i) ∘ b) i = 1)
  unfolding *
  using ‹p > 0› ‹n > 0›
  using label(1)[OF b'']
  by auto
 { fix x :: nat ⇒ nat and i assume ∀i<n. x i ≤ p i < n x i = p ∨ x i = 0
  then have (∑i∈Basis. (real (x (b' i)) / real p) *R i) ∈ (cbox 0 One::'a set)
   using b'_Basis
   by (auto simp: cbox_def bij_betw_def zero_le_divide_iff divide_le_eq_1) }
 note cube = this
 have q2: ∀x. (∀i<n. x i ≤ p) ⟶ (∀i<n. x i = 0 ⟶
   (label (∑i∈Basis. (real (x (b' i)) / real p) *R i) ∘ b) i = 0)
  unfolding o_def using cube ‹p > 0› by (intro allI impI label(2)) (auto simp:
b'')
 have q3: ∀x. (∀i<n. x i ≤ p) ⟶ (∀i<n. x i = p ⟶
   (label (∑i∈Basis. (real (x (b' i)) / real p) *R i) ∘ b) i = 1)
  using cube ‹p > 0› unfolding o_def by (intro allI impI label(3)) (auto simp:
b'')
 obtain q where q:
   ∀i<n. q i < p
   ∀i<n.
    ∃r s. (∀j<n. q j ≤ r j ∧ r j ≤ q j + 1) ∧
```

```
            (∀j<n. q j ≤ s j ∧ s j ≤ q j + 1) ∧
            (label (∑i∈Basis. (real (r (b' i)) / real p) *R i) ∘ b) i ≠
            (label (∑i∈Basis. (real (s (b' i)) / real p) *R i) ∘ b) i
    by (rule kuhn_lemma[OF q1 q2 q3])
  define z :: 'a where z = (∑i∈Basis. (real (q (b' i)) / real p) *R i)
  have ∃i∈Basis. d / real n ≤ |(f z − z)•i|
  proof (rule ccontr)
    have ∀i∈Basis. q (b' i) ∈ {0..p}
      using q(1) b'
      by (auto intro: less_imp_le simp: bij_betw_def)
    then have z ∈ cbox 0 One
      unfolding z_def cbox_def
      using b'_Basis
      by (auto simp: bij_betw_def zero_le_divide_iff divide_le_eq_1)
    then have d_fz_z: d ≤ norm (f z − z)
      by (rule d)
    assume ¬ ?thesis
    then have as: ∀i∈Basis. |f z • i − z • i| < d / real n
      using ‹n > 0›
      by (auto simp: not_le inner_diff)
    have norm (f z − z) ≤ (∑i∈Basis. |f z • i − z • i|)
      unfolding inner_diff_left[symmetric]
      by (rule norm_le_l1)
    also have ... < (∑(i::'a) ∈ Basis. d / real n)
      by (meson as finite_Basis nonempty_Basis sum_strict_mono)
    also have ... = d
      using DIM_positive[where 'a='a] by (auto simp: n_def)
    finally show False
      using d_fz_z by auto
  qed
  then obtain i where i: i ∈ Basis d / real n ≤ |(f z − z) • i| ..
  have *: b' i < n
    using i and b'[unfolded bij_betw_def]
    by auto
  obtain r s where rs:
    ⋀j. j < n ⟹ q j ≤ r j ∧ r j ≤ q j + 1
    ⋀j. j < n ⟹ q j ≤ s j ∧ s j ≤ q j + 1
    (label (∑i∈Basis. (real (r (b' i)) / real p) *R i) ∘ b) (b' i) ≠
      (label (∑i∈Basis. (real (s (b' i)) / real p) *R i) ∘ b) (b' i)
    using q(2)[rule_format,OF *] by blast
  have b'_im: ⋀i. i ∈ Basis ⟹ b' i < n
    using b' unfolding bij_betw_def by auto
  define r' ::'a where r' = (∑i∈Basis. (real (r (b' i)) / real p) *R i)
  have ⋀i. i ∈ Basis ⟹ r (b' i) ≤ p
    using b'_im q(1) rs(1) by fastforce
  then have r' ∈ cbox 0 One
    unfolding r'_def cbox_def
    using b'_Basis
    by (auto simp: bij_betw_def zero_le_divide_iff divide_le_eq_1)
```

```
  define s' :: 'a where s' = (∑ i∈Basis. (real (s (b' i)) / real p) *R i)
  have ⋀i. i ∈ Basis ⟹ s (b' i) ≤ p
    using b'_im q(1) rs(2) by fastforce
  then have s' ∈ cbox 0 One
    unfolding s'_def cbox_def
   using b'_Basis by (auto simp: bij_betw_def zero_le_divide_iff divide_le_eq_1)
  have z ∈ cbox 0 One
    unfolding z_def cbox_def
    using b'_Basis q(1)[rule_format,OF b'_im] ‹p > 0›
    by (auto simp: bij_betw_def zero_le_divide_iff divide_le_eq_1 less_imp_le)
  {
    have (∑ i∈Basis. |real (r (b' i)) − real (q (b' i))|) ≤ (∑ (i::'a)∈Basis. 1)
      by (rule sum_mono) (use rs(1)[OF b'_im] in force)
    also have ... < e * real p
      using p ‹e > 0› ‹p > 0›
      by (auto simp: field_simps n_def)
    finally have (∑ i∈Basis. |real (r (b' i)) − real (q (b' i))|) < e * real p .
  }
  moreover
  {
    have (∑ i∈Basis. |real (s (b' i)) − real (q (b' i))|) ≤ (∑ (i::'a)∈Basis. 1)
      by (rule sum_mono) (use rs(2)[OF b'_im] in force)
    also have ... < e * real p
      using p ‹e > 0› ‹p > 0›
      by (auto simp: field_simps n_def)
    finally have (∑ i∈Basis. |real (s (b' i)) − real (q (b' i))|) < e * real p .
  }
  ultimately
  have norm (r' − z) < e and norm (s' − z) < e
    unfolding r'_def s'_def z_def
    using ‹p > 0›
    apply (rule_tac[!] le_less_trans[OF norm_le_l1])
   apply (auto simp: field_simps sum_divide_distrib[symmetric] inner_diff_left)
    done
  then have |(f z − z) • i| < d / real n
    using rs(3) i
    unfolding r'_def[symmetric] s'_def[symmetric] o_def bb'
    by (intro e(2)[OF ‹r'∈cbox 0 One› ‹s'∈cbox 0 One› ‹z∈cbox 0 One›]) auto
  then show False
    using i by auto
qed
```

Next step is to prove it for nonempty interiors.

```
lemma brouwer_weak:
  fixes f :: 'a::euclidean_space ⇒ 'a
  assumes compact S
    and convex S
    and interior S ≠ {}
    and continuous_on S f
```

```
    and f ∈ S → S
  obtains x where x ∈ S and f x = x
proof −
  let ?U = cbox 0 One :: 'a set
  have ∑ Basis /R 2 ∈ interior ?U
  proof (rule interiorI)
    let ?I = (⋂i∈Basis. {x::'a. 0 < x • i} ∩ {x. x • i < 1})
    show open ?I
     by (intro open_INT finite_Basis ballI open_Int, auto intro: open_Collect_less
simp: continuous_on_inner)
    show ∑ Basis /R 2 ∈ ?I
      by simp
    show ?I ⊆ cbox 0 One
      unfolding cbox_def by force
  qed
  then have ∗: interior ?U ≠ {} by fast
  have ∗: ?U homeomorphic S
     using homeomorphic_convex_compact[OF convex_box(1) compact_cbox ∗
assms(2,1,3)] .
  have ∀f. continuous_on ?U f ∧ f ∈ ?U → ?U ⟶ (∃x∈?U. f x = x)
    using brouwer_cube by auto
  then show ?thesis
    unfolding homeomorphic_fixpoint_property[OF ∗]
    using assms
    by (auto intro: that)
qed
```

Then the particular case for closed balls.

```
lemma brouwer_ball:
  fixes f :: 'a::euclidean_space ⇒ 'a
  assumes e > 0
    and continuous_on (cball a e) f
    and f ∈ cball a e → cball a e
  obtains x where x ∈ cball a e and f x = x
  using brouwer_weak[OF compact_cball convex_cball, of a e f]
  unfolding interior_cball ball_eq_empty
  using assms by auto
```

And finally we prove Brouwer's fixed point theorem in its general version.

```
theorem brouwer:
  fixes f :: 'a::euclidean_space ⇒ 'a
  assumes S: compact S convex S S ≠ {}
    and contf: continuous_on S f
    and fim: f ∈ S → S
  obtains x where x ∈ S and f x = x
proof −
  have ∃e>0. S ⊆ cball 0 e
    using compact_imp_bounded[OF ‹compact S›] unfolding bounded_pos
    by auto
```

```
  then obtain e where e: e > 0 S ⊆ cball 0 e
    by blast
  have ∃x∈ cball 0 e. (f ∘ closest_point S) x = x
  proof (rule_tac brouwer_ball[OF e(1)])
    show continuous_on (cball 0 e) (f ∘ closest_point S)
      by (meson assms closest_point_in_set compact_eq_bounded_closed contf
continuous_on_closest_point
          continuous_on_compose continuous_on_subset image_subsetI)
    show f ∘ closest_point S ∈ cball 0 e → cball 0 e
      by (smt (verit) Pi_iff assms(1) assms(3) closest_point_in_set comp_apply
compact_eq_bounded_closed e(2) fim subset_eq)
  qed (use assms in auto)
  then obtain x where x: x ∈ cball 0 e (f ∘ closest_point S) x = x ..
  with S have x ∈ S
    by (metis PiE closest_point_in_set comp_apply compact_imp_closed fim)
  then have ∗: closest_point S x = x
    by (rule closest_point_self)
  show thesis
  proof
    show closest_point S x ∈ S
      by (simp add: ∗ ‹x ∈ S›)
    show f (closest_point S x) = closest_point S x
      using ∗ x by auto
  qed
qed
```

6.28.4 Applications

So we get the no-retraction theorem.

```
corollary no_retraction_cball:
  fixes a :: 'a::euclidean_space
  assumes e > 0
  shows ¬ (frontier (cball a e) retract_of (cball a e))
proof
  assume ∗: frontier (cball a e) retract_of (cball a e)
  have ∗∗: ⋀xa. a − (2 ∗R a − xa) = − (a − xa)
    using scaleR_left_distrib[of 1 1 a] by auto
  obtain x where x: x ∈ {x. norm (a − x) = e} 2 ∗R a − x = x
  proof (rule retract_fixpoint_property[OF ∗, of λx. scaleR 2 a − x])
    show continuous_on (frontier (cball a e)) ((−) (2 ∗R a))
      by (intro continuous_intros)
    show (−) (2 ∗R a) ∈ frontier (cball a e) → frontier (cball a e)
      by clarsimp (metis ∗∗ dist_norm norm_minus_cancel)
  qed (auto simp: dist_norm intro: brouwer_ball[OF assms])
  then have scaleR 2 a = scaleR 1 x + scaleR 1 x
    by (auto simp: algebra_simps)
  then have a = x
    unfolding scaleR_left_distrib[symmetric] by auto
  then show False
```

```
    using x assms by auto
qed

corollary contractible_sphere:
  fixes a :: 'a::euclidean_space
  shows contractible(sphere a r) ⟷ r ≤ 0
proof (cases 0 < r)
  case True
  then show ?thesis
    unfolding contractible_def nullhomotopic_from_sphere_extension
    using no_retraction_cball [OF True, of a]
    by (auto simp: retract_of_def retraction_def)
next
  case False
  then show ?thesis
    unfolding contractible_def nullhomotopic_from_sphere_extension
    using less_eq_real_def by auto
qed

corollary connected_sphere_eq:
  fixes a :: 'a :: euclidean_space
  shows connected(sphere a r) ⟷ 2 ≤ DIM('a) ∨ r ≤ 0
    (is ?lhs = ?rhs)
proof (cases r 0::real rule: linorder_cases)
  case less
  then show ?thesis by auto
next
  case equal
  then show ?thesis by auto
next
  case greater
  show ?thesis
  proof
    assume L: ?lhs
    have False if 1: DIM('a) = 1
    proof −
      obtain x y where xy: sphere a r = {x,y} x ≠ y
        using sphere_1D_doubleton [OF 1 greater]
        by (metis dist_self greater insertI1 less_add_same_cancel1 mem_sphere
mult_2 not_le zero_le_dist)
      then have finite (sphere a r)
        by auto
      with L ‹r > 0› xy show False
        using connected_finite_iff_sing by auto
    qed
    with greater show ?rhs
      by (metis DIM_ge_Suc0 One_nat_def Suc_1 le_antisym not_less_eq_eq)
  next
    assume ?rhs
```

```
    then show ?lhs
      using connected_sphere greater by auto
  qed
qed

corollary path_connected_sphere_eq:
  fixes a :: 'a :: euclidean_space
  shows path_connected(sphere a r) ⟷ 2 ≤ DIM('a) ∨ r ≤ 0
        (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    using connected_sphere_eq path_connected_imp_connected by blast
next
  assume R: ?rhs
  then show ?lhs
   by (auto simp: contractible_imp_path_connected contractible_sphere path_connected_sphere)
qed

proposition frontier_subset_retraction:
  fixes S :: 'a::euclidean_space set
  assumes bounded S and fros: frontier S ⊆ T
      and contf: continuous_on (closure S) f
      and fim: f ∈ S → T
      and fid: ⋀x. x ∈ T ⟹ f x = x
    shows S ⊆ T
proof (rule ccontr)
  assume ¬ S ⊆ T
  then obtain a where a ∈ S a ∉ T by blast
  define g where g ≡ λz. if z ∈ closure S then f z else z
  have continuous_on (closure S ∪ closure(−S)) g
    unfolding g_def
    apply (rule continuous_on_cases)
    using fros fid frontier_closures by (auto simp: contf)
  moreover have closure S ∪ closure(− S) = UNIV
    using closure_Un by fastforce
  ultimately have contg: continuous_on UNIV g by metis
  obtain B where 0 < B and B: closure S ⊆ ball a B
    using ‹bounded S› bounded_subset_ballD by blast
  have notga: g x ≠ a for x
    unfolding g_def using fros fim ‹a ∉ T›
    by (metis PiE Un_iff ‹a ∈ S› closure_Un_frontier fid subsetD)
  define h where h ≡ (λy. a + (B / norm(y − a)) *R (y − a)) ∘ g
  have ¬ (frontier (cball a B) retract_of (cball a B))
    by (metis no_retraction_cball ‹0 < B›)
  then have ⋀k. ¬ retraction (cball a B) (frontier (cball a B)) k
    by (simp add: retract_of_def)
  moreover have retraction (cball a B) (frontier (cball a B)) h
    unfolding retraction_def
```

proof (*intro conjI ballI*)
show *frontier* (*cball a B*) ⊆ *cball a B*
by *force*
show *continuous_on* (*cball a B*) *h*
unfolding *h_def*
by (*intro continuous_intros*) (*use contg continuous_on_subset notga* **in** *auto*)
show *h* ∈ *cball a B* → *frontier* (*cball a B*)
using ‹*0* < *B*› **by** (*auto simp*: *h_def notga dist_norm*)
show ⋀*x*. *x* ∈ *frontier* (*cball a B*) ⟹ *h x* = *x*
using *notga* ‹*0* < *B*›
apply (*simp add*: *g_def h_def field_simps*)
by (*metis B dist_commute dist_norm mem_ball order_less_irrefl subset_eq*)
qed
ultimately show *False* **by** *simp*
qed

Punctured affine hulls, etc

lemma *rel_frontier_deformation_retract_of_punctured_convex*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *convex S convex T bounded S*
and *arelS*: *a* ∈ *rel_interior S*
and *relS*: *rel_frontier S* ⊆ *T*
and *affS*: *T* ⊆ *affine hull S*
obtains *r* **where** *homotopic_with_canon* (λ*x*. *True*) (*T* − {*a*}) (*T* − {*a*}) *id r*
retraction (*T* − {*a*}) (*rel_frontier S*) *r*
proof −
have ∃ *d*. *0* < *d* ∧ (*a* + *d* ∗*R* *l*) ∈ *rel_frontier S* ∧
(∀ *e*. *0* ≤ *e* ∧ *e* < *d* ⟶ (*a* + *e* ∗*R* *l*) ∈ *rel_interior S*)
if (*a* + *l*) ∈ *affine hull S l* ≠ *0* **for** *l*
apply (*rule ray_to_rel_frontier* [*OF* ‹*bounded S*› *arelS*])
apply (*rule that*)+
by *metis*
then obtain *dd*
where *dd1*: ⋀*l*. ⟦(*a* + *l*) ∈ *affine hull S*; *l* ≠ *0*⟧ ⟹ *0* < *dd l* ∧ (*a* + *dd l* ∗*R* *l*) ∈ *rel_frontier S*
and *dd2*: ⋀*l e*. ⟦(*a* + *l*) ∈ *affine hull S*; *e* < *dd l*; *0* ≤ *e*; *l* ≠ *0*⟧
⟹ (*a* + *e* ∗*R* *l*) ∈ *rel_interior S*
by *metis*+
have *aaffS*: *a* ∈ *affine hull S*
by (*meson arelS subsetD hull_inc rel_interior_subset*)
have ((λ*z*. *z* − *a*) ‘ (*affine hull S* − {*a*})) = ((λ*z*. *z* − *a*) ‘ (*affine hull S*)) − {*0*}
by *auto*
moreover have *continuous_on* (((λ*z*. *z* − *a*) ‘ (*affine hull S*)) − {*0*}) (λ*x*. *dd x* ∗*R* *x*)
proof (*rule continuous_on_compact_surface_projection*)
show *compact* (*rel_frontier* ((λ*z*. *z* − *a*) ‘ *S*))
by (*simp add*: ‹*bounded S*› *bounded_translation_minus compact_rel_frontier_bounded*)

```
    have releq: rel_frontier ((λz. z − a) ‘ S) = (λz. z − a) ‘ rel_frontier S
      using rel_frontier_translation [of −a] add.commute by simp
    also have ... ⊆ (λz. z − a) ‘ (affine hull S) − {0}
      using rel_frontier_affine_hull arelS rel_frontier_def by fastforce
    finally show rel_frontier ((λz. z − a) ‘ S) ⊆ (λz. z − a) ‘ (affine hull S) −
{0} .
    show cone ((λz. z − a) ‘ (affine hull S))
      by (rule subspace_imp_cone)
       (use aaffS in ‹simp add: subspace_affine image_comp o_def affine_translation_aux
[of a]›)
    show (0 < k ∧ k *R x ∈ rel_frontier ((λz. z − a) ‘ S)) ⟷ (dd x = k)
        if x: x ∈ (λz. z − a) ‘ (affine hull S) − {0} for k x
    proof
      show dd x = k ⟹ 0 < k ∧ k *R x ∈ rel_frontier ((λz. z − a) ‘ S)
      using dd1 [of x] that image_iff by (fastforce simp add: releq)
    next
      assume k: 0 < k ∧ k *R x ∈ rel_frontier ((λz. z − a) ‘ S)
      have False if dd x < k
      proof −
        have k ≠ 0 a + k *R x ∈ closure S
          using k closure_translation [of −a]
          by (auto simp: rel_frontier_def cong: image_cong_simp)
        then have segsub: open_segment a (a + k *R x) ⊆ rel_interior S
          by (metis rel_interior_closure_convex_segment [OF ‹convex S› arelS])
        have x ≠ 0 and xaffS: a + x ∈ affine hull S
          using x by auto
        then have 0 < dd x and inS: a + dd x *R x ∈ rel_frontier S
          using dd1 by auto
        moreover have a + dd x *R x ∈ open_segment a (a + k *R x)
          using k ‹x ≠ 0› ‹0 < dd x›
          apply (simp add: in_segment)
          apply (rule_tac x = dd x / k in exI)
          apply (simp add: that vector_add_divide_simps algebra_simps)
          done
        ultimately show ?thesis
          using segsub by (auto simp: rel_frontier_def)
      qed
      moreover have False if k < dd x
        using x k that rel_frontier_def
        by (fastforce simp: algebra_simps releq dest!: dd2)
      ultimately show dd x = k
        by fastforce
    qed
  qed
  ultimately have ∗: continuous_on ((λz. z − a) ‘ (affine hull S − {a})) (λx. dd
x *R x)
    by auto
  have continuous_on (affine hull S − {a}) ((λx. a + dd x *R x) ∘ (λz. z − a))
    by (intro ∗ continuous_intros continuous_on_compose)
```

```
  with affS have contdd: continuous_on (T - {a}) ((λx. a + dd x *R x) ∘ (λz. z - a))
    by (blast intro: continuous_on_subset)
  show ?thesis
  proof
    show homotopic_with_canon (λx. True) (T - {a}) (T - {a}) id (λx. a + dd (x - a) *R (x - a))
    proof (rule homotopic_with_linear)
      show continuous_on (T - {a}) id
        by (intro continuous_intros continuous_on_compose)
      show continuous_on (T - {a}) (λx. a + dd (x - a) *R (x - a))
        using contdd by (simp add: o_def)
      show closed_segment (id x) (a + dd (x - a) *R (x - a)) ⊆ T - {a}
          if x ∈ T - {a} for x
      proof (clarsimp simp: in_segment, intro conjI)
        fix u::real assume u: 0 ≤ u u ≤ 1
        have a + dd (x - a) *R (x - a) ∈ T
       by (metis DiffD1 DiffD2 add.commute add.right_neutral affS dd1 diff_add_cancel relS singletonI subsetCE that)
        then show (1 - u) *R x + u *R (a + dd (x - a) *R (x - a)) ∈ T
          using convexD [OF ‹convex T›] that u by simp
        have iff: (1 - u) *R x + u *R (a + d *R (x - a)) = a ⟷
                  (1 - u + u * d) *R (x - a) = 0 for d
          by (auto simp: algebra_simps)
        have x ∈ T x ≠ a using that by auto
        then have axa: a + (x - a) ∈ affine hull S
          by (metis (no_types) add.commute affS diff_add_cancel rev_subsetD)
       then have ¬ dd (x - a) ≤ 0 ∧ a + dd (x - a) *R (x - a) ∈ rel_frontier S
          using ‹x ≠ a› dd1 by fastforce
       with ‹x ≠ a› show (1 - u) *R x + u *R (a + dd (x - a) *R (x - a)) ≠ a
          using less_eq_real_def mult_le_0_iff not_less u by (fastforce simp: iff)
      qed
    qed
    show retraction (T - {a}) (rel_frontier S) (λx. a + dd (x - a) *R (x - a))
    proof (simp add: retraction_def, intro conjI ballI)
      show rel_frontier S ⊆ T - {a}
        using arelS relS rel_frontier_def by fastforce
      show continuous_on (T - {a}) (λx. a + dd (x - a) *R (x - a))
        using contdd by (simp add: o_def)
      show (λx. a + dd (x - a) *R (x - a)) ∈ (T - {a}) → rel_frontier S
        apply (auto simp: rel_frontier_def)
     apply (metis Diff_subset add.commute affS dd1 diff_add_cancel eq_iff_diff_eq_0 rel_frontier_def subset_iff)
       by (metis DiffE add.commute affS dd1 diff_add_cancel eq_iff_diff_eq_0 rel_frontier_def rev_subsetD)
      show a + dd (x - a) *R (x - a) = x if x: x ∈ rel_frontier S for x
      proof -
        have x ≠ a
          using that arelS by (auto simp: rel_frontier_def)
```

```
      have False if dd (x − a) < 1
      proof −
        have x ∈ closure S
          using x by (auto simp: rel_frontier_def)
        then have segsub: open_segment a x ⊆ rel_interior S
          by (metis rel_interior_closure_convex_segment [OF ‹convex S› arelS])
        have  xaffS: x ∈ affine hull S
          using affS relS x by auto
          then have 0 < dd (x − a) and inS: a + dd (x − a) *R (x − a) ∈
rel_frontier S
          using dd1 by (auto simp: ‹x ≠ a›)
        moreover have a + dd (x − a) *R (x − a) ∈ open_segment a x
          using  ‹x ≠ a› ‹0 < dd (x − a)›
          apply (simp add: in_segment)
          apply (rule_tac x = dd (x − a) in exI)
          apply (simp add: algebra_simps that)
          done
        ultimately show ?thesis
          using segsub by (auto simp: rel_frontier_def)
      qed
      moreover have False if 1 < dd (x − a)
        using x that dd2 [of x − a 1] ‹x ≠ a› closure_affine_hull
        by (auto simp: rel_frontier_def)
      ultimately have dd (x − a) = 1 — similar to another proof above
        by fastforce
      with that show ?thesis
        by (simp add: rel_frontier_def)
    qed
  qed
 qed
qed

corollary rel_frontier_retract_of_punctured_affine_hull:
  fixes S :: 'a::euclidean_space set
  assumes bounded S convex S a ∈ rel_interior S
    shows rel_frontier S retract_of (affine hull S − {a})
  by (meson assms convex_affine_hull dual_order.refl rel_frontier_affine_hull
      rel_frontier_deformation_retract_of_punctured_convex retract_of_def)

corollary rel_boundary_retract_of_punctured_affine_hull:
  fixes S :: 'a::euclidean_space set
  assumes compact S convex S a ∈ rel_interior S
    shows (S − rel_interior S) retract_of (affine hull S − {a})
by (metis assms closure_closed compact_eq_bounded_closed rel_frontier_def
        rel_frontier_retract_of_punctured_affine_hull)

lemma homotopy_eqv_rel_frontier_punctured_convex:
  fixes S :: 'a::euclidean_space set
  assumes convex S bounded S a ∈ rel_interior S convex T rel_frontier S ⊆ T T
```

$\subseteq$ *affine hull S*
shows (*rel_frontier S*) *homotopy_eqv* (*T* − {*a*})
by (*meson assms deformation_retract_imp_homotopy_eqv homotopy_equivalent_space_sym*

rel_frontier_deformation_retract_of_punctured_convex[*of S T*])

lemma *homotopy_eqv_rel_frontier_punctured_affine_hull*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *convex S bounded S a* ∈ *rel_interior S*
shows (*rel_frontier S*) *homotopy_eqv* (*affine hull S* − {*a*})
by (*simp add*: *assms homotopy_eqv_rel_frontier_punctured_convex rel_frontier_affine_hull*)

lemma *path_connected_sphere_gen*:
assumes *convex S bounded S aff_dim S* ≠ *1*
shows *path_connected*(*rel_frontier S*)
proof −
have *convex* (*closure S*)
using *assms* **by** *auto*
then show *?thesis*
by (*metis Diff_empty aff_dim_affine_hull assms convex_affine_hull convex_imp_path_connected equals0I*
path_connected_punctured_convex rel_frontier_def rel_frontier_retract_of_punctured_affine_hull retract_of_path_connected)
qed

lemma *connected_sphere_gen*:
assumes *convex S bounded S aff_dim S* ≠ *1*
shows *connected*(*rel_frontier S*)
by (*simp add*: *assms path_connected_imp_connected path_connected_sphere_gen*)

Borsuk-style characterization of separation

lemma *continuous_on_Borsuk_map*:
a ∉ *S* ⟹ *continuous_on S* (λ*x*. *inverse*(*norm* (*x* − *a*)) ∗$_R$ (*x* − *a*))
by (*rule continuous_intros* | *force*)+

lemma *Borsuk_map_into_sphere*:
(λ*x*. *inverse*(*norm* (*x* − *a*)) ∗$_R$ (*x* − *a*)) ∈ *S* → *sphere 0 1* ⟷ (*a* ∉ *S*)
proof −
have ⋀*x*. ⟦*a* ∉ *S*; *x* ∈ *S*⟧ ⟹ *inverse* (*norm* (*x* − *a*)) ∗ *norm* (*x* − *a*) = *1*
by (*metis left_inverse norm_eq_zero right_minus_eq*)
then show *?thesis*
by *force*
qed

lemma *Borsuk_maps_homotopic_in_path_component*:
assumes *path_component* (− *S*) *a b*
shows *homotopic_with_canon* (λ*x*. *True*) *S* (*sphere 0 1*)
(λ*x*. *inverse*(*norm*(*x* − *a*)) ∗$_R$ (*x* − *a*))

```
                (λx. inverse(norm(x − b)) *R (x − b))
proof −
  obtain g where path g path_image g ⊆ −S pathstart g = a pathfinish g = b
    using assms by (auto simp: path_component_def)
  then show ?thesis
    apply (simp add: path_def path_image_def pathstart_def pathfinish_def ho-
motopic_with_def)
    apply (rule_tac x = λz. inverse(norm(snd z − (g ∘ fst)z)) *R (snd z − (g ∘
fst)z) in exI)
    apply (rule conjI continuous_intros | erule continuous_on_subset | fastforce
simp: divide_simps sphere_def)+
    done
qed

lemma non_extensible_Borsuk_map:
  fixes a :: 'a :: euclidean_space
  assumes compact S and cin: C ∈ components(− S) and boc: bounded C and a
∈ C
    shows ¬ (∃ g. continuous_on (S ∪ C) g ∧
                  g ∈ (S ∪ C) → sphere 0 1 ∧
                  (∀ x ∈ S. g x = inverse(norm(x − a)) *R (x − a)))
proof −
  have closed S using assms by (simp add: compact_imp_closed)
  have C ⊆ −S
    using assms by (simp add: in_components_subset)
  with ‹a ∈ C› have a ∉ S by blast
  then have ceq: C = connected_component_set (− S) a
    by (metis ‹a ∈ C› cin components_iff connected_component_eq)
  then have bounded (S ∪ connected_component_set (− S) a)
    using ‹compact S› boc compact_imp_bounded by auto
 with bounded_subset_ballD obtain r where 0 < r and r: (S ∪ connected_component_set
(− S) a) ⊆ ball a r
    by blast
  { fix g
    assume continuous_on (S ∪ C) g
           g ∈ (S ∪ C) → sphere 0 1
      and [simp]: ⋀x. x ∈ S ⟹ g x = (x − a) /R norm (x − a)
    then have norm_g1[simp]: ⋀x. x ∈ S ∪ C ⟹ norm (g x) = 1
      by force
    have cb_eq: cball a r = (S ∪ connected_component_set (− S) a) ∪
                            (cball a r − connected_component_set (− S) a)
      using ball_subset_cball [of a r] r by auto
    have cont1: continuous_on (S ∪ connected_component_set (− S) a)
                              (λx. a + r *R g x)
      using ‹continuous_on (S ∪ C) g› ceq
      by (intro continuous_intros) blast
    have cont2: continuous_on (cball a r − connected_component_set (− S) a)
          (λx. a + r *R ((x − a) /R norm (x − a)))
      by (rule continuous_intros | force simp: ‹a ∉ S›)+
```

```
    have 1: continuous_on (cball a r)
            (λx. if connected_component (− S) a x
                 then a + r *R g x
                 else a + r *R ((x − a) /R norm (x − a)))
      apply (subst cb_eq)
      apply (rule continuous_on_cases [OF _ _ cont1 cont2])
      using ‹closed S› ceq cin
     by (force simp: closed_Diff open_Compl closed_Un_complement_component
open_connected_component)+
    have 2: (λx. a + r *R g x) ‘ (cball a r ∩ connected_component_set (− S) a)
            ⊆ sphere a r
      using ‹0 < r› by (force simp: dist_norm ceq)
    have retraction (cball a r) (sphere a r)
            (λx. if x ∈ connected_component_set (− S) a
                 then a + r *R g x
                 else a + r *R ((x − a) /R norm (x − a)))
      using ‹0 < r› ‹a ∉ S› ‹a ∈ C› r
      by (auto simp: norm_minus_commute retraction_def Pi_iff ceq dist_norm
abs_if
          mult_less_0_iff divide_simps 1 2)
    then have False
      using no_retraction_cball
            [OF ‹0 < r›, of a, unfolded retract_of_def, simplified, rule_format,
             of λx. if x ∈ connected_component_set (− S) a
                    then a + r *R g x
                    else a + r *R inverse(norm(x − a)) *R (x − a)]
      by blast
  }
  then show ?thesis
    by blast
qed
```

Proving surjectivity via Brouwer fixpoint theorem

```
lemma brouwer_surjective:
  fixes f :: 'n::euclidean_space ⇒ 'n
  assumes compact T
    and convex T
    and T ≠ {}
    and continuous_on T f
    and ⋀x y. ⟦x∈S; y∈T⟧ ⟹ x + (y − f y) ∈ T
    and x ∈ S
  shows ∃ y∈T. f y = x
proof −
  have ∗: ⋀x y. f y = x ⟷ x + (y − f y) = y
    by (auto simp add: algebra_simps)
  show ?thesis
    unfolding ∗
    apply (rule brouwer[OF assms(1−3), of λy. x + (y − f y)])
```

apply (*intro continuous_intros*)
using *assms*
apply *auto*
done
qed

lemma *brouwer_surjective_cball*:
fixes $f :: 'n::euclidean\_space \Rightarrow 'n$
assumes *continuous_on* (*cball a e*) *f*
and $e > 0$
and $x \in S$
and $\bigwedge x\ y.\ [\![x \in S;\ y \in cball\ a\ e]\!] \Longrightarrow x + (y - f\ y) \in cball\ a\ e$
shows $\exists\, y \in cball\ a\ e.\ f\ y = x$
by (*smt* (*verit*, *best*) *assms brouwer_surjective cball_eq_empty compact_cball convex_cball*)

Inverse function theorem

See Sussmann: "Multidifferential calculus", Theorem 2.1.1

lemma *sussmann_open_mapping*:
fixes $f :: 'a::real\_normed\_vector \Rightarrow 'b::euclidean\_space$
assumes *open S*
and *contf*: *continuous_on S f*
and $x \in S$
and *derf*: (*f has_derivative f'*) (*at x*)
and *bounded_linear g' f' ∘ g' = id*
and $T \subseteq S$
and *x*: $x \in interior\ T$
shows $f\ x \in interior\ (f\ `\ T)$
proof −
interpret *f'*: *bounded_linear f'*
using *assms* **unfolding** *has_derivative_def* **by** *auto*
interpret *g'*: *bounded_linear g'*
using *assms* **by** *auto*
obtain *B* **where** *B*: $0 < B\ \forall x.\ norm\ (g'\ x) \le norm\ x * B$
using *bounded_linear.pos_bounded*[*OF assms*(*5*)] **by** *blast*
hence ∗: $1\ /\ (2 * B) > 0$ **by** *auto*
obtain *e0* **where** *e0*:
$0 < e0$
$\forall y.\ norm\ (y - x) < e0 \longrightarrow norm\ (f\ y - f\ x - f'\ (y - x)) \le 1\ /\ (2 * B) * norm\ (y - x)$
using *derf* **unfolding** *has_derivative_at_alt*
using ∗ **by** *blast*
obtain *e1* **where** *e1*: $0 < e1\ cball\ x\ e1 \subseteq T$
using *mem_interior_cball x* **by** *blast*
have ∗: $0 < e0\ /\ B\ 0 < e1\ /\ B$ **using** *e0 e1 B* **by** *auto*
obtain *e* **where** *e*: $0 < e\ e < e0\ /\ B\ e < e1\ /\ B$
using *field_lbound_gt_zero*[*OF* ∗] **by** *blast*
have *lem*: $\exists\, y \in cball\ (f\ x)\ e.\ f\ (x + g'\ (y - f\ x)) = z$ **if** $z \in cball\ (f\ x)\ (e\ /\ 2)$ **for**

```
z
  proof (rule brouwer_surjective_cball)
    have z: z ∈ S if as: y ∈cball (f x) e z = x + (g' y - g' (f x)) for y z
    proof−
      have dist x z = norm (g' (f x) - g' y)
        unfolding as(2) and dist_norm by auto
      also have ... ≤ norm (f x - y) * B
        by (metis B(2) g'.diff)
      also have ... ≤ e * B
        by (metis B(1) dist_norm mem_cball mult_le_cancel_iff1 that(1))
      also have ... ≤ e1
        using B(1) e(3) pos_less_divide_eq by fastforce
      finally have z ∈ cball x e1
        by force
      then show z ∈ S
        using e1 assms(7) by auto
    qed
    show continuous_on (cball (f x) e) (λy. f (x + g' (y - f x)))
      unfolding g'.diff
    proof (rule continuous_on_compose2 [OF _ _ order_refl, of _ _ f])
      show continuous_on ((λy. x + (g' y - g' (f x))) ` cball (f x) e) f
        by (rule continuous_on_subset[OF contf]) (use z in blast)
      show continuous_on (cball (f x) e) (λy. x + (g' y - g' (f x)))
       by (intro continuous_intros linear_continuous_on[OF ‹bounded_linear g'›])
    qed
  next
    fix y z
    assume y: y ∈ cball (f x) (e / 2) and z: z ∈ cball (f x) e
    have norm (g' (z - f x)) ≤ norm (z - f x) * B
      using B by auto
    also have ... ≤ e * B
     by (metis B(1) z dist_norm mem_cball norm_minus_commute mult_le_cancel_iff1)
    also have ... < e0
      using B(1) e(2) pos_less_divide_eq by blast
    finally have *: norm (x + g' (z - f x) - x) < e0
      by auto
    have **: f x + f' (x + g' (z - f x) - x) = z
      using assms(6)[unfolded o_def id_def,THEN cong]
      by auto
    have norm (f x - (y + (z - f (x + g' (z - f x))))) ≤
          norm (f (x + g' (z - f x)) - z) + norm (f x - y)
      using norm_triangle_ineq[of f (x + g'(z - f x)) - z f x - y]
      by (auto simp add: algebra_simps)
    also have ... ≤ 1 / (B * 2) * norm (g' (z - f x)) + norm (f x - y)
      using e0(2)[rule_format, OF *]
      by (simp only: algebra_simps **) auto
    also have ... ≤ 1 / (B * 2) * norm (g' (z - f x)) + e/2
      using y by (auto simp: dist_norm)
    also have ... ≤ 1 / (B * 2) * B * norm (z - f x) + e/2
```

```
    using * B by (auto simp add: field_simps)
  also have ... ≤ 1 / 2 * norm (z - f x) + e/2
    by auto
  also have ... ≤ e/2 + e/2
    using B(1) ‹norm (z - f x) * B ≤ e * B› by auto
  finally show y + (z - f (x + g' (z - f x))) ∈ cball (f x) e
    by (auto simp: dist_norm)
 qed (use e that in auto)
 show ?thesis
  unfolding mem_interior
 proof (intro exI conjI subsetI)
  fix y
  assume y ∈ ball (f x) (e / 2)
  then have *: y ∈ cball (f x) (e / 2)
    by auto
  obtain z where z: z ∈ cball (f x) e f (x + g' (z - f x)) = y
    using lem * by blast
  then have norm (g' (z - f x)) ≤ norm (z - f x) * B
    using B
    by (auto simp add: field_simps)
  also have ... ≤ e * B
  by (metis B(1) dist_norm mem_cball norm_minus_commute mult_le_cancel_iff1
z(1))
  also have ... ≤ e1
    using e B unfolding less_divide_eq by auto
  finally have x + g'(z - f x) ∈ T
  by (metis add_diff_cancel diff_diff_add dist_norm e1(2) mem_cball norm_minus_commute
subset_eq)
  then show y ∈ f ‘ T
    using z by auto
 qed (use e in auto)
qed
```

Hence the following eccentric variant of the inverse function theorem. This has no continuity assumptions, but we do need the inverse function. We could put $f' \circ g = I$ but this happens to fit with the minimal linear algebra theory I've set up so far.

```
lemma has_derivative_inverse_strong:
 fixes f :: 'n::euclidean_space ⇒ 'n
 assumes S: open S x ∈ S
   and contf: continuous_on S f
   and gf: ⋀x. x ∈ S ⟹ g (f x) = x
   and derf: (f has_derivative f') (at x)
   and id: f' ∘ g' = id
 shows (g has_derivative g') (at (f x))
proof -
 have linf: bounded_linear f'
   using derf unfolding has_derivative_def by auto
 then have ling: bounded_linear g'
```

```
    unfolding linear_conv_bounded_linear[symmetric]
    using id right_inverse_linear by blast
  moreover have g' ∘ f' = id
    using id linear_inverse_left linear_linear linf ling by blast
  moreover have *: ⋀T. ⟦T ⊆ S; x ∈ interior T⟧ ⟹ f x ∈ interior (f ` T)
    using S derf contf id ling sussmann_open_mapping by blast
  have continuous (at (f x)) g
    unfolding continuous_at Lim_at
  proof (intro strip)
    fix e :: real
    assume e > 0
    then have f x ∈ interior (f ` (ball x e ∩ S))
      by (simp add: * S interior_open)
    then obtain d where d: 0 < d ball (f x) d ⊆ f ` (ball x e ∩ S)
      unfolding mem_interior by blast
    show ∃d>0. ∀y. 0 < dist y (f x) ∧ dist y (f x) < d ⟶ dist (g y) (g (f x)) <
e
    proof (intro exI allI impI conjI)
      fix y
      assume 0 < dist y (f x) ∧ dist y (f x) < d
      then have g y ∈ g ` f ` (ball x e ∩ S)
        by (metis d(2) dist_commute mem_ball rev_image_eqI subset_iff)
      then show dist (g y) (g (f x)) < e
        using ‹x ∈ S› by (simp add: gf dist_commute image_iff)
    qed (use d in auto)
  qed
  moreover have f x ∈ interior (f ` S)
    using * S interior_eq by blast
  moreover have f (g y) = y if y ∈ interior (f ` S) for y
    by (metis gf imageE interiorE subsetD that)
  ultimately show ?thesis using assms
    by (metis has_derivative_inverse_basic_x open_interior)
qed
```

A rewrite based on the other domain.

```
lemma has_derivative_inverse_strong_x:
  fixes f :: 'a::euclidean_space ⇒ 'a
  assumes open S
    and g y ∈ S
    and continuous_on S f
    and ⋀x. x ∈ S ⟹ g (f x) = x
    and (f has_derivative f') (at (g y))
    and f' ∘ g' = id
    and f: f (g y) = y
  shows (g has_derivative g') (at y)
  using has_derivative_inverse_strong[OF assms(1−6)] by (simp add: f)
```

On a region.

```
theorem has_derivative_inverse_on:
```

fixes $f :: 'n::euclidean\_space \Rightarrow 'n$
assumes *open S*
and $\bigwedge x.\ x \in S \Longrightarrow (f\ has\_derivative\ f'(x))\ (at\ x)$
and $\bigwedge x.\ x \in S \Longrightarrow g\ (f\ x) = x$
and $f'\ x \circ g'\ x = id$
and $x \in S$
shows $(g\ has\_derivative\ g'(x))\ (at\ (f\ x))$
by (*meson assms continuous_on_eq_continuous_at has_derivative_continuous has_derivative_inverse_strong*)

end

6.29 Fashoda Meet Theorem

theory *Fashoda_Theorem*
imports *Brouwer_Fixpoint Path_Connected Cartesian_Euclidean_Space*
begin

6.29.1 Bijections between intervals

definition $interval\_bij :: 'a \times 'a \Rightarrow 'a \times 'a \Rightarrow 'a \Rightarrow 'a::euclidean\_space$
where *interval_bij* =
$(\lambda(a,\ b)\ (u,\ v)\ x.\ (\sum i{\in}Basis.\ (u{\cdot}i + (x{\cdot}i - a{\cdot}i) / (b{\cdot}i - a{\cdot}i) * (v{\cdot}i - u{\cdot}i)) *_R i))$

lemma *interval_bij_affine*:
$interval\_bij\ (a,b)\ (u,v) = (\lambda x.\ (\sum i{\in}Basis.\ ((v{\cdot}i - u{\cdot}i) / (b{\cdot}i - a{\cdot}i) * (x{\cdot}i)) *_R i) +$
$(\sum i{\in}Basis.\ (u{\cdot}i - (v{\cdot}i - u{\cdot}i) / (b{\cdot}i - a{\cdot}i) * (a{\cdot}i)) *_R i))$
by (*simp add*: *interval_bij_def algebra_simps add_divide_distrib diff_divide_distrib flip*: *sum.distrib scaleR_add_left*)

lemma *continuous_interval_bij*:
fixes $a\ b :: 'a::euclidean\_space$
shows $continuous\ (at\ x)\ (interval\_bij\ (a,\ b)\ (u,\ v))$
by (*auto simp add*: *divide_inverse interval_bij_def intro*!: *continuous_sum continuous_intros*)

lemma *continuous_on_interval_bij*: $continuous\_on\ s\ (interval\_bij\ (a,\ b)\ (u,\ v))$
by (*metis continuous_at_imp_continuous_on continuous_interval_bij*)

lemma *in_interval_interval_bij*:
fixes $a\ b\ u\ v\ x :: 'a::euclidean\_space$
assumes $x \in cbox\ a\ b$
and $cbox\ u\ v \neq \{\}$
shows $interval\_bij\ (a,\ b)\ (u,\ v)\ x \in cbox\ u\ v$
proof −
have $\bigwedge i.\ i \in Basis \Longrightarrow u \cdot i \leq u \cdot i + (x \cdot i - a \cdot i) / (b \cdot i - a \cdot i) * (v \cdot i$

$- u \cdot i$)
by (*smt* (*verit*) *assms box_ne_empty*(1) *divide_nonneg_nonneg mem_box*(2) *mult_nonneg_nonneg*)
moreover
have $\bigwedge i.\ i \in Basis \Longrightarrow u \cdot i + (x \cdot i - a \cdot i) / (b \cdot i - a \cdot i) * (v \cdot i - u \cdot i) \leq v \cdot i$
apply (*simp add*: *divide_simps algebra_simps*)
by (*smt* (*verit*, *best*) *assms box_ne_empty*(1) *left_diff_distrib mem_box*(2) *mult.commute mult_left_mono*)
ultimately show *?thesis*
by (*force simp only*: *interval_bij_def split_conv mem_box inner_sum_left_Basis*)
qed

lemma *interval_bij_bij*:
$\forall (i::'a::euclidean\_space) \in Basis.\ a \cdot i < b \cdot i \wedge u \cdot i < v \cdot i \Longrightarrow$
interval_bij (*a*, *b*) (*u*, *v*) (*interval_bij* (*u*, *v*) (*a*, *b*) *x*) = *x*
by (*auto simp*: *interval_bij_def euclidean_eq_iff*[**where** *'a*='*a*])

lemma *interval_bij_bij_cart*: **fixes** *x*::*real*^'*n* **assumes** $\forall i.\ a\$i < b\$i \wedge u\$i < v\$i$
shows *interval_bij* (*a*,*b*) (*u*,*v*) (*interval_bij* (*u*,*v*) (*a*,*b*) *x*) = *x*
using *assms* **by** (*intro interval_bij_bij*) (*auto simp*: *Basis_vec_def inner_axis*)

6.29.2 Fashoda meet theorem

lemma *infnorm_2*:
fixes *x* :: *real*^2
shows *infnorm x* = *max* |*x*$1| |*x*$2|
unfolding *infnorm_cart UNIV_2* **by** (*rule cSup_eq*) *auto*

lemma *infnorm_eq_1_2*:
fixes *x* :: *real*^2
shows *infnorm x* = 1 ⟷
|*x*$1| ≤ 1 ∧ |*x*$2| ≤ 1 ∧ (*x*$1 = −1 ∨ *x*$1 = 1 ∨ *x*$2 = −1 ∨ *x*$2 = 1)
unfolding *infnorm_2* **by** *auto*

lemma *infnorm_eq_1_imp*:
fixes *x* :: *real*^2
assumes *infnorm x* = 1
shows |*x*$1| ≤ 1 **and** |*x*$2| ≤ 1
using *assms* **unfolding** *infnorm_eq_1_2* **by** *auto*

proposition *fashoda_unit*:
fixes *f g* :: *real* ⇒ *real*^2
assumes *f* ' {−1 .. 1} ⊆ *cbox* (−1) 1
and *g* ' {−1 .. 1} ⊆ *cbox* (−1) 1
and *continuous_on* {−1 .. 1} *f*
and *continuous_on* {−1 .. 1} *g*
and *f* (− 1)$1 = − 1

```
    and f 1$1 = 1 g (− 1) $2 = −1
    and g 1 $2 = 1
  shows ∃ s∈{−1 .. 1}. ∃ t∈{−1 .. 1}. f s = g t
proof (rule ccontr)
  assume ¬ ?thesis
  note as = this[unfolded bex_simps,rule_format]
  define sqprojection
    where [abs_def]: sqprojection z = (inverse (infnorm z)) *R z for z :: real^2
  define negatex :: real^2 ⇒ real^2
    where negatex x = (vector [−(x$1), x$2]) for x
  have inf_nega: ⋀z::real^2. infnorm (negatex z) = infnorm z
    unfolding negatex_def infnorm_2 vector_2 by auto
  have inf_eq1: ⋀z. z ≠ 0 ⟹ infnorm (sqprojection z) = 1
    unfolding sqprojection_def infnorm_mul[unfolded scalar_mult_eq_scaleR]
    by (simp add: real_abs_infnorm infnorm_eq_0)
  let ?F = λw::real^2. (f ∘ (λx. x$1)) w − (g ∘ (λx. x$2)) w
  have *: ⋀i. (λx::real^2. x $ i) ` cbox (− 1) 1 = {−1..1}
  proof
    show (λx::real^2. x $ i) ` cbox (− 1) 1 ⊆ {−1..1} for i
      by (auto simp: mem_box_cart)
    show {−1..1} ⊆ (λx::real^2. x $ i) ` cbox (− 1) 1 for i
      by (clarsimp simp: image_iff mem_box_cart Bex_def) (metis (no_types,
opaque_lifting) vec_component)
  qed
  {
    fix x
    assume x ∈ (λw. (f ∘ (λx. x $ 1)) w − (g ∘ (λx. x $ 2)) w) ` (cbox (− 1)
(1::real^2))
    then obtain w :: real^2 where w:
      w ∈ cbox (− 1) 1
      x = (f ∘ (λx. x $ 1)) w − (g ∘ (λx. x $ 2)) w
      unfolding image_iff ..
    then have x ≠ 0
      using as[of w$1 w$2] by (auto simp: mem_box_cart atLeastAtMost_iff)
  } note x0 = this
  let ?CB11 = cbox (− 1) (1::real^2)
  obtain x :: real^2 where x:
      x ∈ cbox (− 1) 1
      (negatex ∘ sqprojection ∘ (λw. (f ∘ (λx. x $ 1)) w − (g ∘ (λx. x $ 2)) w)) x
= x
  proof (rule brouwer_weak[of ?CB11 negatex ∘ sqprojection ∘ ?F])
    show compact ?CB11 convex ?CB11
      by (rule compact_cbox convex_box)+
    have box (− 1) (1::real^2) ≠ {}
      unfolding interval_eq_empty_cart by auto
    then show interior ?CB11 ≠ {}
      by simp
    have negatex (x + y) $ i = (negatex x + negatex y) $ i ∧ negatex (c *R x) $
i = (c *R negatex x) $ i
```

```
      for i x y c
      using exhaust_2 [of i] by (auto simp: negatex_def)
    then have bounded_linear negatex
      by (simp add: bounded_linearI' vec_eq_iff)
    then show continuous_on ?CB11 (negatex ∘ sqprojection ∘ ?F)
      unfolding sqprojection_def
      apply (intro continuous_intros continuous_on_component | use ∗ assms in
presburger)+
       apply (simp_all add: infnorm_eq_0 x0 linear_continuous_on)
      done
    have (negatex ∘ sqprojection ∘ ?F) ‘ ?CB11 ⊆ ?CB11
    proof clarsimp
      fix y :: real^2
      assume y: y ∈ ?CB11
      have ?F y ≠ 0
        by (rule x0) (use y in auto)
      then have ∗: infnorm (sqprojection (?F y)) = 1
        using inf_eq1 by blast
      show negatex (sqprojection (f (y $ 1) − g (y $ 2))) ∈ cbox (−1) 1
        unfolding mem_box_cart interval_cbox_cart infnorm_2
      by (smt (verit, del_insts) ∗ component_le_infnorm_cart inf_nega neg_one_index
o_apply one_index)
    qed
    then show negatex ∘ sqprojection ∘ ?F ∈ ?CB11 → ?CB11
      by blast
  qed
  have ?F x ≠ 0
    by (rule x0) (use x in auto)
  then have ∗: infnorm (sqprojection (?F x)) = 1
    using inf_eq1 by blast
  have nx: infnorm x = 1
    by (metis (no_types, lifting) ∗ inf_nega o_apply x(2))
  have iff: 0 < sqprojection x$i ⟷ 0 < x$i sqprojection x$i < 0 ⟷ x$i < 0
if x ≠ 0 for x i
  proof −
    have ∗: inverse (infnorm x) > 0
      by (simp add: infnorm_pos_lt that)
    then show (0 < sqprojection x $ i) = (0 < x $ i)
      by (simp add: sqprojection_def zero_less_mult_iff)
    show (sqprojection x $ i < 0) = (x $ i < 0)
      unfolding sqprojection_def
      by (metis ∗ pos_less_divideR_eq scaleR_zero_right vector_scaleR_component)
  qed
  have x1: x $ 1 ∈ {− 1..1::real} x $ 2 ∈ {− 1..1::real}
    using x(1) unfolding mem_box_cart by auto
  then have nz: f (x $ 1) − g (x $ 2) ≠ 0
    using as by auto
  consider x $ 1 = −1 | x $ 1 = 1 | x $ 2 = −1 | x $ 2 = 1
    using nx unfolding infnorm_eq_1_2 by auto
```

```
  then show False
  proof cases
    case 1
    then have *: f (x $ 1) $ 1 = - 1
      using assms(5) by auto
    have sqprojection (f (x$1) - g (x$2)) $ 1 > 0
      by (smt (verit) 1 negatex_def o_apply vector_2(1) x(2))
    moreover
    from x1 have g (x $ 2) ∈ cbox (-1) 1
      using assms(2) by blast
    ultimately show False
      unfolding iff[OF nz] vector_component_simps * mem_box_cart
      using not_le by auto
  next
    case 2
    then have *: f (x $ 1) $ 1 = 1
      using assms(6) by auto
    have sqprojection (f (x$1) - g (x$2)) $ 1 < 0
      by (smt (verit) 2 negatex_def o_apply vector_2(1) x(2) zero_less_one)
    moreover have g (x $ 2) ∈ cbox (-1) 1
      using assms(2) x1 by blast
    ultimately show False
      unfolding iff[OF nz] vector_component_simps * mem_box_cart
      using not_le by auto
  next
    case 3
    then have *: g (x $ 2) $ 2 = - 1
      using assms(7) by auto
    moreover have sqprojection (f (x$1) - g (x$2)) $ 2 < 0
      by (smt (verit, ccfv_SIG) 3 negatex_def o_apply vector_2(2) x(2))
    moreover from x1 have f (x $ 1) ∈ cbox (-1) 1
      using assms(1) by blast
    ultimately show False
      by (smt (verit, del_insts) iff(2) mem_box_cart(2) neg_one_index nz vec-
tor_minus_component)
  next
    case 4
    then have *: g (x $ 2) $ 2 = 1
      using assms(8) by auto
    have sqprojection (f (x$1) - g (x$2)) $ 2 > 0
      by (smt (verit, best) 4 negatex_def o_apply vector_2(2) x(2))
    moreover
    from x1 have f (x $ 1) ∈ cbox (-1) 1
      using assms(1) by blast
    ultimately show False
    by (smt (verit) * iff(1) mem_box_cart(2) nz one_index vector_minus_component)
  qed
qed
```

proposition *fashoda_unit_path*:
fixes *f g* :: *real* ⇒ *real^2*
assumes *path f*
and *path g*
and *path_image f* ⊆ *cbox* (−*1*) *1*
and *path_image g* ⊆ *cbox* (−*1*) *1*
and (*pathstart f*)$*1* = −*1*
and (*pathfinish f*)$*1* = *1*
and (*pathstart g*)$*2* = −*1*
and (*pathfinish g*)$*2* = *1*
obtains *z* **where** *z* ∈ *path_image f* **and** *z* ∈ *path_image g*
proof −
note *assms* = *assms*[*unfolded path_def pathstart_def pathfinish_def path_image_def*]
define *iscale* **where** [*abs_def*]: *iscale z* = *inverse 2* ∗*R* (*z* + *1*) **for** *z* :: *real*
have *isc*: *iscale* ‘ {− *1..1*} ⊆ {*0..1*}
unfolding *iscale_def* **by** *auto*
have ∃ *s*∈{− *1..1*}. ∃ *t*∈{− *1..1*}. (*f* ∘ *iscale*) *s* = (*g* ∘ *iscale*) *t*
proof (*rule fashoda_unit*)
show (*f* ∘ *iscale*) ‘ {− *1..1*} ⊆ *cbox* (− *1*) *1* (*g* ∘ *iscale*) ‘ {− *1..1*} ⊆ *cbox* (−
1) *1*
using *isc* **and** *assms*(*3*−*4*) **by** (*auto simp add*: *image_comp* [*symmetric*])
have ∗: *continuous_on* {− *1..1*} *iscale*
unfolding *iscale_def* **by** (*rule continuous_intros*)+
show *continuous_on* {− *1..1*} (*f* ∘ *iscale*)
using ∗ *assms*(*1*) *continuous_on_compose continuous_on_subset isc* **by** *blast*
show *continuous_on* {− *1..1*} (*g* ∘ *iscale*)
by (*meson* ∗ *assms*(*2*) *continuous_on_compose continuous_on_subset isc*)
have ∗: (*1* / *2*) ∗*R* (*1* + (*1*::*real^1*)) = *1*
unfolding *vec_eq_iff* **by** *auto*
show (*f* ∘ *iscale*) (− *1*) $ *1* = − *1*
and (*f* ∘ *iscale*) *1* $ *1* = *1*
and (*g* ∘ *iscale*) (− *1*) $ *2* = −*1*
and (*g* ∘ *iscale*) *1* $ *2* = *1*
unfolding *o_def iscale_def* **using** *assms* **by** (*auto simp add*: ∗)
qed
then obtain *s t* **where** *st*: *s* ∈ {− *1..1*} *t* ∈ {− *1..1*} (*f* ∘ *iscale*) *s* = (*g* ∘
iscale) *t*
by *auto*
show *thesis*
proof
show *f* (*iscale s*) ∈ *path_image f*
by (*metis image_eqI image_subset_iff isc path_image_def st*(*1*))
show *f* (*iscale s*) ∈ *path_image g*
by (*metis comp_def image_eqI image_subset_iff isc path_image_def st*(*2*)
st(*3*))
qed
qed

theorem *fashoda*:

```
  fixes b :: real^2
  assumes path f
    and path g
    and path_image f ⊆ cbox a b
    and path_image g ⊆ cbox a b
    and (pathstart f)$1 = a$1
    and (pathfinish f)$1 = b$1
    and (pathstart g)$2 = a$2
    and (pathfinish g)$2 = b$2
  obtains z where z ∈ path_image f and z ∈ path_image g
proof −
  fix P Q S
  presume P ∨ Q ∨ S P ⟹ thesis and Q ⟹ thesis and S ⟹ thesis
  then show thesis
    by auto
next
  have cbox a b ≠ {}
    using assms(3) using path_image_nonempty[of f] by auto
  then have a ≤ b
   unfolding interval_eq_empty_cart less_eq_vec_def by (auto simp add: not_less)
  then show a$1 = b$1 ∨ a$2 = b$2 ∨ (a$1 < b$1 ∧ a$2 < b$2)
    unfolding less_eq_vec_def forall_2 by auto
next
  assume as: a$1 = b$1
  have ∃ z∈path_image g. z$2 = (pathstart f)$2
  proof (rule connected_ivt_component_cart)
    show pathstart g $ 2 ≤ pathstart f $ 2
      by (metis assms(3) assms(7) mem_box_cart(2) pathstart_in_path_image
subset_iff)
    show pathstart f $ 2 ≤ pathfinish g $ 2
     by (metis assms(3) assms(8) in_mono mem_box_cart(2) pathstart_in_path_image)
    show connected (path_image g)
      using assms(2) by blast
  qed (auto simp: path_defs)
  then obtain z :: real^2 where z: z ∈ path_image g z $ 2 = pathstart f $ 2 ..
  have z ∈ cbox a b
    using assms(4) z(1) by blast
  then have z = f 0
   by (smt (verit) as assms(5) exhaust_2 mem_box_cart(2) nle_le pathstart_def
vec_eq_iff z(2))
  then show thesis
    by (metis path_defs(2) pathstart_in_path_image that z(1))
next
  assume as: a$2 = b$2
  have ∃ z∈path_image f. z$1 = (pathstart g)$1
  proof (rule connected_ivt_component_cart)
    show pathstart f $ 1 ≤ pathstart g $ 1
      using assms(4) assms(5) mem_box_cart(2) by fastforce
    show pathstart g $ 1 ≤ pathfinish f $ 1
```

```
      using assms(4) assms(6) mem_box_cart(2) pathstart_in_path_image by
fastforce
    show connected (path_image f)
      by (simp add: assms(1) connected_path_image)
  qed (auto simp: path_defs)
  then obtain z where z: z ∈ path_image f z $ 1 = pathstart g $ 1 ..
  have z ∈ cbox a b
    using assms(3) z(1) by auto
  then have z = g 0
   by (smt (verit) as assms(7) exhaust_2 mem_box_cart(2) pathstart_def vec_eq_iff
z(2))
  then show thesis
    by (metis path_defs(2) pathstart_in_path_image that z(1))
next
  assume as: a $ 1 < b $ 1 ∧ a $ 2 < b $ 2
  have int_nem: cbox (-1) (1::real^2) ≠ {}
    unfolding interval_eq_empty_cart by auto
  obtain z :: real^2 where z:
      z ∈ (interval_bij (a, b) (- 1, 1) ∘ f) ` {0..1}
      z ∈ (interval_bij (a, b) (- 1, 1) ∘ g) ` {0..1}
  proof (rule fashoda_unit_path)
    show path (interval_bij (a, b) (- 1, 1) ∘ f)
      by (meson assms(1) continuous_on_interval_bij path_continuous_image)
    show path (interval_bij (a, b) (- 1, 1) ∘ g)
      by (meson assms(2) continuous_on_interval_bij path_continuous_image)
    show path_image (interval_bij (a, b) (- 1, 1) ∘ f) ⊆ cbox (- 1) 1
      using assms(3)
      by (simp add: path_image_def in_interval_interval_bij int_nem subset_eq)
    show path_image (interval_bij (a, b) (- 1, 1) ∘ g) ⊆ cbox (- 1) 1
      using assms(4)
      by (simp add: path_image_def in_interval_interval_bij int_nem subset_eq)
    show pathstart (interval_bij (a, b) (- 1, 1) ∘ f) $ 1 = - 1
         pathfinish (interval_bij (a, b) (- 1, 1) ∘ f) $ 1 = 1
         pathstart (interval_bij (a, b) (- 1, 1) ∘ g) $ 2 = - 1
         pathfinish (interval_bij (a, b) (- 1, 1) ∘ g) $ 2 = 1
      using assms as
       by (simp_all add: cart_eq_inner_axis pathstart_def pathfinish_def inter-
val_bij_def)
         (simp_all add: inner_axis)
  qed (auto simp: path_defs)
  then obtain zf zg where zf: zf ∈ {0..1} z = (interval_bij (a, b) (- 1, 1) ∘ f)
zf
                      and zg: zg ∈ {0..1} z = (interval_bij (a, b) (- 1, 1) ∘ g) zg
    by blast
  have ∗: ∀ i. (- 1) $ i < (1::real^2) $ i ∧ a $ i < b $ i
    unfolding forall_2 using as by auto
  show thesis
  proof (rule_tac z=interval_bij (- 1,1) (a,b) z in that)
    show interval_bij (- 1, 1) (a, b) z ∈ path_image f
```

```
    using zf by (simp add: interval_bij_bij_cart[OF *] path_image_def)
  show interval_bij (- 1, 1) (a, b) z ∈ path_image g
    using zg by (simp add: interval_bij_bij_cart[OF *] path_image_def)
 qed
qed
```

6.29.3 Some slightly ad hoc lemmas I use below

```
lemma segment_vertical:
 fixes a :: real^2
 assumes a$1 = b$1
 shows x ∈ closed_segment a b ⟷
   x$1 = a$1 ∧ x$1 = b$1 ∧ (a$2 ≤ x$2 ∧ x$2 ≤ b$2 ∨ b$2 ≤ x$2 ∧ x$2 ≤
a$2)
 (is _ = ?R)
proof -
 let ?L = ∃ u. (x $ 1 = (1 - u) * a $ 1 + u * b $ 1 ∧ x $ 2 = (1 - u) * a $ 2
+ u * b $ 2) ∧ 0 ≤ u ∧ u ≤ 1
 {
   presume ?L ⟹ ?R and ?R ⟹ ?L
   then show ?thesis
     unfolding closed_segment_def mem_Collect_eq
    unfolding vec_eq_iff forall_2 scalar_mult_eq_scaleR[symmetric] vector_component_simps
     by blast
 }
 {
   assume ?L
   then obtain u where u:
       x $ 1 = (1 - u) * a $ 1 + u * b $ 1
       x $ 2 = (1 - u) * a $ 2 + u * b $ 2
       0 ≤ u u ≤ 1
     by blast
   { fix b a
     assume b + u * a > a + u * b
     then have (1 - u) * b > (1 - u) * a
       by (auto simp add:field_simps)
     then have b ≥ a
       using not_less_iff_gr_or_eq u(4) by fastforce
     then have u * a ≤ u * b
       by (simp add: mult_left_mono u(3))
   }
   moreover
   { fix a b
     assume u * b > u * a
     then have (1 - u) * a ≤ (1 - u) * b
       using less_eq_real_def u(3) u(4) by force
     then have a + u * b ≤ b + u * a
       by (auto simp add: field_simps)
   } ultimately show ?R
```

```
    by (force simp add: u assms field_simps not_le)
  }
  {
    assume ?R
    then show ?L
    proof (cases x$2 = b$2)
      case True
      with ‹?R› show ?L
        by (rule_tac x=(x$2 - a$2) / (b$2 - a$2) in exI) (auto simp add:
field_simps)
    next
      case False
      with ‹?R› show ?L
        by (rule_tac x=1 - (x$2 - b$2) / (a$2 - b$2) in exI) (auto simp add:
field_simps)
    qed
  }
qed
```

Essentially duplicate proof that could be done by swapping co-ordinates

```
lemma segment_horizontal:
  fixes a :: real^2
  assumes a$2 = b$2
  shows x ∈ closed_segment a b ⟷
    x$2 = a$2 ∧ x$2 = b$2 ∧ (a$1 ≤ x$1 ∧ x$1 ≤ b$1 ∨ b$1 ≤ x$1 ∧ x$1 ≤
a$1)
  (is _ = ?R)
proof -
  let ?L = ∃ u. (x $ 1 = (1 - u) * a $ 1 + u * b $ 1 ∧ x $ 2 = (1 - u) * a $ 2
+ u * b $ 2) ∧ 0 ≤ u ∧ u ≤ 1
  {
    presume ?L ⟹ ?R and ?R ⟹ ?L
    then show ?thesis
      unfolding closed_segment_def mem_Collect_eq
    unfolding vec_eq_iff forall_2 scalar_mult_eq_scaleR[symmetric] vector_component_simps
      by blast
  }
  {
    assume ?L
    then obtain u where u:
        x $ 1 = (1 - u) * a $ 1 + u * b $ 1
        x $ 2 = (1 - u) * a $ 2 + u * b $ 2
        0 ≤ u u ≤ 1
      by blast
    { fix b a
      assume b + u * a > a + u * b
      then have (1 - u) * b > (1 - u) * a
        by (auto simp add: field_simps)
      then have b ≥ a
```

```
      by (smt (verit, best) mult_left_mono u(4))
    then have u * a ≤ u * b
      by (simp add: mult_left_mono u(3))
  }
  moreover
  { fix a b
    assume u * b > u * a
    then have (1 - u) * a ≤ (1 - u) * b
      using less_eq_real_def u(3) u(4) by force
    then have a + u * b ≤ b + u * a
      by (auto simp add: field_simps)
  }
  ultimately show ?R
    by (force simp add: u assms field_simps not_le intro: )
 }
 { assume ?R
  then show ?L
  proof (cases x$1 = b$1)
    case True
    with ‹?R› show ?L
      by (rule_tac x=(x$1 - a$1) / (b$1 - a$1) in exI) (auto simp add:
field_simps)
  next
    case False
    with ‹?R› show ?L
      by (rule_tac x=1 - (x$1 - b$1) / (a$1 - b$1) in exI) (auto simp add:
field_simps)
  qed
 }
qed
```

6.29.4 Useful Fashoda corollary pointed out to me by Tom Hales

```
corollary fashoda_interlace:
 fixes a :: real^2
 assumes path f
   and path g
   and paf: path_image f ⊆ cbox a b
   and pag: path_image g ⊆ cbox a b
   and (pathstart f)$2 = a$2
   and (pathfinish f)$2 = a$2
   and (pathstart g)$2 = a$2
   and (pathfinish g)$2 = a$2
   and (pathstart f)$1 < (pathstart g)$1
   and (pathstart g)$1 < (pathfinish f)$1
   and (pathfinish f)$1 < (pathfinish g)$1
 obtains z where z ∈ path_image f and z ∈ path_image g
proof -
```

```
  have cbox a b ≠ {}
    using path_image_nonempty[of f] using assms(3) by auto
  note ab=this[unfolded interval_eq_empty_cart not_ex forall_2 not_less]
  have pathstart f ∈ cbox a b
    and pathfinish f ∈ cbox a b
    and pathstart g ∈ cbox a b
    and pathfinish g ∈ cbox a b
    using pathstart_in_path_image pathfinish_in_path_image
    using assms(3-4)
    by auto
  note startfin = this[unfolded mem_box_cart forall_2]
  let ?P1 = linepath (vector[a$1 - 2, a$2 - 2]) (vector[(pathstart f)$1,a$2 -
2]) +++
     linepath(vector[(pathstart f)$1,a$2 - 2])(pathstart f) +++ f +++
     linepath(pathfinish f)(vector[(pathfinish f)$1,a$2 - 2]) +++
     linepath(vector[(pathfinish f)$1,a$2 - 2])(vector[b$1 + 2,a$2 - 2])
  let ?P2 = linepath(vector[(pathstart g)$1, (pathstart g)$2 - 3])(pathstart g)
+++ g +++
     linepath(pathfinish g)(vector[(pathfinish g)$1,a$2 - 1]) +++
     linepath(vector[(pathfinish g)$1,a$2 - 1])(vector[b$1 + 1,a$2 - 1]) +++
     linepath(vector[b$1 + 1,a$2 - 1])(vector[b$1 + 1,b$2 + 3])
  let ?a = vector[a$1 - 2, a$2 - 3]
  let ?b = vector[b$1 + 2, b$2 + 3]
  have P1P2: path_image ?P1 = path_image (linepath (vector[a$1 - 2, a$2 -
2]) (vector[(pathstart f)$1,a$2 - 2])) ∪
    path_image (linepath(vector[(pathstart f)$1,a$2 - 2])(pathstart f)) ∪ path_image
f ∪
      path_image (linepath(pathfinish f)(vector[(pathfinish f)$1,a$2 - 2])) ∪
      path_image (linepath(vector[(pathfinish f)$1,a$2 - 2])(vector[b$1 + 2,a$2
- 2]))
    path_image ?P2 = path_image(linepath(vector[(pathstart g)$1, (pathstart g)$2
- 3])(pathstart g)) ∪ path_image g ∪
      path_image(linepath(pathfinish g)(vector[(pathfinish g)$1,a$2 - 1])) ∪
     path_image(linepath(vector[(pathfinish g)$1,a$2 - 1])(vector[b$1 + 1,a$2 -
1])) ∪
      path_image(linepath(vector[b$1 + 1,a$2 - 1])(vector[b$1 + 1,b$2 + 3]))
using assms(1-2)
     by(auto simp add: path_image_join)
  have abab: cbox a b ⊆ cbox ?a ?b
    unfolding interval_cbox_cart[symmetric]
    by (auto simp add:less_eq_vec_def forall_2)
  obtain z where
    z ∈ path_image
         (linepath (vector [a $ 1 - 2, a $ 2 - 2]) (vector [pathstart f $ 1, a $ 2
- 2]) +++
          linepath (vector [pathstart f $ 1, a $ 2 - 2]) (pathstart f) +++
          f +++
          linepath (pathfinish f) (vector [pathfinish f $ 1, a $ 2 - 2]) +++
          linepath (vector [pathfinish f $ 1, a $ 2 - 2]) (vector [b $ 1 + 2, a $ 2
```

```
− 2]))
    z ∈ path_image
        (linepath (vector [pathstart g $ 1, pathstart g $ 2 − 3]) (pathstart g) +++
         g +++
         linepath (pathfinish g) (vector [pathfinish g $ 1, a $ 2 − 1]) +++
         linepath (vector [pathfinish g $ 1, a $ 2 − 1]) (vector [b $ 1 + 1, a $ 2
− 1]) +++
         linepath (vector [b $ 1 + 1, a $ 2 − 1]) (vector [b $ 1 + 1, b $ 2 + 3]))
   apply (rule fashoda[of ?P1 ?P2 ?a ?b])
  unfolding pathstart_join pathfinish_join pathstart_linepath pathfinish_linepath
vector_2
  proof −
    show path ?P1 and path ?P2
      using assms by auto
    show path_image ?P1 ⊆ cbox ?a ?b path_image ?P2 ⊆ cbox ?a ?b
      unfolding P1P2 path_image_linepath using startfin paf pag
     by (auto simp: mem_box_cart segment_horizontal segment_vertical forall_2)
    show a $ 1 − 2 = a $ 1 − 2
      and b $ 1 + 2 = b $ 1 + 2
      and pathstart g $ 2 − 3 = a $ 2 − 3
      and b $ 2 + 3 = b $ 2 + 3
      by (auto simp add: assms)
  qed
  note z=this[unfolded P1P2 path_image_linepath]
  show thesis
  proof (rule that[of z])
    have (z ∈ closed_segment (vector [a $ 1 − 2, a $ 2 − 2]) (vector [pathstart f
$ 1, a $ 2 − 2]) ∨
      z ∈ closed_segment (vector [pathstart f $ 1, a $ 2 − 2]) (pathstart f)) ∨
      z ∈ closed_segment (pathfinish f) (vector [pathfinish f $ 1, a $ 2 − 2]) ∨
      z ∈ closed_segment (vector [pathfinish f $ 1, a $ 2 − 2]) (vector [b $ 1 + 2,
a $ 2 − 2]) ⟹
    (((z ∈ closed_segment (vector [pathstart g $ 1, pathstart g $ 2 − 3]) (pathstart
g)) ∨
      z ∈ closed_segment (pathfinish g) (vector [pathfinish g $ 1, a $ 2 − 1])) ∨
      z ∈ closed_segment (vector [pathfinish g $ 1, a $ 2 − 1]) (vector [b $ 1 + 1,
a $ 2 − 1])) ∨
      z ∈ closed_segment (vector [b $ 1 + 1, a $ 2 − 1]) (vector [b $ 1 + 1, b $
2 + 3]) ⟹ False
    proof (simp only: segment_vertical segment_horizontal vector_2, goal_cases)
      case prems: 1
      have pathfinish f ∈ cbox a b
        using assms(3) pathfinish_in_path_image[of f] by auto
      then have 1 + b $ 1 ≤ pathfinish f $ 1 ⟹ False
        unfolding mem_box_cart forall_2 by auto
      then have z$1 ≠ pathfinish f$1
        using assms(10) assms(11) prems(2) by auto
      moreover have pathstart f ∈ cbox a b
        using assms(3) pathstart_in_path_image[of f]
```

```
      by auto
    then have 1 + b $ 1 ≤ pathstart f $ 1 ⟹ False
      unfolding mem_box_cart forall_2
      by auto
    then have z$1 ≠ pathstart f$1
      using prems(2) using assms ab
      by (auto simp add: field_simps)
    ultimately have ∗: z$2 = a$2 − 2
      using prems(1) by auto
    have z$1 ≠ pathfinish g$1
      using prems(2) assms ab
      by (auto simp add: field_simps ∗)
    moreover have pathstart g ∈ cbox a b
      using assms(4) pathstart_in_path_image[of g]
      by auto
    note this[unfolded mem_box_cart forall_2]
    then have z$1 ≠ pathstart g$1
      using prems(1) assms ab
      by (auto simp add: field_simps ∗)
    ultimately have a $ 2 − 1 ≤ z $ 2 ∧ z $ 2 ≤ b $ 2 + 3 ∨ b $ 2 + 3 ≤ z
$ 2 ∧ z $ 2 ≤ a $ 2 − 1
      using prems(2)  unfolding ∗ assms by (auto simp add: field_simps)
    then show False
      unfolding ∗ using ab by auto
  qed
  then have z ∈ path_image f ∨ z ∈ path_image g
    using z unfolding Un_iff by blast
  then have z′: z ∈ cbox a b
    using assms(3−4) by auto
  have a $ 2 = z $ 2 ⟹ (z $ 1 = pathstart f $ 1 ∨ z $ 1 = pathfinish f $ 1)
⟹
    z = pathstart f ∨ z = pathfinish f
    unfolding vec_eq_iff forall_2 assms
    by auto
  with z′ show z ∈ path_image f
    using z(1)
    unfolding Un_iff mem_box_cart forall_2
    using assms(5) assms(6) segment_horizontal segment_vertical by auto
  have a $ 2 = z $ 2 ⟹ (z $ 1 = pathstart g $ 1 ∨ z $ 1 = pathfinish g $ 1)
⟹
    z = pathstart g ∨ z = pathfinish g
    unfolding vec_eq_iff forall_2 assms
    by auto
  with z′ show z ∈ path_image g
    using z(2)
    unfolding Un_iff mem_box_cart forall_2
    using assms(7) assms(8) segment_horizontal segment_vertical by auto
 qed
qed
```

end

6.30 Vector Cross Products in 3 Dimensions

theory *Cross3*
imports *Determinants Cartesian_Euclidean_Space*
begin

context includes *no_Set_Product_syntax*
begin — locally disable syntax for set product, to avoid warnings

definition *cross3* :: [*real^3*, *real^3*] ⇒ *real^3* (**infixr** × *80*)
where $a \times b \equiv$
vector [*a$2* ∗ *b$3* − *a$3* ∗ *b$2*,
a$3 ∗ *b$1* − *a$1* ∗ *b$3*,
a$1 ∗ *b$2* − *a$2* ∗ *b$1*]

end

bundle *cross3_syntax* **begin**
notation *cross3* (**infixr** × *80*)
no_notation *Product_Type.Times* (**infixr** × *80*)
end

bundle *no_cross3_syntax* **begin**
no_notation *cross3* (**infixr** × *80*)
notation *Product_Type.Times* (**infixr** × *80*)
end

unbundle *cross3_syntax*

6.30.1 Basic lemmas

lemmas *cross3_simps* = *cross3_def inner_vec_def sum_3 det_3 vec_eq_iff vector_def algebra_simps*

lemma *dot_cross_self*: $x \cdot (x \times y) = 0$ $x \cdot (y \times x) = 0$ $(x \times y) \cdot y = 0$ $(y \times x) \cdot y = 0$
by (*simp_all add*: *orthogonal_def cross3_simps*)

lemma *orthogonal_cross*: *orthogonal* $(x \times y)$ *x orthogonal* $(x \times y)$ *y*
orthogonal y $(x \times y)$ *orthogonal* $(x \times y)$ *x*
by (*simp_all add*: *orthogonal_def dot_cross_self*)

lemma *cross_zero_left* [*simp*]: $0 \times x = 0$ **and** *cross_zero_right* [*simp*]: $x \times 0 = 0$ **for** *x*::*real^3*
by (*simp_all add*: *cross3_simps*)

lemma *cross_skew*: $(x \times y) = -(y \times x)$ **for** x::*real^3*
by (*simp add*: *cross3_simps*)

lemma *cross_refl* [*simp*]: $x \times x = 0$ **for** x::*real^3*
by (*simp add*: *cross3_simps*)

lemma *cross_add_left*: $(x + y) \times z = (x \times z) + (y \times z)$ **for** x::*real^3*
by (*simp add*: *cross3_simps*)

lemma *cross_add_right*: $x \times (y + z) = (x \times y) + (x \times z)$ **for** x::*real^3*
by (*simp add*: *cross3_simps*)

lemma *cross_mult_left*: $(c *_R x) \times y = c *_R (x \times y)$ **for** x::*real^3*
by (*simp add*: *cross3_simps*)

lemma *cross_mult_right*: $x \times (c *_R y) = c *_R (x \times y)$ **for** x::*real^3*
by (*simp add*: *cross3_simps*)

lemma *cross_minus_left* [*simp*]: $(-x) \times y = -(x \times y)$ **for** x::*real^3*
by (*simp add*: *cross3_simps*)

lemma *cross_minus_right* [*simp*]: $x \times -y = -(x \times y)$ **for** x::*real^3*
by (*simp add*: *cross3_simps*)

lemma *left_diff_distrib*: $(x - y) \times z = x \times z - y \times z$ **for** x::*real^3*
by (*simp add*: *cross3_simps*)

lemma *right_diff_distrib*: $x \times (y - z) = x \times y - x \times z$ **for** x::*real^3*
by (*simp add*: *cross3_simps*)

hide_fact (**open**) *left_diff_distrib right_diff_distrib*

proposition *Jacobi*: $x \times (y \times z) + y \times (z \times x) + z \times (x \times y) = 0$ **for** x::*real^3*
by (*simp add*: *cross3_simps*)

proposition *Lagrange*: $x \times (y \times z) = (x \cdot z) *_R y - (x \cdot y) *_R z$
by (*simp add*: *cross3_simps*) (*metis* (*full_types*) *exhaust_3*)

proposition *cross_triple*: $(x \times y) \cdot z = (y \times z) \cdot x$
by (*simp add*: *cross3_def inner_vec_def sum_3 vec_eq_iff algebra_simps*)

lemma *cross_components*:
$(x \times y)\$1 = x\$2 * y\$3 - y\$2 * x\$3$ $(x \times y)\$2 = x\$3 * y\$1 - y\$3 * x\$1$ $(x \times y)\$3 = x\$1 * y\$2 - y\$1 * x\$2$
by (*simp_all add*: *cross3_def inner_vec_def sum_3 vec_eq_iff algebra_simps*)

lemma *cross_basis*: $(axis\ 1\ 1) \times (axis\ 2\ 1) = axis\ 3\ 1$ $(axis\ 2\ 1) \times (axis\ 1\ 1) = -(axis\ 3\ 1)$
$(axis\ 2\ 1) \times (axis\ 3\ 1) = axis\ 1\ 1$ $(axis\ 3\ 1) \times (axis\ 2\ 1) = -(axis$

1 1)
(*axis 3 1*) × (*axis 1 1*) = *axis 2 1* (*axis 1 1*) × (*axis 3 1*) = −(*axis 2 1*)
using *exhaust_3*
by (*force simp add*: *axis_def cross3_simps*)+

lemma *cross_basis_nonzero*:
$u \neq 0 \Longrightarrow u \times axis\ 1\ 1 \neq 0 \vee u \times axis\ 2\ 1 \neq 0 \vee u \times axis\ 3\ 1 \neq 0$
by (*clarsimp simp add*: *axis_def cross3_simps*) (*metis exhaust_3*)

lemma *cross_dot_cancel*:
fixes *x*::*real*^3
assumes *deq*: $x \cdot y = x \cdot z$ **and** *veq*: $x \times y = x \times z$ **and** *x*: $x \neq 0$
shows $y = z$
proof −
have $x \cdot x \neq 0$
by (*simp add*: *x*)
then have $y - z = 0$
using *veq*
by (*metis* (*no_types*, *lifting*) *Cross3.right_diff_distrib Lagrange deq eq_iff_diff_eq_0 inner_diff_right scale_eq_0_iff*)
then show *?thesis*
using *eq_iff_diff_eq_0* **by** *blast*
qed

lemma *norm_cross_dot*: $(norm\ (x \times y))^2 + (x \cdot y)^2 = (norm\ x * norm\ y)^2$
unfolding *power2_norm_eq_inner power_mult_distrib*
by (*simp add*: *cross3_simps power2_eq_square*)

lemma *dot_cross_det*: $x \cdot (y \times z) = det(vector[x,y,z])$
by (*simp add*: *cross3_simps*)

lemma *cross_cross_det*: $(w \times x) \times (y \times z) = det(vector[w,x,z]) *_R y - det(vector[w,x,y]) *_R z$
using *exhaust_3* **by** (*force simp add*: *cross3_simps*)

proposition *dot_cross*: $(w \times x) \cdot (y \times z) = (w \cdot y) * (x \cdot z) - (w \cdot z) * (x \cdot y)$
by (*force simp add*: *cross3_simps*)

proposition *norm_cross*: $(norm\ (x \times y))^2 = (norm\ x)^2 * (norm\ y)^2 - (x \cdot y)^2$
unfolding *power2_norm_eq_inner power_mult_distrib*
by (*simp add*: *cross3_simps power2_eq_square*)

lemma *cross_eq_0*: $x \times y = 0 \longleftrightarrow collinear\{0,x,y\}$
proof −
have $x \times y = 0 \longleftrightarrow norm\ (x \times y) = 0$
by *simp*
also have ... $\longleftrightarrow (norm\ x * norm\ y)^2 = (x \cdot y)^2$
using *norm_cross* [*of x y*] **by** (*auto simp*: *power_mult_distrib*)

also have ... $\longleftrightarrow$ $|x \cdot y| = norm\ x * norm\ y$
using *power2_eq_iff*
by (*metis* (*mono_tags*, *opaque_lifting*) *abs_minus abs_norm_cancel abs_power2 norm_mult power_abs real_norm_def*)
also have ... $\longleftrightarrow$ *collinear* $\{0, x, y\}$
by (*rule norm_cauchy_schwarz_equal*)
finally show *?thesis* **.**
qed

lemma *cross_eq_self*: $x \times y = x \longleftrightarrow x = 0\ x \times y = y \longleftrightarrow y = 0$
apply (*metis cross_zero_left dot_cross_self*(*1*) *inner_eq_zero_iff*)
by (*metis cross_zero_right dot_cross_self*(*2*) *inner_eq_zero_iff*)

lemma *norm_and_cross_eq_0*:
$x \cdot y = 0 \wedge x \times y = 0 \longleftrightarrow x = 0 \vee y = 0$ (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (*metis cross_dot_cancel cross_zero_right inner_zero_right*)
qed *auto*

lemma *bilinear_cross*: *bilinear*($\times$)
apply (*auto simp add*: *bilinear_def linear_def*)
apply *unfold_locales*
apply (*simp add*: *cross_add_right*)
apply (*simp add*: *cross_mult_right*)
apply (*simp add*: *cross_add_left*)
apply (*simp add*: *cross_mult_left*)
done

6.30.2 Preservation by rotation, or other orthogonal transformation up to sign

lemma *cross_matrix_mult*: *transpose* $A *v ((A *v x) \times (A *v y)) = det\ A *_R (x \times y)$
apply (*simp add*: *vec_eq_iff*)
apply (*simp add*: *vector_matrix_mult_def matrix_vector_mult_def forall_3 cross3_simps*)
done

lemma *cross_orthogonal_matrix*:
assumes *orthogonal_matrix A*
shows $(A *v x) \times (A *v y) = det\ A *_R (A *v (x \times y))$
proof −
have *mat 1* = *transpose* $(A ** transpose\ A)$
by (*metis* (*no_types*) *assms orthogonal_matrix_def transpose_mat*)
then show *?thesis*
by (*metis* (*no_types*) *vector_matrix_mul_rid vector_transpose_matrix cross_matrix_mult matrix_vector_mul_assoc matrix_vector_mult_scaleR*)

qed

lemma *cross_rotation_matrix*: *rotation_matrix A* $\Longrightarrow$ (*A* $*v$ *x*) $\times$ (*A* $*v$ *y*) = *A* $*v$ (*x* $\times$ *y*)
by (*simp add*: *rotation_matrix_def cross_orthogonal_matrix*)

lemma *cross_rotoinversion_matrix*: *rotoinversion_matrix A* $\Longrightarrow$ (*A* $*v$ *x*) $\times$ (*A* $*v$ *y*) = $-$ *A* $*v$ (*x* $\times$ *y*)
by (*simp add*: *rotoinversion_matrix_def cross_orthogonal_matrix scaleR_matrix_vector_assoc*)

lemma *cross_orthogonal_transformation*:
assumes *orthogonal_transformation f*
shows (*f x*) $\times$ (*f y*) = *det*(*matrix f*) $*_R$ *f*(*x* $\times$ *y*)
proof $-$
have *orth*: *orthogonal_matrix* (*matrix f*)
using *assms orthogonal_transformation_matrix* **by** *blast*
have *matrix f* $*v$ *z* = *f z* **for** *z*
using *assms orthogonal_transformation_matrix* **by** *force*
with *cross_orthogonal_matrix* [*OF orth*] **show** *?thesis*
by *simp*
qed

lemma *cross_linear_image*:
⟦*linear f*; $\bigwedge$*x*. *norm*(*f x*) = *norm x*; *det*(*matrix f*) = *1*⟧
$\Longrightarrow$ (*f x*) $\times$ (*f y*) = *f*(*x* $\times$ *y*)
by (*simp add*: *cross_orthogonal_transformation orthogonal_transformation*)

6.30.3 Continuity

lemma *continuous_cross*: ⟦*continuous F f*; *continuous F g*⟧ $\Longrightarrow$ *continuous F* (λ*x*. (*f x*) $\times$ (*g x*))
apply (*subst continuous_componentwise*)
apply (*clarsimp simp add*: *cross3_simps*)
apply (*intro continuous_intros*; *simp*)
done

lemma *continuous_on_cross*:
fixes *f* :: *'a*::*t2_space* $\Rightarrow$ *real*^*3*
shows ⟦*continuous_on S f*; *continuous_on S g*⟧ $\Longrightarrow$ *continuous_on S* (λ*x*. (*f x*) $\times$ (*g x*))
by (*simp add*: *continuous_on_eq_continuous_within continuous_cross*)

unbundle *no_cross3_syntax*

end

6.31 Bounded Continuous Functions

theory *Bounded_Continuous_Function*

imports
Topology_Euclidean_Space
Uniform_Limit
begin

6.31.1 Definition

definition *bcontfun* = $\{f.\ continuous\_on\ UNIV\ f \wedge bounded\ (range\ f)\}$

typedef (**overloaded**) ('*a*, '*b*) *bcontfun* ((_ $\Rightarrow_C$ /_) [*22*, *21*] *21*) =
bcontfun::('*a*::*topological_space* $\Rightarrow$ '*b*::*metric_space*) *set*
morphisms *apply_bcontfun Bcontfun*
by (*auto intro*: *continuous_intros simp*: *bounded_def bcontfun_def*)

declare [[*coercion apply_bcontfun* :: ('*a*::*topological_space* $\Rightarrow_C$ '*b*::*metric_space*) $\Rightarrow$ '*a* $\Rightarrow$ '*b*]]

setup_lifting *type_definition_bcontfun*

lemma *continuous_on_apply_bcontfun*[*intro*, *simp*]: *continuous_on T* (*apply_bcontfun x*)
and *bounded_apply_bcontfun*[*intro*, *simp*]: *bounded* (*range* (*apply_bcontfun x*))
using *apply_bcontfun*[*of x*]
by (*auto simp*: *bcontfun_def intro*: *continuous_on_subset*)

lemma *bcontfun_eqI*: ($\bigwedge x.$ *apply_bcontfun f x* = *apply_bcontfun g x*) $\Longrightarrow$ *f* = *g*
by *transfer auto*

lemma *bcontfunE*:
assumes *f* $\in$ *bcontfun*
obtains *g* **where** *f* = *apply_bcontfun g*
by (*blast intro*: *apply_bcontfun_cases assms*)

lemma *const_bcontfun*: ($\lambda x.$ *b*) $\in$ *bcontfun*
by (*auto simp*: *bcontfun_def image_def*)

lift_definition *const_bcontfun*::'*b*::*metric_space* $\Rightarrow$ ('*a*::*topological_space* $\Rightarrow_C$ '*b*)
is $\lambda c\ \_.\ c$
by (*rule const_bcontfun*)

instantiation *bcontfun* :: (*topological_space*, *metric_space*) *metric_space*
begin

lift_definition *dist_bcontfun* :: '*a* $\Rightarrow_C$ '*b* $\Rightarrow$ '*a* $\Rightarrow_C$ '*b* $\Rightarrow$ *real*
is $\lambda f\ g.$ (*SUP x. dist* (*f x*) (*g x*)) .

definition *uniformity_bcontfun* :: ('*a* $\Rightarrow_C$ '*b* $\times$ '*a* $\Rightarrow_C$ '*b*) *filter*
where *uniformity_bcontfun* = (*INF e*$\in${*0* <..}. *principal* {(*x*, *y*). *dist x y* < *e*})

definition *open_bcontfun* :: ('*a* ⇒$_C$ '*b*) *set* ⇒ *bool*
where *open_bcontfun S* = (∀ *x*∈*S*. ∀$_F$ (*x'*, *y*) *in uniformity*. *x'* = *x* ⟶ *y* ∈ *S*)

lemma *bounded_dist_le_SUP_dist*:
bounded (*range f*) ⟹ *bounded* (*range g*) ⟹ *dist* (*f x*) (*g x*) ≤ (*SUP x*. *dist* (*f x*) (*g x*))
by (*auto intro*!: *cSUP_upper bounded_imp_bdd_above bounded_dist_comp*)

lemma *dist_bounded*:
fixes *f g* :: '*a* ⇒$_C$ '*b*
shows *dist* (*f x*) (*g x*) ≤ *dist f g*
by *transfer* (*auto intro*!: *bounded_dist_le_SUP_dist simp*: *bcontfun_def*)

lemma *dist_bound*:
fixes *f g* :: '*a* ⇒$_C$ '*b*
assumes ⋀*x*. *dist* (*f x*) (*g x*) ≤ *b*
shows *dist f g* ≤ *b*
using *assms*
by *transfer* (*auto intro*!: *cSUP_least*)

lemma *dist_fun_lt_imp_dist_val_lt*:
fixes *f g* :: '*a* ⇒$_C$ '*b*
assumes *dist f g* < *e*
shows *dist* (*f x*) (*g x*) < *e*
using *dist_bounded assms* **by** (*rule le_less_trans*)

instance
proof
fix *f g h* :: '*a* ⇒$_C$ '*b*
show *dist f g* = *0* ⟷ *f* = *g*
proof
have ⋀*x*. *dist* (*f x*) (*g x*) ≤ *dist f g*
by (*rule dist_bounded*)
also assume *dist f g* = *0*
finally show *f* = *g*
by (*auto simp*: *apply_bcontfun_inject*[*symmetric*])
qed (*auto simp*: *dist_bcontfun_def intro*!: *cSup_eq*)
show *dist f g* ≤ *dist f h* + *dist g h*
proof (*rule dist_bound*)
fix *x*
have *dist* (*f x*) (*g x*) ≤ *dist* (*f x*) (*h x*) + *dist* (*g x*) (*h x*)
by (*rule dist_triangle2*)
also have *dist* (*f x*) (*h x*) ≤ *dist f h*
by (*rule dist_bounded*)
also have *dist* (*g x*) (*h x*) ≤ *dist g h*
by (*rule dist_bounded*)
finally show *dist* (*f x*) (*g x*) ≤ *dist f h* + *dist g h*
by *simp*

qed
qed (*rule open_bcontfun_def uniformity_bcontfun_def*)+

end

lift_definition *PiC*::'*a*::*topological_space set* ⇒ ('*a* ⇒ '*b set*) ⇒ ('*a* ⇒$_C$ '*b*::*metric_space*) *set*
is λ*I X*. *Pi I X* ∩ *bcontfun*
by *auto*

lemma *mem_PiC_iff*: *x* ∈ *PiC I X* ⟷ *apply_bcontfun x* ∈ *Pi I X*
by *transfer simp*

lemmas *mem_PiCD* = *mem_PiC_iff*[*THEN iffD1*]
and *mem_PiCI* = *mem_PiC_iff*[*THEN iffD2*]

lemma *tendsto_bcontfun_uniform_limit*:
fixes *f*::'*i* ⇒ '*a*::*topological_space* ⇒$_C$ '*b*::*metric_space*
assumes (*f* ⟶ *l*) *F*
shows *uniform_limit UNIV f l F*
proof (*rule uniform_limitI*)
fix *e*::*real* **assume** *e* > *0*
from *tendstoD*[*OF assms this*] **have** ∀$_F$ *x in F*. *dist* (*f x*) *l* < *e* **.**
then show ∀$_F$ *n in F*. ∀*x*∈*UNIV*. *dist* ((*f n*) *x*) (*l x*) < *e*
by *eventually_elim* (*auto simp*: *dist_fun_lt_imp_dist_val_lt*)
qed

lemma *uniform_limit_tendsto_bcontfun*:
fixes *f*::'*i* ⇒ '*a*::*topological_space* ⇒$_C$ '*b*::*metric_space*
and *l*::'*a*::*topological_space* ⇒$_C$ '*b*::*metric_space*
assumes *uniform_limit UNIV f l F*
shows (*f* ⟶ *l*) *F*
proof (*rule tendstoI*)
fix *e*::*real* **assume** *e* > *0*
then have *e* / *2* > *0* **by** *simp*
from *uniform_limitD*[*OF assms this*]
have ∀$_F$ *i in F*. ∀*x*. *dist* (*f i x*) (*l x*) < *e* / *2* **by** *simp*
then have ∀$_F$ *x in F*. *dist* (*f x*) *l* ≤ *e* / *2*
by *eventually_elim* (*blast intro*: *dist_bound less_imp_le*)
then show ∀$_F$ *x in F*. *dist* (*f x*) *l* < *e*
by *eventually_elim* (*use* ‹*0* < *e*› **in** *auto*)
qed

lemma *uniform_limit_bcontfunE*:
fixes *f*::'*i* ⇒ '*a*::*topological_space* ⇒$_C$ '*b*::*metric_space*
and *l*::'*a*::*topological_space* ⇒ '*b*::*metric_space*
assumes *uniform_limit UNIV f l F F* ≠ *bot*
obtains *l'*::'*a*::*topological_space* ⇒$_C$ '*b*::*metric_space*
where *l* = *l'* (*f* ⟶ *l'*) *F*

by (*metis* (*mono_tags*, *lifting*) *always_eventually apply_bcontfun apply_bcontfun_cases assms*
bcontfun_def mem_Collect_eq uniform_limit_bounded uniform_limit_tendsto_bcontfun uniform_limit_theorem)

lemma *closed_PiC*:
fixes I :: $'a$::*metric_space set*
and X :: $'a \Rightarrow 'b$::*complete_space set*
assumes $\bigwedge i.\ i \in I \Longrightarrow$ *closed* $(X\ i)$
shows *closed* $(PiC\ I\ X)$
unfolding *closed_sequential_limits*
proof *safe*
fix $f\ l$
assume *seq*: $\forall n.\ f\ n \in PiC\ I\ X$ **and** *lim*: $f \longrightarrow l$
show $l \in PiC\ I\ X$
proof (*safe intro*!: *mem_PiCI*)
fix x **assume** $x \in I$
then have *closed* $(X\ x)$
using *assms* **by** *simp*
moreover have *eventually* $(\lambda i.\ f\ i\ x \in X\ x)$ *sequentially*
using *seq* ‹$x \in I$›
by (*auto intro*!: *eventuallyI dest*!: *mem_PiCD simp*: *Pi_iff*)
moreover note *sequentially_bot*
moreover have $(\lambda n.\ (f\ n)\ x) \longrightarrow l\ x$
using *tendsto_bcontfun_uniform_limit*[*OF lim*]
by (*rule tendsto_uniform_limitI*) *simp*
ultimately show $l\ x \in X\ x$
by (*rule Lim_in_closed_set*)
qed
qed

6.31.2 Complete Space

instance *bcontfun* :: (*metric_space*, *complete_space*) *complete_space*
proof
fix f :: *nat* $\Rightarrow$ ($'a$, $'b$) *bcontfun*
assume *Cauchy* f — Cauchy equals uniform convergence
then obtain g **where** *uniform_limit UNIV f g sequentially*
using *uniformly_convergent_eq_cauchy*[*of* λ_. *True f*]
unfolding *Cauchy_def uniform_limit_sequentially_iff*
by (*metis dist_fun_lt_imp_dist_val_lt*)

from *uniform_limit_bcontfunE*[*OF this sequentially_bot*]
obtain l' **where** g = *apply_bcontfun* l' $(f \longrightarrow l')$ **by** *metis*
then show *convergent f*
by (*intro convergentI*)
qed

6.31.3 Supremum norm for a normed vector space

instantiation *bcontfun* :: (*topological_space*, *real_normed_vector*) *real_vector*
begin

lemma *uminus_cont*: $f \in bcontfun \Longrightarrow (\lambda x.\ -\ f\ x) \in bcontfun$ **for** $f{::}'a \Rightarrow 'b$
by (*auto simp*: *bcontfun_def intro*!: *continuous_intros*)

lemma *plus_cont*: $f \in bcontfun \Longrightarrow g \in bcontfun \Longrightarrow (\lambda x.\ f\ x + g\ x) \in bcontfun$
for $f\ g{::}'a \Rightarrow 'b$
by (*auto simp*: *bcontfun_def intro*!: *continuous_intros bounded_plus_comp*)

lemma *minus_cont*: $f \in bcontfun \Longrightarrow g \in bcontfun \Longrightarrow (\lambda x.\ f\ x - g\ x) \in bcontfun$
for $f\ g{::}'a \Rightarrow 'b$
by (*auto simp*: *bcontfun_def intro*!: *continuous_intros bounded_minus_comp*)

lemma *scaleR_cont*: $f \in bcontfun \Longrightarrow (\lambda x.\ a *_R f\ x) \in bcontfun$ **for** $f :: 'a \Rightarrow 'b$
by (*auto simp*: *bcontfun_def intro*!: *continuous_intros bounded_scaleR_comp*)

lemma *bcontfun_normI*: *continuous_on UNIV f* $\Longrightarrow (\bigwedge x.\ norm\ (f\ x) \leq b) \Longrightarrow f \in bcontfun$
by (*auto simp*: *bcontfun_def intro*: *boundedI*)

lift_definition *uminus_bcontfun*::$('a \Rightarrow_C 'b) \Rightarrow 'a \Rightarrow_C 'b$ **is** $\lambda f\ x.\ -\ f\ x$
by (*rule uminus_cont*)

lift_definition *plus_bcontfun*::$('a \Rightarrow_C 'b) \Rightarrow ('a \Rightarrow_C 'b) \Rightarrow 'a \Rightarrow_C 'b$ **is** $\lambda f\ g\ x.\ f\ x + g\ x$
by (*rule plus_cont*)

lift_definition *minus_bcontfun*::$('a \Rightarrow_C 'b) \Rightarrow ('a \Rightarrow_C 'b) \Rightarrow 'a \Rightarrow_C 'b$ **is** $\lambda f\ g\ x.\ f\ x - g\ x$
by (*rule minus_cont*)

lift_definition *zero_bcontfun*::$'a \Rightarrow_C 'b$ **is** $\lambda\_.\ 0$
by (*rule const_bcontfun*)

lemma *const_bcontfun_0_eq_0*[*simp*]: *const_bcontfun 0 = 0*
by *transfer simp*

lift_definition *scaleR_bcontfun*::*real* $\Rightarrow ('a \Rightarrow_C 'b) \Rightarrow 'a \Rightarrow_C 'b$ **is** $\lambda r\ g\ x.\ r *_R g\ x$
by (*rule scaleR_cont*)

lemmas [*simp*] =
const_bcontfun.rep_eq
uminus_bcontfun.rep_eq
plus_bcontfun.rep_eq
minus_bcontfun.rep_eq
zero_bcontfun.rep_eq

scaleR_bcontfun.rep_eq

instance
by *standard* (*auto intro*!: *bcontfun_eqI simp*: *algebra_simps*)

end

lemma *bounded_norm_le_SUP_norm*:
bounded (*range f*) ⟹ *norm* (*f x*) ≤ (*SUP x*. *norm* (*f x*))
by (*auto intro*!: *cSUP_upper bounded_imp_bdd_above simp*: *bounded_norm_comp*)

instantiation *bcontfun* :: (*topological_space*, *real_normed_vector*) *real_normed_vector*
begin

definition *norm_bcontfun* :: ('*a*, '*b*) *bcontfun* ⇒ *real*
where *norm_bcontfun f* = *dist f 0*

definition *sgn* (*f*::('*a*,'*b*) *bcontfun*) = *f* /$_R$ *norm f*

instance
proof
fix *a* :: *real*
fix *f g* :: ('*a*, '*b*) *bcontfun*
show *dist f g* = *norm* (*f* − *g*)
unfolding *norm_bcontfun_def*
by *transfer* (*simp add*: *dist_norm*)
show *norm* (*f* + *g*) ≤ *norm f* + *norm g*
unfolding *norm_bcontfun_def*
by *transfer*
(*auto intro*!: *cSUP_least norm_triangle_le add_mono bounded_norm_le_SUP_norm*
simp: *dist_norm bcontfun_def*)
show *norm* (*a* *$_R$ *f*) = |*a*| * *norm f*
unfolding *norm_bcontfun_def*
apply *transfer*
by (*rule trans*[*OF _ continuous_at_Sup_mono*[*symmetric*]])
(*auto intro*!: *monoI mult_left_mono continuous_intros bounded_imp_bdd_above*
simp: *bounded_norm_comp bcontfun_def image_comp*)
qed (*auto simp*: *norm_bcontfun_def sgn_bcontfun_def*)

end

lemma *norm_bounded*:
fixes *f* :: ('*a*::*topological_space*, '*b*::*real_normed_vector*) *bcontfun*
shows *norm* (*apply_bcontfun f x*) ≤ *norm f*
using *dist_bounded*[*of f x 0*]
by (*simp add*: *dist_norm*)

lemma *norm_bound*:

fixes f :: ($'a$::*topological_space*, $'b$::*real_normed_vector*) *bcontfun*
assumes $\bigwedge x.$ *norm* (*apply_bcontfun f x*) $\leq b$
shows *norm f* $\leq b$
using *dist_bound*[*of f 0 b*] *assms*
by (*simp add*: *dist_norm*)

6.31.4 (bounded) continuous extenstion

lemma *continuous_on_cbox_bcontfunE*:
fixes *f*::$'a$::*euclidean_space* $\Rightarrow$ $'b$::*metric_space*
assumes *continuous_on* (*cbox a b*) *f*
obtains *g*::$'a \Rightarrow_C$ $'b$ **where**
$\bigwedge x.$ $x \in$ *cbox a b* $\Longrightarrow$ *g x* = *f x*
$\bigwedge x.$ *g x* = *f* (*clamp a b x*)
proof −
define *g* **where** *g* ≡ *ext_cont f a b*
have *g* ∈ *bcontfun*
using *assms*
by (*auto intro*!: *continuous_on_ext_cont simp*: *g_def bcontfun_def*)
(*auto simp*: *g_def ext_cont_def*
intro!: *clamp_bounded compact_imp_bounded*[*OF compact_continuous_image*]
assms)
then obtain *h* **where** *h*: *g* = *apply_bcontfun h* **by** (*rule bcontfunE*)
then have *h x* = *f x* **if** *x* ∈ *cbox a b* **for** *x*
by (*auto simp*: *h*[*symmetric*] *g_def that*)
moreover
have *h x* = *f* (*clamp a b x*) **for** *x*
by (*auto simp*: *h*[*symmetric*] *g_def ext_cont_def*)
ultimately show *?thesis* **..**
qed

lifting_update *bcontfun.lifting*
lifting_forget *bcontfun.lifting*

end

6.32 Infinite Products

theory *Infinite_Products*
imports *Topology_Euclidean_Space Complex_Transcendental*
begin

6.32.1 Preliminaries

lemma *sum_le_prod*:
fixes *f* :: $'a \Rightarrow$ $'b$ :: *linordered_semidom*
assumes $\bigwedge x.$ $x \in A \Longrightarrow f\ x \geq 0$
shows *sum f A* $\leq (\prod x \in A.\ 1 + f\ x)$
using *assms*

proof (*induction A rule*: *infinite_finite_induct*)
case (*insert x A*)
from *insert.hyps* **have** *sum f A + f x * (∏ x∈A. 1) ≤ (∏ x∈A. 1 + f x) + f x * (∏ x∈A. 1 + f x)*
by (*intro add_mono insert mult_left_mono prod_mono*) (*auto intro*: *insert.prems*)
with *insert.hyps* **show** *?case* **by** (*simp add*: *algebra_simps*)
qed *simp_all*

lemma *prod_le_exp_sum*:
fixes *f* :: *'a ⇒ real*
assumes ⋀*x*. *x ∈ A ⟹ f x ≥ 0*
shows *prod* (*λx*. *1 + f x*) *A ≤ exp* (*sum f A*)
using *assms*
proof (*induction A rule*: *infinite_finite_induct*)
case (*insert x A*)
have (*1 + f x*) *** (*∏ x∈A. 1 + f x*) *≤ exp* (*f x*) *** *exp* (*sum f A*)
using *insert.prems* **by** (*intro mult_mono insert prod_nonneg exp_ge_add_one_self*) *auto*
with *insert.hyps* **show** *?case* **by** (*simp add*: *algebra_simps exp_add*)
qed *simp_all*

lemma *lim_ln_1_plus_x_over_x_at_0*: (*λx*::*real*. *ln* (*1 + x*) */ x*) *−0→ 1*
proof (*rule lhopital*)
show (*λx*::*real*. *ln* (*1 + x*)) *−0→ 0*
by (*rule tendsto_eq_intros refl | simp*)+
have *eventually* (*λx*::*real*. *x ∈ {−1/2<..<1/2}*) (*nhds 0*)
by (*rule eventually_nhds_in_open*) *auto*
hence ***: *eventually* (*λx*::*real*. *x ∈ {−1/2<..<1/2}*) (*at 0*)
by (*rule filter_leD* [*rotated*]) (*simp_all add*: *at_within_def*)
show *eventually* (*λx*::*real*. ((*λx*. *ln* (*1 + x*)) *has_field_derivative inverse* (*1 + x*)) (*at x*)) (*at 0*)
using *** **by** *eventually_elim* (*auto intro*!: *derivative_eq_intros simp*: *field_simps*)
show *eventually* (*λx*::*real*. ((*λx*. *x*) *has_field_derivative 1*) (*at x*)) (*at 0*)
using *** **by** *eventually_elim* (*auto intro*!: *derivative_eq_intros simp*: *field_simps*)
show ∀ *F x in at 0*. *x ≠ 0* **by** (*auto simp*: *at_within_def eventually_inf_principal*)
show (*λx*::*real*. *inverse* (*1 + x*) */ 1*) *−0→ 1*
by (*rule tendsto_eq_intros refl | simp*)+
qed *auto*

6.32.2 Definitions and basic properties

definition *raw_has_prod* :: [*nat ⇒ 'a*::{*t2_space*, *comm_semiring_1*}, *nat*, *'a*] *⇒ bool*
where *raw_has_prod f M p* ≡ (*λn*. *∏ i≤n*. *f* (*i+M*)) *⟶* *p ∧ p ≠ 0*

The nonzero and zero cases, as in *Complex Analysis* by Joseph Bak and Donald J.Newman, page 241

definition

has_prod :: (*nat* ⇒ *'a*::{*t2_space*, *comm_semiring_1*}) ⇒ *'a* ⇒ *bool* (**infixr** *has'_prod 80*)
where *f has_prod p* ≡ *raw_has_prod f 0 p* ∨ (∃ *i q*. *p* = *0* ∧ *f i* = *0* ∧ *raw_has_prod f* (*Suc i*) *q*)

definition *convergent_prod* :: (*nat* ⇒ *'a* :: {*t2_space*,*comm_semiring_1*}) ⇒ *bool*
where
convergent_prod f ≡ ∃ *M p*. *raw_has_prod f M p*

definition *prodinf* :: (*nat* ⇒ *'a*::{*t2_space*, *comm_semiring_1*}) ⇒ *'a*
(**binder** ∏ *10*)
where *prodinf f* = (*THE p*. *f has_prod p*)

lemmas *prod_defs* = *raw_has_prod_def has_prod_def convergent_prod_def prodinf_def*

lemma *has_prod_subst*[*trans*]: *f* = *g* ⟹ *g has_prod z* ⟹ *f has_prod z*
by *simp*

lemma *has_prod_cong*: (⋀*n*. *f n* = *g n*) ⟹ *f has_prod c* ⟷ *g has_prod c*
by *presburger*

lemma *raw_has_prod_nonzero* [*simp*]: ¬ *raw_has_prod f M 0*
by (*simp add*: *raw_has_prod_def*)

lemma *raw_has_prod_eq_0*:
fixes *f* :: *nat* ⇒ *'a*::{*semidom*,*t2_space*}
assumes *p*: *raw_has_prod f m p* **and** *i*: *f i* = *0 i* ≥ *m*
shows *p* = *0*
proof −
have *eq0*: (∏ *k*≤*n*. *f* (*k*+*m*)) = *0* **if** *i* − *m* ≤ *n* **for** *n*
proof −
have ∃ *k*≤*n*. *f* (*k* + *m*) = *0*
using *i that* **by** *auto*
then show *?thesis*
by *auto*
qed
have (λ*n*. ∏ *i*≤*n*. *f* (*i* + *m*)) ⟶ *0*
by (*rule LIMSEQ_offset* [**where** *k* = *i*−*m*]) (*simp add*: *eq0*)
with *p* **show** *?thesis*
unfolding *raw_has_prod_def*
using *LIMSEQ_unique* **by** *blast*
qed

lemma *raw_has_prod_Suc*:
raw_has_prod f (*Suc M*) *a* ⟷ *raw_has_prod* (λ*n*. *f* (*Suc n*)) *M a*
unfolding *raw_has_prod_def* **by** *auto*

lemma *has_prod_0_iff*: *f has_prod 0* ⟷ (∃ *i*. *f i* = *0* ∧ (∃ *p*. *raw_has_prod f*

(*Suc i*) *p*))
by (*simp add*: *has_prod_def*)

lemma *has_prod_unique2*:
fixes *f* :: *nat* ⇒ *'a*::{*semidom*,*t2_space*}
assumes *f has_prod a f has_prod b* **shows** *a* = *b*
using *assms*
by (*auto simp*: *has_prod_def raw_has_prod_eq_0*) (*meson raw_has_prod_def sequentially_bot tendsto_unique*)

lemma *has_prod_unique*:
fixes *f* :: *nat* ⇒ *'a* :: {*semidom*,*t2_space*}
shows *f has_prod s* ⟹ *s* = *prodinf f*
by (*simp add*: *has_prod_unique2 prodinf_def the_equality*)

lemma *has_prod_eq_0_iff*:
fixes *f* :: *nat* ⇒ *'a* :: {*semidom*, *comm_semiring_1*, *t2_space*}
assumes *f has_prod P*
shows *P* = *0* ⟷ *0* ∈ *range f*
proof
assume *0* ∈ *range f*
then obtain *N* **where** *N*: *f N* = *0*
by *auto*
have *eventually* (λ*n*. *n* > *N*) *at_top*
by (*rule eventually_gt_at_top*)
hence *eventually* (λ*n*. (∏ *k*<*n*. *f k*) = *0*) *at_top*
by *eventually_elim* (*use N* **in** *auto*)
hence (λ*n*. ∏ *k*<*n*. *f k*) ⟶ *0*
by (*simp add*: *tendsto_eventually*)
moreover have (λ*n*. ∏ *k*<*n*. *f k*) ⟶ *P*
using *assms* **by** (*metis N calculation prod_defs*(*2*) *raw_has_prod_eq_0 zero_le*)
ultimately show *P* = *0*
using *tendsto_unique* **by** *force*
qed (*use assms* **in** ‹*auto simp*: *has_prod_def*›)

lemma *has_prod_0D*:
fixes *f* :: *nat* ⇒ *'a* :: {*semidom*, *comm_semiring_1*, *t2_space*}
shows *f has_prod 0* ⟹ *0* ∈ *range f*
using *has_prod_eq_0_iff*[*of f 0*] **by** *auto*

lemma *has_prod_zeroI*:
fixes *f* :: *nat* ⇒ *'a* :: {*semidom*, *comm_semiring_1*, *t2_space*}
assumes *f has_prod P f n* = *0*
shows *P* = *0*
using *assms* **by** (*auto simp*: *has_prod_eq_0_iff*)

lemma *raw_has_prod_in_Reals*:
assumes *raw_has_prod* (*complex_of_real* ∘ *z*) *M p*
shows *p* ∈ ℝ

using *assms* **by** (*auto simp*: *raw_has_prod_def real_lim_sequentially*)

lemma *raw_has_prod_of_real_iff*: *raw_has_prod* (*complex_of_real* ∘ *z*) *M* (*of_real p*) ⟷ *raw_has_prod z M p*
by (*auto simp*: *raw_has_prod_def tendsto_of_real_iff simp flip*: *of_real_prod*)

lemma *convergent_prod_of_real_iff*: *convergent_prod* (*complex_of_real* ∘ *z*) ⟷ *convergent_prod z*
by (*smt* (*verit*, *best*) *Reals_cases convergent_prod_def raw_has_prod_in_Reals raw_has_prod_of_real_iff*)

lemma *convergent_prod_altdef*:
fixes *f* :: *nat* ⇒ *'a* :: {*t2_space*,*comm_semiring_1*}
shows *convergent_prod f* ⟷ (∃ *M L*. (∀ *n*≥*M*. *f n* ≠ *0*) ∧ (λ*n*. ∏ *i*≤*n*. *f* (*i*+*M*)) ⟶ *L* ∧ *L* ≠ *0*)
proof
assume *convergent_prod f*
then obtain *M L* **where** ∗: (λ*n*. ∏ *i*≤*n*. *f* (*i*+*M*)) ⟶ *L L* ≠ *0*
by (*auto simp*: *prod_defs*)
have *f i* ≠ *0* **if** *i* ≥ *M* **for** *i*
proof
assume *f i* = *0*
have ∗∗: *eventually* (λ*n*. (∏ *i*≤*n*. *f* (*i*+*M*)) = *0*) *sequentially*
using *eventually_ge_at_top*[*of i* − *M*]
proof *eventually_elim*
case (*elim n*)
with ‹*f i* = *0*› **and** ‹*i* ≥ *M*› **show** *?case*
by (*auto intro*!: *bexI*[*of _ i* − *M*] *prod_zero*)
qed
have (λ*n*. (∏ *i*≤*n*. *f* (*i*+*M*))) ⟶ *0*
unfolding *filterlim_iff*
by (*auto dest*!: *eventually_nhds_x_imp_x intro*!: *eventually_mono*[*OF* ∗∗])
from *tendsto_unique*[*OF _ this* ∗(*1*)] **and** ∗(*2*)
show *False* **by** *simp*
qed
with ∗ **show** (∃ *M L*. (∀ *n*≥*M*. *f n* ≠ *0*) ∧ (λ*n*. ∏ *i*≤*n*. *f* (*i*+*M*)) ⟶ *L* ∧ *L* ≠ *0*)
by *blast*
qed (*auto simp*: *prod_defs*)

lemma *raw_has_prod_norm*:
fixes *a* :: *'a* ::*real_normed_field*
assumes *raw_has_prod f M a*
shows *raw_has_prod* (λ*n*. *norm* (*f n*)) *M* (*norm a*)
using *assms* **by** (*auto simp*: *raw_has_prod_def prod_norm tendsto_norm*)

lemma *has_prod_norm*:
fixes *a* :: *'a* ::*real_normed_field*
assumes *f*: *f has_prod a*

```
  shows (λn. norm (f n)) has_prod (norm a)
  using f [unfolded has_prod_def]
proof (elim disjE exE conjE)
  assume f0: raw_has_prod f 0 a
  then show (λn. norm (f n)) has_prod norm a
    using has_prod_def raw_has_prod_norm by blast
next
  fix i p
  assume a = 0 and f i = 0 and p: raw_has_prod f (Suc i) p
  then have Ex (raw_has_prod (λn. norm (f n)) (Suc i))
    using raw_has_prod_norm by blast
  then show ?thesis
    by (metis ‹a = 0› ‹f i = 0› has_prod_0_iff norm_zero)
qed
```

6.32.3 Absolutely convergent products

```
definition abs_convergent_prod :: (nat ⇒ _) ⇒ bool where
  abs_convergent_prod f ⟷ convergent_prod (λi. 1 + norm (f i − 1))

lemma abs_convergent_prodI:
  assumes convergent (λn. ∏i≤n. 1 + norm (f i − 1))
  shows   abs_convergent_prod f
proof −
  from assms obtain L where L: (λn. ∏i≤n. 1 + norm (f i − 1)) ⟶ L
    by (auto simp: convergent_def)
  have L ≥ 1
  proof (rule tendsto_le)
    show eventually (λn. (∏i≤n. 1 + norm (f i − 1)) ≥ 1) sequentially
    proof (intro always_eventually allI)
      fix n
      have (∏i≤n. 1 + norm (f i − 1)) ≥ (∏i≤n. 1)
        by (intro prod_mono) auto
      thus (∏i≤n. 1 + norm (f i − 1)) ≥ 1 by simp
    qed
  qed (use L in simp_all)
  hence L ≠ 0 by auto
  with L show ?thesis unfolding abs_convergent_prod_def prod_defs
    by (intro exI[of _ 0::nat] exI[of _ L]) auto
qed

lemma
  fixes f :: nat ⇒ 'a :: {topological_semigroup_mult,t2_space,idom}
  assumes convergent_prod f
  shows   convergent_prod_imp_convergent:     convergent (λn. ∏i≤n. f i)
    and   convergent_prod_to_zero_iff [simp]: (λn. ∏i≤n. f i) ⟶ 0 ⟷
(∃i. f i = 0)
proof −
  from assms obtain M L
```

```
    where M: ⋀n. n ≥ M ⟹ f n ≠ 0 and (λn. ∏ i≤n. f (i + M)) ⟶ L
and L ≠ 0
    by (auto simp: convergent_prod_altdef)
  note this(2)
  also have (λn. ∏ i≤n. f (i + M)) = (λn. ∏ i=M..M+n. f i)
    by (intro ext prod.reindex_bij_witness[of _ λn. n − M λn. n + M]) auto
  finally have (λn. (∏ i<M. f i) * (∏ i=M..M+n. f i)) ⟶ (∏ i<M. f i) * L
    by (intro tendsto_mult tendsto_const)
 also have (λn. (∏ i<M. f i) * (∏ i=M..M+n. f i)) = (λn. (∏ i∈{..<M}∪{M..M+n}.
f i))
    by (subst prod.union_disjoint) auto
  also have (λn. {..<M} ∪ {M..M+n}) = (λn. {..n+M}) by auto
  finally have lim: (λn. prod f {..n}) ⟶ prod f {..<M} * L
    by (rule LIMSEQ_offset)
  thus convergent (λn. ∏ i≤n. f i)
    by (auto simp: convergent_def)

  show (λn. ∏ i≤n. f i) ⟶ 0 ⟷ (∃ i. f i = 0)
  proof
    assume ∃ i. f i = 0
    then obtain i where f i = 0 by auto
    moreover with M have i < M by (cases i < M) auto
    ultimately have (∏ i<M. f i) = 0 by auto
    with lim show (λn. ∏ i≤n. f i) ⟶ 0 by simp
  next
    assume (λn. ∏ i≤n. f i) ⟶ 0
    from tendsto_unique[OF _ this lim] and ‹L ≠ 0›
    show ∃ i. f i = 0 by auto
  qed
qed

lemma convergent_prod_iff_nz_lim:
  fixes f :: nat ⇒ 'a :: {topological_semigroup_mult,t2_space,idom}
  assumes ⋀i. f i ≠ 0
  shows convergent_prod f ⟷ (∃ L. (λn. ∏ i≤n. f i) ⟶ L ∧ L ≠ 0)
    (is ?lhs ⟷ ?rhs)
proof
  assume ?lhs then show ?rhs
   using assms convergentD convergent_prod_imp_convergent convergent_prod_to_zero_iff
by blast
next
  assume ?rhs then show ?lhs
    unfolding prod_defs
    by (rule_tac x=0 in exI) auto
qed

lemma convergent_prod_iff_convergent:
  fixes f :: nat ⇒ 'a :: {topological_semigroup_mult,t2_space,idom}
  assumes ⋀i. f i ≠ 0
```

shows *convergent_prod f* ⟷ *convergent* (λ*n*. ∏*i*≤*n*. *f i*) ∧ *lim* (λ*n*. ∏*i*≤*n*. *f i*) ≠ *0*
by (*force simp*: *convergent_prod_iff_nz_lim assms convergent_def limI*)

lemma *bounded_imp_convergent_prod*:
fixes *a* :: *nat* ⇒ *real*
assumes *1*: ⋀*n*. *a n* ≥ *1* **and** *bounded*: ⋀*n*. (∏*i*≤*n*. *a i*) ≤ *B*
shows *convergent_prod a*
proof −
have *bdd_above* (*range*(λ*n*. ∏*i*≤*n*. *a i*))
by (*meson bdd_aboveI2 bounded*)
moreover have *incseq* (λ*n*. ∏*i*≤*n*. *a i*)
unfolding *mono_def* **by** (*metis 1 prod_mono2 atMost_subset_iff dual_order.trans finite_atMost zero_le_one*)
ultimately obtain *p* **where** *p*: (λ*n*. ∏*i*≤*n*. *a i*) ⟶ *p*
using *LIMSEQ_incseq_SUP* **by** *blast*
then have *p* ≠ *0*
by (*metis 1 not_one_le_zero prod_ge_1 LIMSEQ_le_const*)
with *1 p* **show** *?thesis*
by (*metis convergent_prod_iff_nz_lim not_one_le_zero*)
qed

lemma *abs_convergent_prod_altdef*:
fixes *f* :: *nat* ⇒ *'a* :: {*one*,*real_normed_vector*}
shows *abs_convergent_prod f* ⟷ *convergent* (λ*n*. ∏*i*≤*n*. *1* + *norm* (*f i* − *1*))
proof
assume *abs_convergent_prod f*
thus *convergent* (λ*n*. ∏*i*≤*n*. *1* + *norm* (*f i* − *1*))
by (*auto simp*: *abs_convergent_prod_def intro*!: *convergent_prod_imp_convergent*)
qed (*auto intro*: *abs_convergent_prodI*)

lemma *Weierstrass_prod_ineq*:
fixes *f* :: *'a* ⇒ *real*
assumes ⋀*x*. *x* ∈ *A* ⟹ *f x* ∈ {*0*..*1*}
shows *1* − *sum f A* ≤ (∏*x*∈*A*. *1* − *f x*)
using *assms*
proof (*induction A rule*: *infinite_finite_induct*)
case (*insert x A*)
from *insert.hyps* **and** *insert.prems*
have *1* − *sum f A* + *f x* ∗ (∏*x*∈*A*. *1* − *f x*) ≤ (∏*x*∈*A*. *1* − *f x*) + *f x* ∗ (∏*x*∈*A*. *1*)
by (*intro insert.IH add_mono mult_left_mono prod_mono*) *auto*
with *insert.hyps* **show** *?case* **by** (*simp add*: *algebra_simps*)
qed *simp_all*

lemma *norm_prod_minus1_le_prod_minus1*:
fixes *f* :: *nat* ⇒ *'a* :: {*real_normed_div_algebra*,*comm_ring_1*}

```
  shows norm (prod (λn. 1 + f n) A − 1) ≤ prod (λn. 1 + norm (f n)) A − 1
proof (induction A rule: infinite_finite_induct)
  case (insert x A)
  from insert.hyps have
    norm ((∏ n∈insert x A. 1 + f n) − 1) =
       norm ((∏ n∈A. 1 + f n) − 1 + f x * (∏ n∈A. 1 + f n))
    by (simp add: algebra_simps)
  also have ... ≤ norm ((∏ n∈A. 1 + f n) − 1) + norm (f x * (∏ n∈A. 1 + f
n))
    by (rule norm_triangle_ineq)
  also have norm (f x * (∏ n∈A. 1 + f n)) = norm (f x) * (∏ x∈A. norm (1 +
f x))
    by (simp add: prod_norm norm_mult)
  also have (∏ x∈A. norm (1 + f x)) ≤ (∏ x∈A. norm (1::'a) + norm (f x))
    by (intro prod_mono norm_triangle_ineq ballI conjI) auto
  also have norm (1::'a) = 1 by simp
  also note insert.IH
  also have (∏ n∈A. 1 + norm (f n)) − 1 + norm (f x) * (∏ x∈A. 1 + norm (f
x)) =
            (∏ n∈insert x A. 1 + norm (f n)) − 1
    using insert.hyps by (simp add: algebra_simps)
  finally show ?case by − (simp_all add: mult_left_mono)
qed simp_all

lemma convergent_prod_imp_ev_nonzero:
  fixes f :: nat ⇒ 'a :: {t2_space,comm_semiring_1}
  assumes convergent_prod f
  shows   eventually (λn. f n ≠ 0) sequentially
  using assms by (auto simp: eventually_at_top_linorder convergent_prod_altdef)

lemma convergent_prod_imp_LIMSEQ:
  fixes f :: nat ⇒ 'a :: {real_normed_field}
  assumes convergent_prod f
  shows   f ⟶ 1
proof −
  from assms obtain M L where L: (λn. ∏ i≤n. f (i+M)) ⟶ L ⋀n. n ≥
M ⟹ f n ≠ 0 L ≠ 0
    by (auto simp: convergent_prod_altdef)
 hence L': (λn. ∏ i≤Suc n. f (i+M)) ⟶ L by (subst filterlim_sequentially_Suc)
  have (λn. (∏ i≤Suc n. f (i+M)) / (∏ i≤n. f (i+M))) ⟶ L / L
    using L L' by (intro tendsto_divide) simp_all
  also from L have L / L = 1 by simp
  also have (λn. (∏ i≤Suc n. f (i+M)) / (∏ i≤n. f (i+M))) = (λn. f (n + Suc
M))
    using assms L by (auto simp: fun_eq_iff atMost_Suc)
  finally show ?thesis by (rule LIMSEQ_offset)
qed

lemma abs_convergent_prod_imp_summable:
```

```
  fixes f :: nat ⇒ 'a :: real_normed_div_algebra
  assumes abs_convergent_prod f
  shows summable (λi. norm (f i − 1))
proof −
  from assms have convergent (λn. ∏i≤n. 1 + norm (f i − 1))
   unfolding abs_convergent_prod_def by (rule convergent_prod_imp_convergent)
  then obtain L where L: (λn. ∏i≤n. 1 + norm (f i − 1)) ⟶ L
    unfolding convergent_def by blast
  have convergent (λn. ∑i≤n. norm (f i − 1))
  proof (rule Bseq_monoseq_convergent)
    have eventually (λn. (∏i≤n. 1 + norm (f i − 1)) < L + 1) sequentially
      using L(1) by (rule order_tendstoD) simp_all
    hence ∀F x in sequentially. norm (∑i≤x. norm (f i − 1)) ≤ L + 1
    proof eventually_elim
      case (elim n)
      have norm (∑i≤n. norm (f i − 1)) = (∑i≤n. norm (f i − 1))
       unfolding real_norm_def by (intro abs_of_nonneg sum_nonneg) simp_all
      also have ... ≤ (∏i≤n. 1 + norm (f i − 1)) by (rule sum_le_prod) auto
      also have ... < L + 1 by (rule elim)
      finally show ?case by simp
    qed
    thus Bseq (λn. ∑i≤n. norm (f i − 1)) by (rule BfunI)
  next
    show monoseq (λn. ∑i≤n. norm (f i − 1))
      by (rule mono_SucI1) auto
  qed
  thus summable (λi. norm (f i − 1)) by (simp add: summable_iff_convergent')
qed

lemma summable_imp_abs_convergent_prod:
  fixes f :: nat ⇒ 'a :: real_normed_div_algebra
  assumes summable (λi. norm (f i − 1))
  shows   abs_convergent_prod f
proof (intro abs_convergent_prodI Bseq_monoseq_convergent)
  show monoseq (λn. ∏i≤n. 1 + norm (f i − 1))
    by (intro mono_SucI1)
     (auto simp: atMost_Suc algebra_simps intro!: mult_nonneg_nonneg prod_nonneg)
next
  show Bseq (λn. ∏i≤n. 1 + norm (f i − 1))
  proof (rule Bseq_eventually_mono)
    show eventually (λn. norm (∏i≤n. 1 + norm (f i − 1)) ≤
          norm (exp (∑i≤n. norm (f i − 1)))) sequentially
      by (intro always_eventually allI) (auto simp: abs_prod exp_sum intro!:
prod_mono)
  next
    from assms have (λn. ∑i≤n. norm (f i − 1)) ⟶ (∑i. norm (f i − 1))
      using sums_def_le by blast
    hence (λn. exp (∑i≤n. norm (f i − 1))) ⟶ exp (∑i. norm (f i − 1))
      by (rule tendsto_exp)
```

```
    hence convergent (λn. exp (∑ i≤n. norm (f i − 1)))
      by (rule convergentI)
    thus Bseq (λn. exp (∑ i≤n. norm (f i − 1)))
      by (rule convergent_imp_Bseq)
  qed
qed

theorem abs_convergent_prod_conv_summable:
  fixes f :: nat ⇒ 'a :: real_normed_div_algebra
  shows abs_convergent_prod f ⟷ summable (λi. norm (f i − 1))
 by (blast intro: abs_convergent_prod_imp_summable summable_imp_abs_convergent_prod)

lemma abs_convergent_prod_imp_LIMSEQ:
  fixes f :: nat ⇒ 'a :: {comm_ring_1,real_normed_div_algebra}
  assumes abs_convergent_prod f
  shows   f ——⟶ 1
proof −
  from assms have summable (λn. norm (f n − 1))
    by (rule abs_convergent_prod_imp_summable)
  from summable_LIMSEQ_zero[OF this] have (λn. f n − 1) ——⟶ 0
    by (simp add: tendsto_norm_zero_iff)
  from tendsto_add[OF this tendsto_const[of 1]] show ?thesis by simp
qed

lemma abs_convergent_prod_imp_ev_nonzero:
  fixes f :: nat ⇒ 'a :: {comm_ring_1,real_normed_div_algebra}
  assumes abs_convergent_prod f
  shows   eventually (λn. f n ≠ 0) sequentially
proof −
  from assms have f ——⟶ 1
    by (rule abs_convergent_prod_imp_LIMSEQ)
  hence eventually (λn. dist (f n) 1 < 1) at_top
    by (auto simp: tendsto_iff)
  thus ?thesis by eventually_elim auto
qed
```

6.32.4 Ignoring initial segments

```
lemma convergent_prod_offset:
  assumes convergent_prod (λn. f (n + m))
  shows   convergent_prod f
proof −
  from assms obtain M L where (λn. ∏ k≤n. f (k + (M + m))) ——⟶ L L ≠
0
    by (auto simp: prod_defs add.assoc)
  thus convergent_prod f
    unfolding prod_defs by blast
qed
```

lemma *abs_convergent_prod_offset*:
assumes *abs_convergent_prod* (λ*n*. *f* (*n* + *m*))
shows *abs_convergent_prod f*
using *assms* **unfolding** *abs_convergent_prod_def* **by** (*rule convergent_prod_offset*)

lemma *raw_has_prod_ignore_initial_segment*:
fixes *f* :: *nat* ⇒ *'a* :: *real_normed_field*
assumes *raw_has_prod f M p N* ≥ *M*
obtains *q* **where** *raw_has_prod f N q*
proof −
have *p*: (λ*n*. ∏ *k*≤*n*. *f* (*k* + *M*)) ⟶ *p* **and** *p* ≠ *0*
using *assms* **by** (*auto simp*: *raw_has_prod_def*)
then have *nz*: ⋀*n*. *n* ≥ *M* ⟹ *f n* ≠ *0*
using *assms* **by** (*auto simp*: *raw_has_prod_eq_0*)
define *C* **where** *C* = (∏ *k*<*N*−*M*. *f* (*k* + *M*))
from *nz* **have** [*simp*]: *C* ≠ *0*
by (*auto simp*: *C_def*)

from *p* **have** (λ*i*. ∏ *k*≤*i* + (*N*−*M*). *f* (*k* + *M*)) ⟶ *p*
by (*rule LIMSEQ_ignore_initial_segment*)
also have (λ*i*. ∏ *k*≤*i* + (*N*−*M*). *f* (*k* + *M*)) = (λ*n*. *C* ∗ (∏ *k*≤*n*. *f* (*k* + *N*)))
proof (*rule ext*, *goal_cases*)
case (*1 n*)
have {..*n*+(*N*−*M*)} = {..<(*N*−*M*)} ∪ {(*N*−*M*)..*n*+(*N*−*M*)} **by** *auto*
also have (∏ *k*∈.... *f* (*k* + *M*)) = *C* ∗ (∏ *k*=(*N*−*M*)..*n*+(*N*−*M*). *f* (*k* + *M*))
unfolding *C_def* **by** (*rule prod.union_disjoint*) *auto*
also have (∏ *k*=(*N*−*M*)..*n*+(*N*−*M*). *f* (*k* + *M*)) = (∏ *k*≤*n*. *f* (*k* + (*N*−*M*) + *M*))
by (*intro ext prod.reindex_bij_witness*[*of* _ λ*k*. *k* + (*N*−*M*) λ*k*. *k* − (*N*−*M*)]) *auto*
finally show *?case*
using ‹*N* ≥ *M*› **by** (*simp add*: *add_ac*)
qed
finally have (λ*n*. *C* ∗ (∏ *k*≤*n*. *f* (*k* + *N*)) / *C*) ⟶ *p* / *C*
by (*intro tendsto_divide tendsto_const*) *auto*
hence (λ*n*. ∏ *k*≤*n*. *f* (*k* + *N*)) ⟶ *p* / *C* **by** *simp*
moreover from ‹*p* ≠ *0*› **have** *p* / *C* ≠ *0* **by** *simp*
ultimately show *?thesis*
using *raw_has_prod_def that* **by** *blast*
qed

corollary *convergent_prod_ignore_initial_segment*:
fixes *f* :: *nat* ⇒ *'a* :: *real_normed_field*
assumes *convergent_prod f*
shows *convergent_prod* (λ*n*. *f* (*n* + *m*))
using *assms*
unfolding *convergent_prod_def*

apply *clarify*
apply (*erule_tac N=M+m* **in** *raw_has_prod_ignore_initial_segment*)
apply (*auto simp add*: *raw_has_prod_def add_ac*)
done

corollary *convergent_prod_ignore_nonzero_segment*:
fixes *f* :: *nat* ⇒ *'a* :: *real_normed_field*
assumes *f*: *convergent_prod f* **and** *nz*: ⋀*i*. *i* ≥ *M* ⟹ *f i* ≠ *0*
shows ∃ *p*. *raw_has_prod f M p*
using *convergent_prod_ignore_initial_segment* [*OF f*]
by (*metis convergent_LIMSEQ_iff convergent_prod_iff_convergent le_add_same_cancel2 nz prod_defs*(*1*) *zero_order*(*1*))

corollary *abs_convergent_prod_ignore_initial_segment*:
assumes *abs_convergent_prod f*
shows *abs_convergent_prod* (λ*n*. *f* (*n* + *m*))
using *assms* **unfolding** *abs_convergent_prod_def*
by (*rule convergent_prod_ignore_initial_segment*)

6.32.5 More elementary properties

theorem *abs_convergent_prod_imp_convergent_prod*:
fixes *f* :: *nat* ⇒ *'a* :: {*real_normed_div_algebra*,*complete_space*,*comm_ring_1*}
assumes *abs_convergent_prod f*
shows *convergent_prod f*
proof −
from *assms* **have** *eventually* (λ*n*. *f n* ≠ *0*) *sequentially*
by (*rule abs_convergent_prod_imp_ev_nonzero*)
then obtain *N* **where** *N*: *f n* ≠ *0* **if** *n* ≥ *N* **for** *n*
by (*auto simp*: *eventually_at_top_linorder*)
let *?P* = λ*n*. ∏ *i*≤*n*. *f* (*i* + *N*) **and** *?Q* = λ*n*. ∏ *i*≤*n*. *1* + *norm* (*f* (*i* + *N*) − *1*)

have *Cauchy ?P*
proof (*rule CauchyI'*, *goal_cases*)
case (*1* ε)
from *assms* **have** *abs_convergent_prod* (λ*n*. *f* (*n* + *N*))
by (*rule abs_convergent_prod_ignore_initial_segment*)
hence *Cauchy ?Q*
unfolding *abs_convergent_prod_def*
by (*intro convergent_Cauchy convergent_prod_imp_convergent*)
from *CauchyD*[*OF this 1*] **obtain** *M* **where** *M*: *norm* (*?Q m* − *?Q n*) < ε **if** *m* ≥ *M n* ≥ *M* **for** *m n*
by *blast*
show *?case*
proof (*rule exI*[*of* _ *M*], *safe*, *goal_cases*)
case (*1 m n*)
have *dist* (*?P m*) (*?P n*) = *norm* (*?P n* − *?P m*)
by (*simp add*: *dist_norm norm_minus_commute*)

```
    also from 1 have {..n} = {..m} ∪ {m<..n} by auto
    hence norm (?P n − ?P m) = norm (?P m ∗ (∏k∈{m<..n}. f (k + N)) −
?P m)
      by (subst prod.union_disjoint [symmetric]) (auto simp: algebra_simps)
    also have ... = norm (?P m ∗ ((∏k∈{m<..n}. f (k + N)) − 1))
      by (simp add: algebra_simps)
    also have ... = (∏k≤m. norm (f (k + N))) ∗ norm ((∏k∈{m<..n}. f (k
+ N)) − 1)
      by (simp add: norm_mult prod_norm)
    also have ... ≤ ?Q m ∗ ((∏k∈{m<..n}. 1 + norm (f (k + N) − 1)) − 1)
     using norm_prod_minus1_le_prod_minus1[of λk. f (k + N) − 1 {m<..n}]
           norm_triangle_ineq[of 1 f k − 1 for k]
    by (intro mult_mono prod_mono ballI conjI norm_prod_minus1_le_prod_minus1
prod_nonneg) auto
    also have ... = ?Q m ∗ (∏k∈{m<..n}. 1 + norm (f (k + N) − 1)) − ?Q
m
      by (simp add: algebra_simps)
    also have ?Q m ∗ (∏k∈{m<..n}. 1 + norm (f (k + N) − 1)) =
                (∏k∈{..m}∪{m<..n}. 1 + norm (f (k + N) − 1))
      by (rule prod.union_disjoint [symmetric]) auto
    also from 1 have {..m}∪{m<..n} = {..n} by auto
    also have ?Q n − ?Q m ≤ norm (?Q n − ?Q m) by simp
    also from 1 have ... < ε by (intro M) auto
    finally show ?case .
  qed
 qed
 hence conv: convergent ?P by (rule Cauchy_convergent)
 then obtain L where L: ?P ⟶ L
  by (auto simp: convergent_def)

 have L ≠ 0
 proof
  assume [simp]: L = 0
  from tendsto_norm[OF L] have limit: (λn. ∏k≤n. norm (f (k + N))) ⟶
0
    by (simp add: prod_norm)

  from assms have (λn. f (n + N)) ⟶ 1
  by (intro abs_convergent_prod_imp_LIMSEQ abs_convergent_prod_ignore_initial_segment)
  hence eventually (λn. norm (f (n + N) − 1) < 1) sequentially
    by (auto simp: tendsto_iff dist_norm)
  then obtain M0 where M0: norm (f (n + N) − 1) < 1 if n ≥ M0 for n
    by (auto simp: eventually_at_top_linorder)

  {
    fix M assume M: M ≥ M0
    with M0 have M: norm (f (n + N) − 1) < 1 if n ≥ M for n using that
by simp
```

have ($\lambda n.\ \prod k{\le}n.\ 1 - norm\ (f\ (k{+}M{+}N) - 1)) \longrightarrow 0$
proof (*rule tendsto_sandwich*)
show *eventually* ($\lambda n.\ (\prod k{\le}n.\ 1 - norm\ (f\ (k{+}M{+}N) - 1)) \ge 0$) *sequentially*
using M **by** (*intro always_eventually prod_nonneg allI ballI*) (*auto intro*: *less_imp_le*)
have *norm* ($1::'a$) − *norm* ($f\ (i + M + N) - 1$) ≤ *norm* ($f\ (i + M + N)$) **for** i
using *norm_triangle_ineq3*[*of f* ($i + M + N$) *1*] **by** *simp*
thus *eventually* ($\lambda n.\ (\prod k{\le}n.\ 1 - norm\ (f\ (k{+}M{+}N) - 1)) \le (\prod k{\le}n.\ norm\ (f\ (k{+}M{+}N))))$ *at_top*
using M **by** (*intro always_eventually allI prod_mono ballI conjI*) (*auto intro*: *less_imp_le*)

define C **where** $C = (\prod k{<}M.\ norm\ (f\ (k + N)))$
from N **have** [*simp*]: $C \neq 0$ **by** (*auto simp*: *C_def*)
from L **have** ($\lambda n.\ norm\ (\prod k{\le}n{+}M.\ f\ (k + N))) \longrightarrow 0$
by (*intro LIMSEQ_ignore_initial_segment*) (*simp add*: *tendsto_norm_zero_iff*)
also have ($\lambda n.\ norm\ (\prod k{\le}n{+}M.\ f\ (k + N))) = (\lambda n.\ C * (\prod k{\le}n.\ norm\ (f\ (k + M + N))))$
proof (*rule ext*, *goal_cases*)
case (*1 n*)
have $\{..n{+}M\} = \{..{<}M\} \cup \{M..n{+}M\}$ **by** *auto*
also have *norm* ($\prod k{\in}\ldots.\ f\ (k + N)) = C * norm\ (\prod k{=}M..n{+}M.\ f\ (k + N))$
unfolding *C_def* **by** (*subst prod.union_disjoint*) (*auto simp*: *norm_mult prod_norm*)
also have ($\prod k{=}M..n{+}M.\ f\ (k + N)) = (\prod k{\le}n.\ f\ (k + N + M))$
by (*intro prod.reindex_bij_witness*[*of* _ $\lambda i.\ i + M$ $\lambda i.\ i - M$]) *auto*
finally show *?case* **by** (*simp add*: *add_ac prod_norm*)
qed
finally have ($\lambda n.\ C * (\prod k{\le}n.\ norm\ (f\ (k + M + N))) / C) \longrightarrow 0 / C$
by (*intro tendsto_divide tendsto_const*) *auto*
thus ($\lambda n.\ \prod k{\le}n.\ norm\ (f\ (k + M + N))) \longrightarrow 0$ **by** *simp*
qed *simp_all*

have $1 - (\sum i.\ norm\ (f\ (i + M + N) - 1)) \le 0$
proof (*rule tendsto_le*)
show *eventually* ($\lambda n.\ 1 - (\sum k{\le}n.\ norm\ (f\ (k{+}M{+}N) - 1)) \le (\prod k{\le}n.\ 1 - norm\ (f\ (k{+}M{+}N) - 1)))$ *at_top*
using M **by** (*intro always_eventually allI Weierstrass_prod_ineq*) (*auto intro*: *less_imp_le*)
show ($\lambda n.\ \prod k{\le}n.\ 1 - norm\ (f\ (k{+}M{+}N) - 1)) \longrightarrow 0$ **by** *fact*
show ($\lambda n.\ 1 - (\sum k{\le}n.\ norm\ (f\ (k + M + N) - 1))) \longrightarrow 1 - (\sum i.\ norm\ (f\ (i + M + N) - 1))$
by (*intro tendsto_intros summable_LIMSEQ' summable_ignore_initial_segment abs_convergent_prod_imp_summable assms*)

```
      qed simp_all
      hence (∑ i. norm (f (i + M + N) − 1)) ≥ 1 by simp
      also have ... + (∑ i<M. norm (f (i + N) − 1)) = (∑ i. norm (f (i + N)
− 1))
      by (intro suminf_split_initial_segment [symmetric] summable_ignore_initial_segment
            abs_convergent_prod_imp_summable assms)
      finally have 1 + (∑ i<M. norm (f (i + N) − 1)) ≤ (∑ i. norm (f (i +
N) − 1)) by simp
    } note ∗ = this

    have 1 + (∑ i. norm (f (i + N) − 1)) ≤ (∑ i. norm (f (i + N) − 1))
    proof (rule tendsto_le)
      show (λM. 1 + (∑ i<M. norm (f (i + N) − 1))) ⟶ 1 + (∑ i. norm
(f (i + N) − 1))
      by (intro tendsto_intros summable_LIMSEQ summable_ignore_initial_segment

            abs_convergent_prod_imp_summable assms)
      show eventually (λM. 1 + (∑ i<M. norm (f (i + N) − 1)) ≤ (∑ i. norm
(f (i + N) − 1))) at_top
        using eventually_ge_at_top[of M0] by eventually_elim (use ∗ in auto)
    qed simp_all
    thus False by simp
  qed
  with L show ?thesis by (auto simp: prod_defs)
qed

lemma raw_has_prod_cases:
  fixes f :: nat ⇒ 'a :: {idom,topological_semigroup_mult,t2_space}
  assumes raw_has_prod f M p
  obtains i where i<M f i = 0 | p where raw_has_prod f 0 p
proof −
  have (λn. ∏ i≤n. f (i + M)) ⟶ p p ≠ 0
    using assms unfolding raw_has_prod_def by blast+
  then have (λn. prod f {..<M} ∗ (∏ i≤n. f (i + M))) ⟶ prod f {..<M} ∗
p
    by (metis tendsto_mult_left)
  moreover have prod f {..<M} ∗ (∏ i≤n. f (i + M)) = prod f {..n+M} for n
  proof −
    have {..n+M} = {..<M} ∪ {M..n+M}
      by auto
    then have prod f {..n+M} = prod f {..<M} ∗ prod f {M..n+M}
      by simp (subst prod.union_disjoint; force)
    also have ... = prod f {..<M} ∗ (∏ i≤n. f (i + M))
    by (metis (mono_tags, lifting) add.left_neutral atMost_atLeast0 prod.shift_bounds_cl_nat_ivl)
    finally show ?thesis by metis
  qed
  ultimately have (λn. prod f {..n}) ⟶ prod f {..<M} ∗ p
    by (auto intro: LIMSEQ_offset [where k=M])
  then have raw_has_prod f 0 (prod f {..<M} ∗ p) if ∀ i<M. f i ≠ 0
```

```
    using ‹p ≠ 0› assms that by (auto simp: raw_has_prod_def)
  then show thesis
    using that by blast
qed

corollary convergent_prod_offset_0:
  fixes f :: nat ⇒ 'a :: {idom,topological_semigroup_mult,t2_space}
  assumes convergent_prod f ⋀i. f i ≠ 0
  shows ∃ p. raw_has_prod f 0 p
  using assms convergent_prod_def raw_has_prod_cases by blast

lemma prodinf_eq_lim:
  fixes f :: nat ⇒ 'a :: {idom,topological_semigroup_mult,t2_space}
  assumes convergent_prod f ⋀i. f i ≠ 0
  shows prodinf f = lim (λn. ∏ i≤n. f i)
  using assms convergent_prod_offset_0 [OF assms]
 by (simp add: prod_defs lim_def) (metis (no_types) assms(1) convergent_prod_to_zero_iff)

lemma prodinf_eq_lim':
  fixes f :: nat ⇒ 'a :: {idom,topological_semigroup_mult,t2_space}
  assumes convergent_prod f ⋀i. f i ≠ 0
  shows prodinf f = lim (λn. ∏ i<n. f i)
 by (metis assms prodinf_eq_lim LIMSEQ_lessThan_iff_atMost convergent_prod_iff_nz_lim
limI)

lemma prodinf_eq_prod_lim:
  fixes a:: 'a :: {topological_semigroup_mult,t2_space,idom}
  assumes (λn. ∏ k≤n. f k) ⟶ a a ≠ 0
  shows(∏ k. f k) = a
 by (metis LIMSEQ_prod_0 LIMSEQ_unique assms convergent_prod_iff_nz_lim
limI prodinf_eq_lim)

lemma prodinf_eq_prod_lim':
  fixes a:: 'a :: {topological_semigroup_mult,t2_space,idom}
  assumes (λn. ∏ k<n. f k) ⟶ a a ≠ 0
  shows(∏ k. f k) = a
  using LIMSEQ_lessThan_iff_atMost assms prodinf_eq_prod_lim by blast

lemma has_prod_one[simp, intro]: (λn. 1) has_prod 1
  unfolding prod_defs by auto

lemma convergent_prod_one[simp, intro]: convergent_prod (λn. 1)
  unfolding prod_defs by auto

lemma prodinf_cong: (⋀n. f n = g n) ⟹ prodinf f = prodinf g
  by presburger

lemma convergent_prod_cong:
  fixes f g :: nat ⇒ 'a::{field,topological_semigroup_mult,t2_space}
```

assumes *ev*: *eventually* (λ*x*. *f x* = *g x*) *sequentially* **and** *f*: ⋀*i*. *f i* ≠ *0* **and** *g*: ⋀*i*. *g i* ≠ *0*
shows *convergent_prod f* = *convergent_prod g*
proof −
from *assms* **obtain** *N* **where** *N*: ∀ *n*≥*N*. *f n* = *g n*
by (*auto simp*: *eventually_at_top_linorder*)
define *C* **where** *C* = (∏ *k*<*N*. *f k* / *g k*)
with *g* **have** *C* ≠ *0*
by (*simp add*: *f*)
have ∗: *eventually* (λ*n*. *prod f* {..*n*} = *C* ∗ *prod g* {..*n*}) *sequentially*
using *eventually_ge_at_top*[*of N*]
proof *eventually_elim*
case (*elim n*)
then have {..*n*} = {..<*N*} ∪ {*N*..*n*}
by *auto*
also have *prod f* ... = *prod f* {..<*N*} ∗ *prod f* {*N*..*n*}
by (*intro prod.union_disjoint*) *auto*
also from *N* **have** *prod f* {*N*..*n*} = *prod g* {*N*..*n*}
by (*intro prod.cong*) *simp_all*
also have *prod f* {..<*N*} ∗ *prod g* {*N*..*n*} = *C* ∗ (*prod g* {..<*N*} ∗ *prod g* {*N*..*n*})
unfolding *C_def* **by** (*simp add*: *g prod_dividef*)
also have *prod g* {..<*N*} ∗ *prod g* {*N*..*n*} = *prod g* ({..<*N*} ∪ {*N*..*n*})
by (*intro prod.union_disjoint* [*symmetric*]) *auto*
also from *elim* **have** {..<*N*} ∪ {*N*..*n*} = {..*n*}
by *auto*
finally show *prod f* {..*n*} = *C* ∗ *prod g* {..*n*} **.**
qed
then have *cong*: *convergent* (λ*n*. *prod f* {..*n*}) = *convergent* (λ*n*. *C* ∗ *prod g* {..*n*})
by (*rule convergent_cong*)
show *?thesis*
proof
assume *cf*: *convergent_prod f*
with *f* **have** ¬ (λ*n*. *prod f* {..*n*}) ⟶ *0*
by *simp*
then have ¬ (λ*n*. *prod g* {..*n*}) ⟶ *0*
using ∗ ‹*C* ≠ *0*› *filterlim_cong* **by** *fastforce*
then show *convergent_prod g*
by (*metis convergent_mult_const_iff* ‹*C* ≠ *0*› *cong cf convergent_LIMSEQ_iff convergent_prod_iff_convergent convergent_prod_imp_convergent g*)
next
assume *cg*: *convergent_prod g*
have ∃ *a*. *C* ∗ *a* ≠ *0* ∧ (λ*n*. *prod g* {..*n*}) ⟶ *a*
by (*metis* (*no_types*) ‹*C* ≠ *0*› *cg convergent_prod_iff_nz_lim divide_eq_0_iff g nonzero_mult_div_cancel_right*)
then show *convergent_prod f*
using ∗ *tendsto_mult_left filterlim_cong*
by (*fastforce simp add*: *convergent_prod_iff_nz_lim f*)

```
  qed
qed

lemma has_prod_finite:
  fixes f :: nat ⇒ 'a::{semidom,t2_space}
  assumes [simp]: finite N
    and f: ⋀n. n ∉ N ⟹ f n = 1
  shows f has_prod (∏ n∈N. f n)
proof −
  have eq: prod f {..n + Suc (Max N)} = prod f N for n
  proof (rule prod.mono_neutral_right)
    show N ⊆ {..n + Suc (Max N)}
      by (auto simp: le_Suc_eq trans_le_add2)
    show ∀ i∈{..n + Suc (Max N)} − N. f i = 1
      using f by blast
  qed auto
  show ?thesis
  proof (cases ∀ n∈N. f n ≠ 0)
    case True
    then have prod f N ≠ 0
      by simp
    moreover have (λn. prod f {..n}) ⟶ prod f N
     by (rule LIMSEQ_offset[of _ Suc (Max N)]) (simp add: eq atLeast0LessThan
del: add_Suc_right)
    ultimately show ?thesis
      by (simp add: raw_has_prod_def has_prod_def)
  next
    case False
    then obtain k where k ∈ N f k = 0
      by auto
    let ?Z = {n ∈ N. f n = 0}
    have maxge: Max ?Z ≥ n if f n = 0 for n
      using Max_ge [of ?Z] ‹finite N› ‹f n = 0›
    by (metis (mono_tags) Collect_mem_eq f finite_Collect_conjI mem_Collect_eq
zero_neq_one)
    let ?q = prod f {Suc (Max ?Z)..Max N}
    have [simp]: ?q ≠ 0
      using maxge Suc_n_not_le_n le_trans by force
    have eq: (∏ i≤n + Max N. f (Suc (i + Max ?Z))) = ?q for n
    proof −
      have (∏ i≤n + Max N. f (Suc (i + Max ?Z))) = prod f {Suc (Max ?Z)..n
+ Max N + Suc (Max ?Z)}
       proof (rule prod.reindex_cong [where l = λi. i + Suc (Max ?Z), THEN
sym])
        show {Suc (Max ?Z)..n + Max N + Suc (Max ?Z)} = (λi. i + Suc (Max
?Z)) ‘ {..n + Max N}
          using le_Suc_ex by fastforce
      qed (auto simp: inj_on_def)
      also have ... = ?q
```

```
      by (rule prod.mono_neutral_right)
        (use Max.coboundedI [OF ‹finite N›] f in ‹force+›)
    finally show ?thesis .
  qed
  have q: raw_has_prod f (Suc (Max ?Z)) ?q
  proof (simp add: raw_has_prod_def)
    show (λn. ∏i≤n. f (Suc (i + Max ?Z))) ⟶ ?q
      by (rule LIMSEQ_offset[of _ (Max N)]) (simp add: eq)
  qed
  show ?thesis
    unfolding has_prod_def
  proof (intro disjI2 exI conjI)
    show prod f N = 0
      using ‹f k = 0› ‹k ∈ N› ‹finite N› prod_zero by blast
    show f (Max ?Z) = 0
      using Max_in [of ?Z] ‹finite N› ‹f k = 0› ‹k ∈ N› by auto
  qed (use q in auto)
 qed
qed

corollary has_prod_0:
 fixes f :: nat ⇒ 'a::{semidom,t2_space}
 assumes ⋀n. f n = 1
 shows f has_prod 1
 by (simp add: assms has_prod_cong)

lemma prodinf_zero[simp]: prodinf (λn. 1::'a::real_normed_field) = 1
 using has_prod_unique by force

lemma convergent_prod_finite:
 fixes f :: nat ⇒ 'a::{idom,t2_space}
 assumes finite N ⋀n. n ∉ N ⟹ f n = 1
 shows convergent_prod f
proof −
 have ∃ n p. raw_has_prod f n p
   using assms has_prod_def has_prod_finite by blast
 then show ?thesis
   by (simp add: convergent_prod_def)
qed

lemma has_prod_If_finite_set:
 fixes f :: nat ⇒ 'a::{idom,t2_space}
 shows finite A ⟹ (λr. if r ∈ A then f r else 1) has_prod (∏ r∈A. f r)
 using has_prod_finite[of A (λr. if r ∈ A then f r else 1)]
 by simp

lemma has_prod_If_finite:
 fixes f :: nat ⇒ 'a::{idom,t2_space}
 shows finite {r. P r} ⟹ (λr. if P r then f r else 1) has_prod (∏ r | P r. f r)
```

using *has_prod_If_finite_set*[*of* {*r*. *P r*}] **by** *simp*

lemma *convergent_prod_If_finite_set*[*simp*, *intro*]:
fixes *f* :: *nat* ⇒ *'a*::{*idom*,*t2_space*}
shows *finite A* ⟹ *convergent_prod* (λ*r*. *if r* ∈ *A then f r else 1*)
by (*simp add*: *convergent_prod_finite*)

lemma *convergent_prod_If_finite*[*simp*, *intro*]:
fixes *f* :: *nat* ⇒ *'a*::{*idom*,*t2_space*}
shows *finite* {*r*. *P r*} ⟹ *convergent_prod* (λ*r*. *if P r then f r else 1*)
using *convergent_prod_def has_prod_If_finite has_prod_def* **by** *fastforce*

lemma *has_prod_single*:
fixes *f* :: *nat* ⇒ *'a*::{*idom*,*t2_space*}
shows (λ*r*. *if r* = *i then f r else 1*) *has_prod f i*
using *has_prod_If_finite*[*of* λ*r*. *r* = *i*] **by** *simp*

The ge1 assumption can probably be weakened, at the expense of extra work

lemma *uniform_limit_prodinf*:
fixes *f*:: *nat* ⇒ *real* ⇒ *real*
assumes *uniformly_convergent_on X* (λ*n x*. ∏*k*<*n*. *f k x*)
and *ge1*: ⋀*x k* . *x* ∈ *X* ⟹ *f k x* ≥ *1*
shows *uniform_limit X* (λ*n x*. ∏*k*<*n*. *f k x*) (λ*x*. ∏*k*. *f k x*) *sequentially*
proof −
have *ul*: *uniform_limit X* (λ*n x*. ∏*k*<*n*. *f k x*) (λ*x*. *lim* (λ*n*. ∏*k*<*n*. *f k x*)) *sequentially*
using *assms uniformly_convergent_uniform_limit_iff* **by** *blast*
moreover have (∏*k*. *f k x*) = *lim* (λ*n*. ∏*k*<*n*. *f k x*) **if** *x* ∈ *X* **for** *x*
proof (*intro prodinf_eq_lim'*)
have *tends*: (λ*n*. ∏*k*<*n*. *f k x*) ⟶ *lim* (λ*n*. ∏*k*<*n*. *f k x*)
using *tendsto_uniform_limitI* [*OF ul*] *that* **by** *metis*
moreover have (∏*k*<*n*. *f k x*) ≥ *1* **for** *n*
using *ge1* **by** (*simp add*: *prod_ge_1 that*)
ultimately have *lim* (λ*n*. ∏*k*<*n*. *f k x*) ≥ *1*
by (*meson LIMSEQ_le_const*)
then have *raw_has_prod* (λ*k*. *f k x*) *0* (*lim* (λ*n*. ∏*k*<*n*. *f k x*))
using *LIMSEQ_lessThan_iff_atMost tends* **by** (*auto simp*: *raw_has_prod_def*)
then show *convergent_prod* (λ*k*. *f k x*)
unfolding *convergent_prod_def* **by** *blast*
show ⋀*k*. *f k x* ≠ *0*
by (*smt* (*verit*) *ge1 that*)
qed
ultimately show *?thesis*
by (*metis* (*mono_tags*, *lifting*) *uniform_limit_cong'*)
qed

context
fixes *f* :: *nat* ⇒ *'a* :: *real_normed_field*
begin

lemma *convergent_prod_imp_has_prod*:
assumes *convergent_prod f*
shows $\exists p.$ *f has_prod p*
proof −
obtain *M p* **where** *p*: *raw_has_prod f M p*
using *assms convergent_prod_def* **by** *blast*
then have $p \neq 0$
using *raw_has_prod_nonzero* **by** *blast*
with *p* **have** *fnz*: $f\ i \neq 0$ **if** $i \geq M$ **for** *i*
using *raw_has_prod_eq_0 that* **by** *blast*
define *C* **where** $C = (\prod n{<}M.\ f\ n)$
show *?thesis*
proof (*cases* $\forall\, n{\leq}M.\ f\ n \neq 0$)
case *True*
then have $C \neq 0$
by (*simp add*: *C_def*)
then show *?thesis*
by (*meson True assms convergent_prod_offset_0 fnz has_prod_def nat_le_linear*)
next
case *False*
let *?N* = *GREATEST n. f n* = *0*
have *0*: *f ?N* = *0*
using *fnz False*
by (*metis* (*mono_tags*, *lifting*) *GreatestI_ex_nat nat_le_linear*)
have $f\ i \neq 0$ **if** $i > ?N$ **for** *i*
by (*metis* (*mono_tags*, *lifting*) *Greatest_le_nat fnz leD linear that*)
then have $\exists p.$ *raw_has_prod f* (*Suc ?N*) *p*
using *assms* **by** (*auto simp*: *intro*!: *convergent_prod_ignore_nonzero_segment*)
then show *?thesis*
unfolding *has_prod_def* **using** *0* **by** *blast*
qed
qed

lemma *convergent_prod_has_prod* [*intro*]:
shows *convergent_prod f* $\Longrightarrow$ *f has_prod* (*prodinf f*)
unfolding *prodinf_def*
by (*metis convergent_prod_imp_has_prod has_prod_unique theI'*)

lemma *convergent_prod_LIMSEQ*:
shows *convergent_prod f* $\Longrightarrow$ $(\lambda n.\ \prod i{\leq}n.\ f\ i) \longrightarrow$ *prodinf f*
by (*metis convergent_LIMSEQ_iff convergent_prod_has_prod convergent_prod_imp_convergent*

convergent_prod_to_zero_iff raw_has_prod_eq_0 has_prod_def prodinf_eq_lim
zero_le)

theorem *has_prod_iff*: *f has_prod x* $\longleftrightarrow$ *convergent_prod f* $\wedge$ *prodinf f* = *x*
proof
assume *f has_prod x*

then show *convergent_prod f* ∧ *prodinf f* = *x*
apply *safe*
using *convergent_prod_def has_prod_def* **apply** *blast*
using *has_prod_unique* **by** *blast*
qed *auto*

lemma *convergent_prod_has_prod_iff*: *convergent_prod f* ⟷ *f has_prod prodinf f*
by (*auto simp*: *has_prod_iff convergent_prod_has_prod*)

lemma *prodinf_finite*:
assumes *N*: *finite N*
and *f*: ⋀*n*. *n* ∉ *N* ⟹ *f n* = *1*
shows *prodinf f* = (∏ *n*∈*N*. *f n*)
using *has_prod_finite*[*OF assms*, *THEN has_prod_unique*] **by** *simp*

end

6.32.6 Infinite products on ordered topological monoids

context
fixes *f* :: *nat* ⇒ *'a*::{*linordered_semidom*,*linorder_topology*}
begin

lemma *has_prod_nonzero*:
assumes *f has_prod a a* ≠ *0*
shows *f k* ≠ *0*
using *assms* **by** (*auto simp*: *has_prod_def raw_has_prod_def LIMSEQ_prod_0 LIMSEQ_unique*)

lemma *has_prod_le*:
assumes *f*: *f has_prod a* **and** *g*: *g has_prod b* **and** *le*: ⋀*n*. *0* ≤ *f n* ∧ *f n* ≤ *g n*
shows *a* ≤ *b*
proof (*cases a=0* ∨ *b=0*)
case *True*
then show *?thesis*
proof
assume [*simp*]: *a=0*
have *b* ≥ *0*
proof (*rule LIMSEQ_prod_nonneg*)
show (λ*n*. *prod g* {..*n*}) ⟶ *b*
using *g* **by** (*auto simp*: *has_prod_def raw_has_prod_def LIMSEQ_prod_0*)
qed (*use le order_trans* **in** *auto*)
then show *?thesis*
by *auto*
next
assume [*simp*]: *b=0*
then obtain *i* **where** *g i* = *0*
using *g* **by** (*auto simp*: *prod_defs*)

```
    then have f i = 0
      using antisym le by force
    then have a=0
      using f by (auto simp: prod_defs LIMSEQ_prod_0 LIMSEQ_unique)
    then show ?thesis
      by auto
  qed
next
  case False
  then show ?thesis
    using assms
    unfolding has_prod_def raw_has_prod_def
    by (force simp: LIMSEQ_prod_0 intro!: LIMSEQ_le prod_mono)
qed

lemma prodinf_le:
  assumes f: f has_prod a and g: g has_prod b and le: ⋀n. 0 ≤ f n ∧ f n ≤ g n
  shows prodinf f ≤ prodinf g
  using has_prod_le [OF assms] has_prod_unique f g  by blast

end


lemma prod_le_prodinf:
  fixes f :: nat ⇒ 'a::{linordered_idom,linorder_topology}
  assumes f has_prod a ⋀i. 0 ≤ f i ⋀i. i≥n ⟹ 1 ≤ f i
  shows prod f {..<n} ≤ prodinf f
  by(rule has_prod_le[OF has_prod_If_finite_set]) (use assms has_prod_unique
in auto)

lemma prodinf_nonneg:
  fixes f :: nat ⇒ 'a::{linordered_idom,linorder_topology}
  assumes f has_prod a ⋀i. 1 ≤ f i
  shows 1 ≤ prodinf f
  using prod_le_prodinf[of f a 0] assms
  by (metis order_trans prod_ge_1 zero_le_one)

lemma prodinf_le_const:
  fixes f :: nat ⇒ real
  assumes convergent_prod f ⋀n. n ≥ N ⟹ prod f {..<n} ≤ x
  shows prodinf f ≤ x
 by (metis lessThan_Suc_atMost assms convergent_prod_LIMSEQ LIMSEQ_le_const2
atMost_iff lessThan_iff less_le)

lemma prodinf_eq_one_iff [simp]:
  fixes f :: nat ⇒ real
  assumes f: convergent_prod f and ge1: ⋀n. 1 ≤ f n
  shows prodinf f = 1 ⟷ (∀ n. f n = 1)
proof
```

```
  assume prodinf f = 1
  then have (λn. ∏ i<n. f i) ⟶ 1
   using convergent_prod_LIMSEQ[of f] assms by (simp add: LIMSEQ_lessThan_iff_atMost)
  then have ⋀i. (∏ n∈{i}. f n) ≤ 1
  proof (rule LIMSEQ_le_const)
    have 1 ≤ prod f n for n
      by (simp add: ge1 prod_ge_1)
    have prod f {..<n} = 1 for n
    by (metis ‹⋀n. 1 ≤ prod f n› ‹prodinf f = 1› antisym f convergent_prod_has_prod
ge1 order_trans prod_le_prodinf zero_le_one)
    then have (∏ n∈{i}. f n) ≤ prod f {..<n} if n ≥ Suc i for i n
    by (metis mult.left_neutral order_refl prod.cong prod.neutral_const prod.lessThan_Suc)
    then show ∃ N. ∀ n≥N. (∏ n∈{i}. f n) ≤ prod f {..<n} for i
      by blast
  qed
  with ge1 show ∀ n. f n = 1
    by (auto intro!: antisym)
qed (metis prodinf_zero fun_eq_iff)

lemma prodinf_pos_iff:
  fixes f :: nat ⇒ real
  assumes convergent_prod f ⋀n. 1 ≤ f n
  shows 1 < prodinf f ⟷ (∃ i. 1 < f i)
  using prod_le_prodinf[of f 1] prodinf_eq_one_iff
  by (metis convergent_prod_has_prod assms less_le prodinf_nonneg)

lemma less_1_prodinf2:
  fixes f :: nat ⇒ real
  assumes convergent_prod f ⋀n. 1 ≤ f n 1 < f i
  shows 1 < prodinf f
proof −
  have 1 < (∏ n<Suc i. f n)
    using assms  by (intro less_1_prod2[where i=i]) auto
  also have ... ≤ prodinf f
    by (intro prod_le_prodinf) (use assms order_trans zero_le_one in ‹blast+›)
  finally show ?thesis .
qed

lemma less_1_prodinf:
  fixes f :: nat ⇒ real
  shows ⟦convergent_prod f; ⋀n. 1 < f n⟧ ⟹ 1 < prodinf f
  by (intro less_1_prodinf2[where i=1]) (auto intro: less_imp_le)

lemma prodinf_nonzero:
  fixes f :: nat ⇒ 'a :: {idom,topological_semigroup_mult,t2_space}
  assumes convergent_prod f ⋀i. f i ≠ 0
  shows prodinf f ≠ 0
 by (metis assms convergent_prod_offset_0 has_prod_unique raw_has_prod_def
has_prod_def)
```

lemma *less_0_prodinf*:
fixes $f :: nat \Rightarrow real$
assumes *f*: *convergent_prod f* **and** *0*: $\bigwedge i.\ f\ i > 0$
shows $0 < prodinf\ f$
proof −
have $prodinf\ f \neq 0$
by (*metis assms less_irrefl prodinf_nonzero*)
moreover have $0 < (\prod n{<}i.\ f\ n)$ **for** *i*
by (*simp add*: *0 prod_pos*)
then have $prodinf\ f \geq 0$
using *convergent_prod_LIMSEQ* [*OF f*] *LIMSEQ_prod_nonneg 0 less_le* **by** *blast*
ultimately show *?thesis*
by *auto*
qed

lemma *prod_less_prodinf2*:
fixes $f :: nat \Rightarrow real$
assumes *f*: *convergent_prod f* **and** *1*: $\bigwedge m.\ m{\geq}n \Longrightarrow 1 \leq f\ m$ **and** *0*: $\bigwedge m.\ 0 < f\ m$ **and** *i*: $n \leq i\ 1 < f\ i$
shows $prod\ f\ \{..{<}n\} < prodinf\ f$
proof −
have $prod\ f\ \{..{<}n\} \leq prod\ f\ \{..{<}i\}$
by (*rule prod_mono2*) (*use assms less_le* **in** *auto*)
then have $prod\ f\ \{..{<}n\} < f\ i * prod\ f\ \{..{<}i\}$
using *mult_less_le_imp_less*[*of 1 f i prod f* $\{..{<}n\}$ *prod f* $\{..{<}i\}$] *assms*
by (*simp add*: *prod_pos*)
moreover have $prod\ f\ \{..{<}Suc\ i\} \leq prodinf\ f$
using *prod_le_prodinf*[*of f _ Suc i*]
by (*meson 0 1 Suc_leD convergent_prod_has_prod f* ‹$n \leq i$› *le_trans less_eq_real_def*)
ultimately show *?thesis*
by (*metis le_less_trans mult.commute not_le prod.lessThan_Suc*)
qed

lemma *prod_less_prodinf*:
fixes $f :: nat \Rightarrow real$
assumes *f*: *convergent_prod f* **and** *1*: $\bigwedge m.\ m{\geq}n \Longrightarrow 1 < f\ m$ **and** *0*: $\bigwedge m.\ 0 < f\ m$
shows $prod\ f\ \{..{<}n\} < prodinf\ f$
by (*meson 0 1 f le_less prod_less_prodinf2*)

lemma *raw_has_prodI_bounded*:
fixes $f :: nat \Rightarrow real$
assumes *pos*: $\bigwedge n.\ 1 \leq f\ n$
and *le*: $\bigwedge n.\ (\prod i{<}n.\ f\ i) \leq x$
shows $\exists p.\ raw\_has\_prod\ f\ 0\ p$
unfolding *raw_has_prod_def add_0_right*
proof (*rule exI LIMSEQ_incseq_SUP conjI*)+

show *bdd_above* (*range* (λ*n*. *prod f* {..*n*}))
by (*metis bdd_aboveI2 le lessThan_Suc_atMost*)
then have (*SUP i*. *prod f* {..*i*}) > *0*
by (*metis UNIV_I cSUP_upper less_le_trans pos prod_pos zero_less_one*)
then show (*SUP i*. *prod f* {..*i*}) ≠ *0*
by *auto*
show *incseq* (λ*n*. *prod f* {..*n*})
using *pos order_trans* [*OF zero_le_one*] **by** (*auto simp*: *mono_def intro*!: *prod_mono2*)
qed

lemma *convergent_prodI_nonneg_bounded*:
fixes *f* :: *nat* ⇒ *real*
assumes ⋀*n*. *1* ≤ *f n* ⋀*n*. (∏ *i*<*n*. *f i*) ≤ *x*
shows *convergent_prod f*
using *convergent_prod_def raw_has_prodI_bounded* [*OF assms*] **by** *blast*

6.32.7 Infinite products on topological spaces

context
fixes *f g* :: *nat* ⇒ *'a*::{*t2_space*,*topological_semigroup_mult*,*idom*}
begin

lemma *raw_has_prod_mult*: ⟦*raw_has_prod f M a*; *raw_has_prod g M b*⟧ ⟹ *raw_has_prod* (λ*n*. *f n* ∗ *g n*) *M* (*a* ∗ *b*)
by (*force simp add*: *prod.distrib tendsto_mult raw_has_prod_def*)

lemma *has_prod_mult_nz*: ⟦*f has_prod a*; *g has_prod b*; *a* ≠ *0*; *b* ≠ *0*⟧ ⟹ (λ*n*. *f n* ∗ *g n*) *has_prod* (*a* ∗ *b*)
by (*simp add*: *raw_has_prod_mult has_prod_def*)

end

context
fixes *f g* :: *nat* ⇒ *'a*::*real_normed_field*
begin

lemma *has_prod_mult*:
assumes *f*: *f has_prod a* **and** *g*: *g has_prod b*
shows (λ*n*. *f n* ∗ *g n*) *has_prod* (*a* ∗ *b*)
using *f* [*unfolded has_prod_def*]
proof (*elim disjE exE conjE*)
assume *f0*: *raw_has_prod f 0 a*
show *?thesis*
using *g* [*unfolded has_prod_def*]
proof (*elim disjE exE conjE*)
assume *g0*: *raw_has_prod g 0 b*
with *f0* **show** *?thesis*

```
    by (force simp add: has_prod_def prod.distrib tendsto_mult raw_has_prod_def)
  next
    fix j q
    assume b = 0 and g j = 0 and q: raw_has_prod g (Suc j) q
    obtain p where p: raw_has_prod f (Suc j) p
      using f0 raw_has_prod_ignore_initial_segment by blast
    then have Ex (raw_has_prod (λn. f n * g n) (Suc j))
      using q raw_has_prod_mult by blast
    then show ?thesis
      using ‹b = 0› ‹g j = 0› has_prod_0_iff by fastforce
  qed
next
  fix i p
  assume a = 0 and f i = 0 and p: raw_has_prod f (Suc i) p
  show ?thesis
    using g [unfolded has_prod_def]
  proof (elim disjE exE conjE)
    assume g0: raw_has_prod g 0 b
    obtain q where q: raw_has_prod g (Suc i) q
      using g0 raw_has_prod_ignore_initial_segment by blast
    then have Ex (raw_has_prod (λn. f n * g n) (Suc i))
      using raw_has_prod_mult p by blast
    then show ?thesis
      using ‹a = 0› ‹f i = 0› has_prod_0_iff by fastforce
  next
    fix j q
    assume b = 0 and g j = 0 and q: raw_has_prod g (Suc j) q
    obtain p' where p': raw_has_prod f (Suc (max i j)) p'
      by (metis raw_has_prod_ignore_initial_segment max_Suc_Suc max_def p)
    moreover
    obtain q' where q': raw_has_prod g (Suc (max i j)) q'
    by (metis raw_has_prod_ignore_initial_segment max.cobounded2 max_Suc_Suc
q)
    ultimately show ?thesis
       using ‹b = 0› by (simp add: has_prod_def) (metis ‹f i = 0› ‹g j = 0›
raw_has_prod_mult max_def)
  qed
qed

lemma convergent_prod_mult:
  assumes f: convergent_prod f and g: convergent_prod g
  shows convergent_prod (λn. f n * g n)
  unfolding convergent_prod_def
proof −
  obtain M p N q where p: raw_has_prod f M p and q: raw_has_prod g N q
    using convergent_prod_def f g by blast+
 then obtain p' q' where p': raw_has_prod f (max M N) p' and q': raw_has_prod
g (max M N) q'
   by (meson raw_has_prod_ignore_initial_segment max.cobounded1 max.cobounded2)
```

then show $\exists M\ p.\ raw\_has\_prod\ (\lambda n.\ f\ n * g\ n)\ M\ p$
using *raw_has_prod_mult* **by** *blast*
qed

lemma *prodinf_mult*: *convergent_prod f* $\Longrightarrow$ *convergent_prod g* $\Longrightarrow$ *prodinf f * prodinf g* $= (\prod n.\ f\ n * g\ n)$
by (*intro has_prod_unique has_prod_mult convergent_prod_has_prod*)

end

context
fixes $f :: 'i \Rightarrow nat \Rightarrow 'a$::*real_normed_field*
and $I :: 'i\ set$
begin

lemma *has_prod_prod*: $(\bigwedge i.\ i \in I \Longrightarrow (f\ i)\ has\_prod\ (x\ i)) \Longrightarrow (\lambda n.\ \prod i{\in}I.\ f\ i\ n)\ has\_prod\ (\prod i{\in}I.\ x\ i)$
by (*induct I rule*: *infinite_finite_induct*) (*auto intro*!: *has_prod_mult*)

lemma *prodinf_prod*: $(\bigwedge i.\ i \in I \Longrightarrow convergent\_prod\ (f\ i)) \Longrightarrow (\prod n.\ \prod i{\in}I.\ f\ i\ n) = (\prod i{\in}I.\ \prod n.\ f\ i\ n)$
using *has_prod_unique*[*OF has_prod_prod*, *OF convergent_prod_has_prod*] **by** *simp*

lemma *convergent_prod_prod*: $(\bigwedge i.\ i \in I \Longrightarrow convergent\_prod\ (f\ i)) \Longrightarrow convergent\_prod\ (\lambda n.\ \prod i{\in}I.\ f\ i\ n)$
using *convergent_prod_has_prod_iff has_prod_prod prodinf_prod* **by** *force*

end

6.32.8 Infinite summability on real normed fields

context
fixes $f :: nat \Rightarrow 'a$::*real_normed_field*
begin

lemma *raw_has_prod_Suc_iff*: *raw_has_prod f M (a * f M)* $\longleftrightarrow$ *raw_has_prod* $(\lambda n.\ f\ (Suc\ n))\ M\ a \wedge f\ M \neq 0$
proof −
have *raw_has_prod f M (a * f M)* $\longleftrightarrow (\lambda i.\ \prod j{\leq}Suc\ i.\ f\ (j{+}M)) \longrightarrow a * f\ M \wedge a * f\ M \neq 0$
by (*subst filterlim_sequentially_Suc*) (*simp add*: *raw_has_prod_def*)
also have ... $\longleftrightarrow (\lambda i.\ (\prod j{\leq}i.\ f\ (Suc\ j + M)) * f\ M) \longrightarrow a * f\ M \wedge a * f\ M \neq 0$
by (*simp add*: *ac_simps atMost_Suc_eq_insert_0 image_Suc_atMost prod.atLeast1_atMost_eq lessThan_Suc_atMost*
del: *prod.cl_ivl_Suc*)
also have ... $\longleftrightarrow$ *raw_has_prod* $(\lambda n.\ f\ (Suc\ n))\ M\ a \wedge f\ M \neq 0$
proof *safe*

```
    assume tends: (λi. (∏j≤i. f (Suc j + M)) * f M) ⟶ a * f M and 0: a
* f M ≠ 0
    with tendsto_divide[OF tends tendsto_const, of f M]
    show raw_has_prod (λn. f (Suc n)) M a
      by (simp add: raw_has_prod_def)
  qed (auto intro: tendsto_mult_right simp:  raw_has_prod_def)
  finally show ?thesis .
qed

lemma has_prod_Suc_iff:
  assumes f 0 ≠ 0 shows (λn. f (Suc n)) has_prod a ⟷ f has_prod (a * f 0)
proof (cases a = 0)
  case True
  then show ?thesis
  proof (simp add: has_prod_def, safe)
    fix i x
    assume f (Suc i) = 0 and raw_has_prod (λn. f (Suc n)) (Suc i) x
    then obtain y where raw_has_prod f (Suc (Suc i)) y
    by (metis (no_types) raw_has_prod_eq_0 Suc_n_not_le_n raw_has_prod_Suc_iff
raw_has_prod_ignore_initial_segment raw_has_prod_nonzero linear)
    then show ∃ i. f i = 0 ∧ Ex (raw_has_prod f (Suc i))
      using ‹f (Suc i) = 0› by blast
  next
    fix i x
    assume f i = 0 and x: raw_has_prod f (Suc i) x
    then obtain j where j: i = Suc j
      by (metis assms not0_implies_Suc)
    moreover have ∃ y. raw_has_prod (λn. f (Suc n)) i y
      using x by (auto simp: raw_has_prod_def)
    then show ∃ i. f (Suc i) = 0 ∧ Ex (raw_has_prod (λn. f (Suc n)) (Suc i))
      using ‹f i = 0› j by blast
  qed
next
  case False
  then show ?thesis
    by (auto simp: has_prod_def raw_has_prod_Suc_iff assms)
qed

lemma convergent_prod_Suc_iff [simp]:
  shows convergent_prod (λn. f (Suc n)) = convergent_prod f
proof
  assume convergent_prod f
  then obtain M L where M_nz:∀ n≥M. f n ≠ 0 and
        M_L:(λn. ∏ i≤n. f (i + M)) ⟶ L and L ≠ 0
    unfolding convergent_prod_altdef by auto
  have (λn. ∏ i≤n. f (Suc (i + M))) ⟶ L / f M
  proof −
    have (λn. ∏ i∈{0..Suc n}. f (i + M)) ⟶ L
      using M_L
```

```
    apply (subst (asm) filterlim_sequentially_Suc[symmetric])
    using atLeast0AtMost by auto
  then have (λn. f M * (∏ i∈{0..n}. f (Suc (i + M)))) ⟶ L
    apply (subst (asm) prod.atLeast0_atMost_Suc_shift)
    by simp
  then have (λn. (∏ i∈{0..n}. f (Suc (i + M)))) ⟶ L/f M
    apply (drule_tac tendsto_divide)
    using M_nz[rule_format,of M,simplified] by auto
  then show ?thesis unfolding atLeast0AtMost .
 qed
 then show convergent_prod (λn. f (Suc n)) unfolding convergent_prod_altdef
  apply (rule_tac exI[where x=M])
  apply (rule_tac exI[where x=L/f M])
  using M_nz ‹L≠0› by auto
next
 assume convergent_prod (λn. f (Suc n))
 then obtain M where ∃ L. (∀ n≥M. f (Suc n) ≠ 0) ∧ (λn. ∏ i≤n. f (Suc (i
+ M))) ⟶ L ∧ L ≠ 0
  unfolding convergent_prod_altdef by auto
 then show convergent_prod f unfolding convergent_prod_altdef
  apply (rule_tac exI[where x=Suc M])
  using Suc_le_D by auto
qed

lemma raw_has_prod_inverse:
  assumes raw_has_prod f M a shows raw_has_prod (λn. inverse (f n)) M
(inverse a)
 using assms unfolding raw_has_prod_def by (auto dest: tendsto_inverse simp:
prod_inversef [symmetric])

lemma has_prod_inverse:
 assumes f has_prod a shows (λn. inverse (f n)) has_prod (inverse a)
using assms raw_has_prod_inverse unfolding has_prod_def by auto

lemma convergent_prod_inverse:
 assumes convergent_prod f
 shows convergent_prod (λn. inverse (f n))
 using assms unfolding convergent_prod_def by (blast intro: raw_has_prod_inverse
elim: )

end

context
 fixes f :: nat ⇒ 'a::real_normed_field
begin

lemma raw_has_prod_Suc_iff': raw_has_prod f M a ⟷ raw_has_prod (λn. f
(Suc n)) M (a / f M) ∧ f M ≠ 0
 by (metis raw_has_prod_eq_0 add.commute add.left_neutral raw_has_prod_Suc_iff
```

raw_has_prod_nonzero le_add1 nonzero_mult_div_cancel_right times_divide_eq_left)

lemma *has_prod_divide*: *f has_prod a* $\Longrightarrow$ *g has_prod b* $\Longrightarrow$ $(\lambda n.\ f\ n\ /\ g\ n)$ *has_prod* $(a\ /\ b)$
unfolding *divide_inverse* **by** (*intro has_prod_inverse has_prod_mult*)

lemma *convergent_prod_divide*:
assumes *f*: *convergent_prod f* **and** *g*: *convergent_prod g*
shows *convergent_prod* $(\lambda n.\ f\ n\ /\ g\ n)$
using *f g has_prod_divide has_prod_iff* **by** *blast*

lemma *prodinf_divide*: *convergent_prod f* $\Longrightarrow$ *convergent_prod g* $\Longrightarrow$ *prodinf f* / *prodinf g* $= (\prod n.\ f\ n\ /\ g\ n)$
by (*intro has_prod_unique has_prod_divide convergent_prod_has_prod*)

lemma *prodinf_inverse*: *convergent_prod f* $\Longrightarrow$ $(\prod n.\ inverse\ (f\ n))$ $=$ *inverse* $(\prod n.\ f\ n)$
by (*intro has_prod_unique* [*symmetric*] *has_prod_inverse convergent_prod_has_prod*)

lemma *has_prod_Suc_imp*:
assumes $(\lambda n.\ f\ (Suc\ n))$ *has_prod a*
shows *f has_prod* $(a * f\ 0)$
proof −
have *f has_prod* $(a * f\ 0)$ **when** *raw_has_prod* $(\lambda n.\ f\ (Suc\ n))\ 0\ a$
apply (*cases f 0=0*)
using *that* **unfolding** *has_prod_def raw_has_prod_Suc*
by (*auto simp add*: *raw_has_prod_Suc_iff*)
moreover have *f has_prod* $(a * f\ 0)$ **when**
$(\exists i\ q.\ a = 0 \wedge f\ (Suc\ i) = 0 \wedge$ *raw_has_prod* $(\lambda n.\ f\ (Suc\ n))\ (Suc\ i)\ q)$
proof −
from *that*
obtain *i q* **where** $a = 0\ f\ (Suc\ i) = 0$ *raw_has_prod* $(\lambda n.\ f\ (Suc\ n))\ (Suc\ i)\ q$
by *auto*
then show *?thesis* **unfolding** *has_prod_def*
by (*auto intro*!:*exI*[**where** *x*=*Suc i*] *simp*:*raw_has_prod_Suc*)
qed
ultimately show *f has_prod* $(a * f\ 0)$ **using** *assms* **unfolding** *has_prod_def*
by *auto*
qed

lemma *has_prod_iff_shift*:
assumes $\bigwedge i.\ i < n \Longrightarrow f\ i \neq 0$
shows $(\lambda i.\ f\ (i + n))$ *has_prod a* $\longleftrightarrow$ *f has_prod* $(a * (\prod i{<}n.\ f\ i))$
using *assms*
proof (*induct n arbitrary*: *a*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)

then have $(\lambda i.\ f\ (Suc\ i + n))\ has\_prod\ a \longleftrightarrow (\lambda i.\ f\ (i + n))\ has\_prod\ (a * f\ n)$
by (*subst has_prod_Suc_iff*) *auto*
with *Suc* **show** *?case*
by (*simp add*: *ac_simps*)
qed

corollary *has_prod_iff_shift'*:
assumes $\bigwedge i.\ i < n \Longrightarrow f\ i \neq 0$
shows $(\lambda i.\ f\ (i + n))\ has\_prod\ (a\ /\ (\prod i{<}n.\ f\ i)) \longleftrightarrow f\ has\_prod\ a$
by (*simp add*: *assms has_prod_iff_shift*)

lemma *has_prod_one_iff_shift*:
assumes $\bigwedge i.\ i < n \Longrightarrow f\ i = 1$
shows $(\lambda i.\ f\ (i{+}n))\ has\_prod\ a \longleftrightarrow (\lambda i.\ f\ i)\ has\_prod\ a$
by (*simp add*: *assms has_prod_iff_shift*)

lemma *convergent_prod_iff_shift* [*simp*]:
shows *convergent_prod* $(\lambda i.\ f\ (i + n)) \longleftrightarrow$ *convergent_prod f*
apply *safe*
using *convergent_prod_offset* **apply** *blast*
using *convergent_prod_ignore_initial_segment convergent_prod_def* **by** *blast*

lemma *has_prod_split_initial_segment*:
assumes *f has_prod a* $\bigwedge i.\ i < n \Longrightarrow f\ i \neq 0$
shows $(\lambda i.\ f\ (i + n))\ has\_prod\ (a\ /\ (\prod i{<}n.\ f\ i))$
using *assms has_prod_iff_shift'* **by** *blast*

lemma *prodinf_divide_initial_segment*:
assumes *convergent_prod f* $\bigwedge i.\ i < n \Longrightarrow f\ i \neq 0$
shows $(\prod i.\ f\ (i + n)) = (\prod i.\ f\ i)\ /\ (\prod i{<}n.\ f\ i)$
by (*rule has_prod_unique*[*symmetric*]) (*auto simp*: *assms has_prod_iff_shift*)

lemma *prodinf_split_initial_segment*:
assumes *convergent_prod f* $\bigwedge i.\ i < n \Longrightarrow f\ i \neq 0$
shows *prodinf f* $= (\prod i.\ f\ (i + n)) * (\prod i{<}n.\ f\ i)$
by (*auto simp add*: *assms prodinf_divide_initial_segment*)

lemma *prodinf_split_head*:
assumes *convergent_prod f f 0* $\neq 0$
shows $(\prod n.\ f\ (Suc\ n)) =$ *prodinf f* $/$ *f 0*
using *prodinf_split_initial_segment*[*of 1*] *assms* **by** *simp*

lemma *has_prod_ignore_initial_segment'*:
assumes *convergent_prod f*
shows *f has_prod* $((\prod k{<}n.\ f\ k) * (\prod k.\ f\ (k + n)))$
proof (*cases* $\exists k{<}n.\ f\ k = 0$)
case *True*
hence [*simp*]: $(\prod k{<}n.\ f\ k) = 0$

```
    by (meson finite_lessThan lessThan_iff prod_zero)
  thus ?thesis using True assms
    by (metis convergent_prod_has_prod_iff has_prod_zeroI mult_not_zero)
next
  case False
  hence (λi. f (i + n)) has_prod (prodinf f / prod f {..<n})
    using assms by (intro has_prod_split_initial_segment) (auto simp: convergent_prod_has_prod_iff)
  hence prodinf f = prod f {..<n} * (∏ k. f (k + n))
    using False by (simp add: has_prod_iff divide_simps mult_ac)
  thus ?thesis
    using assms by (simp add: convergent_prod_has_prod_iff)
qed

end

context
  fixes f :: nat ⇒ 'a::real_normed_field
begin

lemma convergent_prod_inverse_iff [simp]: convergent_prod (λn. inverse (f n))
⟷ convergent_prod f
  by (auto dest: convergent_prod_inverse)

lemma convergent_prod_const_iff [simp]:
  fixes c :: 'a :: {real_normed_field}
  shows convergent_prod (λ_. c) ⟷ c = 1
proof
  assume convergent_prod (λ_. c)
  then show c = 1
    using convergent_prod_imp_LIMSEQ LIMSEQ_unique by blast
next
  assume c = 1
  then show convergent_prod (λ_. c)
    by auto
qed

lemma has_prod_power: f has_prod a ⟹ (λi. f i ^ n) has_prod (a ^ n)
  by (induction n) (auto simp: has_prod_mult)

lemma convergent_prod_power: convergent_prod f ⟹ convergent_prod (λi. f i
^ n)
  by (induction n) (auto simp: convergent_prod_mult)

lemma prodinf_power: convergent_prod f ⟹ prodinf (λi. f i ^ n) = prodinf f ^
n
  by (metis has_prod_unique convergent_prod_imp_has_prod has_prod_power)

end
```

6.32.9 Exponentials and logarithms

context
fixes $f :: nat \Rightarrow 'a::\{real\_normed\_field,banach\}$
begin

lemma *sums_imp_has_prod_exp*:
assumes *f sums s*
shows *raw_has_prod* $(\lambda i.\ exp\ (f\ i))\ 0\ (exp\ s)$
using *assms continuous_on_exp* [*of UNIV* $\lambda x::'a.\ x$]
using *continuous_on_tendsto_compose* [*of UNIV exp* $(\lambda n.\ sum\ f\ \{..n\})\ s$]
by (*simp add*: *prod_defs sums_def_le exp_sum*)

lemma *convergent_prod_exp*:
assumes *summable f*
shows *convergent_prod* $(\lambda i.\ exp\ (f\ i))$
using *sums_imp_has_prod_exp assms* **unfolding** *summable_def convergent_prod_def*
by *blast*

lemma *prodinf_exp*:
assumes *summable f*
shows *prodinf* $(\lambda i.\ exp\ (f\ i)) = exp\ (suminf\ f)$
proof −
have *f sums suminf f*
using *assms* **by** *blast*
then have $(\lambda i.\ exp\ (f\ i))$ *has_prod exp* $(suminf\ f)$
by (*simp add*: *has_prod_def sums_imp_has_prod_exp*)
then show *?thesis*
by (*rule has_prod_unique* [*symmetric*])
qed

end

theorem *convergent_prod_iff_summable_real*:
fixes $a :: nat \Rightarrow real$
assumes $\bigwedge n.\ a\ n > 0$
shows *convergent_prod* $(\lambda k.\ 1 + a\ k) \longleftrightarrow$ *summable a* (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then obtain *p* **where** *raw_has_prod* $(\lambda k.\ 1 + a\ k)\ 0\ p$
by (*metis assms add_less_same_cancel2 convergent_prod_offset_0 not_one_less_zero*)
then have *to_p*: $(\lambda n.\ \prod k{\le}n.\ 1 + a\ k) \longrightarrow p$
by (*auto simp*: *raw_has_prod_def*)
moreover have *le*: $(\sum k{\le}n.\ a\ k) \le (\prod k{\le}n.\ 1 + a\ k)$ **for** *n*
by (*rule sum_le_prod*) (*use assms less_le* **in** *force*)
have $(\prod k{\le}n.\ 1 + a\ k) \le p$ **for** *n*
proof (*rule incseq_le* [*OF _ to_p*])
show *incseq* $(\lambda n.\ \prod k{\le}n.\ 1 + a\ k)$
using *assms* **by** (*auto simp*: *mono_def order.strict_implies_order intro*!:
prod_mono2)

```
  qed
  with le have (∑k≤n. a k) ≤ p for n
    by (metis order_trans)
  with assms bounded_imp_summable show ?rhs
    by (metis not_less order.asym)
next
  assume R: ?rhs
  have (∏k≤n. 1 + a k) ≤ exp (suminf a) for n
  proof −
    have (∏k≤n. 1 + a k) ≤ exp (∑k≤n. a k) for n
      by (rule prod_le_exp_sum) (use assms less_le in force)
    moreover have exp (∑k≤n. a k) ≤ exp (suminf a) for n
      unfolding exp_le_cancel_iff
      by (meson sum_le_suminf R assms finite_atMost less_eq_real_def)
    ultimately show ?thesis
      by (meson order_trans)
  qed
  then obtain L where L: (λn. ∏k≤n. 1 + a k) ⟶ L
    by (metis assms bounded_imp_convergent_prod convergent_prod_iff_nz_lim
le_add_same_cancel1 le_add_same_cancel2 less_le not_le zero_le_one)
  moreover have L ≠ 0
  proof
    assume L = 0
    with L have (λn. ∏k≤n. 1 + a k) ⟶ 0
      by simp
    moreover have (∏k≤n. 1 + a k) > 1 for n
      by (simp add: assms less_1_prod)
    ultimately show False
      by (meson Lim_bounded2 not_one_le_zero less_imp_le)
  qed
  ultimately show ?lhs
    using assms convergent_prod_iff_nz_lim
    by (metis add_less_same_cancel1 less_le not_le zero_less_one)
qed

lemma exp_suminf_prodinf_real:
  fixes f :: nat ⇒ real
  assumes ge0:⋀n. f n ≥ 0 and ac: abs_convergent_prod (λn. exp (f n))
  shows prodinf (λi. exp (f i)) = exp (suminf f)
proof −
  have summable f
    using ac unfolding abs_convergent_prod_conv_summable
  proof (elim summable_comparison_test')
    fix n
    have |f n| = f n
      by (simp add: ge0)
    also have ... ≤ exp (f n) − 1
      by (metis diff_diff_add exp_ge_add_one_self ge_iff_diff_ge_0)
    finally show norm (f n) ≤ norm (exp (f n) − 1)
```

```
    by simp
  qed
  then show ?thesis
    by (simp add: prodinf_exp)
qed

lemma has_prod_imp_sums_ln_real:
  fixes f :: nat ⇒ real
  assumes raw_has_prod f 0 p and 0: ⋀x. f x > 0
  shows (λi. ln (f i)) sums (ln p)
proof −
  have p > 0
   using assms unfolding prod_defs by (metis LIMSEQ_prod_nonneg less_eq_real_def)
  then show ?thesis
  using assms continuous_on_ln [of {0<..} λx. x]
  using continuous_on_tendsto_compose [of {0<..} ln (λn. prod f {..n}) p]
  by (auto simp: prod_defs sums_def_le ln_prod order_tendstoD)
qed

lemma summable_ln_real:
  fixes f :: nat ⇒ real
  assumes f: convergent_prod f and 0: ⋀x. f x > 0
  shows summable (λi. ln (f i))
proof −
  obtain M p where raw_has_prod f M p
    using f convergent_prod_def by blast
  then consider i<M f i = 0 | p where raw_has_prod f 0 p
    using raw_has_prod_cases by blast
  then show ?thesis
  proof cases
    case 1
    with 0 show ?thesis
      by (metis less_irrefl)
  next
    case 2
    then show ?thesis
      using 0 has_prod_imp_sums_ln_real summable_def by blast
  qed
qed

lemma suminf_ln_real:
  fixes f :: nat ⇒ real
  assumes f: convergent_prod f and 0: ⋀x. f x > 0
  shows suminf (λi. ln (f i)) = ln (prodinf f)
proof −
  have f has_prod prodinf f
    by (simp add: f has_prod_iff)
  then have raw_has_prod f 0 (prodinf f)
    by (metis 0 has_prod_def less_irrefl)
```

then have $(\lambda i.\ ln\ (f\ i))\ sums\ ln\ (prodinf\ f)$
using *0 has_prod_imp_sums_ln_real* **by** *blast*
then show *?thesis*
by (*rule sums_unique* [*symmetric*])
qed

lemma *prodinf_exp_real*:
fixes $f :: nat \Rightarrow real$
assumes *f*: *convergent_prod f* **and** *0*: $\bigwedge x.\ f\ x > 0$
shows $prodinf\ f = exp\ (suminf\ (\lambda i.\ ln\ (f\ i)))$
by (*simp add*: *0 f less_0_prodinf suminf_ln_real*)

theorem *Ln_prodinf_complex*:
fixes $z :: nat \Rightarrow complex$
assumes *z*: $\bigwedge j.\ z\ j \neq 0$ **and** ξ: $\xi \neq 0$
shows $((\lambda n.\ \prod j{\le}n.\ z\ j) \longrightarrow \xi) \longleftrightarrow (\exists k.\ (\lambda n.\ (\sum j{\le}n.\ Ln\ (z\ j))) \longrightarrow Ln\ \xi + of\_int\ k * (of\_real(2{*}pi) * \mathrm{i}))$ (**is** *?lhs = ?rhs*)
proof
assume *L*: *?lhs*
have *pnz*: $(\prod j{\le}n.\ z\ j) \neq 0$ **for** *n*
using *z* **by** *auto*
define Θ **where** $\Theta \equiv Arg\ \xi + 2{*}pi$
then have $\Theta > pi$
using *Arg_def mpi_less_Im_Ln* **by** *fastforce*
have ξ_eq: $\xi = cmod\ \xi * exp\ (\mathrm{i} * \Theta)$
using *Arg_def Arg_eq* ξ **unfolding** Θ_def **by** (*simp add*: *algebra_simps exp_add*)
define ϑ **where** $\vartheta \equiv \lambda n.\ THE\ t.\ is\_Arg\ (\prod j{\le}n.\ z\ j)\ t \wedge t \in \{\Theta{-}pi{<}..\Theta{+}pi\}$
have *uniq*: $\exists!s.\ is\_Arg\ (\prod j{\le}n.\ z\ j)\ s \wedge s \in \{\Theta{-}pi{<}..\Theta{+}pi\}$ **for** *n*
using *Argument_exists_unique* [*OF pnz*] **by** *metis*
have ϑ: $is\_Arg\ (\prod j{\le}n.\ z\ j)\ (\vartheta\ n)$ **and** $\vartheta\_interval$: $\vartheta\ n \in \{\Theta{-}pi{<}..\Theta{+}pi\}$ **for** *n*
unfolding ϑ_def
using *theI'* [*OF uniq*] **by** *metis+*
have ϑ_pos: $\bigwedge j.\ \vartheta\ j > 0$
using $\vartheta\_interval$ ‹$\Theta > pi$› **by** *simp* (*meson diff_gt_0_iff_gt less_trans*)
have $(\prod j{\le}n.\ z\ j) = cmod\ (\prod j{\le}n.\ z\ j) * exp\ (\mathrm{i} * \vartheta\ n)$ **for** *n*
using ϑ **by** (*auto simp*: *is_Arg_def*)
then have *eq*: $(\lambda n.\ \prod j{\le}n.\ z\ j) = (\lambda n.\ cmod\ (\prod j{\le}n.\ z\ j) * exp\ (\mathrm{i} * \vartheta\ n))$
by *simp*
then have $(\lambda n.\ (cmod\ (\prod j{\le}n.\ z\ j)) * exp\ (\mathrm{i} * (\vartheta\ n))) \longrightarrow \xi$
using *L* **by** *force*
then obtain *k* **where** *k*: $(\lambda j.\ \vartheta\ j - of\_int\ (k\ j) * (2 * pi)) \longrightarrow \Theta$
using *L* **by** (*subst* (*asm*) ξ_eq) (*auto simp add*: *eq z* ξ *polar_convergence*)
moreover have $\forall_F\ n\ in\ sequentially.\ k\ n = 0$
proof −
have $*$: $kj = 0$ **if** $dist\ (vj - real\_of\_int\ kj * 2)\ V < 1\ vj \in \{V - 1{<}..V + 1\}$ **for** *kj vj V*

```
    using that  by (auto simp: dist_norm)
   have ∀F j in sequentially. dist (ϑ j − of_int (k j) ∗ (2 ∗ pi)) Θ < pi
    using tendstoD [OF k] pi_gt_zero by blast
   then show ?thesis
   proof (rule eventually_mono)
    fix j
    assume d: dist (ϑ j − real_of_int (k j) ∗ (2 ∗ pi)) Θ < pi
    show k j = 0
     by (rule ∗ [of ϑ j/pi _ Θ/pi])
       (use ϑ_interval [of j] d in ‹simp_all add: divide_simps dist_norm›)
   qed
  qed
  ultimately have ϑtoΘ: ϑ ⟶ Θ
   apply (simp only: tendsto_def)
   apply (erule all_forward imp_forward asm_rl)+
   apply (drule (1) eventually_conj)
   apply (auto elim: eventually_mono)
   done
  then have to0: (λn. |ϑ (Suc n) − ϑ n|) ⟶ 0
   by (metis (full_types) diff_self filterlim_sequentially_Suc tendsto_diff tendsto_rabs_zero)
  have ∃k. Im (∑j≤n. Ln (z j)) − of_int k ∗ (2∗pi) = ϑ n for n
  proof (rule is_Arg_exp_diff_2pi)
   show is_Arg (exp (∑j≤n. Ln (z j))) (ϑ n)
    using pnz ϑ by (simp add: is_Arg_def exp_sum prod_norm)
  qed
  then have ∃k. (∑j≤n. Im (Ln (z j))) = ϑ n + of_int k ∗ (2∗pi) for n
   by (simp add: algebra_simps)
  then obtain k where k: ⋀n. (∑j≤n. Im (Ln (z j))) = ϑ n + of_int (k n) ∗ (2∗pi)
   by metis
  obtain K where ∀F n in sequentially. k n = K
  proof −
   have k_le: (2∗pi) ∗ |k (Suc n) − k n| ≤ |ϑ (Suc n) − ϑ n| + |Im (Ln (z (Suc n)))| for n
   proof −
    have (∑j≤Suc n. Im (Ln (z j))) − (∑j≤n. Im (Ln (z j))) = Im (Ln (z (Suc n)))
     by simp
    then show ?thesis
     using k [of Suc n] k [of n] by (auto simp: abs_if algebra_simps)
   qed
   have z ⟶ 1
   using L ξ convergent_prod_iff_nz_lim z by (blast intro: convergent_prod_imp_LIMSEQ)
   with z have (λn. Ln (z n)) ⟶ Ln 1
   using isCont_tendsto_compose [OF continuous_at_Ln] nonpos_Reals_one_I
by blast
   then have (λn. Ln (z n)) ⟶ 0
    by simp
```

```
    then have (λn. |Im (Ln (z (Suc n)))|) ⟶ 0
    by (metis LIMSEQ_unique ‹z ⟶ 1› continuous_at_Ln filterlim_sequentially_Suc
isCont_tendsto_compose nonpos_Reals_one_I tendsto_Im tendsto_rabs_zero_iff
zero_complex.simps(2))
    then have ∀F n in sequentially. |Im (Ln (z (Suc n)))| < 1
      by (simp add: order_tendsto_iff)
    moreover have ∀F n in sequentially. |ϑ (Suc n) − ϑ n| < 1
      using to0 by (simp add: order_tendsto_iff)
    ultimately have ∀F n in sequentially. (2*pi) * |k (Suc n) − k n| < 1 + 1
    proof (rule eventually_elim2)
      fix n
      assume |Im (Ln (z (Suc n)))| < 1 and |ϑ (Suc n) − ϑ n| < 1
      with k_le [of n] show 2 * pi * real_of_int |k (Suc n) − k n| < 1 + 1
        by linarith
    qed
    then have ∀F n in sequentially. real_of_int|k (Suc n) − k n| < 1
    proof (rule eventually_mono)
      fix n :: nat
      assume 2 * pi * |k (Suc n) − k n| < 1 + 1
      then have |k (Suc n) − k n| < 2 / (2*pi)
        by (simp add: field_simps)
      also have ... < 1
        using pi_ge_two by auto
      finally show real_of_int |k (Suc n) − k n| < 1 .
    qed
  then obtain N where N: ⋀n. n≥N ⟹ |k (Suc n) − k n| = 0
    using eventually_sequentially less_irrefl of_int_abs by fastforce
  have k (N+i) = k N for i
  proof (induction i)
    case (Suc i)
    with N [of N+i] show ?case
      by auto
  qed simp
  then have ⋀n. n≥N ⟹ k n = k N
    using le_Suc_ex by auto
  then show ?thesis
    by (force simp add: eventually_sequentially intro: that)
  qed
  with ϑtoΘ have (λn. (∑j≤n. Im (Ln (z j)))) ⟶ Θ + of_int K * (2*pi)
    by (simp add: k tendsto_add tendsto_mult tendsto_eventually)
  moreover have (λn. (∑k≤n. Re (Ln (z k)))) ⟶ Re (Ln ξ)
    using assms continuous_imp_tendsto [OF isCont_ln tendsto_norm [OF L]]
    by (simp add: o_def flip: prod_norm ln_prod)
  ultimately show ?rhs
   by (rule_tac x=K+1 in exI) (auto simp: tendsto_complex_iff Θ_def Arg_def
assms algebra_simps)
next
  assume ?rhs
  then obtain r where r: (λn. (∑k≤n. Ln (z k))) ⟶ Ln ξ + of_int r *
```

```
(of_real(2*pi) * i) ..
  have (λn. exp (∑k≤n. Ln (z k))) ⟶ ξ
    using assms continuous_imp_tendsto [OF isCont_exp r] exp_integer_2pi [of
r]
    by (simp add: o_def exp_add algebra_simps)
  moreover have exp (∑k≤n. Ln (z k)) = (∏k≤n. z k) for n
    by (simp add: exp_sum add_eq_0_iff assms)
  ultimately show ?lhs
    by auto
qed
```

Prop 17.2 of Bak and Newman, Complex Analysis, p.242

```
proposition convergent_prod_iff_summable_complex:
  fixes z :: nat ⇒ complex
  assumes ⋀k. z k ≠ 0
  shows convergent_prod (λk. z k) ⟷ summable (λk. Ln (z k)) (is ?lhs = ?rhs)
proof
  assume ?lhs
  then obtain p where p: (λn. ∏k≤n. z k) ⟶ p and p ≠ 0
     using convergent_prod_LIMSEQ prodinf_nonzero add_eq_0_iff assms by
fastforce
  then show ?rhs
    using Ln_prodinf_complex assms
    by (auto simp: prodinf_nonzero summable_def sums_def_le)
next
  assume R: ?rhs
  have (∏k≤n. z k) = exp (∑k≤n. Ln (z k)) for n
    by (simp add: exp_sum add_eq_0_iff assms)
  then have (λn. ∏k≤n. z k) ⟶ exp (suminf (λk. Ln (z k)))
    using continuous_imp_tendsto [OF isCont_exp summable_LIMSEQ' [OF R]]
by (simp add: o_def)
  then show ?lhs
    by (subst convergent_prod_iff_convergent) (auto simp: convergent_def tend-
sto_Lim assms add_eq_0_iff)
qed
```

Prop 17.3 of Bak and Newman, Complex Analysis

```
proposition summable_imp_convergent_prod_complex:
  fixes z :: nat ⇒ complex
  assumes z: summable (λk. norm (z k)) and non0: ⋀k. z k ≠ −1
  shows convergent_prod (λk. 1 + z k)
proof −
  obtain N where ⋀k. k≥N ⟹ norm (z k) < 1/2
    using summable_LIMSEQ_zero [OF z]
   by (metis diff_zero dist_norm half_gt_zero_iff less_numeral_extra(1) lim_sequentially
tendsto_norm_zero_iff)
  then have summable (λk. Ln (1 + z k))
    by (metis norm_Ln_le summable_comparison_test summable_mult z)
  with non0 show ?thesis
```

by (*simp add*: *add_eq_0_iff convergent_prod_iff_summable_complex*)
qed

corollary *summable_imp_convergent_prod_real*:
fixes $z :: nat \Rightarrow real$
assumes *z*: *summable* $(\lambda k.\ |z\ k|)$ **and** *non0*: $\bigwedge k.\ z\ k \neq -1$
shows *convergent_prod* $(\lambda k.\ 1 + z\ k)$
proof −
have $\bigwedge k.\ (complex\_of\_real \circ z)\ k \neq -1$
by (*metis non0 o_apply of_real_1 of_real_eq_iff of_real_minus*)
with *z*
have *convergent_prod* $(\lambda k.\ 1 + (complex\_of\_real \circ z)\ k)$
by (*auto intro*: *summable_imp_convergent_prod_complex*)
then show *?thesis*
using *convergent_prod_of_real_iff* [*of* $\lambda k.\ 1 + z\ k$] **by** (*simp add*: *o_def*)
qed

lemma *summable_Ln_complex*:
fixes $z :: nat \Rightarrow complex$
assumes *convergent_prod z* $\bigwedge k.\ z\ k \neq 0$
shows *summable* $(\lambda k.\ Ln\ (z\ k))$
using *convergent_prod_def assms convergent_prod_iff_summable_complex* **by** *blast*

6.32.10 Embeddings from the reals into some complete real normed field

lemma *tendsto_eq_of_real_lim*:
assumes $(\lambda n.\ of\_real\ (f\ n) :: 'a{::}\{complete\_space, real\_normed\_field\}) \longrightarrow q$
shows $q = of\_real\ (lim\ f)$
proof −
have *convergent* $(\lambda n.\ of\_real\ (f\ n) :: 'a)$
using *assms convergent_def* **by** *blast*
then have *convergent f*
unfolding *convergent_def*
by (*simp add*: *convergent_eq_Cauchy Cauchy_def*)
then show *?thesis*
by (*metis LIMSEQ_unique assms convergentD sequentially_bot tendsto_Lim tendsto_of_real*)
qed

lemma *tendsto_eq_of_real*:
assumes $(\lambda n.\ of\_real\ (f\ n) :: 'a{::}\{complete\_space, real\_normed\_field\}) \longrightarrow q$
obtains *r* **where** $q = of\_real\ r$
using *tendsto_eq_of_real_lim assms* **by** *blast*

lemma *has_prod_of_real_iff* [*simp*]:
$(\lambda n.\ of\_real\ (f\ n) :: 'a{::}\{complete\_space, real\_normed\_field\})\ has\_prod\ of\_real\ c \longleftrightarrow f\ has\_prod\ c$

(**is** *?lhs = ?rhs*)
proof
assume *?lhs*
then show *?rhs*
apply (*auto simp*: *prod_defs LIMSEQ_prod_0 tendsto_of_real_iff simp flip*: *of_real_prod*)
using *tendsto_eq_of_real*
by (*metis of_real_0 tendsto_of_real_iff*)
next
assume *?rhs*
with *tendsto_of_real_iff* **show** *?lhs*
by (*fastforce simp*: *prod_defs simp flip*: *of_real_prod*)
qed

end

6.33 Sums over Infinite Sets

theory *Infinite_Set_Sum*
imports *Set_Integral Infinite_Sum*
begin

Conflicting notation from *HOL−Analysis.Infinite_Sum*

no_notation *Infinite_Sum.abs_summable_on* (**infixr** *abs'_summable'_on 46*)

lemma *sets_eq_countable*:
assumes *countable A space M = A* $\bigwedge x.\ x \in A \Longrightarrow \{x\} \in$ *sets M*
shows *sets M = Pow A*
proof (*intro equalityI subsetI*)
fix *X* **assume** *X ∈ Pow A*
hence $(\bigcup x{\in}X.\ \{x\}) \in$ *sets M*
by (*intro sets.countable_UN' countable_subset*[*OF _ assms*(*1*)]) (*auto intro*!: *assms*(*3*))
also have $(\bigcup x{\in}X.\ \{x\}) = X$ **by** *auto*
finally show *X ∈ sets M* **.**
next
fix *X* **assume** *X ∈ sets M*
from *sets.sets_into_space*[*OF this*] **and** *assms*
show *X ∈ Pow A* **by** *simp*
qed

lemma *measure_eqI_countable'*:
assumes *spaces*: *space M = A space N = A*
assumes *sets*: $\bigwedge x.\ x \in A \Longrightarrow \{x\} \in$ *sets M* $\bigwedge x.\ x \in A \Longrightarrow \{x\} \in$ *sets N*
assumes *A*: *countable A*
assumes *eq*: $\bigwedge a.\ a \in A \Longrightarrow$ *emeasure M* $\{a\}$ = *emeasure N* $\{a\}$
shows *M = N*
proof (*rule measure_eqI_countable*)

```
  show sets M = Pow A
    by (intro sets_eq_countable assms)
  show sets N = Pow A
    by (intro sets_eq_countable assms)
qed fact+

lemma count_space_PiM_finite:
  fixes B :: 'a ⇒ 'b set
  assumes finite A ⋀i. countable (B i)
  shows   PiM A (λi. count_space (B i)) = count_space (PiE A B)
proof (rule measure_eqI_countable')
  show space (PiM A (λi. count_space (B i))) = PiE A B
    by (simp add: space_PiM)
  show space (count_space (PiE A B)) = PiE A B by simp
next
  fix f assume f: f ∈ PiE A B
  hence PiE A (λx. {f x}) ∈ sets (Pi_M A (λi. count_space (B i)))
    by (intro sets_PiM_I_finite assms) auto
  also from f have PiE A (λx. {f x}) = {f}
    by (intro PiE_singleton) (auto simp: PiE_def)
  finally show {f} ∈ sets (Pi_M A (λi. count_space (B i))) .
next
  interpret product_sigma_finite (λi. count_space (B i))
   by (intro product_sigma_finite.intro sigma_finite_measure_count_space_countable
assms)
  thm sigma_finite_measure_count_space
  fix f assume f: f ∈ PiE A B
  hence {f} = PiE A (λx. {f x})
    by (intro PiE_singleton [symmetric]) (auto simp: PiE_def)
  also have emeasure (Pi_M A (λi. count_space (B i))) ... =
              (∏ i∈A. emeasure (count_space (B i)) {f i})
    using f assms by (subst emeasure_PiM) auto
  also have ... = (∏ i∈A. 1)
    by (intro prod.cong refl, subst emeasure_count_space_finite) (use f in auto)
  also have ... = emeasure (count_space (PiE A B)) {f}
    using f by (subst emeasure_count_space_finite) auto
  finally show emeasure (Pi_M A (λi. count_space (B i))) {f} =
                 emeasure (count_space (Pi_E A B)) {f} .
qed (simp_all add: countable_PiE assms)



definition abs_summable_on ::
    ('a ⇒ 'b :: {banach, second_countable_topology}) ⇒ 'a set ⇒ bool
    (infix abs'_summable'_on 50)
 where
   f abs_summable_on A ⟷ integrable (count_space A) f
```

definition *infsetsum* ::
  $('a \Rightarrow 'b :: \{banach, second\_countable\_topology\}) \Rightarrow 'a\ set \Rightarrow 'b$
where
  $infsetsum\ f\ A = lebesgue\_integral\ (count\_space\ A)\ f$

syntax (*ASCII*)
  $\_infsetsum :: pttrn \Rightarrow 'a\ set \Rightarrow 'b \Rightarrow 'b::\{banach, second\_countable\_topology\}$
  $((3INFSETSUM\ \_:\_./\ \_)\ [0, 51, 10]\ 10)$
syntax
  $\_infsetsum :: pttrn \Rightarrow 'a\ set \Rightarrow 'b \Rightarrow 'b::\{banach, second\_countable\_topology\}$
  $((2\sum_a \_\in\_./\ \_)\ [0, 51, 10]\ 10)$
translations — Beware of argument permutation!
  $\sum_a i\in A.\ b \rightleftharpoons CONST\ infsetsum\ (\lambda i.\ b)\ A$

syntax (*ASCII*)
  $\_uinfsetsum :: pttrn \Rightarrow 'a\ set \Rightarrow 'b \Rightarrow 'b::\{banach, second\_countable\_topology\}$
  $((3INFSETSUM\ \_:\_./\ \_)\ [0, 51, 10]\ 10)$
syntax
  $\_uinfsetsum :: pttrn \Rightarrow 'b \Rightarrow 'b::\{banach, second\_countable\_topology\}$
  $((2\sum_a \_./\ \_)\ [0, 10]\ 10)$
translations — Beware of argument permutation!
  $\sum_a i.\ b \rightleftharpoons CONST\ infsetsum\ (\lambda i.\ b)\ (CONST\ UNIV)$

syntax (*ASCII*)
  $\_qinfsetsum :: pttrn \Rightarrow bool \Rightarrow 'a \Rightarrow 'a::\{banach, second\_countable\_topology\}$
  $((3INFSETSUM\ \_\ |/\ \_./\ \_)\ [0, 0, 10]\ 10)$
syntax
  $\_qinfsetsum :: pttrn \Rightarrow bool \Rightarrow 'a \Rightarrow 'a::\{banach, second\_countable\_topology\}$
  $((2\sum_a \_\ |\ (\_)./\ \_)\ [0, 0, 10]\ 10)$
translations
  $\sum_a x|P.\ t => CONST\ infsetsum\ (\lambda x.\ t)\ \{x.\ P\}$

print_translation ‹
let
  *fun sum_tr' [Abs (x, Tx, t), Const (**const_syntax**‹Collect›, _) $ Abs (y, Ty, P)] =*
    if x <> y then raise Match
    else
      let
        val x' = Syntax_Trans.mark_bound_body (x, Tx);
        val t' = subst_bound (x', t);
        val P' = subst_bound (x', P);
      in
       *Syntax.const **syntax_const**‹_qinfsetsum› $ Syntax_Trans.mark_bound_abs (x, Tx) $ P' $ t'*
      end
  | sum_tr' _ = raise Match;
*in [(**const_syntax**‹infsetsum›, K sum_tr')] end*
›

lemma *restrict_count_space_subset*:
A ⊆ *B* ⟹ *restrict_space* (*count_space B*) *A* = *count_space A*
by (*subst restrict_count_space*) (*simp_all add*: *Int_absorb2*)

lemma *abs_summable_on_restrict*:
fixes *f* :: *'a* ⇒ *'b* :: {*banach*, *second_countable_topology*}
assumes *A* ⊆ *B*
shows *f abs_summable_on A* ⟷ (λ*x*. *indicator A x* ∗*R f x*) *abs_summable_on B*
proof −
have *count_space A* = *restrict_space* (*count_space B*) *A*
by (*rule restrict_count_space_subset* [*symmetric*]) *fact*+
also have *integrable* ... *f* ⟷ *set_integrable* (*count_space B*) *A f*
by (*simp add*: *integrable_restrict_space set_integrable_def*)
finally show *?thesis*
unfolding *abs_summable_on_def set_integrable_def* .
qed

lemma *abs_summable_on_altdef*: *f abs_summable_on A* ⟷ *set_integrable* (*count_space UNIV*) *A f*
unfolding *abs_summable_on_def set_integrable_def*
by (*metis* (*no_types*) *inf_top.right_neutral integrable_restrict_space restrict_count_space sets_UNIV*)

lemma *abs_summable_on_altdef'*:
A ⊆ *B* ⟹ *f abs_summable_on A* ⟷ *set_integrable* (*count_space B*) *A f*
unfolding *abs_summable_on_def set_integrable_def*
by (*metis* (*no_types*) *Pow_iff abs_summable_on_def inf.orderE integrable_restrict_space restrict_count_space_subset sets_count_space space_count_space*)

lemma *abs_summable_on_norm_iff* [*simp*]:
(λ*x*. *norm* (*f x*)) *abs_summable_on A* ⟷ *f abs_summable_on A*
by (*simp add*: *abs_summable_on_def integrable_norm_iff*)

lemma *abs_summable_on_normI*: *f abs_summable_on A* ⟹ (λ*x*. *norm* (*f x*)) *abs_summable_on A*
by *simp*

lemma *abs_summable_complex_of_real* [*simp*]: (λ*n*. *complex_of_real* (*f n*)) *abs_summable_on A* ⟷ *f abs_summable_on A*
by (*simp add*: *abs_summable_on_def complex_of_real_integrable_eq*)

lemma *abs_summable_on_comparison_test*:
assumes *g abs_summable_on A*
assumes ⋀*x*. *x* ∈ *A* ⟹ *norm* (*f x*) ≤ *norm* (*g x*)
shows *f abs_summable_on A*
using *assms Bochner_Integration.integrable_bound*[*of count_space A g f*]

unfolding *abs_summable_on_def* **by** (*auto simp*: *AE_count_space*)

lemma *abs_summable_on_comparison_test'*:
assumes *g abs_summable_on A*
assumes $\bigwedge x.\ x \in A \Longrightarrow \mathit{norm}\ (f\ x) \leq g\ x$
shows *f abs_summable_on A*
proof (*rule abs_summable_on_comparison_test*[*OF assms*(*1*), *of f*])
fix *x* **assume** $x \in A$
with *assms*(*2*) **have** $\mathit{norm}\ (f\ x) \leq g\ x$ **.**
also have $\ldots \leq \mathit{norm}\ (g\ x)$ **by** *simp*
finally show $\mathit{norm}\ (f\ x) \leq \mathit{norm}\ (g\ x)$ **.**
qed

lemma *abs_summable_on_cong* [*cong*]:
$(\bigwedge x.\ x \in A \Longrightarrow f\ x = g\ x) \Longrightarrow A = B \Longrightarrow (f$ *abs_summable_on* $A) \longleftrightarrow (g$ *abs_summable_on* $B)$
unfolding *abs_summable_on_def* **by** (*intro integrable_cong*) *auto*

lemma *abs_summable_on_cong_neutral*:
assumes $\bigwedge x.\ x \in A - B \Longrightarrow f\ x = 0$
assumes $\bigwedge x.\ x \in B - A \Longrightarrow g\ x = 0$
assumes $\bigwedge x.\ x \in A \cap B \Longrightarrow f\ x = g\ x$
shows *f abs_summable_on A* $\longleftrightarrow$ *g abs_summable_on B*
unfolding *abs_summable_on_altdef set_integrable_def* **using** *assms*
by (*intro Bochner_Integration.integrable_cong refl*)
(*auto simp*: *indicator_def split*: *if_splits*)

lemma *abs_summable_on_restrict'*:
fixes $f :: 'a \Rightarrow 'b :: \{$*banach*, *second_countable_topology*$\}$
assumes $A \subseteq B$
shows *f abs_summable_on A* $\longleftrightarrow$ $(\lambda x.\ \mathit{if}\ x \in A\ \mathit{then}\ f\ x\ \mathit{else}\ 0)$ *abs_summable_on B*
by (*subst abs_summable_on_restrict*[*OF assms*]) (*intro abs_summable_on_cong*, *auto*)

lemma *abs_summable_on_nat_iff*:
f abs_summable_on $(A :: \mathit{nat\ set}) \longleftrightarrow \mathit{summable}\ (\lambda n.\ \mathit{if}\ n \in A\ \mathit{then}\ \mathit{norm}\ (f\ n)\ \mathit{else}\ 0)$
proof −
have *f abs_summable_on A* $\longleftrightarrow \mathit{summable}\ (\lambda x.\ \mathit{norm}\ (\mathit{if}\ x \in A\ \mathit{then}\ f\ x\ \mathit{else}\ 0))$
by (*subst abs_summable_on_restrict'*[*of _ UNIV*])
(*simp_all add*: *abs_summable_on_def integrable_count_space_nat_iff*)
also have $(\lambda x.\ \mathit{norm}\ (\mathit{if}\ x \in A\ \mathit{then}\ f\ x\ \mathit{else}\ 0)) = (\lambda x.\ \mathit{if}\ x \in A\ \mathit{then}\ \mathit{norm}\ (f\ x)\ \mathit{else}\ 0)$
by *auto*
finally show *?thesis* **.**
qed

lemma *abs_summable_on_nat_iff'*:
f abs_summable_on (*UNIV* :: *nat set*) ⟷ *summable* (λ*n*. *norm* (*f n*))
by (*subst abs_summable_on_nat_iff*) *auto*

lemma *nat_abs_summable_on_comparison_test*:
fixes *f* :: *nat* ⇒ *'a* :: {*banach*, *second_countable_topology*}
assumes *g abs_summable_on I*
assumes ⋀*n*. ⟦*n*≥*N*; *n* ∈ *I*⟧ ⟹ *norm* (*f n*) ≤ *g n*
shows *f abs_summable_on I*
using *assms* **by** (*fastforce simp add*: *abs_summable_on_nat_iff intro*: *summable_comparison_test'*)

lemma *abs_summable_comparison_test_ev*:
assumes *g abs_summable_on I*
assumes *eventually* (λ*x*. *x* ∈ *I* ⟶ *norm* (*f x*) ≤ *g x*) *sequentially*
shows *f abs_summable_on I*
by (*metis* (*no_types*, *lifting*) *nat_abs_summable_on_comparison_test eventually_at_top_linorder assms*)

lemma *abs_summable_on_Cauchy*:
f abs_summable_on (*UNIV* :: *nat set*) ⟷ (∀ *e*>*0*. ∃ *N*. ∀ *m*≥*N*. ∀ *n*. (∑ *x* = *m*..<*n*. *norm* (*f x*)) < *e*)
by (*simp add*: *abs_summable_on_nat_iff' summable_Cauchy sum_nonneg*)

lemma *abs_summable_on_finite* [*simp*]: *finite A* ⟹ *f abs_summable_on A*
unfolding *abs_summable_on_def* **by** (*rule integrable_count_space*)

lemma *abs_summable_on_empty* [*simp*, *intro*]: *f abs_summable_on* {}
by *simp*

lemma *abs_summable_on_subset*:
assumes *f abs_summable_on B* **and** *A* ⊆ *B*
shows *f abs_summable_on A*
unfolding *abs_summable_on_altdef*
by (*rule set_integrable_subset*) (*insert assms*, *auto simp*: *abs_summable_on_altdef*)

lemma *abs_summable_on_union* [*intro*]:
assumes *f abs_summable_on A* **and** *f abs_summable_on B*
shows *f abs_summable_on* (*A* ∪ *B*)
using *assms* **unfolding** *abs_summable_on_altdef* **by** (*intro set_integrable_Un*) *auto*

lemma *abs_summable_on_insert_iff* [*simp*]:
f abs_summable_on insert x A ⟷ *f abs_summable_on A*
proof *safe*
assume *f abs_summable_on insert x A*
thus *f abs_summable_on A*
by (*rule abs_summable_on_subset*) *auto*
next
assume *f abs_summable_on A*

```
  from abs_summable_on_union[OF this, of {x}]
    show f abs_summable_on insert x A by simp
qed

lemma abs_summable_sum:
  assumes ⋀x. x ∈ A ⟹ f x abs_summable_on B
  shows   (λy. ∑x∈A. f x y) abs_summable_on B
 using assms unfolding abs_summable_on_def by (intro Bochner_Integration.integrable_sum)

lemma abs_summable_Re: f abs_summable_on A ⟹ (λx. Re (f x)) abs_summable_on
A
  by (simp add: abs_summable_on_def)

lemma abs_summable_Im: f abs_summable_on A ⟹ (λx. Im (f x)) abs_summable_on
A
  by (simp add: abs_summable_on_def)

lemma abs_summable_on_finite_diff:
  assumes f abs_summable_on A A ⊆ B finite (B − A)
  shows   f abs_summable_on B
proof −
  have f abs_summable_on (A ∪ (B − A))
    by (intro abs_summable_on_union assms abs_summable_on_finite)
  also from assms have A ∪ (B − A) = B by blast
  finally show ?thesis .
qed

lemma abs_summable_on_reindex_bij_betw:
  assumes bij_betw g A B
  shows   (λx. f (g x)) abs_summable_on A ⟷ f abs_summable_on B
proof −
  have ∗: count_space B = distr (count_space A) (count_space B) g
    by (rule distr_bij_count_space [symmetric]) fact
  show ?thesis unfolding abs_summable_on_def
    by (subst ∗, subst integrable_distr_eq[of _ _ count_space B])
       (insert assms, auto simp: bij_betw_def)
qed

lemma abs_summable_on_reindex:
  assumes (λx. f (g x)) abs_summable_on A
  shows   f abs_summable_on (g ‘ A)
proof −
  define g′ where g′ = inv_into A g
  from assms have (λx. f (g x)) abs_summable_on (g′ ‘ g ‘ A)
    by (rule abs_summable_on_subset) (auto simp: g′_def inv_into_into)
  also have ?this ⟷ (λx. f (g (g′ x))) abs_summable_on (g ‘ A) unfolding
g′_def
   by (intro abs_summable_on_reindex_bij_betw [symmetric] inj_on_imp_bij_betw
inj_on_inv_into) auto
```

```
  also have ... ⟷ f abs_summable_on (g ‘ A)
    by (intro abs_summable_on_cong refl) (auto simp: g'_def f_inv_into_f)
  finally show ?thesis .
qed

lemma abs_summable_on_reindex_iff:
  inj_on g A ⟹ (λx. f (g x)) abs_summable_on A ⟷ f abs_summable_on (g
‘ A)
  by (intro abs_summable_on_reindex_bij_betw inj_on_imp_bij_betw)

lemma abs_summable_on_Sigma_project2:
  fixes A :: 'a set and B :: 'a ⇒ 'b set
  assumes f abs_summable_on (Sigma A B) x ∈ A
  shows   (λy. f (x, y)) abs_summable_on (B x)
proof −
  from assms(2) have f abs_summable_on (Sigma {x} B)
    by (intro abs_summable_on_subset [OF assms(1)]) auto
  also have ?this ⟷ (λz. f (x, snd z)) abs_summable_on (Sigma {x} B)
    by (rule abs_summable_on_cong) auto
  finally have (λy. f (x, y)) abs_summable_on (snd ‘ Sigma {x} B)
    by (rule abs_summable_on_reindex)
  also have snd ‘ Sigma {x} B = B x
    using assms by (auto simp: image_iff)
  finally show ?thesis .
qed

lemma abs_summable_on_Times_swap:
  f abs_summable_on A × B ⟷ (λ(x,y). f (y,x)) abs_summable_on B × A
proof −
  have bij: bij_betw (λ(x,y). (y,x)) (B × A) (A × B)
    by (auto simp: bij_betw_def inj_on_def)
  show ?thesis
    by (subst abs_summable_on_reindex_bij_betw[OF bij, of f, symmetric])
       (simp_all add: case_prod_unfold)
qed

lemma abs_summable_on_0 [simp, intro]: (λ_. 0) abs_summable_on A
  by (simp add: abs_summable_on_def)

lemma abs_summable_on_uminus [intro]:
  f abs_summable_on A ⟹ (λx. −f x) abs_summable_on A
 unfolding abs_summable_on_def by (rule Bochner_Integration.integrable_minus)

lemma abs_summable_on_add [intro]:
  assumes f abs_summable_on A and g abs_summable_on A
  shows   (λx. f x + g x) abs_summable_on A
 using assms unfolding abs_summable_on_def by (rule Bochner_Integration.integrable_add)

lemma abs_summable_on_diff [intro]:
```

assumes *f abs_summable_on A* **and** *g abs_summable_on A*
shows (λx. *f x* − *g x*) *abs_summable_on A*
using *assms* **unfolding** *abs_summable_on_def* **by** (*rule Bochner_Integration.integrable_diff*)

lemma *abs_summable_on_scaleR_left* [*intro*]:
assumes *c* ≠ *0* ⟹ *f abs_summable_on A*
shows (λx. *f x* *$_R$ *c*) *abs_summable_on A*
using *assms* **unfolding** *abs_summable_on_def* **by** (*intro Bochner_Integration.integrable_scaleR_left*)

lemma *abs_summable_on_scaleR_right* [*intro*]:
assumes *c* ≠ *0* ⟹ *f abs_summable_on A*
shows (λx. *c* *$_R$ *f x*) *abs_summable_on A*
using *assms* **unfolding** *abs_summable_on_def* **by** (*intro Bochner_Integration.integrable_scaleR_righ*

lemma *abs_summable_on_cmult_right* [*intro*]:
fixes *f* :: *'a* ⇒ *'b* :: {*banach*, *real_normed_algebra*, *second_countable_topology*}
assumes *c* ≠ *0* ⟹ *f abs_summable_on A*
shows (λx. *c* ∗ *f x*) *abs_summable_on A*
using *assms* **unfolding** *abs_summable_on_def* **by** (*intro Bochner_Integration.integrable_mult_right*)

lemma *abs_summable_on_cmult_left* [*intro*]:
fixes *f* :: *'a* ⇒ *'b* :: {*banach*, *real_normed_algebra*, *second_countable_topology*}
assumes *c* ≠ *0* ⟹ *f abs_summable_on A*
shows (λx. *f x* ∗ *c*) *abs_summable_on A*
using *assms* **unfolding** *abs_summable_on_def* **by** (*intro Bochner_Integration.integrable_mult_left*)

lemma *abs_summable_on_prod_PiE*:
fixes *f* :: *'a* ⇒ *'b* ⇒ *'c* :: {*real_normed_field*,*banach*,*second_countable_topology*}
assumes *finite*: *finite A* **and** *countable*: ⋀x. *x* ∈ *A* ⟹ *countable* (*B x*)
assumes *summable*: ⋀x. *x* ∈ *A* ⟹ *f x abs_summable_on B x*
shows (λg. ∏ x∈*A*. *f x* (*g x*)) *abs_summable_on PiE A B*
proof −
define *B′* **where** *B′* = (λx. *if x* ∈ *A then B x else* {})
from *assms* **have** [*simp*]: *countable* (*B′ x*) **for** *x*
by (*auto simp*: *B′_def*)
then interpret *product_sigma_finite count_space* ∘ *B′*
unfolding *o_def* **by** (*intro product_sigma_finite.intro sigma_finite_measure_count_space_countable*
from *assms* **have** *integrable* (*PiM A* (*count_space* ∘ *B′*)) (λg. ∏ x∈*A*. *f x* (*g x*))
by (*intro product_integrable_prod*) (*auto simp*: *abs_summable_on_def B′_def*)
also have *PiM A* (*count_space* ∘ *B′*) = *count_space* (*PiE A B′*)
unfolding *o_def* **using** *finite* **by** (*intro count_space_PiM_finite*) *simp_all*
also have *PiE A B′* = *PiE A B* **by** (*intro PiE_cong*) (*simp_all add*: *B′_def*)
finally show *?thesis* **by** (*simp add*: *abs_summable_on_def*)
qed

lemma *not_summable_infsetsum_eq*:
¬*f abs_summable_on A* ⟹ *infsetsum f A* = *0*

by (*simp add*: *abs_summable_on_def infsetsum_def not_integrable_integral_eq*)

lemma *infsetsum_altdef*:
infsetsum f A = *set_lebesgue_integral* (*count_space UNIV*) *A f*
unfolding *set_lebesgue_integral_def*
by (*subst integral_restrict_space* [*symmetric*])
(*auto simp*: *restrict_count_space_subset infsetsum_def*)

lemma *infsetsum_altdef'*:
A ⊆ *B* ⟹ *infsetsum f A* = *set_lebesgue_integral* (*count_space B*) *A f*
unfolding *set_lebesgue_integral_def*
by (*subst integral_restrict_space* [*symmetric*])
(*auto simp*: *restrict_count_space_subset infsetsum_def*)

lemma *nn_integral_conv_infsetsum*:
assumes *f abs_summable_on A* ⋀*x*. *x* ∈ *A* ⟹ *f x* ≥ *0*
shows *nn_integral* (*count_space A*) *f* = *ennreal* (*infsetsum f A*)
using *assms* **unfolding** *infsetsum_def abs_summable_on_def*
by (*subst nn_integral_eq_integral*) *auto*

lemma *infsetsum_conv_nn_integral*:
assumes *nn_integral* (*count_space A*) *f* ≠ ∞ ⋀*x*. *x* ∈ *A* ⟹ *f x* ≥ *0*
shows *infsetsum f A* = *enn2real* (*nn_integral* (*count_space A*) *f*)
unfolding *infsetsum_def* **using** *assms*
by (*subst integral_eq_nn_integral*) *auto*

lemma *infsetsum_cong* [*cong*]:
(⋀*x*. *x* ∈ *A* ⟹ *f x* = *g x*) ⟹ *A* = *B* ⟹ *infsetsum f A* = *infsetsum g B*
unfolding *infsetsum_def* **by** (*intro Bochner_Integration.integral_cong*) *auto*

lemma *infsetsum_0* [*simp*]: *infsetsum* (λ_. *0*) *A* = *0*
by (*simp add*: *infsetsum_def*)

lemma *infsetsum_all_0*: (⋀*x*. *x* ∈ *A* ⟹ *f x* = *0*) ⟹ *infsetsum f A* = *0*
by *simp*

lemma *infsetsum_nonneg*: (⋀*x*. *x* ∈ *A* ⟹ *f x* ≥ (*0*::*real*)) ⟹ *infsetsum f A* ≥
0
unfolding *infsetsum_def* **by** (*rule Bochner_Integration.integral_nonneg*) *auto*

lemma *sum_infsetsum*:
assumes ⋀*x*. *x* ∈ *A* ⟹ *f x abs_summable_on B*
shows (∑ *x*∈*A*. ∑ *a y*∈*B*. *f x y*) = (∑ *a y*∈*B*. ∑ *x*∈*A*. *f x y*)
using *assms* **by** (*simp add*: *infsetsum_def abs_summable_on_def Bochner_Integration.integral_sum*)

lemma *Re_infsetsum*: *f abs_summable_on A* ⟹ *Re* (*infsetsum f A*) = (∑ *a x*∈*A*.
Re (*f x*))
by (*simp add*: *infsetsum_def abs_summable_on_def*)

lemma *Im_infsetsum*: *f abs_summable_on A* $\Longrightarrow$ *Im* (*infsetsum f A*) = ($\sum_a x{\in}A$. *Im* (*f x*))
by (*simp add*: *infsetsum_def abs_summable_on_def*)

lemma *infsetsum_of_real*:
shows *infsetsum* (λx. *of_real* (*f x*)
:: *'a* :: {*real_normed_algebra_1*,*banach*,*second_countable_topology*,*real_inner*}) *A* =
of_real (*infsetsum f A*)
unfolding *infsetsum_def*
by (*rule integral_bounded_linear'*[*OF bounded_linear_of_real bounded_linear_inner_left*[*of 1*]]) *auto*

lemma *infsetsum_finite* [*simp*]: *finite A* $\Longrightarrow$ *infsetsum f A* = ($\sum x{\in}A$. *f x*)
by (*simp add*: *infsetsum_def lebesgue_integral_count_space_finite*)

lemma *infsetsum_nat*:
assumes *f abs_summable_on A*
shows *infsetsum f A* = ($\sum$ *n*. *if n* $\in$ *A then f n else 0*)
proof −
from *assms* **have** *infsetsum f A* = ($\sum$ *n*. *indicator A n* $*_R$ *f n*)
unfolding *infsetsum_altdef abs_summable_on_altdef set_lebesgue_integral_def set_integrable_def*
by (*subst integral_count_space_nat*) *auto*
also have (λn. *indicator A n* $*_R$ *f n*) = (λn. *if n* $\in$ *A then f n else 0*)
by *auto*
finally show *?thesis* **.**
qed

lemma *infsetsum_nat'*:
assumes *f abs_summable_on UNIV*
shows *infsetsum f UNIV* = ($\sum$ *n*. *f n*)
using *assms* **by** (*subst infsetsum_nat*) *auto*

lemma *sums_infsetsum_nat*:
assumes *f abs_summable_on A*
shows (λn. *if n* $\in$ *A then f n else 0*) *sums infsetsum f A*
proof −
from *assms* **have** *summable* (λn. *if n* $\in$ *A then norm* (*f n*) *else 0*)
by (*simp add*: *abs_summable_on_nat_iff*)
also have (λn. *if n* $\in$ *A then norm* (*f n*) *else 0*) = (λn. *norm* (*if n* $\in$ *A then f n else 0*))
by *auto*
finally have *summable* (λn. *if n* $\in$ *A then f n else 0*)
by (*rule summable_norm_cancel*)
with *assms* **show** *?thesis*
by (*auto simp*: *sums_iff infsetsum_nat*)
qed

lemma *sums_infsetsum_nat'*:
assumes *f abs_summable_on UNIV*
shows *f sums infsetsum f UNIV*
using *sums_infsetsum_nat* [*OF assms*] **by** *simp*

lemma *infsetsum_Un_disjoint*:
assumes *f abs_summable_on A f abs_summable_on B A* ∩ *B* = {}
shows *infsetsum f* (*A* ∪ *B*) = *infsetsum f A* + *infsetsum f B*
using *assms* **unfolding** *infsetsum_altdef abs_summable_on_altdef*
by (*subst set_integral_Un*) *auto*

lemma *infsetsum_Diff*:
assumes *f abs_summable_on B A* ⊆ *B*
shows *infsetsum f* (*B* − *A*) = *infsetsum f B* − *infsetsum f A*
proof −
have *infsetsum f* ((*B* − *A*) ∪ *A*) = *infsetsum f* (*B* − *A*) + *infsetsum f A*
using *assms*(*2*) **by** (*intro infsetsum_Un_disjoint abs_summable_on_subset*[*OF assms*(*1*)]) *auto*
also from *assms*(*2*) **have** (*B* − *A*) ∪ *A* = *B*
by *auto*
ultimately show *?thesis*
by (*simp add*: *algebra_simps*)
qed

lemma *infsetsum_Un_Int*:
assumes *f abs_summable_on* (*A* ∪ *B*)
shows *infsetsum f* (*A* ∪ *B*) = *infsetsum f A* + *infsetsum f B* − *infsetsum f* (*A* ∩ *B*)
proof −
have *A* ∪ *B* = *A* ∪ (*B* − *A* ∩ *B*)
by *auto*
also have *infsetsum f* ... = *infsetsum f A* + *infsetsum f* (*B* − *A* ∩ *B*)
by (*intro infsetsum_Un_disjoint abs_summable_on_subset*[*OF assms*]) *auto*
also have *infsetsum f* (*B* − *A* ∩ *B*) = *infsetsum f B* − *infsetsum f* (*A* ∩ *B*)
by (*intro infsetsum_Diff abs_summable_on_subset*[*OF assms*]) *auto*
finally show *?thesis*
by (*simp add*: *algebra_simps*)
qed

lemma *infsetsum_reindex_bij_betw*:
assumes *bij_betw g A B*
shows *infsetsum* (λ*x*. *f* (*g x*)) *A* = *infsetsum f B*
proof −
have ∗: *count_space B* = *distr* (*count_space A*) (*count_space B*) *g*
by (*rule distr_bij_count_space* [*symmetric*]) *fact*
show *?thesis* **unfolding** *infsetsum_def*
by (*subst* ∗, *subst integral_distr*[*of* _ _ *count_space B*])
(*insert assms*, *auto simp*: *bij_betw_def*)
qed

theorem *infsetsum_reindex*:
assumes *inj_on g A*
shows *infsetsum f (g ‘ A) = infsetsum (λx. f (g x)) A*
by (*intro infsetsum_reindex_bij_betw* [*symmetric*] *inj_on_imp_bij_betw assms*)

lemma *infsetsum_cong_neutral*:
assumes $\bigwedge x.\ x \in A - B \Longrightarrow f\ x = 0$
assumes $\bigwedge x.\ x \in B - A \Longrightarrow g\ x = 0$
assumes $\bigwedge x.\ x \in A \cap B \Longrightarrow f\ x = g\ x$
shows *infsetsum f A = infsetsum g B*
unfolding *infsetsum_altdef set_lebesgue_integral_def* **using** *assms*
by (*intro Bochner_Integration.integral_cong refl*)
(*auto simp*: *indicator_def split*: *if_splits*)

lemma *infsetsum_mono_neutral*:
fixes *f g* :: *'a* ⇒ *real*
assumes *f abs_summable_on A* **and** *g abs_summable_on B*
assumes $\bigwedge x.\ x \in A \Longrightarrow f\ x \leq g\ x$
assumes $\bigwedge x.\ x \in A - B \Longrightarrow f\ x \leq 0$
assumes $\bigwedge x.\ x \in B - A \Longrightarrow g\ x \geq 0$
shows *infsetsum f A ≤ infsetsum g B*
using *assms* **unfolding** *infsetsum_altdef set_lebesgue_integral_def abs_summable_on_altdef set_integrable_def*
by (*intro Bochner_Integration.integral_mono*) (*auto simp*: *indicator_def*)

lemma *infsetsum_mono_neutral_left*:
fixes *f g* :: *'a* ⇒ *real*
assumes *f abs_summable_on A* **and** *g abs_summable_on B*
assumes $\bigwedge x.\ x \in A \Longrightarrow f\ x \leq g\ x$
assumes $A \subseteq B$
assumes $\bigwedge x.\ x \in B - A \Longrightarrow g\ x \geq 0$
shows *infsetsum f A ≤ infsetsum g B*
using ‹$A \subseteq B$› **by** (*intro infsetsum_mono_neutral assms*) *auto*

lemma *infsetsum_mono_neutral_right*:
fixes *f g* :: *'a* ⇒ *real*
assumes *f abs_summable_on A* **and** *g abs_summable_on B*
assumes $\bigwedge x.\ x \in A \Longrightarrow f\ x \leq g\ x$
assumes $B \subseteq A$
assumes $\bigwedge x.\ x \in A - B \Longrightarrow f\ x \leq 0$
shows *infsetsum f A ≤ infsetsum g B*
using ‹$B \subseteq A$› **by** (*intro infsetsum_mono_neutral assms*) *auto*

lemma *infsetsum_mono*:
fixes *f g* :: *'a* ⇒ *real*
assumes *f abs_summable_on A* **and** *g abs_summable_on A*
assumes $\bigwedge x.\ x \in A \Longrightarrow f\ x \leq g\ x$
shows *infsetsum f A ≤ infsetsum g A*

by (*intro infsetsum_mono_neutral assms*) *auto*

lemma *norm_infsetsum_bound*:
norm (*infsetsum f A*) ≤ *infsetsum* (λ*x*. *norm* (*f x*)) *A*
unfolding *abs_summable_on_def infsetsum_def*
by (*rule Bochner_Integration.integral_norm_bound*)

theorem *infsetsum_Sigma*:
fixes *A* :: *'a set* **and** *B* :: *'a* ⇒ *'b set*
assumes [*simp*]: *countable A* **and** ⋀*i*. *countable* (*B i*)
assumes *summable*: *f abs_summable_on* (*Sigma A B*)
shows *infsetsum f* (*Sigma A B*) = *infsetsum* (λ*x*. *infsetsum* (λ*y*. *f* (*x*, *y*)) (*B x*)) *A*
proof −
define *B'* **where** *B'* = (⋃*i*∈*A*. *B i*)
have [*simp*]: *countable B'*
unfolding *B'_def* **by** (*intro countable_UN assms*)
interpret *pair_sigma_finite count_space A count_space B'*
by (*intro pair_sigma_finite.intro sigma_finite_measure_count_space_countable*) *fact*+

have *integrable* (*count_space* (*A* × *B'*)) (λ*z*. *indicator* (*Sigma A B*) *z* ∗*R f z*)
using *summable*
by (*metis* (*mono_tags*, *lifting*) *abs_summable_on_altdef abs_summable_on_def integrable_cong integrable_mult_indicator set_integrable_def sets_UNIV*)
also have *?this* ⟷ *integrable* (*count_space A* ⨂*M count_space B'*) (λ(*x*, *y*). *indicator* (*B x*) *y* ∗*R f* (*x*, *y*))
by (*intro Bochner_Integration.integrable_cong*)
(*auto simp*: *pair_measure_countable indicator_def split*: *if_splits*)
finally have *integrable*: ... **.**

have *infsetsum* (λ*x*. *infsetsum* (λ*y*. *f* (*x*, *y*)) (*B x*)) *A* =
(∫ *x*. *infsetsum* (λ*y*. *f* (*x*, *y*)) (*B x*) ∂*count_space A*)
unfolding *infsetsum_def* **by** *simp*
also have ... = (∫ *x*. ∫ *y*. *indicator* (*B x*) *y* ∗*R f* (*x*, *y*) ∂*count_space B'* ∂*count_space A*)
proof (*rule Bochner_Integration.integral_cong* [*OF refl*])
show ⋀*x*. *x* ∈ *space* (*count_space A*) ⟹
(∑*a y*∈*B x*. *f* (*x*, *y*)) = *LINT y*|*count_space B'*. *indicat_real* (*B x*) *y* ∗*R f* (*x*, *y*)
using *infsetsum_altdef'*[*of _ B'*]
unfolding *set_lebesgue_integral_def B'_def*
by *auto*
qed
also have ... = (∫ (*x*,*y*). *indicator* (*B x*) *y* ∗*R f* (*x*, *y*) ∂(*count_space A* ⨂*M count_space B'*))
by (*subst integral_fst* [*OF integrable*]) *auto*
also have ... = (∫ *z*. *indicator* (*Sigma A B*) *z* ∗*R f z* ∂*count_space* (*A* × *B'*))
by (*intro Bochner_Integration.integral_cong*)

```
      (auto simp: pair_measure_countable indicator_def split: if_splits)
  also have ... = infsetsum f (Sigma A B)
    unfolding set_lebesgue_integral_def [symmetric]
    by (rule infsetsum_altdef' [symmetric]) (auto simp: B'_def)
  finally show ?thesis ..
qed

lemma infsetsum_Sigma':
  fixes A :: 'a set and B :: 'a ⇒ 'b set
  assumes [simp]: countable A and ⋀i. countable (B i)
  assumes summable: (λ(x,y). f x y) abs_summable_on (Sigma A B)
  shows   infsetsum (λx. infsetsum (λy. f x y) (B x)) A = infsetsum (λ(x,y). f x
y) (Sigma A B)
  using assms by (subst infsetsum_Sigma) auto

lemma infsetsum_Times:
  fixes A :: 'a set and B :: 'b set
  assumes [simp]: countable A and countable B
  assumes summable: f abs_summable_on (A × B)
  shows   infsetsum f (A × B) = infsetsum (λx. infsetsum (λy. f (x, y)) B) A
  using assms by (subst infsetsum_Sigma) auto

lemma infsetsum_Times':
  fixes A :: 'a set and B :: 'b set
  fixes f :: 'a ⇒ 'b ⇒ 'c :: {banach, second_countable_topology}
  assumes [simp]: countable A and [simp]: countable B
  assumes summable: (λ(x,y). f x y) abs_summable_on (A × B)
  shows   infsetsum (λx. infsetsum (λy. f x y) B) A = infsetsum (λ(x,y). f x y)
(A × B)
  using assms by (subst infsetsum_Times) auto

lemma infsetsum_swap:
  fixes A :: 'a set and B :: 'b set
  fixes f :: 'a ⇒ 'b ⇒ 'c :: {banach, second_countable_topology}
  assumes [simp]: countable A and [simp]: countable B
  assumes summable: (λ(x,y). f x y) abs_summable_on A × B
  shows   infsetsum (λx. infsetsum (λy. f x y) B) A = infsetsum (λy. infsetsum
(λx. f x y) A) B
proof −
  from summable have summable': (λ(x,y). f y x) abs_summable_on B × A
    by (subst abs_summable_on_Times_swap) auto
  have bij: bij_betw (λ(x, y). (y, x)) (B × A) (A × B)
    by (auto simp: bij_betw_def inj_on_def)
  have infsetsum (λx. infsetsum (λy. f x y) B) A = infsetsum (λ(x,y). f x y) (A
× B)
    using summable by (subst infsetsum_Times) auto
  also have ... = infsetsum (λ(x,y). f y x) (B × A)
    by (subst infsetsum_reindex_bij_betw[OF bij, of λ(x,y). f x y, symmetric])
       (simp_all add: case_prod_unfold)
```

```
  also have ... = infsetsum (λy. infsetsum (λx. f x y) A) B
    using summable' by (subst infsetsum_Times) auto
  finally show ?thesis .
qed

theorem abs_summable_on_Sigma_iff:
  assumes [simp]: countable A and ⋀x. x ∈ A ⟹ countable (B x)
  shows   f abs_summable_on Sigma A B ⟷
            (∀x∈A. (λy. f (x, y)) abs_summable_on B x) ∧
            ((λx. infsetsum (λy. norm (f (x, y))) (B x)) abs_summable_on A)
proof safe
  define B' where B' = (⋃x∈A. B x)
  have [simp]: countable B'
    unfolding B'_def using assms by auto
  interpret pair_sigma_finite count_space A count_space B'
   by (intro pair_sigma_finite.intro sigma_finite_measure_count_space_countable)
fact+
  {
    assume ∗: f abs_summable_on Sigma A B
    thus (λy. f (x, y)) abs_summable_on B x if x ∈ A for x
      using that by (rule abs_summable_on_Sigma_project2)

    have set_integrable (count_space (A × B')) (Sigma A B) (λz. norm (f z))
      using abs_summable_on_normI[OF ∗]
      by (subst abs_summable_on_altdef' [symmetric]) (auto simp: B'_def)
    also have count_space (A × B') = count_space A ⨂M count_space B'
      by (simp add: pair_measure_countable)
    finally have integrable (count_space A)
                   (λx. lebesgue_integral (count_space B')
                     (λy. indicator (Sigma A B) (x, y) ∗R norm (f (x, y))))
      unfolding set_integrable_def by (rule integrable_fst')
    also have ?this ⟷ integrable (count_space A)
                   (λx. lebesgue_integral (count_space B')
                     (λy. indicator (B x) y ∗R norm (f (x, y))))
      by (intro integrable_cong refl) (simp_all add: indicator_def)
    also have ... ⟷ integrable (count_space A) (λx. infsetsum (λy. norm (f (x,
y))) (B x))
      unfolding set_lebesgue_integral_def [symmetric]
        by (intro integrable_cong refl infsetsum_altdef' [symmetric]) (auto simp:
B'_def)
   also have ... ⟷ (λx. infsetsum (λy. norm (f (x, y))) (B x)) abs_summable_on
A
      by (simp add: abs_summable_on_def)
    finally show ... .
  }
  {
    assume ∗: ∀x∈A. (λy. f (x, y)) abs_summable_on B x
    assume (λx. ∑a y∈B x. norm (f (x, y))) abs_summable_on A
   also have ?this ⟷ (λx. ∫ y∈B x. norm (f (x, y)) ∂count_space B') abs_summable_on
```

```
A
    by (intro abs_summable_on_cong refl infsetsum_altdef') (auto simp: B'_def)
   also have ... ⟷ (λx. ∫ y. indicator (Sigma A B) (x, y) *R norm (f (x, y)))
∂count_space B')
                    abs_summable_on A (is _ ⟷ ?h abs_summable_on _)
     unfolding set_lebesgue_integral_def
     by (intro abs_summable_on_cong) (auto simp: indicator_def)
   also have ... ⟷ integrable (count_space A) ?h
     by (simp add: abs_summable_on_def)
   finally have **: ... .

   have integrable (count_space A ⨂M count_space B') (λz. indicator (Sigma A
B) z *R f z)
    proof (rule Fubini_integrable, goal_cases)
     case 3
     {
      fix x assume x: x ∈ A
      with * have (λy. f (x, y)) abs_summable_on B x
        by blast
      also have ?this ⟷ integrable (count_space B')
                  (λy. indicator (B x) y *R f (x, y))
        unfolding set_integrable_def [symmetric]
       using x by (intro abs_summable_on_altdef') (auto simp: B'_def)
      also have (λy. indicator (B x) y *R f (x, y)) =
                 (λy. indicator (Sigma A B) (x, y) *R f (x, y))
        using x by (auto simp: indicator_def)
      finally have integrable (count_space B')
                    (λy. indicator (Sigma A B) (x, y) *R f (x, y)) .
     }
     thus ?case by (auto simp: AE_count_space)
    qed (insert **, auto simp: pair_measure_countable)
   moreover have count_space A ⨂M count_space B' = count_space (A × B')
     by (simp add: pair_measure_countable)
   moreover have set_integrable (count_space (A × B')) (Sigma A B) f ⟷
             f abs_summable_on Sigma A B
     by (rule abs_summable_on_altdef' [symmetric]) (auto simp: B'_def)
   ultimately show f abs_summable_on Sigma A B
     by (simp add: set_integrable_def)
  }
qed

lemma abs_summable_on_Sigma_project1:
  assumes (λ(x,y). f x y) abs_summable_on Sigma A B
  assumes [simp]: countable A and ⋀x. x ∈ A ⟹ countable (B x)
  shows   (λx. infsetsum (λy. norm (f x y)) (B x)) abs_summable_on A
  using assms by (subst (asm) abs_summable_on_Sigma_iff) auto

lemma abs_summable_on_Sigma_project1':
  assumes (λ(x,y). f x y) abs_summable_on Sigma A B
```

assumes [*simp*]: *countable A* **and** $\bigwedge x.\ x \in A \Longrightarrow$ *countable* (*B x*)
shows (λ*x*. *infsetsum* (λ*y*. *f x y*) (*B x*)) *abs_summable_on A*
by (*intro abs_summable_on_comparison_test'* [*OF abs_summable_on_Sigma_project1* [*OF assms*]]
norm_infsetsum_bound)

theorem *infsetsum_prod_PiE*:
fixes *f* :: *'a* ⇒ *'b* ⇒ *'c* :: {*real_normed_field*,*banach*,*second_countable_topology*}
assumes *finite*: *finite A* **and** *countable*: $\bigwedge x.\ x \in A \Longrightarrow$ *countable* (*B x*)
assumes *summable*: $\bigwedge x.\ x \in A \Longrightarrow$ *f x abs_summable_on B x*
shows *infsetsum* (λ*g*. $\prod x{\in}A.$ *f x* (*g x*)) (*PiE A B*) = ($\prod x{\in}A.$ *infsetsum* (*f x*) (*B x*))
proof −
define *B'* **where** *B'* = (λ*x*. *if x* ∈ *A then B x else* {})
from *assms* **have** [*simp*]: *countable* (*B' x*) **for** *x*
by (*auto simp*: *B'_def*)
then interpret *product_sigma_finite count_space* ∘ *B'*
unfolding *o_def* **by** (*intro product_sigma_finite.intro sigma_finite_measure_count_space_countable*)
have *infsetsum* (λ*g*. $\prod x{\in}A.$ *f x* (*g x*)) (*PiE A B*) =
($\int g.$ ($\prod x{\in}A.$ *f x* (*g x*)) ∂*count_space* (*PiE A B*))
by (*simp add*: *infsetsum_def*)
also have *PiE A B* = *PiE A B'*
by (*intro PiE_cong*) (*simp_all add*: *B'_def*)
hence *count_space* (*PiE A B*) = *count_space* (*PiE A B'*)
by *simp*
also have ... = *PiM A* (*count_space* ∘ *B'*)
unfolding *o_def* **using** *finite* **by** (*intro count_space_PiM_finite* [*symmetric*]) *simp_all*
also have ($\int g.$ ($\prod x{\in}A.$ *f x* (*g x*)) ∂...) = ($\prod x{\in}A.$ *infsetsum* (*f x*) (*B' x*))
by (*subst product_integral_prod*)
(*insert summable finite*, *simp_all add*: *infsetsum_def B'_def abs_summable_on_def*)
also have ... = ($\prod x{\in}A.$ *infsetsum* (*f x*) (*B x*))
by (*intro prod.cong refl*) (*simp_all add*: *B'_def*)
finally show *?thesis* **.**
qed

lemma *infsetsum_uminus*: *infsetsum* (λ*x*. −*f x*) *A* = −*infsetsum f A*
unfolding *infsetsum_def abs_summable_on_def*
by (*rule Bochner_Integration.integral_minus*)

lemma *infsetsum_add*:
assumes *f abs_summable_on A* **and** *g abs_summable_on A*
shows *infsetsum* (λ*x*. *f x* + *g x*) *A* = *infsetsum f A* + *infsetsum g A*
using *assms* **unfolding** *infsetsum_def abs_summable_on_def*
by (*rule Bochner_Integration.integral_add*)

lemma *infsetsum_diff*:
assumes *f abs_summable_on A* **and** *g abs_summable_on A*
shows *infsetsum* (λ*x*. *f x* − *g x*) *A* = *infsetsum f A* − *infsetsum g A*

using *assms* **unfolding** *infsetsum_def abs_summable_on_def*
by (*rule Bochner_Integration.integral_diff*)

lemma *infsetsum_scaleR_left*:
assumes $c \neq 0 \Longrightarrow f$ *abs_summable_on* A
shows *infsetsum* $(\lambda x.\ f\ x *_R c)\ A =$ *infsetsum* $f\ A *_R c$
using *assms* **unfolding** *infsetsum_def abs_summable_on_def*
by (*rule Bochner_Integration.integral_scaleR_left*)

lemma *infsetsum_scaleR_right*:
infsetsum $(\lambda x.\ c *_R f\ x)\ A = c *_R$ *infsetsum* $f\ A$
unfolding *infsetsum_def abs_summable_on_def*
by (*subst Bochner_Integration.integral_scaleR_right*) *auto*

lemma *infsetsum_cmult_left*:
fixes $f :: 'a \Rightarrow 'b :: \{$*banach*, *real_normed_algebra*, *second_countable_topology*$\}$
assumes $c \neq 0 \Longrightarrow f$ *abs_summable_on* A
shows *infsetsum* $(\lambda x.\ f\ x * c)\ A =$ *infsetsum* $f\ A * c$
using *assms* **unfolding** *infsetsum_def abs_summable_on_def*
by (*rule Bochner_Integration.integral_mult_left*)

lemma *infsetsum_cmult_right*:
fixes $f :: 'a \Rightarrow 'b :: \{$*banach*, *real_normed_algebra*, *second_countable_topology*$\}$
assumes $c \neq 0 \Longrightarrow f$ *abs_summable_on* A
shows *infsetsum* $(\lambda x.\ c * f\ x)\ A = c *$ *infsetsum* $f\ A$
using *assms* **unfolding** *infsetsum_def abs_summable_on_def*
by (*rule Bochner_Integration.integral_mult_right*)

lemma *infsetsum_cdiv*:
fixes $f :: 'a \Rightarrow 'b :: \{$*banach*, *real_normed_field*, *second_countable_topology*$\}$
assumes $c \neq 0 \Longrightarrow f$ *abs_summable_on* A
shows *infsetsum* $(\lambda x.\ f\ x\ /\ c)\ A =$ *infsetsum* $f\ A\ /\ c$
using *assms* **unfolding** *infsetsum_def abs_summable_on_def* **by** *auto*

lemma
fixes $f :: 'a \Rightarrow 'c :: \{$*banach*, *real_normed_field*, *second_countable_topology*$\}$
assumes [*simp*]: *countable* A **and** [*simp*]: *countable* B
assumes f *abs_summable_on* A **and** g *abs_summable_on* B
shows *abs_summable_on_product*: $(\lambda(x,y).\ f\ x * g\ y)$ *abs_summable_on* $A \times B$
and *infsetsum_product*: *infsetsum* $(\lambda(x,y).\ f\ x * g\ y)\ (A \times B) =$ *infsetsum* $f\ A *$ *infsetsum* $g\ B$
proof −
from *assms* **show** $(\lambda(x,y).\ f\ x * g\ y)$ *abs_summable_on* $A \times B$
by (*subst abs_summable_on_Sigma_iff*)
(*auto intro*!: *abs_summable_on_cmult_right simp*: *norm_mult infsetsum_cmult_right*)

with *assms* **show** *infsetsum* (λ(*x*,*y*). *f x* ∗ *g y*) (*A* × *B*) = *infsetsum f A* ∗ *infsetsum g B*
by (*subst infsetsum_Sigma*)
(*auto simp*: *infsetsum_cmult_left infsetsum_cmult_right*)
qed

lemma *abs_summable_finite_sumsI*:
assumes ⋀*F*. *finite F* ⟹ *F*⊆*S* ⟹ *sum* (λ*x*. *norm* (*f x*)) *F* ≤ *B*
shows *f abs_summable_on S*
proof −
have *main*: *f abs_summable_on S* ∧ *infsetsum* (λ*x*. *norm* (*f x*)) *S* ≤ *B* **if** ‹*B* ≥ *0*› **and** ‹*S* ≠ {}›
proof −
define *M normf* **where** *M* = *count_space S* **and** *normf x* = *ennreal* (*norm* (*f x*)) **for** *x*
have *sum normf F* ≤ *ennreal B*
if *finite F* **and** *F* ⊆ *S* **and**
⋀*F*. *finite F* ⟹ *F* ⊆ *S* ⟹ (∑ *i*∈*F*. *ennreal* (*norm* (*f i*))) ≤ *ennreal B* **and**
ennreal 0 ≤ *ennreal B* **for** *F*
using *that* **unfolding** *normf_def*[*symmetric*] **by** *simp*
hence *normf_B*: *finite F* ⟹ *F*⊆*S* ⟹ *sum normf F* ≤ *ennreal B* **for** *F*
using *assms*[*THEN ennreal_leI*]
by *auto*
have *integral*^{S} *M g* ≤ *B* **if** *simple_function M g* **and** *g* ≤ *normf* **for** *g*
proof −
define *gS* **where** *gS* = *g* ‘ *S*
have *finite gS*
using *that* **unfolding** *gS_def M_def simple_function_count_space* **by** *simp*
have *gS* ≠ {} **unfolding** *gS_def* **using** ‹*S* ≠ {}› **by** *auto*
define *part* **where** *part r* = *g* −‘ {*r*} ∩ *S* **for** *r*
have *r_finite*: *r* < ∞ **if** *r* : *gS* **for** *r*
using ‹*g* ≤ *normf*› *that* **unfolding** *gS_def le_fun_def normf_def* **apply** *auto*
using *ennreal_less_top neq_top_trans top.not_eq_extremum* **by** *blast*
define *B*′ **where** *B*′ *r* = (*SUP F*∈{*F*. *finite F* ∧ *F*⊆*part r*}. *sum normf F*) **for** *r*
have *B*′*fin*: *B*′ *r* < ∞ **for** *r*
proof −
have *B*′ *r* ≤ (*SUP F*∈{*F*. *finite F* ∧ *F*⊆*part r*}. *sum normf F*)
unfolding *B*′*_def*
by (*metis* (*mono_tags*, *lifting*) *SUP_least SUP_upper*)
also have ... ≤ *B*
using *normf_B* **unfolding** *part_def*
by (*metis* (*no_types*, *lifting*) *Int_subset_iff SUP_least mem_Collect_eq*)
also have ... < ∞
by *simp*
finally show *?thesis* **by** *simp*
qed

```
    have sumB': sum B' gS ≤ ennreal B + ε if ε>0 for ε
    proof −
      define N εN where N = card gS and εN = ε / N
      have N > 0
        unfolding N_def using ‹gS≠{}› ‹finite gS›
        by (simp add: card_gt_0_iff)
      from εN_def that have εN > 0
       by (simp add: ennreal_of_nat_eq_real_of_nat ennreal_zero_less_divide)
      have c1: ∃ y. B' r ≤ sum normf y + εN ∧ finite y ∧ y ⊆ part r
        if B' r = 0 for r
        using that by auto
      have c2: ∃ y. B' r ≤ sum normf y + εN ∧ finite y ∧ y ⊆ part r if B' r ≠
0 for r
      proof−
        have B' r − εN < B' r
          using B'fin ‹0 < εN› ennreal_between that by fastforce
        have B' r − εN < Sup (sum normf ‘ {F. finite F ∧ F ⊆ part r}) ⟹
            ∃ F. B' r − εN ≤ sum normf F ∧ finite F ∧ F ⊆ part r
         by (metis (no_types, lifting) leD le_cases less_SUP_iff mem_Collect_eq)
        hence B' r − εN < B' r ⟹ ∃ F. B' r − εN ≤ sum normf F ∧ finite F
∧ F ⊆ part r
          by (subst (asm) (2) B'_def)
         then obtain F where B' r − εN ≤ sum normf F and finite F and F
⊆ part r
          using ‹B' r − εN < B' r› by auto
        thus ∃ F. B' r ≤ sum normf F + εN ∧ finite F ∧ F ⊆ part r
          by (metis add.commute ennreal_minus_le_iff)
      qed
      have ∀ x. ∃ y. B' x ≤ sum normf y + εN ∧
        finite y ∧ y ⊆ part x
        using c1 c2
        by blast
      hence ∃ F. ∀ x. B' x ≤ sum normf (F x) + εN ∧ finite (F x) ∧ F x ⊆ part
x
        by metis
      then obtain F where F: sum normf (F r) + εN ≥ B' r and Ffin: finite
(F r) and Fpartr: F r ⊆ part r for r
        using atomize_elim by auto
      have w1: finite gS
        by (simp add: ‹finite gS›)
      have w2: ∀ i∈gS. finite (F i)
        by (simp add: Ffin)
      have False
        if ⋀r. F r ⊆ g −‘ {r} ∧ F r ⊆ S
          and i ∈ gS and j ∈ gS and i ≠ j and x ∈ F i and x ∈ F j
        for i j x
        by (metis subsetD that(1) that(4) that(5) that(6) vimage_singleton_eq)

      hence w3: ∀ i∈gS. ∀ j∈gS. i ≠ j ⟶ F i ∩ F j = {}
```

```
      using Fpartr[unfolded part_def] by auto
    have w4: sum normf (⋃ (F ‘ gS)) + ε = sum normf (⋃ (F ‘ gS)) + ε
      by simp
    have sum B' gS ≤ (∑ r∈gS. sum normf (F r) + εN)
      using F by (simp add: sum_mono)
    also have ... = (∑ r∈gS. sum normf (F r)) + (∑ r∈gS. εN)
      by (simp add: sum.distrib)
    also have ... = (∑ r∈gS. sum normf (F r)) + (card gS * εN)
      by auto
    also have ... = (∑ r∈gS. sum normf (F r)) + ε
      unfolding εN_def N_def[symmetric] using ‹N>0›
   by (simp add: ennreal_times_divide mult.commute mult_divide_eq_ennreal)
    also have ... = sum normf (⋃ (F ‘ gS)) + ε
      using w1 w2 w3 w4
      by (subst sum.UNION_disjoint[symmetric])
    also have ... ≤ B + ε
        using ‹finite gS› normf_B add_right_mono Ffin Fpartr unfolding
part_def
      by (simp add: ‹gS ≠ {}› cSUP_least)
    finally show ?thesis
      by auto
  qed
  hence sumB': sum B' gS ≤ B
    using ennreal_le_epsilon ennreal_less_zero_iff by blast
  have ∀ r. ∃ y. r ∈ gS ⟶ B' r = ennreal y
    using B'fin less_top_ennreal by auto
  hence ∃ B''. ∀ r. r ∈ gS ⟶ B' r = ennreal (B'' r)
    by (rule_tac choice)
  then obtain B'' where B'': B' r = ennreal (B'' r) if r ∈ gS for r
    by atomize_elim
  have cases[case_names zero finite infinite]: P if r=0 ⟹ P and finite (part
r) ⟹ P
    and infinite (part r) ⟹ r≠0 ⟹ P for P r
    using that by metis
  have emeasure_B': r * emeasure M (part r) ≤ B' r if r : gS for r
  proof (cases rule:cases[of r])
    case zero
    thus ?thesis by simp
  next
    case finite
    have s1: sum g F ≤ sum normf F
      if F ∈ {F. finite F ∧ F ⊆ part r}
      for F
      using ‹g ≤ normf›
      by (simp add: le_fun_def sum_mono)

    have r * of_nat (card (part r)) = r * (∑ x∈part r. 1) by simp
    also have ... = (∑ x∈part r. r)
      using mult.commute by auto
```

```
    also have ... = (∑ x∈part r. g x)
      unfolding part_def by auto
    also have ... ≤ (SUP F∈{F. finite F ∧ F⊆part r}. sum g F)
      using finite
      by (simp add: Sup_upper)
    also have ... ≤ B' r
      unfolding B'_def
      using s1 SUP_subset_mono by blast
    finally have r * of_nat (card (part r)) ≤ B' r by assumption
    thus ?thesis
      unfolding M_def
      using part_def finite by auto
  next
    case infinite
    from r_finite[OF ‹r : gS›] obtain r' where r': r = ennreal r'
      using ennreal_cases by auto
    with infinite have r' > 0
      using ennreal_less_zero_iff not_gr_zero by blast
    obtain N::nat where N:N > B / r' and real N > 0 apply atomize_elim
      using reals_Archimedean2
      by (metis less_trans linorder_neqE_linordered_idom)
    obtain F where finite F and card F = N and F ⊆ part r
      using infinite(1) infinite_arbitrarily_large by blast
    from ‹F ⊆ part r› have F ⊆ S unfolding part_def by simp
    have B < r * N
      unfolding r' ennreal_of_nat_eq_real_of_nat
      using N ‹0 < r'› ‹B ≥ 0› r'
          by (metis enn2real_ennreal enn2real_less_iff ennreal_less_top en-
nreal_mult' less_le mult_less_cancel_left_pos nonzero_mult_div_cancel_left times_divide_eq_right)
    also have r * N = (∑ x∈F. r)
      using ‹card F = N› by (simp add: mult.commute)
    also have (∑ x∈F. r) = (∑ x∈F. g x)
      using ‹F ⊆ part r›  part_def sum.cong subsetD by fastforce
    also have (∑ x∈F. g x) ≤ (∑ x∈F. ennreal (norm (f x)))
      by (metis (mono_tags, lifting) ‹g ≤ normf› ‹normf ≡ λx. ennreal (norm
(f x))› le_fun_def
          sum_mono)
    also have (∑ x∈F. ennreal (norm (f x))) ≤ B
      using ‹F ⊆ S› ‹finite F› ‹normf ≡ λx. ennreal (norm (f x))› normf_B
by blast
    finally have B < B by auto
    thus ?thesis by simp
  qed

  have integral^S M g = (∑ r ∈ gS. r * emeasure M (part r))
    unfolding simple_integral_def gS_def M_def part_def by simp
  also have ... ≤ (∑ r ∈ gS. B' r)
    by (simp add: emeasure_B' sum_mono)
  also have ... ≤ B
```

```
      using sumB' by blast
    finally show ?thesis by assumption
  qed
  hence int_leq_B: integral^N M normf ≤ B
  unfolding nn_integral_def by (metis (no_types, lifting) SUP_least mem_Collect_eq)
  hence integral^N M normf < ∞
    using le_less_trans by fastforce
  hence integrable M f
    unfolding M_def normf_def by (rule integrableI_bounded[rotated], simp)
  hence v1: f abs_summable_on S
    unfolding abs_summable_on_def M_def by simp

  have (λx. norm (f x)) abs_summable_on S
   using v1 Infinite_Set_Sum.abs_summable_on_norm_iff[where A = S and
f = f]
    by auto
  moreover have 0 ≤ norm (f x)
    if x ∈ S for x
    by simp
  moreover have (∫⁺ x. ennreal (norm (f x)) ∂count_space S) ≤ ennreal B
    using M_def ‹normf ≡ λx. ennreal (norm (f x))› int_leq_B by auto
  ultimately have ennreal (∑ a x∈S. norm (f x)) ≤ ennreal B
    by (simp add: nn_integral_conv_infsetsum)
  hence v2: (∑ a x∈S. norm (f x)) ≤ B
    by (subst ennreal_le_iff[symmetric], simp add: assms ‹B ≥ 0›)
  show ?thesis
    using v1 v2 by auto
 qed
 then show f abs_summable_on S
  by (metis abs_summable_on_finite assms empty_subsetI finite.emptyI sum_clauses(1))
qed


lemma infsetsum_nonneg_is_SUPREMUM_ennreal:
 fixes f :: 'a ⇒ real
 assumes summable: f abs_summable_on A
   and fnn: ⋀x. x∈A ⟹ f x ≥ 0
 shows ennreal (infsetsum f A) = (SUP F∈{F. finite F ∧ F ⊆ A}. (ennreal (sum
f F)))
proof −
 have sum_F_A: sum f F ≤ infsetsum f A
   if F ∈ {F. finite F ∧ F ⊆ A}
   for F
 proof−
   from that have finite F and F ⊆ A by auto
   from ‹finite F› have sum f F = infsetsum f F by auto
   also have ... ≤ infsetsum f A
   proof (rule infsetsum_mono_neutral_left)
     show f abs_summable_on F
```

```
      by (simp add: ‹finite F›)
    show f abs_summable_on A
      by (simp add: local.summable)
    show f x ≤ f x
      if x ∈ F
      for x :: 'a
      by simp
    show F ⊆ A
      by (simp add: ‹F ⊆ A›)
    show 0 ≤ f x
      if x ∈ A − F
      for x :: 'a
      using that fnn by auto
  qed
  finally show ?thesis by assumption
 qed
 hence geq: ennreal (infsetsum f A) ≥ (SUP F∈{G. finite G ∧ G ⊆ A}. (ennreal
(sum f F)))
  by (meson SUP_least ennreal_leI)

 define fe where fe x = ennreal (f x) for x

 have sum_f_int: infsetsum f A = ∫⁺ x. fe x ∂(count_space A)
  unfolding infsetsum_def fe_def
 proof (rule nn_integral_eq_integral [symmetric])
  show integrable (count_space A) f
   using abs_summable_on_def local.summable by blast
  show AE x in count_space A. 0 ≤ f x
   using fnn by auto
 qed
 also have ... = (SUP g ∈ {g. finite (g‘A) ∧ g ≤ fe}. integral^S (count_space A)
g)
  unfolding nn_integral_def simple_function_count_space by simp
 also have ... ≤ (SUP F∈{F. finite F ∧ F ⊆ A}. (ennreal (sum f F)))
 proof (rule Sup_least)
  fix x assume x ∈ integral^S (count_space A) ‘ {g. finite (g ‘ A) ∧ g ≤ fe}
  then obtain g where xg: x = integral^S (count_space A) g and fin_gA: finite
(g‘A)
   and g_fe: g ≤ fe by auto
  define F where F = {z:A. g z ≠ 0}
  hence F ⊆ A by simp

  have fin: finite {z:A. g z = t} if t ≠ 0 for t
  proof (rule ccontr)
   assume inf: infinite {z:A. g z = t}
   hence tgA: t ∈ g ‘ A
    by (metis (mono_tags, lifting) image_eqI not_finite_existsD)
   have x = (∑ x ∈ g ‘ A. x ∗ emeasure (count_space A) (g −‘ {x} ∩ A))
    unfolding xg simple_integral_def space_count_space by simp
```

```
    also have ... ≥ (∑ x ∈ {t}. x * emeasure (count_space A) (g -` {x} ∩ A))
(is _ ≥ ...)
    proof (rule sum_mono2)
      show finite (g ` A)
        by (simp add: fin_gA)
      show {t} ⊆ g ` A
        by (simp add: tgA)
      show 0 ≤ b * emeasure (count_space A) (g -` {b} ∩ A)
        if b ∈ g ` A - {t}
        for b :: ennreal
        using that
        by simp
    qed
    also have ... = t * emeasure (count_space A) (g -` {t} ∩ A)
      by auto
    also have ... = t * ∞
    proof (subst emeasure_count_space_infinite)
      show g -` {t} ∩ A ⊆ A
        by simp
      have {a ∈ A. g a = t} = {a ∈ g -` {t}. a ∈ A}
        by auto
      thus infinite (g -` {t} ∩ A)
        by (metis (full_types) Int_def inf)
      show t * ∞ = t * ∞
        by simp
    qed
    also have ... = ∞ using ‹t ≠ 0›
      by (simp add: ennreal_mult_eq_top_iff)
    finally have x_inf: x = ∞
      using neq_top_trans by auto
    have x = integral^S (count_space A) g by (fact xg)
    also have ... = integral^N (count_space A) g
      by (simp add: fin_gA nn_integral_eq_simple_integral)
    also have ... ≤ integral^N (count_space A) fe
      using g_fe
      by (simp add: le_funD nn_integral_mono)
    also have ... < ∞
      by (metis sum_f_int ennreal_less_top infinity_ennreal_def)
    finally have x_fin: x < ∞ by simp
    from x_inf x_fin show False by simp
  qed
  have F: F = (⋃ t∈g`A-{0}. {z∈A. g z = t})
    unfolding F_def by auto
  hence finite F
    unfolding F using fin_gA fin by auto
  have x = integral^N (count_space A) g
    unfolding xg
    by (simp add: fin_gA nn_integral_eq_simple_integral)
  also have ... = set_nn_integral (count_space UNIV) A g
```

```
    by (simp add: nn_integral_restrict_space[symmetric] restrict_count_space)
  also have ... = set_nn_integral (count_space UNIV) F g
  proof -
    have ∀a. g a * (if a ∈ {a ∈ A. g a ≠ 0} then 1 else 0) = g a * (if a ∈ A
then 1 else 0)
      by auto
    hence (∫⁺ a. g a * (if a ∈ A then 1 else 0) ∂count_space UNIV)
        = (∫⁺ a. g a * (if a ∈ {a ∈ A. g a ≠ 0} then 1 else 0) ∂count_space
UNIV)
      by presburger
    thus ?thesis unfolding F_def indicator_def
      using mult.right_neutral mult_zero_right nn_integral_cong
      by (simp add: of_bool_def)
  qed
  also have ... = integral^N (count_space F) g
    by (simp add: nn_integral_restrict_space[symmetric] restrict_count_space)
  also have ... = sum g F
    using ‹finite F› by (rule nn_integral_count_space_finite)
  also have sum g F ≤ sum fe F
    using g_fe unfolding le_fun_def
    by (simp add: sum_mono)
  also have ... ≤ (SUP F ∈ {G. finite G ∧ G ⊆ A}. (sum fe F))
    using ‹finite F› ‹F⊆A›
    by (simp add: SUP_upper)
  also have ... = (SUP F ∈ {F. finite F ∧ F ⊆ A}. (ennreal (sum f F)))
  proof (rule SUP_cong [OF refl])
    have finite x ⟹ x ⊆ A ⟹ (∑x∈x. ennreal (f x)) = ennreal (sum f x)
      for x
      by (metis fnn subsetCE sum_ennreal)
    thus sum fe x = ennreal (sum f x)
      if x ∈ {G. finite G ∧ G ⊆ A}
      for x :: 'a set
      using that unfolding fe_def by auto
  qed
  finally show x ≤ ... by simp
 qed
 finally have leq: ennreal (infsetsum f A) ≤ (SUP F∈{F. finite F ∧ F ⊆ A}.
(ennreal (sum f F)))
  by assumption
 from leq geq show ?thesis by simp
qed

lemma infsetsum_nonneg_is_SUPREMUM_ereal:
 fixes f :: 'a ⇒ real
 assumes summable: f abs_summable_on A
  and fnn: ⋀x. x∈A ⟹ f x ≥ 0
 shows ereal (infsetsum f A) = (SUP F∈{F. finite F ∧ F ⊆ A}. (ereal (sum f
F)))
proof -
```

```
  have ereal (infsetsum f A) = enn2ereal (ennreal (infsetsum f A))
    by (simp add: fnn infsetsum_nonneg)
  also have ... = enn2ereal (SUP F∈{F. finite F ∧ F ⊆ A}. ennreal (sum f F))
    apply (subst infsetsum_nonneg_is_SUPREMUM_ennreal)
    using fnn by (auto simp add: local.summable)
  also have ... = (SUP F∈{F. finite F ∧ F ⊆ A}. (ereal (sum f F)))
  proof (simp add: image_def Sup_ennreal.rep_eq)
    have 0 ≤ Sup {y. ∃x. (∃xa. finite xa ∧ xa ⊆ A ∧ x = ennreal (sum f xa)) ∧
                    y = enn2ereal x}
      by (metis (mono_tags, lifting) Sup_upper empty_subsetI ennreal_0 finite.emptyI
          mem_Collect_eq sum.empty zero_ennreal.rep_eq)
    moreover have (∃x. (∃y. finite y ∧ y ⊆ A ∧ x = ennreal (sum f y)) ∧ y =
enn2ereal x) =
                  (∃x. finite x ∧ x ⊆ A ∧ y = ereal (sum f x)) for y
    proof −
      have (∃x. (∃y. finite y ∧ y ⊆ A ∧ x = ennreal (sum f y)) ∧ y = enn2ereal
x) ⟷
            (∃X x. finite X ∧ X ⊆ A ∧ x = ennreal (sum f X) ∧ y = enn2ereal x)
        by blast
      also have ... ⟷ (∃X. finite X ∧ X ⊆ A ∧ y = ereal (sum f X))
        by (rule arg_cong[of _ _ Ex])
        (auto simp: fun_eq_iff intro!: enn2ereal_ennreal sum_nonneg enn2ereal_ennreal[symmetric]
fnn)
      finally show ?thesis .
    qed
    hence Sup {y. ∃x. (∃y. finite y ∧ y ⊆ A ∧ x = ennreal (sum f y)) ∧ y =
enn2ereal x} =
          Sup {y. ∃x. finite x ∧ x ⊆ A ∧ y = ereal (sum f x)}
      by simp
    ultimately show max 0 (Sup {y. ∃x. (∃xa. finite xa ∧ xa ⊆ A ∧ x
                                  = ennreal (sum f xa)) ∧ y = enn2ereal x})
        = Sup {y. ∃x. finite x ∧ x ⊆ A ∧ y = ereal (sum f x)}
      by linarith
  qed
  finally show ?thesis
    by simp
qed
```

The following theorem relates (*abs_summable_on*) with *Infinite_Sum.abs_summable_on*. Note that while this theorem expresses an equivalence, the notion on the lhs is more general nonetheless because it applies to a wider range of types. (The rhs requires second-countable Banach spaces while the lhs is well-defined on arbitrary real vector spaces.)

```
lemma abs_summable_equivalent: ‹Infinite_Sum.abs_summable_on f A ⟷ f
abs_summable_on A›
proof (rule iffI)
  define n where ‹n x = norm (f x)› for x
  assume ‹n summable_on A›
```

```
  then have ‹sum n F ≤ infsum n A› if ‹finite F› and ‹F⊆A› for F
    using that by (auto simp flip: infsum_finite simp: n_def[abs_def] intro!:
infsum_mono_neutral)

  then show ‹f abs_summable_on A›
   by (auto intro!: abs_summable_finite_sumsI simp: n_def)
next
  define n where ‹n x = norm (f x)› for x
  assume ‹f abs_summable_on A›
  then have ‹n abs_summable_on A›
   by (simp add: ‹f abs_summable_on A› n_def)
  then have ‹sum n F ≤ infsetsum n A› if ‹finite F› and ‹F⊆A› for F
    using that by (auto simp flip: infsetsum_finite simp: n_def[abs_def] intro!:
infsetsum_mono_neutral)
  then show ‹n summable_on A›
   apply (rule_tac nonneg_bdd_above_summable_on)
   by (auto simp add: n_def bdd_above_def)
qed

lemma infsetsum_infsum:
  assumes f abs_summable_on A
  shows infsetsum f A = infsum f A
proof −
  have conv_sum_norm[simp]: (λx. norm (f x)) summable_on A
   using abs_summable_equivalent assms by blast
  have norm (infsetsum f A − infsum f A) ≤ ε if ε>0 for ε
  proof −
   define δ where δ = ε/2
   with that have [simp]: δ > 0 by simp
   obtain F1 where F1A: F1 ⊆ A and finF1: finite F1 and leq_eps: infsetsum
(λx. norm (f x)) (A−F1) ≤ δ
   proof −
     have sum_SUP: ereal (infsetsum (λx. norm (f x)) A) = (SUP F∈{F. finite
F ∧ F ⊆ A}. ereal (sum (λx. norm (f x)) F))
       (is _ = ?SUP)
       apply (rule infsetsum_nonneg_is_SUPREMUM_ereal)
       using assms by auto

     have (SUP F∈{F. finite F ∧ F ⊆ A}. ereal (∑x∈F. norm (f x))) − ereal δ
         < (SUP i∈{F. finite F ∧ F ⊆ A}. ereal (∑x∈i. norm (f x)))
       using ‹δ>0›
      by (metis diff_strict_left_mono diff_zero ereal_less_eq(3) ereal_minus(1)
not_le sum_SUP)
     then obtain F where F∈{F. finite F ∧ F ⊆ A} and ereal (sum (λx. norm
(f x)) F) > ?SUP − ereal (δ)
       by (meson less_SUP_iff)

     hence sum (λx. norm (f x)) F > infsetsum (λx. norm (f x)) A − (δ)
       unfolding sum_SUP[symmetric] by auto
```

```
  hence δ > infsetsum (λx. norm (f x)) (A−F)
  proof (subst infsetsum_Diff)
    show (λx. norm (f x)) abs_summable_on A
      if (∑ₐx∈A. norm (f x)) − δ < (∑ x∈F. norm (f x))
      using that
      by (simp add: assms)
    show F ⊆ A
      if (∑ₐx∈A. norm (f x)) − δ < (∑ x∈F. norm (f x))
      using that ‹F ∈ {F. finite F ∧ F ⊆ A}› by blast
    show (∑ₐx∈A. norm (f x)) − (∑ₐx∈F. norm (f x)) < δ
      if (∑ₐx∈A. norm (f x)) − δ < (∑ x∈F. norm (f x))
      using that ‹F ∈ {F. finite F ∧ F ⊆ A}› by auto
  qed
  thus ?thesis using that
    apply atomize_elim
    using ‹F ∈ {F. finite F ∧ F ⊆ A}› less_imp_le by blast
qed
obtain F2 where F2A: F2 ⊆ A and finF2: finite F2
  and dist: dist (sum (λx. norm (f x)) F2) (infsum (λx. norm (f x)) A) ≤ δ
  apply atomize_elim
  by (metis ‹0 < δ› conv_sum_norm infsum_finite_approximation)
have  leq_eps': infsum (λx. norm (f x)) (A−F2) ≤ δ
  apply (subst infsum_Diff)
  using finF2 F2A dist by (auto simp: dist_norm)
define F where F = F1 ∪ F2
have FA: F ⊆ A and finF: finite F
  unfolding F_def using F1A F2A finF1 finF2 by auto

have (∑ₐx∈A − (F1 ∪ F2). norm (f x)) ≤ (∑ₐx∈A − F1. norm (f x))
  apply (rule infsetsum_mono_neutral_left)
  using abs_summable_on_subset assms by fastforce+
hence leq_eps: infsetsum (λx. norm (f x)) (A−F) ≤ δ
  unfolding F_def
  using leq_eps by linarith
have infsum (λx. norm (f x)) (A − (F1 ∪ F2))
    ≤ infsum (λx. norm (f x)) (A − F2)
  apply (rule infsum_mono_neutral)
  using finF by (auto simp add: finF2 summable_on_cofin_subset F_def)
hence leq_eps': infsum (λx. norm (f x)) (A−F) ≤ δ
  unfolding F_def
  by (rule order.trans[OF _ leq_eps'])
have norm (infsetsum f A − infsetsum f F) = norm (infsetsum f (A−F))
  apply (subst infsetsum_Diff [symmetric])
  by (auto simp: FA assms)
also have ... ≤ infsetsum (λx. norm (f x)) (A−F)
  using norm_infsetsum_bound by blast
also have ... ≤ δ
  using leq_eps by simp
finally have diff1: norm (infsetsum f A − infsetsum f F) ≤ δ
```

by *assumption*
have *norm* (*infsum f A − infsum f F*) = *norm* (*infsum f* (*A−F*))
apply (*subst infsum_Diff* [*symmetric*])
by (*auto simp*: *assms abs_summable_summable finF FA*)
also have ... ≤ *infsum* (λ*x*. *norm* (*f x*)) (*A−F*)
by (*simp add*: *finF summable_on_cofin_subset norm_infsum_bound*)
also have ... ≤ *δ*
using *leq_eps′* **by** *simp*
finally have *diff2*: *norm* (*infsum f A − infsum f F*) ≤ *δ*
by *assumption*

have *x1*: *infsetsum f F* = *infsum f F*
using *finF* **by** *simp*
have *norm* (*infsetsum f A − infsum f A*) ≤ *norm* (*infsetsum f A − infsetsum f F*) + *norm* (*infsum f A − infsum f F*)
apply (*rule_tac norm_diff_triangle_le*)
apply *auto*
by (*simp_all add*: *x1 norm_minus_commute*)
also have ... ≤ *ε*
using *diff1 diff2 δ_def* **by** *linarith*
finally show *?thesis*
by *assumption*
qed
hence *norm* (*infsetsum f A − infsum f A*) = *0*
by (*meson antisym_conv1 dense_ge norm_not_less_zero*)
thus *?thesis*
by *auto*
qed

end

6.34 Faces, Extreme Points, Polytopes, Polyhedra etc

Ported from HOL Light by L C Paulson

theory *Polytope*
imports *Cartesian_Euclidean_Space Path_Connected*
begin

6.34.1 Faces of a (usually convex) set

definition *face_of* :: [*′a*::*real_vector set*, *′a set*] ⇒ *bool* (**infixr** (*face′_of*) *50*)
where
T face_of S ⟷
T ⊆ *S* ∧ *convex T* ∧
(∀ *a* ∈ *S*. ∀ *b* ∈ *S*. ∀ *x* ∈ *T*. *x* ∈ *open_segment a b* ⟶ *a* ∈ *T* ∧ *b* ∈ *T*)

lemma *face_ofD*: ⟦*T face_of S*; *x* ∈ *open_segment a b*; *a* ∈ *S*; *b* ∈ *S*; *x* ∈ *T*⟧ ⟹ *a* ∈ *T* ∧ *b* ∈ *T*
unfolding *face_of_def* **by** *blast*

lemma *face_of_translation_eq* [*simp*]:
((+) *a* ‘ *T face_of* (+) *a* ‘ *S*) ⟷ *T face_of S*
proof −
have ∗: ⋀*a T S*. *T face_of S* ⟹ ((+) *a* ‘ *T face_of* (+) *a* ‘ *S*)
by (*simp add*: *face_of_def*)
show *?thesis*
by (*force simp*: *image_comp o_def dest*: ∗ [**where** *a* = −*a*] *intro*: ∗)
qed

lemma *face_of_linear_image*:
assumes *linear f inj f*
shows (*f* ‘ *c face_of f* ‘ *S*) ⟷ *c face_of S*
by (*simp add*: *face_of_def inj_image_subset_iff inj_image_mem_iff open_segment_linear_image assms*)

lemma *face_of_refl*: *convex S* ⟹ *S face_of S*
by (*auto simp*: *face_of_def*)

lemma *face_of_refl_eq*: *S face_of S* ⟷ *convex S*
by (*auto simp*: *face_of_def*)

lemma *empty_face_of* [*iff*]: {} *face_of S*
by (*simp add*: *face_of_def*)

lemma *face_of_empty* [*simp*]: *S face_of* {} ⟷ *S* = {}
by (*meson empty_face_of face_of_def subset_empty*)

lemma *face_of_trans* [*trans*]: ⟦*S face_of T*; *T face_of u*⟧ ⟹ *S face_of u*
unfolding *face_of_def* **by** (*safe*; *blast*)

lemma *face_of_face*: *T face_of S* ⟹ (*f face_of T* ⟷ *f face_of S* ∧ *f* ⊆ *T*)
unfolding *face_of_def* **by** (*safe*; *blast*)

lemma *face_of_subset*: ⟦*F face_of S*; *F* ⊆ *T*; *T* ⊆ *S*⟧ ⟹ *F face_of T*
unfolding *face_of_def* **by** (*safe*; *blast*)

lemma *face_of_slice*: ⟦*F face_of S*; *convex T*⟧ ⟹ (*F* ∩ *T*) *face_of* (*S* ∩ *T*)
unfolding *face_of_def* **by** (*blast intro*: *convex_Int*)

lemma *face_of_Int*: ⟦*t1 face_of S*; *t2 face_of S*⟧ ⟹ (*t1* ∩ *t2*) *face_of S*
unfolding *face_of_def* **by** (*blast intro*: *convex_Int*)

lemma *face_of_Inter*: ⟦*A* ≠ {}; ⋀*T*. *T* ∈ *A* ⟹ *T face_of S*⟧ ⟹ (⋂ *A*) *face_of S*
unfolding *face_of_def* **by** (*blast intro*: *convex_Inter*)

lemma *face_of_Int_Int*: ⟦*F face_of T*; *F′ face_of t′*⟧ ⟹ (*F* ∩ *F′*) *face_of* (*T* ∩ *t′*)
unfolding *face_of_def* **by** (*blast intro*: *convex_Int*)

lemma *face_of_imp_subset*: *T face_of S* ⟹ *T* ⊆ *S*
unfolding *face_of_def* **by** *blast*

proposition *face_of_imp_eq_affine_Int*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *S*: *convex S* **and** *T*: *T face_of S*
shows *T* = (*affine hull T*) ∩ *S*
proof −
have *convex T* **using** *T* **by** (*simp add*: *face_of_def*)
have ∗: *False* **if** *x*: *x* ∈ *affine hull T* **and** *x* ∈ *S x* ∉ *T* **and** *y*: *y* ∈ *rel_interior T* **for** *x y*
proof −
obtain *e* **where** *e*>*0* **and** *e*: *cball y e* ∩ *affine hull T* ⊆ *T*
using *y* **by** (*auto simp*: *rel_interior_cball*)
have *y* ≠ *x y* ∈ *S y* ∈ *T*
using *face_of_imp_subset rel_interior_subset T that* **by** *blast*+
then have *zne*: ⋀*u*. ⟦*u* ∈ {*0*<..<*1*}; (*1* − *u*) ∗*R y* + *u* ∗*R x* ∈ *T*⟧ ⟹ *False*
using ‹*x* ∈ *S*› ‹*x* ∉ *T*› ‹*T face_of S*› **unfolding** *face_of_def*
by (*meson greaterThanLessThan_iff in_segment*(*2*))
define *u* **where** *u* ≡ *min* (*1*/*2*) (*e* / *norm* (*x* − *y*))
have *in01*: *u* ∈ {*0*<..<*1*}
using ‹*y* ≠ *x*› ‹*e* > *0*› **by** (*simp add*: *u_def*)
have *norm* (*u* ∗*R y* − *u* ∗*R x*) ≤ *e*
using ‹*e* > *0*›
by (*simp add*: *u_def norm_minus_commute min_mult_distrib_right flip*: *scaleR_diff_right*)
then have *dist y* ((*1* − *u*) ∗*R y* + *u* ∗*R x*) ≤ *e*
by (*simp add*: *dist_norm algebra_simps*)
then show *False*
using *zne* [*OF in01 e* [*THEN subsetD*]] **by** (*simp add*: ‹*y* ∈ *T*› *hull_inc mem_affine x*)
qed
show *?thesis*
proof (*rule subset_antisym*)
show *T* ⊆ *affine hull T* ∩ *S*
using *assms* **by** (*simp add*: *hull_subset face_of_imp_subset*)
show *affine hull T* ∩ *S* ⊆ *T*
using ∗ ‹*convex T*› *rel_interior_eq_empty* **by** *fastforce*
qed
qed

lemma *face_of_imp_closed*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *convex S closed S T face_of S* **shows** *closed T*

by (*metis affine_affine_hull affine_closed closed_Int face_of_imp_eq_affine_Int assms*)

lemma *face_of_Int_supporting_hyperplane_le_strong*:
assumes *convex*$(S \cap \{x.\ a \cdot x = b\})$ **and** *aleb*: $\bigwedge x.\ x \in S \Longrightarrow a \cdot x \le b$
shows $(S \cap \{x.\ a \cdot x = b\})$ *face_of S*
proof −
have ∗: $a \cdot u = a \cdot x$ **if** $x \in$ *open_segment u v* $u \in S$ $v \in S$ **and** b: $b = a \cdot x$
for *u v x*
proof (*rule antisym*)
show $a \cdot u \le a \cdot x$
using *aleb* ‹$u \in S$› ‹$b = a \cdot x$› **by** *blast*
next
obtain ξ **where** $b = a \cdot ((1 - \xi) *_R u + \xi *_R v)$ $0 < \xi$ $\xi < 1$
using ‹$b = a \cdot x$› ‹$x \in$ *open_segment u v*› *in_segment*
by (*auto simp*: *open_segment_image_interval split*: *if_split_asm*)
then have $b + \xi * (a \cdot u) \le a \cdot u + \xi * b$
using *aleb* [*OF* ‹$v \in S$›] **by** (*simp add*: *algebra_simps*)
then have $(1 - \xi) * b \le (1 - \xi) * (a \cdot u)$
by (*simp add*: *algebra_simps*)
then have $b \le a \cdot u$
using ‹$\xi < 1$› **by** *auto*
with b **show** $a \cdot x \le a \cdot u$ **by** *simp*
qed
show *?thesis*
using ∗ *open_segment_commute* **by** (*fastforce simp add*: *face_of_def assms*)
qed

lemma *face_of_Int_supporting_hyperplane_ge_strong*:
⟦*convex*$(S \cap \{x.\ a \cdot x = b\})$; $\bigwedge x.\ x \in S \Longrightarrow a \cdot x \ge b$⟧
$\Longrightarrow (S \cap \{x.\ a \cdot x = b\})$ *face_of S*
using *face_of_Int_supporting_hyperplane_le_strong* [*of S* $-a$ $-b$] **by** *simp*

lemma *face_of_Int_supporting_hyperplane_le*:
⟦*convex S*; $\bigwedge x.\ x \in S \Longrightarrow a \cdot x \le b$⟧ $\Longrightarrow (S \cap \{x.\ a \cdot x = b\})$ *face_of S*
by (*simp add*: *convex_Int convex_hyperplane face_of_Int_supporting_hyperplane_le_strong*)

lemma *face_of_Int_supporting_hyperplane_ge*:
⟦*convex S*; $\bigwedge x.\ x \in S \Longrightarrow a \cdot x \ge b$⟧ $\Longrightarrow (S \cap \{x.\ a \cdot x = b\})$ *face_of S*
by (*simp add*: *convex_Int convex_hyperplane face_of_Int_supporting_hyperplane_ge_strong*)

lemma *face_of_imp_convex*: *T face_of S* $\Longrightarrow$ *convex T*
using *face_of_def* **by** *blast*

lemma *face_of_imp_compact*:
fixes *S* :: *'a*::*euclidean_space set*
shows ⟦*convex S*; *compact S*; *T face_of S*⟧ $\Longrightarrow$ *compact T*
by (*meson bounded_subset compact_eq_bounded_closed face_of_imp_closed face_of_imp_subset*)

lemma *face_of_Int_subface*:
  ⟦*A* ∩ *B face_of A*; *A* ∩ *B face_of B*; *C face_of A*; *D face_of B*⟧
  ⟹ (*C* ∩ *D*) *face_of C* ∧ (*C* ∩ *D*) *face_of D*
 by (*meson face_of_Int_Int face_of_face inf_le1 inf_le2*)

lemma *subset_of_face_of*:
  fixes *S* :: *'a*::*real_normed_vector set*
  assumes *T face_of S u* ⊆ *S T* ∩ (*rel_interior u*) ≠ {}
   shows *u* ⊆ *T*
proof
 fix *c*
 assume *c* ∈ *u*
 obtain *b* **where** *b* ∈ *T b* ∈ *rel_interior u* **using** *assms* **by** *auto*
 then obtain *e* **where** *e*>*0 b* ∈ *u* **and** *e*: *cball b e* ∩ *affine hull u* ⊆ *u*
  by (*auto simp*: *rel_interior_cball*)
 show *c* ∈ *T*
 proof (*cases b*=*c*)
  case *True* **with** ‹*b* ∈ *T*› **show** *?thesis* **by** *blast*
 next
  case *False*
  define *d* **where** *d* = *b* + (*e* / *norm*(*b* − *c*)) *∗R* (*b* − *c*)
  have *d* ∈ *cball b e* ∩ *affine hull u*
   using ‹*e* > *0*› ‹*b* ∈ *u*› ‹*c* ∈ *u*›
   by (*simp add*: *d_def dist_norm hull_inc mem_affine_3_minus False*)
  with *e* **have** *d* ∈ *u* **by** *blast*
  have *nbc*: *norm* (*b* − *c*) + *e* > *0* **using** ‹*e* > *0*›
  by (*metis add.commute le_less_trans less_add_same_cancel2 norm_ge_zero*)
  then have [*simp*]: *d* ≠ *c* **using** *False scaleR_cancel_left* [*of 1* + (*e* / *norm* (*b* − *c*)) *b c*]
   by (*simp add*: *algebra_simps d_def*) (*simp add*: *field_split_simps*)
  have [*simp*]: ((*e* − *e* ∗ *e* / (*e* + *norm* (*b* − *c*))) / *norm* (*b* − *c*)) = (*e* / (*e* + *norm* (*b* − *c*)))
   using *False nbc*
   by (*simp add*: *divide_simps*) (*simp add*: *algebra_simps*)
  have *b* ∈ *open_segment d c*
   apply (*simp add*: *open_segment_image_interval*)
   apply (*simp add*: *d_def algebra_simps*)
   apply (*rule_tac x*=*e* / (*e* + *norm* (*b* − *c*)) **in** *image_eqI*)
   using *False nbc* ‹*0* < *e*› **by** (*auto simp*: *algebra_simps*)
  then have *d* ∈ *T* ∧ *c* ∈ *T*
   by (*meson* ‹*b* ∈ *T*› ‹*c* ∈ *u*› ‹*d* ∈ *u*› *assms face_ofD subset_iff*)
  then show *?thesis* **..**
 qed
qed

lemma *face_of_eq*:
  fixes *S* :: *'a*::*real_normed_vector set*
  assumes *T face_of S U face_of S* (*rel_interior T*) ∩ (*rel_interior U*) ≠ {}

shows $T = U$
using *assms*
unfolding *disjoint_iff_not_equal*
by (*metis IntI empty_iff face_of_imp_subset mem_rel_interior_ball subset_antisym subset_of_face_of*)

lemma *face_of_disjoint_rel_interior*:
fixes $S :: 'a::real\_normed\_vector\ set$
assumes $T\ face\_of\ S\ T \neq S$
shows $T \cap rel\_interior\ S = \{\}$
by (*meson assms subset_of_face_of face_of_imp_subset order_refl subset_antisym*)

lemma *face_of_disjoint_interior*:
fixes $S :: 'a::real\_normed\_vector\ set$
assumes $T\ face\_of\ S\ T \neq S$
shows $T \cap interior\ S = \{\}$
using *assms face_of_disjoint_rel_interior interior_subset_rel_interior* **by** *fastforce*

lemma *face_of_subset_rel_boundary*:
fixes $S :: 'a::real\_normed\_vector\ set$
assumes $T\ face\_of\ S\ T \neq S$
shows $T \subseteq (S - rel\_interior\ S)$
by (*meson DiffI assms disjoint_iff_not_equal face_of_disjoint_rel_interior face_of_imp_subset subset_iff*)

lemma *face_of_subset_rel_frontier*:
fixes $S :: 'a::real\_normed\_vector\ set$
assumes $T\ face\_of\ S\ T \neq S$
shows $T \subseteq rel\_frontier\ S$
using *assms closure_subset face_of_disjoint_rel_interior face_of_imp_subset rel_frontier_def* **by** *fastforce*

lemma *face_of_aff_dim_lt*:
fixes $S :: 'a::euclidean\_space\ set$
assumes *convex S* $T\ face\_of\ S\ T \neq S$
shows $aff\_dim\ T < aff\_dim\ S$
proof −
have $aff\_dim\ T \leq aff\_dim\ S$
by (*simp add*: *face_of_imp_subset aff_dim_subset assms*)
moreover have $aff\_dim\ T \neq aff\_dim\ S$
by (*metis aff_dim_empty assms convex_rel_frontier_aff_dim face_of_imp_convex face_of_subset_rel_frontier order_less_irrefl*)
ultimately show *?thesis*
by *simp*
qed

lemma *subset_of_face_of_affine_hull*:

```
    fixes S :: 'a::euclidean_space set
  assumes T: T face_of S and convex S U ⊆ S and dis: ¬ disjnt (affine hull T)
(rel_interior U)
  shows U ⊆ T
proof (rule subset_of_face_of [OF T ‹U ⊆ S›])
  show T ∩ rel_interior U ≠ {}
    using face_of_imp_eq_affine_Int [OF ‹convex S› T] rel_interior_subset dis
‹U ⊆ S› disjnt_def
    by fastforce
qed

lemma affine_hull_face_of_disjoint_rel_interior:
  fixes S :: 'a::euclidean_space set
  assumes convex S F face_of S F ≠ S
  shows affine hull F ∩ rel_interior S = {}
 by (meson antisym assms disjnt_def equalityD2 face_of_def subset_of_face_of_affine_hull)

lemma affine_diff_divide:
    assumes affine S k ≠ 0 k ≠ 1 and xy: x ∈ S y /R (1 − k) ∈ S
      shows (x − y) /R k ∈ S
proof −
  have inverse(k) ∗R (x − y) = (1 − inverse k) ∗R inverse(1 − k) ∗R y +
inverse(k) ∗R x
    using assms
      by (simp add: algebra_simps) (simp add: scaleR_left_distrib [symmetric]
field_split_simps)
  then show ?thesis
    using ‹affine S› xy by (auto simp: affine_alt)
qed

proposition face_of_convex_hulls:
     assumes S: finite S T ⊆ S and disj: affine hull T ∩ convex hull (S − T) =
{}
     shows (convex hull T) face_of (convex hull S)
proof −
  have fin: finite T finite (S − T) using assms
    by (auto simp: finite_subset)
  have ∗: x ∈ convex hull T
        if x: x ∈ convex hull S and y: y ∈ convex hull S and w: w ∈ convex hull
T w ∈ open_segment x y
        for x y w
  proof −
    have waff: w ∈ affine hull T
      using convex_hull_subset_affine_hull w by blast
    obtain a b where a: ⋀i. i ∈ S ⟹ 0 ≤ a i and asum: sum a S = 1 and
aeqx: (∑ i∈S. a i ∗R i) = x
                and b: ⋀i. i ∈ S ⟹ 0 ≤ b i and bsum: sum b S = 1 and beqy:
(∑ i∈S. b i ∗R i) = y
      using x y by (auto simp: assms convex_hull_finite)
```

```
    obtain u where (1 − u) *R x + u *R y ∈ convex hull T x ≠ y and weq: w
= (1 − u) *R x + u *R y
              and u01: 0 < u u < 1
      using w by (auto simp: open_segment_image_interval split: if_split_asm)
    define c where c i = (1 − u) * a i + u * b i for i
    have cge0: ⋀i. i ∈ S ⟹ 0 ≤ c i
      using a b u01 by (simp add: c_def)
    have sumc1: sum c S = 1
      by (simp add: c_def sum.distrib sum_distrib_left [symmetric] asum bsum)
    have sumci_xy: (∑ i∈S. c i *R i) = (1 − u) *R x + u *R y
      apply (simp add: c_def sum.distrib scaleR_left_distrib)
     by (simp only: scaleR_scaleR [symmetric] Real_Vector_Spaces.scaleR_right.sum
[symmetric] aeqx beqy)
    show ?thesis
    proof (cases sum c (S − T) = 0)
      case True
      have ci0: ⋀i. i ∈ (S − T) ⟹ c i = 0
        using True cge0 fin(2) sum_nonneg_eq_0_iff by auto
      have a0: a i = 0 if i ∈ (S − T) for i
        using ci0 [OF that] u01 a [of i] b [of i] that
       by (simp add: c_def Groups.ordered_comm_monoid_add_class.add_nonneg_eq_0_iff)
      have  sum a T = 1
        using assms by (metis sum.mono_neutral_cong_right a0 asum)
      moreover have (∑ x∈T. a x *R x) = x
      using a0 assms by (auto simp: cge0 a aeqx [symmetric] sum.mono_neutral_right)
      ultimately show ?thesis
        using a assms(2) by (auto simp add: convex_hull_finite ‹finite T› )
    next
      case False
      define k where k = sum c (S − T)
      have k > 0 using False
      unfolding k_def by (metis DiffD1 antisym_conv cge0 sum_nonneg not_less)
      have weq_sumsum: w = sum (λx. c x *R x) T + sum (λx. c x *R x) (S −
T)
        by (metis (no_types) add.commute S(1) S(2) sum.subset_diff sumci_xy
weq)
      show ?thesis
      proof (cases k = 1)
        case True
        then have sum c T = 0
          by (simp add: S k_def sum_diff sumc1)
        then have sum c (S − T) = 1
          by (simp add: S sum_diff sumc1)
        moreover have ci0: ⋀i. i ∈ T ⟹ c i = 0
         by (meson ‹finite T› ‹sum c T = 0› ‹T ⊆ S› cge0 sum_nonneg_eq_0_iff
subsetCE)
        then have (∑ i∈S−T. c i *R i) = w
          by (simp add: weq_sumsum)
        ultimately have w ∈ convex hull (S − T)
```

```
      using cge0 by (auto simp add: convex_hull_finite fin)
     then show ?thesis
      using disj waff by blast
    next
     case False
     then have sumcf: sum c T = 1 - k
      by (simp add: S k_def sum_diff sumc1)
     have ⋀x. x ∈ T ⟹ 0 ≤ inverse (1 - k) * c x
     by (metis ‹T ⊆ S› cge0 inverse_nonnegative_iff_nonnegative mult_nonneg_nonneg
subsetD sum_nonneg sumcf)
     moreover have (∑x∈T. inverse (1 - k) * c x) = 1
     by (metis False eq_iff_diff_eq_0 mult.commute right_inverse sum_distrib_left
sumcf)
     ultimately have (∑ i∈T. c i *R i) /R (1 - k) ∈ convex hull T
      apply (simp add: convex_hull_finite fin)
      by (metis (mono_tags, lifting) scaleR_right.sum scaleR_scaleR sum.cong)
     with ‹0 < k› have inverse(k) *R (w - sum (λi. c i *R i) T) ∈ affine hull
T
        by (simp add: affine_diff_divide [OF affine_affine_hull] False waff
convex_hull_subset_affine_hull [THEN subsetD])
     moreover have inverse(k) *R (w - sum (λx. c x *R x) T) ∈ convex hull
(S - T)
      apply (simp add: weq_sumsum convex_hull_finite fin)
      apply (rule_tac x=λi. inverse k * c i in exI)
      using ‹k > 0› cge0
      apply (auto simp: scaleR_right.sum simp flip: sum_distrib_left k_def)
      done
     ultimately show ?thesis
      using disj by blast
    qed
   qed
  qed
  have [simp]: convex hull T ⊆ convex hull S
   by (simp add: ‹T ⊆ S› hull_mono)
  show ?thesis
   using open_segment_commute by (auto simp: face_of_def intro: *)
qed

proposition face_of_convex_hull_insert:
  assumes finite S a ∉ affine hull S and T: T face_of convex hull S
  shows T face_of convex hull insert a S
proof -
  have convex hull S face_of convex hull insert a S
   by (simp add: assms face_of_convex_hulls insert_Diff_if subset_insertI)
  then show ?thesis
   using T face_of_trans by blast
qed

proposition face_of_affine_trivial:
```

assumes *affine S T face_of S*
shows $T = \{\} \vee T = S$
proof (*rule ccontr*, *clarsimp*)
assume $T \neq \{\}$ $T \neq S$
then obtain *a* **where** $a \in T$ **by** *auto*
then have $a \in S$
using ‹*T face_of S*› *face_of_imp_subset* **by** *blast*
have $S \subseteq T$
proof
fix *b* **assume** $b \in S$
show $b \in T$
proof (*cases* $a = b$)
case *True* **with** ‹$a \in T$› **show** *?thesis* **by** *auto*
next
case *False*
then have $a \in$ *open_segment* $(2 *_R a - b)$ *b*
by (*metis diff_add_cancel inverse_eq_divide midpoint_def midpoint_in_open_segment*
scaleR_2 scaleR_half_double)
moreover have $2 *_R a - b \in S$
by (*rule mem_affine* [*OF* ‹*affine S*› ‹$a \in S$› ‹$b \in S$›, *of 2 −1*, *simplified*])
moreover note ‹$b \in S$› ‹$a \in T$›
ultimately show *?thesis*
by (*rule face_ofD* [*OF* ‹*T face_of S*›, *THEN conjunct2*])
qed
qed
then show *False*
using ‹$T \neq S$› ‹*T face_of S*› *face_of_imp_subset* **by** *blast*
qed

lemma *face_of_affine_eq*:
affine S $\Longrightarrow$ (*T face_of S* $\longleftrightarrow$ $T = \{\} \vee T = S$)
using *affine_imp_convex face_of_affine_trivial face_of_refl* **by** *auto*

proposition *Inter_faces_finite_altbound*:
fixes *T* :: *'a*::*euclidean_space set set*
assumes *cfaI*: $\bigwedge c.\ c \in T \Longrightarrow c$ *face_of S*
shows $\exists F'.$ *finite* $F' \wedge F' \subseteq T \wedge$ *card* $F' \leq DIM('a) + 2 \wedge \bigcap F' = \bigcap T$
proof (*cases* $\forall F'.$ *finite* $F' \wedge F' \subseteq T \wedge$ *card* $F' \leq DIM('a) + 2 \longrightarrow (\exists c.\ c \in T$
$\wedge\ c \cap (\bigcap F') \subset (\bigcap F'))$)
case *True*
then obtain *c* **where** *c*:
$\bigwedge F'.\ [\![$*finite* F'; $F' \subseteq T$; *card* $F' \leq DIM('a) + 2]\!] \Longrightarrow c\ F' \in T \wedge c\ F' \cap$
$(\bigcap F') \subset (\bigcap F')$
by *metis*
define *d* **where** $d \equiv \lambda n.\ ((\lambda r.\ insert\ (c\ r)\ r)\ \widehat{\ }\widehat{\ }\ n)\ \{c\{\}\}$
note *d_def* [*simp*]

```
  have dSuc: ⋀n. d (Suc n) = insert (c (d n)) (d n)
    by simp
  have dn_notempty: d n ≠ {} for n
    by (induction n) auto
  have dn_le_Suc: d n ⊆ T ∧ finite(d n) ∧ card(d n) ≤ Suc n if n ≤ DIM('a) +
2 for n
  using that
  proof (induction n)
    case 0
    then show ?case by (simp add: c)
  next
    case (Suc n)
    then show ?case by (auto simp: c card_insert_if)
  qed
  have aff_dim_le: aff_dim(⋂(d n)) ≤ DIM('a) − int n if n ≤ DIM('a) + 2 for
n
  using that
  proof (induction n)
    case 0
    then show ?case
      by (simp add: aff_dim_le_DIM)
  next
    case (Suc n)
    have fs: ⋂(d (Suc n)) face_of S
     by (meson Suc.prems cfaI dn_le_Suc dn_notempty face_of_Inter subsetCE)
    have condn: convex (⋂(d n))
      using Suc.prems nat_le_linear not_less_eq_eq
      by (blast intro: face_of_imp_convex cfaI convex_Inter dest: dn_le_Suc)
    have fdn: ⋂(d (Suc n)) face_of ⋂(d n)
    by (metis (no_types, lifting) Inter_anti_mono Suc.prems dSuc cfaI dn_le_Suc
dn_notempty face_of_Inter face_of_imp_subset face_of_subset subset_iff sub-
set_insertI)
    have ne: ⋂(d (Suc n)) ≠ ⋂(d n)
    by (metis (no_types, lifting) Suc.prems Suc_leD c complete_lattice_class.Inf_insert
dSuc dn_le_Suc less_irrefl order.trans)
   have ∗: ⋀m::int. ⋀d. ⋀d'::int. d < d' ∧ d' ≤ m − n ⟹ d ≤ m − of_nat(n+1)
      by arith
    have aff_dim (⋂(d (Suc n))) < aff_dim (⋂(d n))
      by (rule face_of_aff_dim_lt [OF condn fdn ne])
    moreover have aff_dim (⋂(d n)) ≤ int (DIM('a)) − int n
      using Suc by auto
    ultimately
    have aff_dim (⋂(d (Suc n))) ≤ int (DIM('a)) − (n+1) by arith
    then show ?case by linarith
  qed
  have aff_dim (⋂(d (DIM('a) + 2))) ≤ −2
     using aff_dim_le [OF order_refl] by simp
  with aff_dim_geq [of ⋂(d (DIM('a) + 2))] show ?thesis
    using order.trans by fastforce
```

next
case *False*
then show *?thesis* **by** *fastforce*
qed

lemma *faces_of_translation*:
$\{F.\ F\ face\_of\ (+)\ a\ `\ S\} = (image\ ((+)\ a))\ `\ \{F.\ F\ face\_of\ S\}$
proof −
have $\bigwedge F.\ F\ face\_of\ (+)\ a\ `\ S \Longrightarrow \exists\, G.\ G\ face\_of\ S \wedge F = (+)\ a\ `\ G$
by (*metis face_of_imp_subset face_of_translation_eq subset_imageE*)
then show *?thesis*
by (*auto simp*: *image_iff*)
qed

proposition *face_of_Times*:
assumes *F face_of S* **and** *F′ face_of S′*
shows $(F \times F')\ face\_of\ (S \times S')$
proof −
have $F \times F' \subseteq S \times S'$
using *assms* [*unfolded face_of_def*] **by** *blast*
moreover
have *convex* $(F \times F')$
using *assms* [*unfolded face_of_def*] **by** (*blast intro*: *convex_Times*)
moreover
have $a \in F \wedge a' \in F' \wedge b \in F \wedge b' \in F'$
if $a \in S\ b \in S\ a' \in S'\ b' \in S'\ x \in F \times F'\ x \in open\_segment\ (a,a')\ (b,b')$
for *a b a′ b′ x*
proof (*cases* $b{=}a \vee b'{=}a'$)
case *True* **with** *that* **show** *?thesis*
using *assms*
by (*force simp*: *in_segment dest*: *face_ofD*)
next
case *False* **with** *assms* [*unfolded face_of_def*] *that* **show** *?thesis*
by (*blast dest*!: *open_segment_PairD*)
qed
ultimately show *?thesis*
unfolding *face_of_def* **by** *blast*
qed

corollary *face_of_Times_decomp*:
fixes *S* :: *'a*::*euclidean_space set* **and** *S′* :: *'b*::*euclidean_space set*
shows $C\ face\_of\ (S \times S') \longleftrightarrow (\exists\, F\ F'.\ F\ face\_of\ S \wedge F'\ face\_of\ S' \wedge C = F \times F')$
(**is** *?lhs* = *?rhs*)
proof
assume *C*: *?lhs*
show *?rhs*
proof (*cases* $C = \{\}$)
case *True* **then show** *?thesis* **by** *auto*

```
  next
    case False
    have 1: fst ‘ C ⊆ S snd ‘ C ⊆ S'
      using C face_of_imp_subset by fastforce+
    have convex C
      using C by (metis face_of_imp_convex)
    have conv: convex (fst ‘ C) convex (snd ‘ C)
      by (simp_all add: ‹convex C› convex_linear_image linear_fst linear_snd)
    have fstab: a ∈ fst ‘ C ∧ b ∈ fst ‘ C
          if a ∈ S b ∈ S x ∈ open_segment a b (x,x') ∈ C for a b x x'
    proof −
      have ∗: (x,x') ∈ open_segment (a,x') (b,x')
        using that by (auto simp: in_segment)
      show ?thesis
        using face_ofD [OF C ∗] that face_of_imp_subset [OF C] by force
    qed
    have fst: fst ‘ C face_of S
      by (force simp: face_of_def 1 conv fstab)
    have sndab: a' ∈ snd ‘ C ∧ b' ∈ snd ‘ C
      if a' ∈ S' b' ∈ S' x' ∈ open_segment a' b' (x,x') ∈ C for a' b' x x'
    proof −
      have ∗: (x,x') ∈ open_segment (x,a') (x,b')
        using that by (auto simp: in_segment)
      show ?thesis
        using face_ofD [OF C ∗] that face_of_imp_subset [OF C] by force
    qed
    have snd: snd ‘ C face_of S'
      by (force simp: face_of_def 1 conv sndab)
    have cc: rel_interior C ⊆ rel_interior (fst ‘ C) × rel_interior (snd ‘ C)
      by (force simp: face_of_Times rel_interior_Times conv fst snd ‹convex C›
linear_fst linear_snd rel_interior_convex_linear_image [symmetric])
    have C = fst ‘ C × snd ‘ C
    proof (rule face_of_eq [OF C])
      show fst ‘ C × snd ‘ C face_of S × S'
        by (simp add: face_of_Times rel_interior_Times conv fst snd)
      show rel_interior C ∩ rel_interior (fst ‘ C × snd ‘ C) ≠ {}
        using False rel_interior_eq_empty ‹convex C› cc
        by (auto simp: face_of_Times rel_interior_Times conv fst)
    qed
    with fst snd show ?thesis by metis
  qed
qed (use face_of_Times in auto)

lemma face_of_Times_eq:
  fixes S :: 'a::euclidean_space set and S' :: 'b::euclidean_space set
  shows (F × F') face_of (S × S') ⟷ F = {} ∨ F' = {} ∨ F face_of S ∧ F'
face_of S'
  by (auto simp: face_of_Times_decomp times_eq_iff)
```

lemma *hyperplane_face_of_halfspace_le*: $\{x.\ a \cdot x = b\}$ *face_of* $\{x.\ a \cdot x \leq b\}$
proof −
have $\{x.\ a \cdot x \leq b\} \cap \{x.\ a \cdot x = b\} = \{x.\ a \cdot x = b\}$
by *auto*
with *face_of_Int_supporting_hyperplane_le* [*OF convex_halfspace_le* [*of a b*], *of a b*]
show *?thesis* **by** *auto*
qed

lemma *hyperplane_face_of_halfspace_ge*: $\{x.\ a \cdot x = b\}$ *face_of* $\{x.\ a \cdot x \geq b\}$
proof −
have $\{x.\ a \cdot x \geq b\} \cap \{x.\ a \cdot x = b\} = \{x.\ a \cdot x = b\}$
by *auto*
with *face_of_Int_supporting_hyperplane_ge* [*OF convex_halfspace_ge* [*of b a*], *of b a*]
show *?thesis* **by** *auto*
qed

lemma *face_of_halfspace_le*:
fixes *a* :: *'n*::*euclidean_space*
shows *F face_of* $\{x.\ a \cdot x \leq b\} \longleftrightarrow F = \{\} \vee F = \{x.\ a \cdot x = b\} \vee F = \{x.\ a \cdot x \leq b\}$
(**is** *?lhs* = *?rhs*)
proof (*cases a* = *0*)
case *True* **then show** *?thesis*
using *face_of_affine_eq affine_UNIV* **by** *auto*
next
case *False*
then have *ine*: *interior* $\{x.\ a \cdot x \leq b\} \neq \{\}$
using *halfspace_eq_empty_lt interior_halfspace_le* **by** *blast*
show *?thesis*
proof
assume *L*: *?lhs*
have *F face_of* $\{x.\ a \cdot x = b\}$ **if** $F \neq \{x.\ a \cdot x \leq b\}$
proof −
have *F face_of rel_frontier* $\{x.\ a \cdot x \leq b\}$
proof (*rule face_of_subset* [*OF L*])
show $F \subseteq$ *rel_frontier* $\{x.\ a \cdot x \leq b\}$
by (*simp add*: *L face_of_subset_rel_frontier that*)
qed (*force simp*: *rel_frontier_def closed_halfspace_le*)
then show *?thesis*
using *False*
by (*simp add*: *frontier_halfspace_le rel_frontier_nonempty_interior* [*OF ine*])
qed
with *L* **show** *?rhs*
using *affine_hyperplane face_of_affine_eq* **by** *blast*
next
assume *?rhs*

```
    then show ?lhs
    by (metis convex_halfspace_le empty_face_of face_of_refl hyperplane_face_of_halfspace_le)
  qed
qed

lemma face_of_halfspace_ge:
  fixes a :: 'n::euclidean_space
  shows F face_of {x. a · x ≥ b} ⟷ F = {} ∨ F = {x. a · x = b} ∨ F = {x.
a · x ≥ b}
  using face_of_halfspace_le [of F −a −b] by simp
```

6.34.2 Exposed faces

That is, faces that are intersection with supporting hyperplane

```
definition exposed_face_of :: ['a::euclidean_space set, 'a set] ⇒ bool
                              (infixr (exposed'_face'_of) 50)
  where T exposed_face_of S ⟷
        T face_of S ∧ (∃ a b. S ⊆ {x. a · x ≤ b} ∧ T = S ∩ {x. a · x = b})

lemma empty_exposed_face_of [iff]: {} exposed_face_of S
proof −
  have S ⊆ {x. 0 · x ≤ 1} ∧ {} = S ∩ {x. 0 · x = 1}
    by force
  then show ?thesis
    using exposed_face_of_def by blast
qed

lemma exposed_face_of_refl_eq [simp]: S exposed_face_of S ⟷ convex S
proof
  assume S: convex S
  have S ⊆ {x. 0 · x ≤ 0} ∧ S = S ∩ {x. 0 · x = 0}
    by auto
  with S show S exposed_face_of S
    using exposed_face_of_def face_of_refl_eq by blast
qed (simp add: exposed_face_of_def face_of_refl_eq)

lemma exposed_face_of_refl: convex S ⟹ S exposed_face_of S
  by simp

lemma exposed_face_of:
    T exposed_face_of S ⟷
     T face_of S ∧ (T = {} ∨ T = S ∨
      (∃ a b. a ≠ 0 ∧ S ⊆ {x. a · x ≤ b} ∧ T = S ∩ {x. a · x = b}))
     (is ?lhs = ?rhs)
proof
  show ?lhs ⟹ ?rhs
   by (smt (verit) Collect_cong exposed_face_of_def hyperplane_eq_empty inf.absorb_iff1
                   inf_bot_right inner_zero_left)
  show ?rhs ⟹ ?lhs
```

using *exposed_face_of_def face_of_imp_convex* **by** *fastforce*
qed

lemma *exposed_face_of_Int_supporting_hyperplane_le*:
⟦*convex S*; ⋀*x*. *x* ∈ *S* ⟹ *a* · *x* ≤ *b*⟧ ⟹ (*S* ∩ {*x*. *a* · *x* = *b*}) *exposed_face_of S*
by (*force simp*: *exposed_face_of_def face_of_Int_supporting_hyperplane_le*)

lemma *exposed_face_of_Int_supporting_hyperplane_ge*:
⟦*convex S*; ⋀*x*. *x* ∈ *S* ⟹ *a* · *x* ≥ *b*⟧ ⟹ (*S* ∩ {*x*. *a* · *x* = *b*}) *exposed_face_of S*
using *exposed_face_of_Int_supporting_hyperplane_le* [*of S* −*a* −*b*] **by** *simp*

proposition *exposed_face_of_Int*:
assumes *T exposed_face_of S*
and *U exposed_face_of S*
shows (*T* ∩ *U*) *exposed_face_of S*
proof −
obtain *a b* **where** *T*: *S* ∩ {*x*. *a* · *x* = *b*} *face_of S*
and *S*: *S* ⊆ {*x*. *a* · *x* ≤ *b*}
and *teq*: *T* = *S* ∩ {*x*. *a* · *x* = *b*}
using *assms* **by** (*auto simp*: *exposed_face_of_def*)
obtain *a′ b′* **where** *U*: *S* ∩ {*x*. *a′* · *x* = *b′*} *face_of S*
and *s′*: *S* ⊆ {*x*. *a′* · *x* ≤ *b′*}
and *ueq*: *U* = *S* ∩ {*x*. *a′* · *x* = *b′*}
using *assms* **by** (*auto simp*: *exposed_face_of_def*)
have *tu*: *T* ∩ *U face_of S*
using *T teq U ueq* **by** (*simp add*: *face_of_Int*)
have *ss*: *S* ⊆ {*x*. (*a* + *a′*) · *x* ≤ *b* + *b′*}
using *S s′* **by** (*force simp*: *inner_left_distrib*)
have *S* ⊆ {*x*. (*a* + *a′*) · *x* ≤ *b* + *b′*} ∧ *T* ∩ *U* = *S* ∩ {*x*. (*a* + *a′*) · *x* = *b* + *b′*}
using *S s′* **by** (*fastforce simp*: *ss inner_left_distrib teq ueq*)
then show *?thesis*
using *exposed_face_of_def tu* **by** *auto*
qed

proposition *exposed_face_of_Inter*:
fixes *P* :: *'a*::*euclidean_space set set*
assumes *P* ≠ {}
and ⋀*T*. *T* ∈ *P* ⟹ *T exposed_face_of S*
shows ⋂ *P exposed_face_of S*
proof −
obtain *Q* **where** *finite Q* **and** *QsubP*: *Q* ⊆ *P card Q* ≤ *DIM*(*'a*) + *2* **and** *IntQ*: ⋂ *Q* = ⋂ *P*
using *Inter_faces_finite_altbound* [*of P S*] *assms* [*unfolded exposed_face_of*]
by *force*
show *?thesis*
proof (*cases Q* = {})
case *True* **then show** *?thesis*

```
    by (metis IntQ Inter_UNIV_conv(2) assms(1) assms(2) ex_in_conv)
  next
    case False
    have Q ⊆ {T. T exposed_face_of S}
      using QsubP assms by blast
    moreover have Q ⊆ {T. T exposed_face_of S} ⟹ ⋂ Q exposed_face_of S
      using ‹finite Q› False
      by (induction Q rule: finite_induct; use exposed_face_of_Int in fastforce)
    ultimately show ?thesis
      by (simp add: IntQ)
  qed
qed

proposition exposed_face_of_sums:
  assumes convex S and convex T
      and F exposed_face_of {x + y | x y. x ∈ S ∧ y ∈ T}
          (is F exposed_face_of ?ST)
  obtains k l
    where k exposed_face_of S l exposed_face_of T
          F = {x + y | x y. x ∈ k ∧ y ∈ l}
proof (cases F = {})
  case True then show ?thesis
    using that by blast
next
  case False
  show ?thesis
  proof (cases F = ?ST)
    case True then show ?thesis
      using assms exposed_face_of_refl_eq that by blast
  next
    case False
    obtain p where p ∈ F using ‹F ≠ {}› by blast
    moreover
    obtain u z where T: ?ST ∩ {x. u • x = z} face_of ?ST
                 and S: ?ST ⊆ {x. u • x ≤ z}
                 and feq: F = ?ST ∩ {x. u • x = z}
      using assms by (auto simp: exposed_face_of_def)
    ultimately obtain a0 b0
          where p: p = a0 + b0 and a0 ∈ S b0 ∈ T and z: u • p = z
      by auto
    have lez: u • (x + y) ≤ z if x ∈ S y ∈ T for x y
      using S that by auto
    have sef: S ∩ {x. u • x = u • a0} exposed_face_of S
    proof (rule exposed_face_of_Int_supporting_hyperplane_le [OF ‹convex S›])
      show ⋀x. x ∈ S ⟹ u • x ≤ u • a0
        by (metis p z add_le_cancel_right inner_right_distrib lez [OF _ ‹b0 ∈
T›])
    qed
    have tef: T ∩ {x. u • x = u • b0} exposed_face_of T
```

```
    proof (rule exposed_face_of_Int_supporting_hyperplane_le [OF ‹convex T›])
      show ⋀x. x ∈ T ⟹ u · x ≤ u · b0
        by (metis p z add.commute add_le_cancel_right inner_right_distrib lez
[OF ‹a0 ∈ S›])
    qed
    have {x + y |x y. x ∈ S ∧ u · x = u · a0 ∧ y ∈ T ∧ u · y = u · b0} ⊆ F
      by (auto simp: feq) (metis inner_right_distrib p z)
    moreover have F ⊆ {x + y |x y. x ∈ S ∧ u · x = u · a0 ∧ y ∈ T ∧ u · y =
u · b0}
    proof −
      have ⋀x y. ⟦z = u · (x + y); x ∈ S; y ∈ T⟧
          ⟹ u · x = u · a0 ∧ u · y = u · b0
        by (smt (verit, best) z p ‹a0 ∈ S› ‹b0 ∈ T› inner_right_distrib lez)
      then show ?thesis
        using feq by blast
    qed
    ultimately have F = {x + y |x y. x ∈ S ∩ {x. u · x = u · a0} ∧ y ∈ T ∩
{x. u · x = u · b0}}
      by blast
    then show ?thesis
      by (rule that [OF sef tef])
  qed
qed

proposition exposed_face_of_parallel:
  T exposed_face_of S ⟷
        T face_of S ∧
        (∃ a b. S ⊆ {x. a · x ≤ b} ∧ T = S ∩ {x. a · x = b} ∧
              (T ≠ {} ⟶ T ≠ S ⟶ a ≠ 0) ∧
              (T ≠ S ⟶ (∀ w ∈ affine hull S. (w + a) ∈ affine hull S)))
  (is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
  proof (clarsimp simp: exposed_face_of_def)
    fix a b
    assume faceS: S ∩ {x. a · x = b} face_of S and Ssub: S ⊆ {x. a · x ≤ b}
    show ∃ c d. S ⊆ {x. c · x ≤ d} ∧
              S ∩ {x. a · x = b} = S ∩ {x. c · x = d} ∧
              (S ∩ {x. a · x = b} ≠ {} ⟶ S ∩ {x. a · x = b} ≠ S ⟶ c ≠ 0) ∧
             (S ∩ {x. a · x = b} ≠ S ⟶ (∀ w ∈ affine hull S. w + c ∈ affine hull
S))
    proof (cases affine hull S ∩ {x. −a · x ≤ −b} = {} ∨ affine hull S ⊆ {x. − a
· x ≤ − b})
      case True
      then show ?thesis
      proof
        assume affine hull S ∩ {x. − a · x ≤ − b} = {}
        then show ?thesis
          apply (rule_tac x=0 in exI)
```

```
        apply (rule_tac x=1 in exI)
        using hull_subset by fastforce
      next
        assume affine hull S ⊆ {x. - a • x ≤ - b}
        then show ?thesis
        apply (rule_tac x=0 in exI)
        apply (rule_tac x=0 in exI)
        using Ssub hull_subset by fastforce
      qed
    next
    case False
    then obtain a' b' where a' ≠ 0
     and le: affine hull S ∩ {x. a' • x ≤ b'} = affine hull S ∩ {x. - a • x ≤ - b}
     and eq: affine hull S ∩ {x. a' • x = b'} = affine hull S ∩ {x. - a • x = - b}
     and mem: ⋀w. w ∈ affine hull S ⟹ w + a' ∈ affine hull S
     using affine_parallel_slice affine_affine_hull by metis
    show ?thesis
    proof (intro conjI impI allI ballI exI)
      have *: S ⊆ - (affine hull S ∩ {x. P x}) ∪ affine hull S ∩ {x. Q x} ⟹ S
⊆ {x. ¬ P x ∨ Q x}
        for P Q
        using hull_subset by fastforce
      have S ⊆ {x. ¬ (a' • x ≤ b') ∨ a' • x = b'}
        by (rule *) (use le eq Ssub in auto)
      then show S ⊆ {x. - a' • x ≤ - b'}
        by auto
      show S ∩ {x. a • x = b} = S ∩ {x. - a' • x = - b'}
        using eq hull_subset [of S affine] by force
      show ⟦S ∩ {x. a • x = b} ≠ {}; S ∩ {x. a • x = b} ≠ S⟧ ⟹ - a' ≠ 0
        using ‹a' ≠ 0› by auto
      show w + - a' ∈ affine hull S
        if S ∩ {x. a • x = b} ≠ S w ∈ affine hull S for w
      proof -
        have w + 1 *R (w - (w + a')) ∈ affine hull S
          using affine_affine_hull mem mem_affine_3_minus that(2) by blast
        then show ?thesis  by simp
      qed
    qed
  qed
qed
next
  assume ?rhs then show ?lhs
    unfolding exposed_face_of_def by blast
qed
```

6.34.3 Extreme points of a set: its singleton faces

```
definition extreme_point_of :: ['a::real_vector, 'a set] ⇒ bool
                               (infixr (extreme'_point'_of) 50)
```

where x *extreme_point_of* $S \longleftrightarrow$
$x \in S \wedge (\forall a \in S.\ \forall b \in S.\ x \notin$ *open_segment* $a\ b)$

lemma *extreme_point_of_stillconvex*:
convex $S \Longrightarrow (x$ *extreme_point_of* $S \longleftrightarrow x \in S \wedge$ *convex*$(S - \{x\}))$
by (*fastforce simp add*: *convex_contains_segment extreme_point_of_def open_segment_def*)

lemma *face_of_singleton*:
$\{x\}$ *face_of* $S \longleftrightarrow x$ *extreme_point_of* S
by (*fastforce simp add*: *extreme_point_of_def face_of_def*)

lemma *extreme_point_not_in_REL_INTERIOR*:
fixes S :: *'a*::*real_normed_vector set*
shows $[\![x$ *extreme_point_of* $S;\ S \neq \{x\}]\!] \Longrightarrow x \notin$ *rel_interior* S
by (*metis disjoint_iff face_of_disjoint_rel_interior face_of_singleton insertI1*)

lemma *extreme_point_not_in_interior*:
fixes S :: *'a*::{*real_normed_vector*, *perfect_space*} *set*
assumes x *extreme_point_of* S **shows** $x \notin$ *interior* S
using *assms extreme_point_not_in_REL_INTERIOR interior_subset_rel_interior*
by *fastforce*

lemma *extreme_point_of_face*:
F *face_of* $S \Longrightarrow v$ *extreme_point_of* $F \longleftrightarrow v$ *extreme_point_of* $S \wedge v \in F$
by (*meson empty_subsetI face_of_face face_of_singleton insert_subset*)

lemma *extreme_point_of_convex_hull*:
x *extreme_point_of* (*convex hull* $S) \Longrightarrow x \in S$
using *hull_minimal* [*of* S (*convex hull* $S) - \{x\}$ *convex*]
using *hull_subset* [*of* S *convex*]
by (*force simp add*: *extreme_point_of_stillconvex*)

proposition *extreme_points_of_convex_hull*:
$\{x.\ x$ *extreme_point_of* (*convex hull* $S)\} \subseteq S$
using *extreme_point_of_convex_hull* **by** *auto*

lemma *extreme_point_of_empty* [*simp*]: $\neg$ (x *extreme_point_of* $\{\}$)
by (*simp add*: *extreme_point_of_def*)

lemma *extreme_point_of_singleton* [*iff*]: x *extreme_point_of* $\{a\} \longleftrightarrow x = a$
using *extreme_point_of_stillconvex* **by** *auto*

lemma *extreme_point_of_translation_eq*:
$(a + x)$ *extreme_point_of* (*image* $(\lambda x.\ a + x)\ S) \longleftrightarrow x$ *extreme_point_of* S
by (*auto simp*: *extreme_point_of_def*)

lemma *extreme_points_of_translation*:
$\{x.\ x$ *extreme_point_of* (*image* $(\lambda x.\ a + x)\ S)\} =$
$(\lambda x.\ a + x)$ ‘ $\{x.\ x$ *extreme_point_of* $S\}$

using *extreme_point_of_translation_eq*
by *auto* (*metis* (*no_types*, *lifting*) *image_iff mem_Collect_eq minus_add_cancel*)

lemma *extreme_points_of_translation_subtract*:
$\{x.\ x\ extreme\_point\_of\ (image\ (\lambda x.\ x - a)\ S)\} =$
$(\lambda x.\ x - a)\ `\ \{x.\ x\ extreme\_point\_of\ S\}$
using *extreme_points_of_translation* [*of* $-\ a\ S$]
by *simp*

lemma *extreme_point_of_Int*:
$[\![x\ extreme\_point\_of\ S;\ x\ extreme\_point\_of\ T]\!] \Longrightarrow x\ extreme\_point\_of\ (S \cap T)$
by (*simp add*: *extreme_point_of_def*)

lemma *extreme_point_of_Int_supporting_hyperplane_le*:
$[\![S \cap \{x.\ a \cdot x = b\} = \{c\};\ \bigwedge x.\ x \in S \Longrightarrow a \cdot x \leq b]\!] \Longrightarrow c\ extreme\_point\_of\ S$
by (*metis convex_singleton face_of_Int_supporting_hyperplane_le_strong face_of_singleton*)

lemma *extreme_point_of_Int_supporting_hyperplane_ge*:
$[\![S \cap \{x.\ a \cdot x = b\} = \{c\};\ \bigwedge x.\ x \in S \Longrightarrow a \cdot x \geq b]\!] \Longrightarrow c\ extreme\_point\_of\ S$
using *extreme_point_of_Int_supporting_hyperplane_le* [*of* $S\ -a\ -b\ c$]
by *simp*

lemma *exposed_point_of_Int_supporting_hyperplane_le*:
$[\![S \cap \{x.\ a \cdot x = b\} = \{c\};\ \bigwedge x.\ x \in S \Longrightarrow a \cdot x \leq b]\!] \Longrightarrow \{c\}\ exposed\_face\_of\ S$
unfolding *exposed_face_of_def*
by (*force simp*: *face_of_singleton extreme_point_of_Int_supporting_hyperplane_le*)

lemma *exposed_point_of_Int_supporting_hyperplane_ge*:
$[\![S \cap \{x.\ a \cdot x = b\} = \{c\};\ \bigwedge x.\ x \in S \Longrightarrow a \cdot x \geq b]\!] \Longrightarrow \{c\}\ exposed\_face\_of\ S$
using *exposed_point_of_Int_supporting_hyperplane_le* [*of* $S\ -a\ -b\ c$]
by *simp*

lemma *extreme_point_of_convex_hull_insert*:
assumes *finite S* $a \notin$ *convex hull S*
shows $a\ extreme\_point\_of\ (convex\ hull\ (insert\ a\ S))$
proof (*cases* $a \in S$)
case *False*
then show *?thesis*
using *face_of_convex_hulls* [*of insert a S* $\{a\}$] *assms*
by (*auto simp*: *face_of_singleton hull_same*)
qed (*use assms* **in** ‹*simp add*: *hull_inc*›)

6.34.4 Facets

definition *facet_of* :: [$'a$::*euclidean_space set*, $'a$ *set*] ⇒ *bool*
(**infixr** (*facet'_of*) *50*)
where $F\ facet\_of\ S \longleftrightarrow F\ face\_of\ S \wedge F \neq \{\} \wedge aff\_dim\ F = aff\_dim\ S - 1$

lemma *facet_of_empty* [*simp*]: ¬ *S facet_of* {}
by (*simp add*: *facet_of_def*)

lemma *facet_of_irrefl* [*simp*]: ¬ *S facet_of S*
by (*simp add*: *facet_of_def*)

lemma *facet_of_imp_face_of*: *F facet_of S* ⟹ *F face_of S*
by (*simp add*: *facet_of_def*)

lemma *facet_of_imp_subset*: *F facet_of S* ⟹ *F* ⊆ *S*
by (*simp add*: *face_of_imp_subset facet_of_def*)

lemma *hyperplane_facet_of_halfspace_le*:
a ≠ *0* ⟹ {*x*. *a* · *x* = *b*} *facet_of* {*x*. *a* · *x* ≤ *b*}
unfolding *facet_of_def hyperplane_eq_empty*
by (*auto simp*: *hyperplane_face_of_halfspace_ge hyperplane_face_of_halfspace_le*
Suc_leI of_nat_diff aff_dim_halfspace_le)

lemma *hyperplane_facet_of_halfspace_ge*:
a ≠ *0* ⟹ {*x*. *a* · *x* = *b*} *facet_of* {*x*. *a* · *x* ≥ *b*}
unfolding *facet_of_def hyperplane_eq_empty*
by (*auto simp*: *hyperplane_face_of_halfspace_le hyperplane_face_of_halfspace_ge*
Suc_leI of_nat_diff aff_dim_halfspace_ge)

lemma *facet_of_halfspace_le*:
F facet_of {*x*. *a* · *x* ≤ *b*} ⟷ *a* ≠ *0* ∧ *F* = {*x*. *a* · *x* = *b*}
(**is** *?lhs* = *?rhs*)
proof
assume *c*: *?lhs*
with *c facet_of_irrefl* **show** *?rhs*
by (*force simp*: *aff_dim_halfspace_le facet_of_def face_of_halfspace_le cong*:
conj_cong split: *if_split_asm*)
next
assume *?rhs* **then show** *?lhs*
by (*simp add*: *hyperplane_facet_of_halfspace_le*)
qed

lemma *facet_of_halfspace_ge*:
F facet_of {*x*. *a* · *x* ≥ *b*} ⟷ *a* ≠ *0* ∧ *F* = {*x*. *a* · *x* = *b*}
using *facet_of_halfspace_le* [*of F* −*a* −*b*] **by** *simp*

6.34.5 Edges: faces of affine dimension 1

definition *edge_of* :: ['*a*::*euclidean_space set*, '*a set*] ⇒ *bool* (**infixr** (*edge'_of*)
50)
where *e edge_of S* ⟷ *e face_of S* ∧ *aff_dim e* = *1*

lemma *edge_of_imp_subset*:

S edge_of T $\Longrightarrow$ *S* $\subseteq$ *T*
by (*simp add*: *edge_of_def face_of_imp_subset*)

6.34.6 Existence of extreme points

proposition *different_norm_3_collinear_points*:
fixes *a* :: *'a*::*euclidean_space*
assumes *x* $\in$ *open_segment a b norm*(*a*) = *norm*(*b*) *norm*(*x*) = *norm*(*b*)
shows *False*
proof −
obtain *u* **where** *norm* ((*1* − *u*) $*_R$ *a* + *u* $*_R$ *b*) = *norm b*
and *a* ≠ *b*
and *u01*: *0* < *u u* < *1*
using *assms* **by** (*auto simp*: *open_segment_image_interval if_splits*)
then have (*1* − *u*) $*_R$ *a* · (*1* − *u*) $*_R$ *a* + ((*1* − *u*) * *2*) $*_R$ *a* · *u* $*_R$ *b* =
(*1* − *u* * *u*) $*_R$ (*a* · *a*)
using *assms* **by** (*simp add*: *norm_eq algebra_simps inner_commute*)
then have (*1* − *u*) $*_R$ ((*1* − *u*) $*_R$ *a* · *a* + (*2* * *u*) $*_R$ *a* · *b*) =
(*1* − *u*) $*_R$ ((*1* + *u*) $*_R$ (*a* · *a*))
by (*simp add*: *algebra_simps*)
then have (*1* − *u*) $*_R$ (*a* · *a*) + (*2* * *u*) $*_R$ (*a* · *b*) = (*1* + *u*) $*_R$ (*a* · *a*)
using *u01* **by** *auto*
then have *a* · *b* = *a* · *a*
using *u01* **by** (*simp add*: *algebra_simps*)
then have *a* = *b*
using ‹*norm*(*a*) = *norm*(*b*)› *norm_eq vector_eq* **by** *fastforce*
then show *?thesis*
using ‹*a* ≠ *b*› **by** *force*
qed

proposition *extreme_point_exists_convex*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *compact S convex S S* ≠ {}
obtains *x* **where** *x extreme_point_of S*
proof −
obtain *x* **where** *x* $\in$ *S* **and** *xsup*: $\bigwedge$*y*. *y* $\in$ *S* $\Longrightarrow$ *norm y* ≤ *norm x*
using *distance_attains_sup* [*of S 0*] *assms* **by** *auto*
have *False* **if** *a* $\in$ *S b* $\in$ *S* **and** *x*: *x* $\in$ *open_segment a b* **for** *a b*
proof −
have *noax*: *norm a* ≤ *norm x* **and** *nobx*: *norm b* ≤ *norm x* **using** *xsup that*
by *auto*
have *a* ≠ *b*
using *empty_iff open_segment_idem x* **by** *auto*
show *False*
by (*metis dist_0_norm dist_decreases_open_segment noax nobx not_le x*)
qed
then show *?thesis*
by (*meson* ‹*x* $\in$ *S*› *extreme_point_of_def that*)
qed

6.34.7 Krein-Milman, the weaker form

proposition *Krein_Milman*:
fixes S :: *'a*::*euclidean_space set*
assumes *compact S convex S*
shows S = *closure*(*convex hull* {*x*. *x extreme_point_of S*})
proof (*cases* $S = \{\}$)
case *True* **then show** *?thesis* **by** *simp*
next
case *False*
have *closed S*
by (*simp add*: ‹*compact S*› *compact_imp_closed*)
have *closure* (*convex hull* {*x*. *x extreme_point_of S*}) ⊆ *S*
by (*simp add*: ‹*closed S*› *assms closure_minimal extreme_point_of_def hull_minimal*)
moreover have *u* ∈ *closure* (*convex hull* {*x*. *x extreme_point_of S*})
if *u* ∈ *S* **for** *u*
proof (*rule ccontr*)
assume *unot*: *u* ∉ *closure*(*convex hull* {*x*. *x extreme_point_of S*})
then obtain *a b* **where** *a* · *u* < *b*
and *ab*: ⋀*x*. *x* ∈ *closure*(*convex hull* {*x*. *x extreme_point_of S*}) ⟹ *b* < *a* · *x*
using *separating_hyperplane_closed_point* [*of closure*(*convex hull* {*x*. *x extreme_point_of S*})]
by *blast*
have *continuous_on S* ((·) *a*)
by (*rule continuous_intros*)+
then obtain *m* **where** *m* ∈ *S* **and** *m*: ⋀*y*. *y* ∈ *S* ⟹ *a* · *m* ≤ *a* · *y*
using *continuous_attains_inf* [*of S* λ*x*. *a* · *x*] ‹*compact S*› ‹*u* ∈ *S*›
by *auto*
define *T* **where** *T* = *S* ∩ {*x*. *a* · *x* = *a* · *m*}
have *m* ∈ *T*
by (*simp add*: *T_def* ‹*m* ∈ *S*›)
moreover have *compact T*
by (*simp add*: *T_def compact_Int_closed* [*OF* ‹*compact S*› *closed_hyperplane*])
moreover have *convex T*
by (*simp add*: *T_def convex_Int* [*OF* ‹*convex S*› *convex_hyperplane*])
ultimately obtain *v* **where** *v*: *v extreme_point_of T*
using *extreme_point_exists_convex* [*of T*] **by** *auto*
then have {*v*} *face_of T*
by (*simp add*: *face_of_singleton*)
also have *T face_of S*
by (*simp add*: *T_def m face_of_Int_supporting_hyperplane_ge* [*OF* ‹*convex S*›])
finally have *v extreme_point_of S*
by (*simp add*: *face_of_singleton*)
then have *b* < *a* · *v*
using *closure_subset* **by** (*simp add*: *closure_hull hull_inc ab*)
then show *False*
using ‹*a* · *u* < *b*› ‹{*v*} *face_of T*› *face_of_imp_subset m T_def that* **by** *fastforce*

```
  qed
  ultimately show ?thesis
    by blast
qed
```

Now the sharper form.

```
lemma Krein_Milman_Minkowski_aux:
  fixes S :: 'a::euclidean_space set
  assumes n: dim S = n and S: compact S convex S 0 ∈ S
    shows 0 ∈ convex hull {x. x extreme_point_of S}
using n S
proof (induction n arbitrary: S rule: less_induct)
  case (less n S) show ?case
  proof (cases 0 ∈ rel_interior S)
    case True with Krein_Milman less.prems
    show ?thesis
    by (metis subsetD convex_convex_hull convex_rel_interior_closure rel_interior_subset)
  next
    case False
    have rel_interior S ≠ {}
      by (simp add: rel_interior_convex_nonempty_aux less)
    then obtain c where c: c ∈ rel_interior S by blast
    obtain a where a ≠ 0
                and le_ay: ⋀y. y ∈ S ⟹ a · 0 ≤ a · y
                and less_ay: ⋀y. y ∈ rel_interior S ⟹ a · 0 < a · y
      by (blast intro: supporting_hyperplane_rel_boundary intro!: less False)
    have face: S ∩ {x. a · x = 0} face_of S
      using face_of_Int_supporting_hyperplane_ge le_ay ‹convex S› by auto
    then have co: compact (S ∩ {x. a · x = 0}) convex (S ∩ {x. a · x = 0})
    using less.prems by (blast intro: face_of_imp_compact face_of_imp_convex)+
    have a · y = 0 if y ∈ span (S ∩ {x. a · x = 0}) for y
    proof −
      have y ∈ span {x. a · x = 0}
        by (metis inf.cobounded2 span_mono subsetCE that)
      then show ?thesis
        by (blast intro: span_induct [OF _ subspace_hyperplane])
    qed
    then have dim (S ∩ {x. a · x = 0}) < n
      by (metis (no_types) less_ay c subsetD dim_eq_span inf.strict_order_iff
          inf_le1 ‹dim S = n› not_le rel_interior_subset span_0 span_base)
    then have 0 ∈ convex hull {x. x extreme_point_of (S ∩ {x. a · x = 0})}
      by (rule less.IH) (auto simp: co less.prems)
    then show ?thesis
     by (metis (mono_tags, lifting) Collect_mono_iff face extreme_point_of_face
hull_mono subset_iff)
  qed
qed
```

theorem *Krein_Milman_Minkowski*:
fixes S :: *'a::euclidean_space set*
assumes *compact S convex S*
shows S = *convex hull* {*x*. *x extreme_point_of S*}
proof
show $S \subseteq$ *convex hull* {*x*. *x extreme_point_of S*}
proof
fix *a* **assume** [*simp*]: $a \in S$
have *1*: *compact* ((+) (− *a*) ‘ *S*)
by (*simp add*: ‹*compact S*› *compact_translation_subtract cong*: *image_cong_simp*)
have *2*: *convex* ((+) (− *a*) ‘ *S*)
by (*simp add*: ‹*convex S*› *compact_translation_subtract*)
show *a_invex*: $a \in$ *convex hull* {*x*. *x extreme_point_of S*}
using *Krein_Milman_Minkowski_aux* [*OF refl 1 2*]
convex_hull_translation [*of* −*a*]
by (*auto simp*: *extreme_points_of_translation_subtract translation_assoc cong*: *image_cong_simp*)
qed
next
show *convex hull* {*x*. *x extreme_point_of S*} ⊆ *S*
using ‹*convex S*› *extreme_point_of_stillconvex subset_hull* **by** *fastforce*
qed

6.34.8 Applying it to convex hulls of explicitly indicated finite sets

corollary *Krein_Milman_polytope*:
fixes S :: *'a::euclidean_space set*
shows
finite S
⟹ *convex hull S* =
convex hull {*x*. *x extreme_point_of* (*convex hull S*)}
by (*simp add*: *Krein_Milman_Minkowski finite_imp_compact_convex_hull*)

lemma *extreme_points_of_convex_hull_eq*:
fixes S :: *'a::euclidean_space set*
shows
⟦*compact S*; ⋀*T*. *T* ⊂ *S* ⟹ *convex hull T* ≠ *convex hull S*⟧
⟹ {*x*. *x extreme_point_of* (*convex hull S*)} = *S*
by (*metis* (*full_types*) *Krein_Milman_Minkowski compact_convex_hull convex_convex_hull extreme_points_of_convex_hull psubsetI*)

lemma *extreme_point_of_convex_hull_eq*:
fixes S :: *'a::euclidean_space set*
shows
⟦*compact S*; ⋀*T*. *T* ⊂ *S* ⟹ *convex hull T* ≠ *convex hull S*⟧
⟹ (*x extreme_point_of* (*convex hull S*) ⟷ *x* ∈ *S*)
using *extreme_points_of_convex_hull_eq* **by** *auto*

lemma *extreme_point_of_convex_hull_convex_independent*:
fixes $S :: 'a::euclidean\_space\ set$
assumes *compact* S **and** S: $\bigwedge a.\ a \in S \Longrightarrow a \notin convex\ hull\ (S - \{a\})$
shows $(x\ extreme\_point\_of\ (convex\ hull\ S) \longleftrightarrow x \in S)$
proof −
have $convex\ hull\ T \neq convex\ hull\ S$ **if** $T \subset S$ **for** T
proof −
obtain a **where** $T \subseteq S\ a \in S\ a \notin T$ **using** ‹$T \subset S$› **by** *blast*
then show *?thesis*
by (*metis* (*full_types*) *Diff_eq_empty_iff Diff_insert0 S hull_mono hull_subset insert_Diff_single subsetCE*)
qed
then show *?thesis*
by (*rule extreme_point_of_convex_hull_eq* [*OF* ‹*compact S*›])
qed

lemma *extreme_point_of_convex_hull_affine_independent*:
fixes $S :: 'a::euclidean\_space\ set$
shows
¬ *affine_dependent S*
$\Longrightarrow (x\ extreme\_point\_of\ (convex\ hull\ S) \longleftrightarrow x \in S)$
by (*metis aff_independent_finite affine_dependent_def affine_hull_convex_hull extreme_point_of_convex_hull_convex_independent finite_imp_compact hull_inc*)

Elementary proofs exist, not requiring Euclidean spaces and all this development

lemma *extreme_point_of_convex_hull_2*:
fixes $x :: 'a::euclidean\_space$
shows $x\ extreme\_point\_of\ (convex\ hull\ \{a,b\}) \longleftrightarrow x = a \vee x = b$
by (*simp add*: *extreme_point_of_convex_hull_affine_independent*)

lemma *extreme_point_of_segment*:
fixes $x :: 'a::euclidean\_space$
shows $x\ extreme\_point\_of\ closed\_segment\ a\ b \longleftrightarrow x = a \vee x = b$
by (*simp add*: *extreme_point_of_convex_hull_2 segment_convex_hull*)

lemma *face_of_convex_hull_subset*:
fixes $S :: 'a::euclidean\_space\ set$
assumes *compact* S **and** T: $T\ face\_of\ (convex\ hull\ S)$
obtains S' **where** $S' \subseteq S\ T = convex\ hull\ S'$
proof
show $\{x.\ x\ extreme\_point\_of\ T\} \subseteq S$
using T *extreme_point_of_convex_hull extreme_point_of_face* **by** *blast*
show $T = convex\ hull\ \{x.\ x\ extreme\_point\_of\ T\}$
by (*metis Krein_Milman_Minkowski assms compact_convex_hull convex_convex_hull face_of_imp_compact face_of_imp_convex*)
qed

lemma *face_of_convex_hull_aux*:
assumes *eq*: $x *_R p = u *_R a + v *_R b + w *_R c$
and x: $u + v + w = x$ $x \neq 0$ **and** S: *affine* S $a \in S$ $b \in S$ $c \in S$
shows $p \in S$
proof −
have $p = (u *_R a + v *_R b + w *_R c) /_R x$
by (*metis* ‹$x \neq 0$› *eq mult.commute right_inverse scaleR_one scaleR_scaleR*)
moreover have *affine hull* {*a*,*b*,*c*} ⊆ S
by (*simp add*: S *hull_minimal*)
moreover have $(u *_R a + v *_R b + w *_R c) /_R x \in$ *affine hull* {*a*,*b*,*c*}
apply (*simp add*: *affine_hull_3*)
apply (*rule_tac* x=u/x **in** *exI*)
apply (*rule_tac* x=v/x **in** *exI*)
apply (*rule_tac* x=w/x **in** *exI*)
using x **apply** (*auto simp*: *field_split_simps*)
done
ultimately show *?thesis* **by** *force*
qed

proposition *face_of_convex_hull_insert_eq*:
fixes a :: $'a$ :: *euclidean_space*
assumes *finite* S **and** a: $a \notin$ *affine hull* S
shows (F *face_of* (*convex hull* (*insert* a S)) ⟷
F *face_of* (*convex hull* S) ∨
(∃F'. F' *face_of* (*convex hull* S) ∧ F = *convex hull* (*insert* a F')))
(**is** F *face_of ?CAS* ⟷ _)
proof *safe*
assume F: F *face_of ?CAS*
and ∗: ∄F'. F' *face_of convex hull* S ∧ F = *convex hull insert* a F'
obtain T **where** T: $T \subseteq$ *insert* a S **and** *FeqT*: F = *convex hull* T
by (*metis* F ‹*finite* S› *compact_insert finite_imp_compact face_of_convex_hull_subset*)
show F *face_of convex hull* S
proof (*cases* $a \in T$)
case *True*
have F = *convex hull insert* a (*convex hull* T ∩ *convex hull* S)
proof
have $T \subseteq$ *insert* a (*convex hull* T ∩ *convex hull* S)
using T *hull_subset* **by** *fastforce*
then show $F \subseteq$ *convex hull insert* a (*convex hull* T ∩ *convex hull* S)
by (*simp add*: *FeqT hull_mono*)
show *convex hull insert* a (*convex hull* T ∩ *convex hull* S) ⊆ F
by (*simp add*: *FeqT True hull_inc hull_minimal*)
qed
moreover have *convex hull* T ∩ *convex hull* S *face_of convex hull* S
by (*metis* F *FeqT convex_convex_hull face_of_slice hull_mono inf.absorb_iff2 subset_insertI*)
ultimately show *?thesis*

```
      using ∗ by force
  next
    case False
    then show ?thesis
     by (metis FeqT F T face_of_subset hull_mono subset_insert subset_insertI)
  qed
next
  assume F face_of convex hull S
  show F face_of ?CAS
    by (simp add: ‹F face_of convex hull S› a face_of_convex_hull_insert ‹finite
S›)
next
  fix F
  assume F: F face_of convex hull S
  show convex hull insert a F face_of ?CAS
  proof (cases S = {})
    case True
    then show ?thesis
      using F face_of_affine_eq by auto
  next
    case False
    have anotc: a ∉ convex hull S
      by (metis (no_types) a affine_hull_convex_hull hull_inc)
    show ?thesis
    proof (cases F = {})
      case True show ?thesis
      using anotc by (simp add: ‹F = {}› ‹finite S› extreme_point_of_convex_hull_insert
face_of_singleton)
    next
      case False
      have convex hull insert a F ⊆ ?CAS
       by (simp add: F a ‹finite S› convex_hull_subset face_of_convex_hull_insert
face_of_imp_subset hull_inc)
      moreover
      have (∃ y v. (1 − ub) ∗R a + ub ∗R b = (1 − v) ∗R a + v ∗R y ∧
                   0 ≤ v ∧ v ≤ 1 ∧ y ∈ F) ∧
            (∃ x u. (1 − uc) ∗R a + uc ∗R c = (1 − u) ∗R a + u ∗R x ∧
                   0 ≤ u ∧ u ≤ 1 ∧ x ∈ F)
        if ∗: (1 − ux) ∗R a + ux ∗R x
              ∈ open_segment ((1 − ub) ∗R a + ub ∗R b) ((1 − uc) ∗R a + uc ∗R
c)
          and 0 ≤ ub ub ≤ 1 0 ≤ uc uc ≤ 1 0 ≤ ux ux ≤ 1
          and b: b ∈ convex hull S and c: c ∈ convex hull S and x ∈ F
        for b c ub uc ux x
      proof −
        have xah: x ∈ affine hull S
         using F convex_hull_subset_affine_hull face_of_imp_subset ‹x ∈ F› by
blast
        have ah: b ∈ affine hull S c ∈ affine hull S
```

```
      using b c convex_hull_subset_affine_hull by blast+
    obtain v where ne: (1 - ub) *R a + ub *R b ≠ (1 - uc) *R a + uc *R c
      and eq: (1 - ux) *R a + ux *R x =
            (1 - v) *R ((1 - ub) *R a + ub *R b) + v *R ((1 - uc) *R a +
uc *R c)
      and 0 < v v < 1
      using * by (auto simp: in_segment)
    then have 0: ((1 - ux) - ((1 - v) * (1 - ub) + v * (1 - uc))) *R a +
              (ux *R x - (((1 - v) * ub) *R b + (v * uc) *R c)) = 0
      by (auto simp: algebra_simps)
    then have ((1 - ux) - ((1 - v) * (1 - ub) + v * (1 - uc))) *R a =
            ((1 - v) * ub) *R b + (v * uc) *R c + (-ux) *R x
      by (auto simp: algebra_simps)
    then have a ∈ affine hull S if 1 - ux - ((1 - v) * (1 - ub) + v * (1 -
uc)) ≠ 0
       by (rule face_of_convex_hull_aux) (use b c xah ah that in ‹auto simp:
algebra_simps›)
    then have 1 - ux - ((1 - v) * (1 - ub) + v * (1 - uc)) = 0
      using a by blast
    with 0 have equx: (1 - v) * ub + v * uc = ux
      and uxx: ux *R x = (((1 - v) * ub) *R b + (v * uc) *R c)
      by auto (auto simp: algebra_simps)
    show ?thesis
    proof (cases uc = 0)
      case True
      then show ?thesis
        using equx ‹0 ≤ ub› ‹ub ≤ 1› ‹v < 1› uxx ‹x ∈ F› by force
    next
      case False
      show ?thesis
      proof (cases ub = 0)
        case True
        then show ?thesis
          using equx ‹0 ≤ uc› ‹uc ≤ 1› ‹0 < v› uxx ‹x ∈ F› by force
      next
        case False
        then have 0 < ub 0 < uc
          using ‹uc ≠ 0› ‹0 ≤ ub› ‹0 ≤ uc› by auto
        then have (1 - v) * ub > 0 v * uc > 0
          by (simp_all add: ‹0 < uc› ‹0 < v› ‹v < 1›)
        then have ux ≠ 0
          using equx ‹0 < v› by auto
        have b ∈ F ∧ c ∈ F
        proof (cases b = c)
          case True
          then show ?thesis
          by (metis ‹ux ≠ 0› equx real_vector.scale_cancel_left scaleR_add_left
uxx ‹x ∈ F›)
        next
```

```
            case False
            have x = (((1 - v) * ub) *R b + (v * uc) *R c) /R ux
                by (metis ‹ux ≠ 0› uxx mult.commute right_inverse scaleR_one
scaleR_scaleR)
            also have ... = (1 - v * uc / ux) *R b + (v * uc / ux) *R c
              using ‹ux ≠ 0› equx apply (auto simp: field_split_simps)
           by (metis add.commute add_diff_eq add_divide_distrib diff_add_cancel
scaleR_add_left)
            finally have x = (1 - v * uc / ux) *R b + (v * uc / ux) *R c .
            then have x ∈ open_segment b c
              apply (simp add: in_segment ‹b ≠ c›)
              apply (rule_tac x=(v * uc) / ux in exI)
              using ‹0 ≤ ux› ‹ux ≠ 0› ‹0 < uc› ‹0 < v› ‹0 < ub› ‹v < 1› equx
              apply (force simp: field_split_simps)
              done
            then show ?thesis
              by (rule face_ofD [OF F _ b c ‹x ∈ F›])
          qed
          with ‹0 ≤ ub› ‹ub ≤ 1› ‹0 ≤ uc› ‹uc ≤ 1› show ?thesis by blast
        qed
      qed
    qed
    moreover have convex hull F = F
      by (meson F convex_hull_eq face_of_imp_convex)
    ultimately show ?thesis
      unfolding face_of_def by (fastforce simp: convex_hull_insert_alt ‹S ≠
{}› ‹F ≠ {}›)
  qed
 qed
qed

lemma face_of_convex_hull_insert2:
 fixes a :: 'a :: euclidean_space
 assumes S: finite S and a: a ∉ affine hull S and F: F face_of convex hull S
 shows convex hull (insert a F) face_of convex hull (insert a S)
 by (metis F face_of_convex_hull_insert_eq [OF S a])

proposition face_of_convex_hull_affine_independent:
 fixes S :: 'a::euclidean_space set
 assumes ¬ affine_dependent S
   shows (T face_of (convex hull S) ⟷ (∃ c. c ⊆ S ∧ T = convex hull c))
       (is ?lhs = ?rhs)
proof
 assume ?lhs
 then show ?rhs
  by (meson ‹T face_of convex hull S› aff_independent_finite assms face_of_convex_hull_subset
finite_imp_compact)
next
 assume ?rhs
```

```
  then obtain C where C ⊆ S and T: T = convex hull C
    by blast
  have affine hull C ∩ affine hull (S − C) = {}
    by (intro disjoint_affine_hull [OF assms ‹C ⊆ S›], auto)
  then have affine hull C ∩ convex hull (S − C) = {}
    using convex_hull_subset_affine_hull by fastforce
  then show ?lhs
    by (metis face_of_convex_hulls ‹C ⊆ S› aff_independent_finite assms T)
qed

lemma facet_of_convex_hull_affine_independent:
  fixes S :: 'a::euclidean_space set
  assumes ¬ affine_dependent S
    shows T facet_of (convex hull S) ⟷
          T ≠ {} ∧ (∃u. u ∈ S ∧ T = convex hull (S − {u}))
          (is ?lhs = ?rhs)
proof
  assume ?lhs
  then have T face_of (convex hull S) T ≠ {}
       and afft: aff_dim T = aff_dim (convex hull S) − 1
    by (auto simp: facet_of_def)
  then obtain c where c ⊆ S and c: T = convex hull c
    by (auto simp: face_of_convex_hull_affine_independent [OF assms])
  then have affs: aff_dim S = aff_dim c + 1
    by (metis aff_dim_convex_hull afft eq_diff_eq)
  have ¬ affine_dependent c
    using ‹c ⊆ S› affine_dependent_subset assms by blast
  with affs have card (S − c) = 1
    by (smt (verit) ‹c ⊆ S› aff_dim_affine_independent aff_independent_finite
assms card_Diff_subset
        card_mono of_nat_diff of_nat_eq_1_iff)
  then obtain u where u: u ∈ S − c
   by (metis DiffI ‹c ⊆ S› aff_independent_finite assms cancel_comm_monoid_add_class.diff_cancel
              card_Diff_subset subsetI subset_antisym zero_neq_one)
  then have u: S = insert u c
    by (metis Diff_subset ‹c ⊆ S› ‹card (S − c) = 1› card_1_singletonE dou-
ble_diff insert_Diff insert_subset singletonD)
  have T = convex hull (c − {u})
   by (metis Diff_empty Diff_insert0 ‹T facet_of convex hull S› c facet_of_irrefl
insert_absorb u)
  with ‹T ≠ {}› show ?rhs
    using c u by auto
next
  assume ?rhs
  then obtain u where T ≠ {} u ∈ S and u: T = convex hull (S − {u})
    by (force simp: facet_of_def)
  then have ¬ S ⊆ {u}
    using ‹T ≠ {}› u by auto
  have aff_dim (S − {u}) = aff_dim S − 1
```

```
    using assms ‹u ∈ S›
    unfolding affine_dependent_def
    by (metis add_diff_cancel_right' aff_dim_insert insert_Diff [of u S])
  then have aff_dim (convex hull (S − {u})) = aff_dim (convex hull S) − 1
    by (simp add: aff_dim_convex_hull)
  then show ?lhs
   by (metis Diff_subset ‹T ≠ {}› assms face_of_convex_hull_affine_independent
facet_of_def u)
qed

lemma facet_of_convex_hull_affine_independent_alt:
  fixes S :: 'a::euclidean_space set
  assumes ¬ affine_dependent S
  shows (T facet_of (convex hull S)) ⟷ 2 ≤ card S ∧ (∃ u. u ∈ S ∧ T = convex
hull (S − {u})))
        (is ?lhs = ?rhs)
proof
  assume L: ?lhs
  then obtain x where
    x ∈ S and x: T = convex hull (S − {x}) and finite S
   using assms facet_of_convex_hull_affine_independent aff_independent_finite
by blast
  moreover have Suc (Suc 0) ≤ card S
    using L  x ‹x ∈ S› ‹finite S›
     by (metis Suc_leI assms card.remove convex_hull_eq_empty card_gt_0_iff
facet_of_convex_hull_affine_independent finite_Diff not_less_eq_eq)
  ultimately show ?rhs
    by auto
next
  assume ?rhs then show ?lhs
    using assms
   by (auto simp: facet_of_convex_hull_affine_independent Set.subset_singleton_iff)
qed

lemma segment_face_of:
  assumes (closed_segment a b) face_of S
  shows a extreme_point_of S b extreme_point_of S
proof −
  have as: {a} face_of S
   by (metis (no_types) assms convex_hull_singleton empty_iff extreme_point_of_convex_hull_insert
face_of_face face_of_singleton finite.emptyI finite.insertI insert_absorb insert_iff
segment_convex_hull)
  moreover have {b} face_of S
  proof −
    have b ∈ convex hull {a} ∨ b extreme_point_of convex hull {b, a}
     by (meson extreme_point_of_convex_hull_insert finite.emptyI finite.insertI)
    moreover have closed_segment a b = convex hull {b, a}
      using closed_segment_commute segment_convex_hull by blast
    ultimately show ?thesis
```

by (*metis as assms face_of_face convex_hull_singleton empty_iff face_of_singleton insertE*)
qed
ultimately show *a extreme_point_of S b extreme_point_of S*
using *face_of_singleton* **by** *blast+*
qed

proposition *Krein_Milman_frontier*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *convex S compact S*
shows *S* = *convex hull* (*frontier S*)
(**is** *?lhs* = *?rhs*)
proof
have *?lhs* ⊆ *convex hull* {*x*. *x extreme_point_of S*}
using *Krein_Milman_Minkowski assms* **by** *blast*
also have ... ⊆ *?rhs*
proof (*rule hull_mono*)
show {*x*. *x extreme_point_of S*} ⊆ *frontier S*
using *closure_subset*
by (*auto simp*: *frontier_def extreme_point_not_in_interior extreme_point_of_def*)
qed
finally show *?lhs* ⊆ *?rhs* **.**
next
have *?rhs* ⊆ *convex hull S*
by (*metis Diff_subset* ‹*compact S*› *closure_closed compact_eq_bounded_closed frontier_def hull_mono*)
also have ... ⊆ *?lhs*
by (*simp add*: ‹*convex S*› *hull_same*)
finally show *?rhs* ⊆ *?lhs* **.**
qed

6.34.9 Polytopes

definition *polytope* **where**
polytope S ≡ ∃ *v*. *finite v* ∧ *S* = *convex hull v*

lemma *polytope_translation_eq*: *polytope* ((+) *a* ' *S*) ⟷ *polytope S*
unfolding *polytope_def*
by (*metis* (*no_types*, *opaque_lifting*) *add.left_inverse convex_hull_translation finite_imageI image_add_0 translation_assoc*)

lemma *polytope_linear_image*: ⟦*linear f*; *polytope p*⟧ ⟹ *polytope*(*image f p*)
unfolding *polytope_def* **using** *convex_hull_linear_image* **by** *blast*

lemma *polytope_empty*: *polytope* {}
using *convex_hull_empty polytope_def* **by** *blast*

lemma *polytope_convex_hull*: *finite S* ⟹ *polytope*(*convex hull S*)

using *polytope_def* **by** *auto*

lemma *polytope_Times*: ⟦*polytope S*; *polytope T*⟧ ⟹ *polytope*(*S* × *T*)
unfolding *polytope_def*
by (*metis finite_cartesian_product convex_hull_Times*)

lemma *face_of_polytope_polytope*:
fixes *S* :: *'a*::*euclidean_space set*
shows ⟦*polytope S*; *F face_of S*⟧ ⟹ *polytope F*
unfolding *polytope_def*
by (*meson face_of_convex_hull_subset finite_imp_compact finite_subset*)

lemma *finite_polytope_faces*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *polytope S*
shows *finite* {*F*. *F face_of S*}
proof −
obtain *v* **where** *finite v S* = *convex hull v*
using *assms polytope_def* **by** *auto*
have *finite* ((*hull*) *convex* ‘ {*T*. *T* ⊆ *v*})
by (*simp add*: ‹*finite v*›)
moreover have {*F*. *F face_of S*} ⊆ ((*hull*) *convex* ‘ {*T*. *T* ⊆ *v*})
by (*metis* (*no_types*, *lifting*) ‹*finite v*› ‹*S* = *convex hull v*› *face_of_convex_hull_subset finite_imp_compact image_eqI mem_Collect_eq subsetI*)
ultimately show *?thesis*
by (*blast intro*: *finite_subset*)
qed

lemma *finite_polytope_facets*:
assumes *polytope S*
shows *finite* {*T*. *T facet_of S*}
by (*simp add*: *assms facet_of_def finite_polytope_faces*)

lemma *polytope_scaling*:
assumes *polytope S* **shows** *polytope* (*image* (λ*x*. *c* ∗*R* *x*) *S*)
by (*simp add*: *assms polytope_linear_image*)

lemma *polytope_imp_compact*:
fixes *S* :: *'a*::*real_normed_vector set*
shows *polytope S* ⟹ *compact S*
by (*metis finite_imp_compact_convex_hull polytope_def*)

lemma *polytope_imp_convex*: *polytope S* ⟹ *convex S*
by (*metis convex_convex_hull polytope_def*)

lemma *polytope_imp_closed*:
fixes *S* :: *'a*::*real_normed_vector set*
shows *polytope S* ⟹ *closed S*
by (*simp add*: *compact_imp_closed polytope_imp_compact*)

lemma *polytope_imp_bounded*:
fixes $S :: 'a::real\_normed\_vector\ set$
shows $polytope\ S \Longrightarrow bounded\ S$
by (*simp add*: *compact_imp_bounded polytope_imp_compact*)

lemma *polytope_interval*: $polytope(cbox\ a\ b)$
unfolding *polytope_def* **by** (*meson closed_interval_as_convex_hull*)

lemma *polytope_sing*: $polytope\ \{a\}$
using *polytope_def* **by** *force*

lemma *face_of_polytope_insert*:
$[\![polytope\ S;\ a \notin affine\ hull\ S;\ F\ face\_of\ S]\!] \Longrightarrow F\ face\_of\ convex\ hull\ (insert\ a\ S)$
by (*metis* (*no_types*, *lifting*) *affine_hull_convex_hull face_of_convex_hull_insert hull_insert polytope_def*)

proposition *face_of_polytope_insert2*:
fixes $a :: 'a :: euclidean\_space$
assumes $polytope\ S\ a \notin affine\ hull\ S\ F\ face\_of\ S$
shows $convex\ hull\ (insert\ a\ F)\ face\_of\ convex\ hull\ (insert\ a\ S)$
proof −
obtain V **where** $finite\ V\ S = convex\ hull\ V$
using *assms* **by** (*auto simp*: *polytope_def*)
then have $convex\ hull\ (insert\ a\ F)\ face\_of\ convex\ hull\ (insert\ a\ V)$
using *affine_hull_convex_hull assms face_of_convex_hull_insert2* **by** *blast*
then show *?thesis*
by (*metis* ‹$S = convex\ hull\ V$› *hull_insert*)
qed

6.34.10 Polyhedra

definition *polyhedron* **where**
$polyhedron\ S \equiv$
$\exists F.\ finite\ F \wedge$
$S = \bigcap F \wedge$
$(\forall h \in F.\ \exists a\ b.\ a \neq 0 \wedge h = \{x.\ a \cdot x \leq b\})$

lemma *polyhedron_Int* [*intro*,*simp*]:
$[\![polyhedron\ S;\ polyhedron\ T]\!] \Longrightarrow polyhedron\ (S \cap T)$
apply (*clarsimp simp add*: *polyhedron_def*)
subgoal for $F\ G$
by (*rule_tac* $x{=}F \cup G$ **in** *exI*, *auto*)
done

lemma *polyhedron_UNIV* [*iff*]: $polyhedron\ UNIV$
using *polyhedron_def* **by** *auto*

lemma *polyhedron_Inter* [*intro*,*simp*]:
  ⟦*finite F*; ⋀*S*. *S* ∈ *F* ⟹ *polyhedron S*⟧ ⟹ *polyhedron*(⋂ *F*)
  by (*induction F rule*: *finite_induct*) *auto*

lemma *polyhedron_empty* [*iff*]: *polyhedron* ({} :: *'a* :: *euclidean_space set*)
proof −
  define *i*::*'a* **where** (*i* ≡ *SOME i*. *i* ∈ *Basis*)
  have ∃ *a*. *a* ≠ *0* ∧ (∃ *b*. {*x*. *i* ⋅ *x* ≤ −*1*} = {*x*. *a* ⋅ *x* ≤ *b*})
    by (*rule_tac x*=*i* **in** *exI*) (*force simp*: *i_def SOME_Basis nonzero_Basis*)
  moreover have ∃ *a b*. *a* ≠ *0* ∧ {*x*. −*i* ⋅ *x* ≤ − *1*} = {*x*. *a* ⋅ *x* ≤ *b*}
    by (*metis Basis_zero SOME_Basis i_def neg_0_equal_iff_equal*)
  ultimately show *?thesis*
    unfolding *polyhedron_def*
    by (*rule_tac x*={{*x*. *i* ⋅ *x* ≤ −*1*}, {*x*. −*i* ⋅ *x* ≤ −*1*}} **in** *exI*) *force*
qed

lemma *polyhedron_halfspace_le*:
  fixes *a* :: *'a* :: *euclidean_space*
  shows *polyhedron* {*x*. *a* ⋅ *x* ≤ *b*}
proof (*cases a* = *0*)
  case *True* **then show** *?thesis* **by** *auto*
next
  case *False*
  then show *?thesis*
    unfolding *polyhedron_def*
    by (*rule_tac x*={{*x*. *a* ⋅ *x* ≤ *b*}} **in** *exI*) *auto*
qed

lemma *polyhedron_halfspace_ge*:
  fixes *a* :: *'a* :: *euclidean_space*
  shows *polyhedron* {*x*. *a* ⋅ *x* ≥ *b*}
  using *polyhedron_halfspace_le* [*of* −*a* −*b*] **by** *simp*

lemma *polyhedron_hyperplane*:
  fixes *a* :: *'a* :: *euclidean_space*
  shows *polyhedron* {*x*. *a* ⋅ *x* = *b*}
proof −
  have {*x*. *a* ⋅ *x* = *b*} = {*x*. *a* ⋅ *x* ≤ *b*} ∩ {*x*. *a* ⋅ *x* ≥ *b*}
    by *force*
  then show *?thesis*
    by (*simp add*: *polyhedron_halfspace_ge polyhedron_halfspace_le*)
qed

lemma *affine_imp_polyhedron*:
  fixes *S* :: *'a* :: *euclidean_space set*
  shows *affine S* ⟹ *polyhedron S*
 by (*metis affine_hull_finite_intersection_hyperplanes hull_same polyhedron_Inter polyhedron_hyperplane*)

lemma *polyhedron_imp_closed*:
fixes $S :: 'a :: euclidean\_space\ set$
shows $polyhedron\ S \Longrightarrow closed\ S$
by (*metis closed_Inter closed_halfspace_le polyhedron_def*)

lemma *polyhedron_imp_convex*:
fixes $S :: 'a :: euclidean\_space\ set$
shows $polyhedron\ S \Longrightarrow convex\ S$
by (*metis convex_Inter convex_halfspace_le polyhedron_def*)

lemma *polyhedron_affine_hull*:
fixes $S :: 'a :: euclidean\_space\ set$
shows $polyhedron(affine\ hull\ S)$
by (*simp add*: *affine_imp_polyhedron*)

6.34.11 Canonical polyhedron representation making facial structure explicit

proposition *polyhedron_Int_affine*:
fixes $S :: 'a :: euclidean\_space\ set$
shows $polyhedron\ S \longleftrightarrow$
$(\exists F.\ finite\ F \wedge S = (affine\ hull\ S) \cap \bigcap F \wedge$
$(\forall h \in F.\ \exists a\ b.\ a \neq 0 \wedge h = \{x.\ a \cdot x \leq b\}))$
by (*metis hull_subset inf.absorb_iff2 polyhedron_Int polyhedron_affine_hull polyhedron_def*)

proposition *rel_interior_polyhedron_explicit*:
assumes *finite F*
and *seq*: $S = affine\ hull\ S \cap \bigcap F$
and *faceq*: $\bigwedge h.\ h \in F \Longrightarrow a\ h \neq 0 \wedge h = \{x.\ a\ h \cdot x \leq b\ h\}$
and *psub*: $\bigwedge F'.\ F' \subset F \Longrightarrow S \subset affine\ hull\ S \cap \bigcap F'$
shows $rel\_interior\ S = \{x \in S.\ \forall h \in F.\ a\ h \cdot x < b\ h\}$
proof −
have *rels*: $\bigwedge x.\ x \in rel\_interior\ S \Longrightarrow x \in S$
by (*meson IntE mem_rel_interior*)
moreover have $a\ i \cdot x < b\ i$ **if** x: $x \in rel\_interior\ S$ **and** $i \in F$ **for** $x\ i$
proof −
have *fif*: $F - \{i\} \subset F$
using ‹$i \in F$› *Diff_insert_absorb Diff_subset set_insert psubsetI* **by** *blast*
then have $S \subset affine\ hull\ S \cap \bigcap (F - \{i\})$
by (*rule psub*)
then obtain z **where** *ssub*: $S \subseteq \bigcap (F - \{i\})$ **and** *zint*: $z \in \bigcap (F - \{i\})$
and $z \notin S$ **and** *zaff*: $z \in affine\ hull\ S$
by *auto*
have $z \neq x$
using ‹$z \notin S$› *rels x* **by** *blast*
have $z \notin affine\ hull\ S \cap \bigcap F$
using ‹$z \notin S$› *seq* **by** *auto*

```
    then have aiz: a i · z > b i
      using faceq zint zaff by fastforce
    obtain e where e > 0 x ∈ S and e: ball x e ∩ affine hull S ⊆ S
      using x by (auto simp: mem_rel_interior_ball)
    then have ins: ⋀y. ⟦norm (x − y) < e; y ∈ affine hull S⟧ ⟹ y ∈ S
      by (metis IntI subsetD dist_norm mem_ball)
    define ξ where ξ = min (1/2) (e / 2 / norm(z − x))
    have norm (ξ *R x − ξ *R z) = norm (ξ *R (x − z))
      by (simp add: ξ_def algebra_simps norm_mult)
    also have ... = ξ * norm (x − z)
      using ‹e > 0› by (simp add: ξ_def)
    also have ... < e
       using ‹z ≠ x› ‹e > 0› by (simp add: ξ_def min_def field_split_simps
norm_minus_commute)
    finally have lte: norm (ξ *R x − ξ *R z) < e .
    have ξ_aff: ξ *R z + (1 − ξ) *R x ∈ affine hull S
      by (simp add: ‹x ∈ S› hull_inc mem_affine zaff)
    have ξ *R z + (1 − ξ) *R x ∈ S
      using ins [OF _ ξ_aff] by (simp add: algebra_simps lte)
    then obtain l where l: 0 < l l < 1 and ls: (l *R z + (1 − l) *R x) ∈ S
      using ‹e > 0› ‹z ≠ x›
      by (rule_tac l = ξ in that) (auto simp: ξ_def)
    then have i: l *R z + (1 − l) *R x ∈ i
      using seq ‹i ∈ F› by auto
    have b i * l + (a i · x) * (1 − l) < a i · (l *R z + (1 − l) *R x)
      using l by (simp add: algebra_simps aiz)
    also have ... ≤ b i using i l
      using faceq mem_Collect_eq ‹i ∈ F› by blast
    finally have (a i · x) * (1 − l) < b i * (1 − l)
      by (simp add: algebra_simps)
    with l show ?thesis
      by simp
  qed
  moreover have x ∈ rel_interior S
        if x ∈ S and less: ⋀h. h ∈ F ⟹ a h · x < b h for x
  proof −
    have 1: ⋀h. h ∈ F ⟹ x ∈ interior h
      by (metis interior_halfspace_le mem_Collect_eq less faceq)
    have 2: ⋀y. ⟦∀h∈F. y ∈ interior h; y ∈ affine hull S⟧ ⟹ y ∈ S
      by (metis IntI Inter_iff subsetD interior_subset seq)
    show ?thesis
      apply (simp add: rel_interior ‹x ∈ S›)
      apply (rule_tac x=⋂h∈F. interior h in exI)
      apply (auto simp: ‹finite F› open_INT 1 2)
      done
  qed
  ultimately show ?thesis by blast
qed
```

lemma *polyhedron_Int_affine_parallel*:
fixes $S :: 'a :: euclidean\_space\ set$
shows $polyhedron\ S \longleftrightarrow$
$(\exists F.\ finite\ F \wedge$
$S = (affine\ hull\ S) \cap (\bigcap F) \wedge$
$(\forall h \in F.\ \exists a\ b.\ a \neq 0 \wedge h = \{x.\ a \cdot x \leq b\} \wedge$
$(\forall x \in affine\ hull\ S.\ (x + a) \in affine\ hull\ S)))$
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then obtain *F* **where** *finite F* **and** *seq*: $S = (affine\ hull\ S) \cap \bigcap F$
and *faces*: $\bigwedge h.\ h \in F \Longrightarrow \exists a\ b.\ a \neq 0 \wedge h = \{x.\ a \cdot x \leq b\}$
by (*fastforce simp add*: *polyhedron_Int_affine*)
then obtain *a b* **where** *ab*: $\bigwedge h.\ h \in F \Longrightarrow a\ h \neq 0 \wedge h = \{x.\ a\ h \cdot x \leq b\ h\}$
by *metis*
show *?rhs*
proof −
have $\exists a'\ b'.\ a' \neq 0 \wedge$
$affine\ hull\ S \cap \{x.\ a' \cdot x \leq b'\} = affine\ hull\ S \cap h \wedge$
$(\forall w \in affine\ hull\ S.\ (w + a') \in affine\ hull\ S)$
if $h \in F$ $\neg(affine\ hull\ S \subseteq h)$ **for** *h*
proof −
have $a\ h \neq 0$ **and** $h = \{x.\ a\ h \cdot x \leq b\ h\}$ $h \cap \bigcap F = \bigcap F$
using ‹$h \in F$› *ab* **by** *auto*
then have $(affine\ hull\ S) \cap \{x.\ a\ h \cdot x \leq b\ h\} \neq \{\}$
by (*metis affine_hull_eq_empty inf.absorb_iff1 inf_assoc inf_bot_left seq that(2)*)
moreover have ¬ $(affine\ hull\ S \subseteq \{x.\ a\ h \cdot x \leq b\ h\})$
using ‹$h = \{x.\ a\ h \cdot x \leq b\ h\}$› *that(2)* **by** *blast*
ultimately show *?thesis*
using *affine_parallel_slice* [*of affine hull S*]
by (*metis* ‹$h = \{x.\ a\ h \cdot x \leq b\ h\}$› *affine_affine_hull*)
qed
then obtain *a b*
where *ab*: $\bigwedge h.\ [\![h \in F;\ \neg\ (affine\ hull\ S \subseteq h)]\!]$
$\Longrightarrow a\ h \neq 0 \wedge$
$affine\ hull\ S \cap \{x.\ a\ h \cdot x \leq b\ h\} = affine\ hull\ S \cap h \wedge$
$(\forall w \in affine\ hull\ S.\ (w + a\ h) \in affine\ hull\ S)$
by *metis*
let *?F* = $(\lambda h.\ \{x.\ a\ h \cdot x \leq b\ h\})$ ‘ $\{h \in F.\ \neg\ affine\ hull\ S \subseteq h\}$
show *?thesis*
proof (*intro exI conjI*)
show *finite ?F*
using ‹*finite F*› **by** *force*
show $S = affine\ hull\ S \cap \bigcap$ *?F*
by (*subst seq*) (*auto simp*: *ab INT_extend_simps*)
qed (*use ab* **in** *blast*)
qed

next
assume *?rhs* **then show** *?lhs*
by (*metis polyhedron_Int_affine*)
qed

proposition *polyhedron_Int_affine_parallel_minimal*:
fixes $S :: 'a :: euclidean\_space\ set$
shows *polyhedron* $S \longleftrightarrow$
$(\exists F.\ finite\ F \wedge$
$S = (affine\ hull\ S) \cap (\bigcap F) \wedge$
$(\forall h \in F.\ \exists a\ b.\ a \neq 0 \wedge h = \{x.\ a \cdot x \leq b\} \wedge$
$(\forall x \in affine\ hull\ S.\ (x + a) \in affine\ hull\ S)) \wedge$
$(\forall F'.\ F' \subset F \longrightarrow S \subset (affine\ hull\ S) \cap (\bigcap F')))$
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then obtain *f0*
where *f0*: *finite f0*
$S = (affine\ hull\ S) \cap (\bigcap f0)$
(**is** *?P f0*)
$\forall h \in f0.\ \exists a\ b.\ a \neq 0 \wedge h = \{x.\ a \cdot x \leq b\} \wedge$
$(\forall x \in affine\ hull\ S.\ (x + a) \in affine\ hull\ S)$
(**is** *?Q f0*)
by (*force simp*: *polyhedron_Int_affine_parallel*)
define *n* **where** $n = (LEAST\ n.\ \exists F.\ card\ F = n \wedge finite\ F \wedge ?P\ F \wedge ?Q\ F)$
have *nf*: $\exists F.\ card\ F = n \wedge finite\ F \wedge ?P\ F \wedge ?Q\ F$
apply (*simp add*: *n_def*)
apply (*rule LeastI* [**where** $k = card\ f0$])
using *f0* **apply** *auto*
done
then obtain *F* **where** *F*: $card\ F = n$ *finite F* **and** *seq*: *?P F* **and** *aff*: *?Q F*
by *blast*
then have $\neg$ (*finite g* $\wedge$ *?P g* $\wedge$ *?Q g*) **if** $card\ g < n$ **for** *g*
using *that* **by** (*auto simp*: *n_def dest*!: *not_less_Least*)
then have $*$: $\neg$ (*?P g* $\wedge$ *?Q g*) **if** $g \subset F$ **for** *g*
using *that* ‹*finite F*› *psubset_card_mono* ‹$card\ F = n$›
by (*metis finite_Int inf.strict_order_iff*)
have *1*: $\bigwedge F'.\ F' \subset F \Longrightarrow S \subseteq affine\ hull\ S \cap \bigcap F'$
by (*subst seq*) *blast*
have *2*: $S \neq affine\ hull\ S \cap \bigcap F'$ **if** $F' \subset F$ **for** F'
using $*$ [*OF that*] **by** (*metis IntE aff inf.strict_order_iff that*)
show *?rhs*
by (*metis* ‹*finite F*› *seq aff psubsetI 1 2*)
next
assume *?rhs* **then show** *?lhs*
by (*auto simp*: *polyhedron_Int_affine_parallel*)
qed

lemma *polyhedron_Int_affine_minimal*:
fixes $S :: 'a :: euclidean\_space\ set$
shows $polyhedron\ S \longleftrightarrow$
$(\exists F.\ finite\ F \wedge S = (affine\ hull\ S) \cap \bigcap F \wedge$
$(\forall h \in F.\ \exists a\ b.\ a \neq 0 \wedge h = \{x.\ a \cdot x \leq b\}) \wedge$
$(\forall F'.\ F' \subset F \longrightarrow S \subset (affine\ hull\ S) \cap \bigcap F'))$
by (*metis polyhedron_Int_affine polyhedron_Int_affine_parallel_minimal*)

proposition *facet_of_polyhedron_explicit*:
assumes *finite F*
and *seq*: $S = affine\ hull\ S \cap \bigcap F$
and *faceq*: $\bigwedge h.\ h \in F \Longrightarrow a\ h \neq 0 \wedge h = \{x.\ a\ h \cdot x \leq b\ h\}$
and *psub*: $\bigwedge F'.\ F' \subset F \Longrightarrow S \subset affine\ hull\ S \cap \bigcap F'$
shows $C\ facet\_of\ S \longleftrightarrow (\exists h.\ h \in F \wedge C = S \cap \{x.\ a\ h \cdot x = b\ h\})$
proof (*cases* $S = \{\}$)
case *True* **with** *psub* **show** *?thesis* **by** *force*
next
case *False*
have *polyhedron S*
unfolding *polyhedron_Int_affine* **by** (*metis* ‹*finite F*› *faceq seq*)
then have *convex S*
by (*rule polyhedron_imp_convex*)
with *False rel_interior_eq_empty* **have** $rel\_interior\ S \neq \{\}$ **by** *blast*
then obtain *x* **where** $x \in rel\_interior\ S$ **by** *auto*
then obtain *T* **where** $open\ T\ x \in T\ x \in S\ T \cap affine\ hull\ S \subseteq S$
by (*force simp*: *mem_rel_interior*)
then have *xaff*: $x \in affine\ hull\ S$ **and** *xint*: $x \in \bigcap F$
using *seq hull_inc* **by** *auto*
have $rel\_interior\ S = \{x \in S.\ \forall h{\in}F.\ a\ h \cdot x < b\ h\}$
by (*rule rel_interior_polyhedron_explicit* [*OF* ‹*finite F*› *seq faceq psub*])
with ‹$x \in rel\_interior\ S$›
have [*simp*]: $\bigwedge h.\ h{\in}F \Longrightarrow a\ h \cdot x < b\ h$ **by** *blast*
have $*$: $(S \cap \{x.\ a\ h \cdot x = b\ h\})\ facet\_of\ S$ **if** $h \in F$ **for** *h*
proof −
have $S \subset affine\ hull\ S \cap \bigcap (F - \{h\})$
using *psub that* **by** (*metis Diff_disjoint Diff_subset insert_disjoint*(*2*) *psubsetI*)
then obtain *z* **where** *zaff*: $z \in affine\ hull\ S$ **and** *zint*: $z \in \bigcap (F - \{h\})$ **and** $z \notin S$
by *force*
then have $z \neq x\ z \notin h$ **using** *seq* ‹$x \in S$› **by** *auto*
have $x \in h$ **using** *that xint* **by** *auto*
then have *able*: $a\ h \cdot x \leq b\ h$
using *faceq that* **by** *blast*
also have $\ldots < a\ h \cdot z$ **using** ‹$z \notin h$› *faceq* [*OF that*] *xint* **by** *auto*
finally have *xltz*: $a\ h \cdot x < a\ h \cdot z$ **.**
define *l* **where** $l = (b\ h - a\ h \cdot x) / (a\ h \cdot z - a\ h \cdot x)$
define *w* **where** $w = (1 - l) *_R x + l *_R z$

```
    have 0 < l l < 1
      using able xltz ‹b h < a h · z› ‹h ∈ F›
      by (auto simp: l_def field_split_simps)
    have awlt: a i · w < b i if i ∈ F i ≠ h for i
    proof −
      have (1 − l) ∗ (a i · x) < (1 − l) ∗ b i
        by (simp add: ‹l < 1› ‹i ∈ F›)
      moreover have l ∗ (a i · z) ≤ l ∗ b i
      proof (rule mult_left_mono)
        show a i · z ≤ b i
          by (metis DiffI Inter_iff empty_iff faceq insertE mem_Collect_eq that
zint)
      qed (use ‹0 < l› in auto)
      ultimately show ?thesis by (simp add: w_def algebra_simps)
    qed
    have weq: a h · w = b h
      using xltz unfolding w_def l_def
      by (simp add: algebra_simps) (simp add: field_simps)
    let ?F = {x. a h · x = b h}
    have faceS: S ∩ ?F face_of S
    proof (rule face_of_Int_supporting_hyperplane_le)
      show ⋀x. x ∈ S ⟹ a h · x ≤ b h
        using faceq seq that by fastforce
    qed fact
    have w ∈ affine hull S
      by (simp add: w_def mem_affine xaff zaff)
    moreover have w ∈ ⋂F
      using ‹a h · w = b h› awlt faceq less_eq_real_def by blast
    ultimately have w ∈ S
      using seq by blast
    with weq have ne: S ∩ ?F ≠ {} by blast
    moreover have affine hull (S ∩ ?F) = (affine hull S) ∩ ?F
    proof
      show affine hull (S ∩ ?F) ⊆ affine hull S ∩ ?F
      proof −
        have affine hull (S ∩ ?F) ⊆ affine hull S
          by (simp add: hull_mono)
        then show ?thesis
          by (simp add: affine_hyperplane subset_hull)
      qed
    next
      show affine hull S ∩ ?F ⊆ affine hull (S ∩ ?F)
      proof
        fix y
        assume yaff: y ∈ affine hull S ∩ {y. a h · y = b h}
        obtain T where 0 < T
            and T: ⋀j. ⟦j ∈ F; j ≠ h⟧ ⟹ T ∗ (a j · y − a j · w) ≤ b j − a j · w
        proof (cases F − {h} = {})
          case True then show ?thesis
```

```
      by (rule_tac T=1 in that) auto
    next
      case False
      then obtain h' where h': h' ∈ F - {h} by auto
      let ?body = (λj. if 0 < a j · y - a j · w
        then (b j - a j · w) / (a j · y - a j · w) else 1) ` (F - {h})
      define inff where inff = Inf ?body
      from ‹finite F› have finite ?body
        by blast
      moreover from h' have ?body ≠ {}
        by blast
      moreover have j > 0 if j ∈ ?body for j
      proof -
        from that obtain x where x ∈ F and x ≠ h and *: j =
          (if 0 < a x · y - a x · w
            then (b x - a x · w) / (a x · y - a x · w) else 1)
          by blast
        with awlt [of x] have a x · w < b x
          by simp
        with * show ?thesis
          by simp
      qed
      ultimately have 0 < inff
        by (simp_all add: finite_less_Inf_iff inff_def)
      moreover have inff * (a j · y - a j · w) ≤ b j - a j · w
                  if j ∈ F j ≠ h for j
      proof (cases a j · w < a j · y)
        case True
        then have inff ≤ (b j - a j · w) / (a j · y - a j · w)
          unfolding inff_def
            using ‹finite F› by (auto intro: cInf_le_finite simp add: that split:
if_split_asm)
        then show ?thesis
          using ‹0 < inff› awlt [OF that] mult_strict_left_mono
          by (fastforce simp add: field_split_simps split: if_split_asm)
      next
        case False
        with ‹0 < inff› have inff * (a j · y - a j · w) ≤ 0
          by (simp add: mult_le_0_iff)
        also have ... < b j - a j · w
          by (simp add: awlt that)
        finally show ?thesis by simp
      qed
      ultimately show ?thesis
        by (blast intro: that)
    qed
    define C where C = (1 - T) *R w + T *R y
    have (1 - T) *R w + T *R y ∈ j if j ∈ F for j
    proof (cases j = h)
```

```
        case True
        have (1 − T) *R w + T *R y ∈ {x. a h • x ≤ b h}
          using weq yaff by (auto simp: algebra_simps)
        with True faceq [OF that] show ?thesis by metis
      next
        case False
        with T that have (1 − T) *R w + T *R y ∈ {x. a j • x ≤ b j}
          by (simp add: algebra_simps)
        with faceq [OF that] show ?thesis by simp
      qed
      moreover have (1 − T) *R w + T *R y ∈ affine hull S
        using yaff ‹w ∈ affine hull S› affine_affine_hull affine_alt by blast
      ultimately have C ∈ S
        using seq by (force simp: C_def)
      moreover have a h • C = b h
        using yaff by (force simp: C_def algebra_simps weq)
      ultimately have caff: C ∈ affine hull (S ∩ {y. a h • y = b h})
        by (simp add: hull_inc)
      have waff: w ∈ affine hull (S ∩ {y. a h • y = b h})
        using ‹w ∈ S› weq by (blast intro: hull_inc)
      have yeq: y = (1 − inverse T) *R w + C /R T
        using ‹0 < T› by (simp add: C_def algebra_simps)
      show y ∈ affine hull (S ∩ {y. a h • y = b h})
        by (metis yeq affine_affine_hull [simplified affine_alt, rule_format, OF
waff caff])
    qed
  qed
  ultimately have aff_dim (affine hull (S ∩ ?F)) = aff_dim S − 1
    using ‹b h < a h • z› zaff by (force simp: aff_dim_affine_Int_hyperplane)
  then show ?thesis
    by (simp add: ne faceS facet_of_def)
 qed
 show ?thesis
 proof
  show ∃h. h ∈ F ∧ C = S ∩ {x. a h • x = b h} ⟹ C facet_of S
    using ∗ by blast
 next
  assume C facet_of S
  then have C face_of S convex C C ≠ {} and affc: aff_dim C = aff_dim S
− 1
    by (auto simp: facet_of_def face_of_imp_convex)
  then obtain x where x: x ∈ rel_interior C
    by (force simp: rel_interior_eq_empty)
  then have x ∈ C
    by (meson subsetD rel_interior_subset)
  then have x ∈ S
    using ‹C facet_of S› facet_of_imp_subset by blast
  have rels: rel_interior S = {x ∈ S. ∀h∈F. a h • x < b h}
    by (rule rel_interior_polyhedron_explicit [OF assms])
```

```
    have C ≠ S
      using ‹C facet_of S› facet_of_irrefl by blast
    then have x ∉ rel_interior S
    by (metis IntI empty_iff ‹x ∈ C› ‹C ≠ S› ‹C face_of S› face_of_disjoint_rel_interior)
    with rels ‹x ∈ S› obtain i where i ∈ F and i: a i · x ≥ b i
      by force
    have x ∈ {u. a i · u ≤ b i}
      by (metis IntD2 InterE ‹i ∈ F› ‹x ∈ S› faceq seq)
    then have a i · x ≤ b i by simp
    then have a i · x = b i using i by auto
    have C ⊆ S ∩ {x. a i · x = b i}
    proof (rule subset_of_face_of [of _ S])
      show S ∩ {x. a i · x = b i} face_of S
        by (simp add: ∗ ‹i ∈ F› facet_of_imp_face_of)
      show C ⊆ S
        by (simp add: ‹C face_of S› face_of_imp_subset)
      show S ∩ {x. a i · x = b i} ∩ rel_interior C ≠ {}
        using ‹a i · x = b i› ‹x ∈ S› x by blast
    qed
    then have cface: C face_of (S ∩ {x. a i · x = b i})
      by (meson ‹C face_of S› face_of_subset inf_le1)
    have con: convex (S ∩ {x. a i · x = b i})
      by (simp add: ‹convex S› convex_Int convex_hyperplane)
    show ∃h. h ∈ F ∧ C = S ∩ {x. a h · x = b h}
      apply (rule_tac x=i in exI)
    by (metis (no_types) ∗ ‹i ∈ F› affc facet_of_def less_irrefl face_of_aff_dim_lt
[OF con cface])
  qed
qed


lemma face_of_polyhedron_subset_explicit:
  fixes S :: 'a :: euclidean_space set
  assumes finite F
      and seq: S = affine hull S ∩ ⋂F
      and faceq: ⋀h. h ∈ F ⟹ a h ≠ 0 ∧ h = {x. a h · x ≤ b h}
      and psub: ⋀F'. F' ⊂ F ⟹ S ⊂ affine hull S ∩ ⋂F'
      and C: C face_of S and C ≠ {} C ≠ S
   obtains h where h ∈ F C ⊆ S ∩ {x. a h · x = b h}
proof −
  have C ⊆ S using ‹C face_of S›
    by (simp add: face_of_imp_subset)
  have polyhedron S
   by (metis ‹finite F› faceq polyhedron_Int polyhedron_Inter polyhedron_affine_hull
polyhedron_halfspace_le seq)
  then have convex S
    by (simp add: polyhedron_imp_convex)
  then have ∗: (S ∩ {x. a h · x = b h}) face_of S if h ∈ F for h
    using faceq seq face_of_Int_supporting_hyperplane_le that by fastforce
```

```
  have rel_interior C ≠ {}
    using C ‹C ≠ {}› face_of_imp_convex rel_interior_eq_empty by blast
  then obtain x where x ∈ rel_interior C by auto
  have rels: rel_interior S = {x ∈ S. ∀ h∈F. a h · x < b h}
    by (rule rel_interior_polyhedron_explicit [OF ‹finite F› seq faceq psub])
  then have xnot: x ∉ rel_interior S
     by (metis IntI ‹x ∈ rel_interior C› C ‹C ≠ S› contra_subsetD empty_iff
face_of_disjoint_rel_interior rel_interior_subset)
  then have x ∈ S
    using ‹C ⊆ S› ‹x ∈ rel_interior C› rel_interior_subset by auto
  then have xint: x ∈ ⋂F
    using seq by blast
  have F ≠ {} using assms
   by (metis affine_Int affine_Inter affine_affine_hull ex_in_conv face_of_affine_trivial)
  then obtain i where i ∈ F ¬ (a i · x < b i)
    using ‹x ∈ S› rels xnot by auto
  with xint have a i · x = b i
    by (metis eq_iff mem_Collect_eq not_le Inter_iff faceq)
  have face: S ∩ {x. a i · x = b i} face_of S
    by (simp add: ∗ ‹i ∈ F›)
  show ?thesis
  proof
    show C ⊆ S ∩ {x. a i · x = b i}
     using subset_of_face_of [OF face ‹C ⊆ S›] ‹a i · x = b i› ‹x ∈ rel_interior
C› ‹x ∈ S› by blast
  qed fact
qed
```

Initial part of proof duplicates that above

```
proposition face_of_polyhedron_explicit:
  fixes S :: 'a :: euclidean_space set
  assumes finite F
      and seq: S = affine hull S ∩ ⋂F
      and faceq: ⋀h. h ∈ F ⟹ a h ≠ 0 ∧ h = {x. a h · x ≤ b h}
      and psub: ⋀F'. F' ⊂ F ⟹ S ⊂ affine hull S ∩ ⋂F'
      and C: C face_of S and C ≠ {} C ≠ S
    shows C = ⋂{S ∩ {x. a h · x = b h} | h. h ∈ F ∧ C ⊆ S ∩ {x. a h · x = b
h}}
proof −
  let ?ab = λh. {x. a h · x = b h}
  have C ⊆ S using ‹C face_of S›
    by (simp add: face_of_imp_subset)
  have polyhedron S
   by (metis ‹finite F› faceq polyhedron_Int polyhedron_Inter polyhedron_affine_hull
polyhedron_halfspace_le seq)
  then have convex S
    by (simp add: polyhedron_imp_convex)
  then have ∗: (S ∩ ?ab h) face_of S if h ∈ F for h
    using faceq seq face_of_Int_supporting_hyperplane_le that by fastforce
```

have *rel_interior C* ≠ {}
using *C* ‹*C* ≠ {}› *face_of_imp_convex rel_interior_eq_empty* **by** *blast*
then obtain *z* **where** *z*: *z* ∈ *rel_interior C* **by** *auto*
have *rels*: *rel_interior S* = {*z* ∈ *S*. ∀ *h*∈*F*. *a h* ∙ *z* < *b h*}
by (*rule rel_interior_polyhedron_explicit* [*OF* ‹*finite F*› *seq faceq psub*])
then have *xnot*: *z* ∉ *rel_interior S*
by (*metis IntI* ‹*z* ∈ *rel_interior C*› *C* ‹*C* ≠ *S*› *contra_subsetD empty_iff face_of_disjoint_rel_interior rel_interior_subset*)
then have *z* ∈ *S*
using ‹*C* ⊆ *S*› ‹*z* ∈ *rel_interior C*› *rel_interior_subset* **by** *auto*
with *seq* **have** *xint*: *z* ∈ ⋂ *F* **by** *blast*
have *open* (⋂ *h*∈{*h* ∈ *F*. *a h* ∙ *z* < *b h*}. {*w*. *a h* ∙ *w* < *b h*})
by (*auto simp*: ‹*finite F*› *open_halfspace_lt open_INT*)
then obtain *e* **where** *0* < *e*
ball z e ⊆ (⋂ *h*∈{*h* ∈ *F*. *a h* ∙ *z* < *b h*}. {*w*. *a h* ∙ *w* < *b h*})
by (*auto intro*: *openE* [*of* _ *z*])
then have *e*: ⋀*h*. ⟦*h* ∈ *F*; *a h* ∙ *z* < *b h*⟧ ⟹ *ball z e* ⊆ {*w*. *a h* ∙ *w* < *b h*}
by *blast*
have *C* ⊆ (*S* ∩ *?ab h*) ⟷ *z* ∈ *S* ∩ *?ab h* **if** *h* ∈ *F* **for** *h*
proof
show *z* ∈ *S* ∩ *?ab h* ⟹ *C* ⊆ *S* ∩ *?ab h*
by (*metis* ∗ *Collect_cong IntI* ‹*C* ⊆ *S*› *empty_iff subset_of_face_of that z*)
next
show *C* ⊆ *S* ∩ *?ab h* ⟹ *z* ∈ *S* ∩ *?ab h*
using ‹*z* ∈ *rel_interior C*› *rel_interior_subset* **by** *force*
qed
then have ∗∗: {*S* ∩ *?ab h* | *h*. *h* ∈ *F* ∧ *C* ⊆ *S* ∧ *C* ⊆ *?ab h*} =
{*S* ∩ *?ab h* |*h*. *h* ∈ *F* ∧ *z* ∈ *S* ∩ *?ab h*}
by *blast*
have *bsub*: *ball z e* ∩ *affine hull* ⋂{*S* ∩ *?ab h* |*h*. *h* ∈ *F* ∧ *a h* ∙ *z* = *b h*}
⊆ *affine hull S* ∩ ⋂ *F* ∩ ⋂{*?ab h* |*h*. *h* ∈ *F* ∧ *a h* ∙ *z* = *b h*}
if *i* ∈ *F* **and** *i*: *a i* ∙ *z* = *b i* **for** *i*
proof −
have *sub*: *ball z e* ∩ ⋂{*?ab h* |*h*. *h* ∈ *F* ∧ *a h* ∙ *z* = *b h*} ⊆ *j*
if *j* ∈ *F* **for** *j*
proof −
have *a j* ∙ *z* ≤ *b j* **using** *faceq that xint* **by** *auto*
then consider *a j* ∙ *z* < *b j* | *a j* ∙ *z* = *b j* **by** *linarith*
then have ∃ *G*. *G* ∈ {*?ab h* |*h*. *h* ∈ *F* ∧ *a h* ∙ *z* = *b h*} ∧ *ball z e* ∩ *G* ⊆ *j*
proof *cases*
assume *a j* ∙ *z* < *b j*
then have *ball z e* ∩ {*x*. *a i* ∙ *x* = *b i*} ⊆ *j*
using *e* [*OF* ‹*j* ∈ *F*›] *faceq that*
by (*fastforce simp*: *ball_def*)
then show *?thesis*
by (*rule_tac x*={*x*. *a i* ∙ *x* = *b i*} **in** *exI*) (*force simp*: ‹*i* ∈ *F*› *i*)
next
assume *eq*: *a j* ∙ *z* = *b j*
with *faceq that* **show** *?thesis*

```
      by (rule_tac x={x. a j · x = b j} in exI) (fastforce simp add: ‹j ∈ F›)
    qed
    then show ?thesis by blast
  qed
  have 1: affine hull ⋂{S ∩ ?ab h |h. h ∈ F ∧ a h · z = b h} ⊆ affine hull S
    using that ‹z ∈ S› by (intro hull_mono) auto
  have 2: affine hull ⋂{S ∩ ?ab h |h. h ∈ F ∧ a h · z = b h}
      ⊆ ⋂{?ab h |h. h ∈ F ∧ a h · z = b h}
    by (rule hull_minimal) (auto intro: affine_hyperplane)
  have 3: ball z e ∩ ⋂{?ab h |h. h ∈ F ∧ a h · z = b h} ⊆ ⋂F
    by (iprover intro: sub Inter_greatest)
  have ∗: ⟦A ⊆ (B :: 'a set); A ⊆ C; E ∩ C ⊆ D⟧ ⟹ E ∩ A ⊆ (B ∩ D) ∩ C
      for A B C D E by blast
  show ?thesis by (intro ∗ 1 2 3)
 qed
 have ∃h. h ∈ F ∧ C ⊆ ?ab h
  using assms
  by (metis face_of_polyhedron_subset_explicit [OF ‹finite F› seq faceq psub]
le_inf_iff)
 then have fac: ⋂{S ∩ ?ab h |h. h ∈ F ∧ C ⊆ S ∩ ?ab h} face_of S
  using ∗ by (force simp: ‹C ⊆ S› intro: face_of_Inter)
 have red: (⋀a. P a ⟹ T ⊆ S ∩ ⋂{F X |X. P X}) ⟹ T ⊆ ⋂{S ∩ F X |X::'a
set. P X} for P T F
  by blast
 have ball z e ∩ affine hull ⋂{S ∩ ?ab h |h. h ∈ F ∧ a h · z = b h}
     ⊆ ⋂{S ∩ ?ab h |h. h ∈ F ∧ a h · z = b h}
  by (rule red) (metis seq bsub)
 with ‹0 < e› have zinrel: z ∈ rel_interior
                (⋂{S ∩ ?ab h |h. h ∈ F ∧ z ∈ S ∧ a h · z = b h})
  by (auto simp: mem_rel_interior_ball ‹z ∈ S›)
 show ?thesis
  using z zinrel
  by (intro face_of_eq [OF C fac]) (force simp: ∗∗)
qed
```

6.34.12 More general corollaries from the explicit representation

```
corollary facet_of_polyhedron:
 assumes polyhedron S and C facet_of S
 obtains a b where a ≠ 0 S ⊆ {x. a · x ≤ b} C = S ∩ {x. a · x = b}
proof −
 obtain F where finite F and seq: S = affine hull S ∩ ⋂F
          and faces: ⋀h. h ∈ F ⟹ ∃a b. a ≠ 0 ∧ h = {x. a · x ≤ b}
          and min: ⋀F'. F' ⊂ F ⟹ S ⊂ (affine hull S) ∩ ⋂F'
  using assms by (simp add: polyhedron_Int_affine_minimal) meson
 then obtain a b where ab: ⋀h. h ∈ F ⟹ a h ≠ 0 ∧ h = {x. a h · x ≤ b h}
  by metis
 obtain i where i ∈ F and C: C = S ∩ {x. a i · x = b i}
```

using *facet_of_polyhedron_explicit* [*OF* ‹*finite F*› *seq ab min*] *assms*
by *force*
moreover have *ssub*: $S \subseteq \{x.\ a\ i \cdot x \leq b\ i\}$
using ‹$i \in F$› *ab* **by** (*subst seq*) *auto*
ultimately show *?thesis*
by (*rule_tac* $a = a\ i$ **and** $b = b\ i$ **in** *that*) (*simp_all add*: *ab*)
qed

corollary *face_of_polyhedron*:
assumes *polyhedron S* **and** *C face_of S* **and** $C \neq \{\}$ **and** $C \neq S$
shows $C = \bigcap \{F.\ F\ facet\_of\ S \wedge C \subseteq F\}$
proof −
obtain *F* **where** *finite F* **and** *seq*: $S = affine\ hull\ S \cap \bigcap F$
and *faces*: $\bigwedge h.\ h \in F \Longrightarrow \exists a\ b.\ a \neq 0 \wedge h = \{x.\ a \cdot x \leq b\}$
and *min*: $\bigwedge F'.\ F' \subset F \Longrightarrow S \subset (affine\ hull\ S) \cap \bigcap F'$
using *assms* **by** (*simp add*: *polyhedron_Int_affine_minimal*) *meson*
then obtain *a b* **where** *ab*: $\bigwedge h.\ h \in F \Longrightarrow a\ h \neq 0 \wedge h = \{x.\ a\ h \cdot x \leq b\ h\}$
by *metis*
show *?thesis*
apply (*subst face_of_polyhedron_explicit* [*OF* ‹*finite F*› *seq ab min*])
apply (*auto simp*: *assms facet_of_polyhedron_explicit* [*OF* ‹*finite F*› *seq ab min*] *cong*: *Collect_cong*)
done
qed

lemma *face_of_polyhedron_subset_facet*:
assumes *polyhedron S* **and** *C face_of S* **and** $C \neq \{\}$ **and** $C \neq S$
obtains *F* **where** *F facet_of S C* ⊆ *F*
using *face_of_polyhedron assms*
by (*metis* (*no_types*, *lifting*) *Inf_greatest antisym_conv face_of_imp_subset mem_Collect_eq*)

lemma *exposed_face_of_polyhedron*:
assumes *polyhedron S*
shows *F exposed_face_of S* ⟷ *F face_of S*
proof
show *F exposed_face_of S* ⟹ *F face_of S*
by (*simp add*: *exposed_face_of_def*)
next
assume *F face_of S*
show *F exposed_face_of S*
proof (*cases* $F = \{\} \vee F = S$)
case *True* **then show** *?thesis*
using ‹*F face_of S*› *exposed_face_of* **by** *blast*
next
case *False*
then have $\{g.\ g\ facet\_of\ S \wedge F \subseteq g\} \neq \{\}$
by (*metis Collect_empty_eq_bot* ‹*F face_of S*› *assms empty_def face_of_polyhedron_subset_facet*)

```
    moreover have ⋀T. ⟦T facet_of S; F ⊆ T⟧ ⟹ T exposed_face_of S
     by (metis assms exposed_face_of facet_of_imp_face_of facet_of_polyhedron)
    ultimately have ⋂{G. G facet_of S ∧ F ⊆ G} exposed_face_of S
      by (metis (no_types, lifting) mem_Collect_eq exposed_face_of_Inter)
    then show ?thesis
      using False ‹F face_of S› assms face_of_polyhedron by fastforce
  qed
qed

lemma face_of_polyhedron_polyhedron:
  fixes S :: 'a :: euclidean_space set
  assumes polyhedron S c face_of S shows polyhedron c
by (metis assms face_of_imp_eq_affine_Int polyhedron_Int polyhedron_affine_hull
polyhedron_imp_convex)

lemma finite_polyhedron_faces:
  fixes S :: 'a :: euclidean_space set
  assumes polyhedron S
    shows finite {F. F face_of S}
proof −
  obtain F where finite F and seq: S = affine hull S ∩ ⋂F
            and faces: ⋀h. h ∈ F ⟹ ∃a b. a ≠ 0 ∧ h = {x. a · x ≤ b}
            and min:   ⋀F'. F' ⊂ F ⟹ S ⊂ (affine hull S) ∩ ⋂F'
    using assms by (simp add: polyhedron_Int_affine_minimal) meson
  then obtain a b where ab: ⋀h. h ∈ F ⟹ a h ≠ 0 ∧ h = {x. a h · x ≤ b h}
    by metis
  have finite {⋂{S ∩ {x. a h · x = b h} |h. h ∈ F'}| F'. F' ∈ Pow F}
    by (simp add: ‹finite F›)
  moreover have {F. F face_of S} − {{}, S} ⊆ {⋂{S ∩ {x. a h · x = b h} |h.
h ∈ F'}| F'. F' ∈ Pow F}
    apply clarify
    apply (rename_tac c)
   apply (drule face_of_polyhedron_explicit [OF ‹finite F› seq ab min, simplified],
simp_all)
    apply (rule_tac x={h ∈ F. c ⊆ S ∩ {x. a h · x = b h}} in exI, auto)
    done
  ultimately show ?thesis
    by (meson finite.emptyI finite.insertI finite_Diff2 finite_subset)
qed

lemma finite_polyhedron_exposed_faces:
   polyhedron S ⟹ finite {F. F exposed_face_of S}
using exposed_face_of_polyhedron finite_polyhedron_faces by fastforce

lemma finite_polyhedron_extreme_points:
  fixes S :: 'a :: euclidean_space set
  assumes polyhedron S shows finite {v. v extreme_point_of S}
proof −
  have finite {v. {v} face_of S}
```

using *assms* **by** (*intro finite_subset* [*OF* _ *finite_vimageI* [*OF finite_polyhedron_faces*]], *auto*)
then show *?thesis*
by (*simp add*: *face_of_singleton*)
qed

lemma *finite_polyhedron_facets*:
fixes $S :: 'a :: euclidean\_space\ set$
shows *polyhedron* $S \Longrightarrow$ *finite* $\{F.\ F\ facet\_of\ S\}$
unfolding *facet_of_def*
by (*blast intro*: *finite_subset* [*OF* _ *finite_polyhedron_faces*])

proposition *rel_interior_of_polyhedron*:
fixes $S :: 'a :: euclidean\_space\ set$
assumes *polyhedron S*
shows $rel\_interior\ S = S - \bigcup \{F.\ F\ facet\_of\ S\}$
proof −
obtain *F* **where** *finite F* **and** *seq*: $S = affine\ hull\ S \cap \bigcap F$
and *faces*: $\bigwedge h.\ h \in F \Longrightarrow \exists\, a\ b.\ a \neq 0 \wedge h = \{x.\ a \cdot x \leq b\}$
and *min*: $\bigwedge F'.\ F' \subset F \Longrightarrow S \subset (affine\ hull\ S) \cap \bigcap F'$
using *assms* **by** (*simp add*: *polyhedron_Int_affine_minimal*) *meson*
then obtain *a b* **where** *ab*: $\bigwedge h.\ h \in F \Longrightarrow a\ h \neq 0 \wedge h = \{x.\ a\ h \cdot x \leq b\ h\}$
by *metis*
have *facet*: $(c\ facet\_of\ S) \longleftrightarrow (\exists\, h.\ h \in F \wedge c = S \cap \{x.\ a\ h \cdot x = b\ h\})$ **for** *c*
by (*rule facet_of_polyhedron_explicit* [*OF* ‹*finite F*› *seq ab min*])
have *rel*: $rel\_interior\ S = \{x \in S.\ \forall\, h{\in}F.\ a\ h \cdot x < b\ h\}$
by (*rule rel_interior_polyhedron_explicit* [*OF* ‹*finite F*› *seq ab min*])
have $a\ h \cdot x < b\ h$ **if** $x \in S\ h \in F$ **and** *xnot*: $x \notin \bigcup \{F.\ F\ facet\_of\ S\}$ **for** *x h*
proof −
have $x \in \bigcap F$ **using** *seq that* **by** *force*
with ‹$h \in F$› *ab* **have** $a\ h \cdot x \leq b\ h$ **by** *auto*
then consider $a\ h \cdot x < b\ h \mid a\ h \cdot x = b\ h$ **by** *linarith*
then show *?thesis*
proof *cases*
case *1* **then show** *?thesis* **.**
next
case *2*
have $Collect\ ((\in)\ x) \notin Collect\ ((\in)\ (\bigcup \{A.\ A\ facet\_of\ S\}))$
using *xnot* **by** *fastforce*
then have $F \notin Collect\ ((\in)\ h)$
using *2* ‹$x \in S$› *facet* **by** *blast*
with *2 that* ‹$x \in \bigcap F$› **show** *?thesis*
by *blast*
qed
qed
moreover have $\exists\, h{\in}F.\ a\ h \cdot x \geq b\ h$ **if** $x \in \bigcup \{F.\ F\ facet\_of\ S\}$ **for** *x*
using *that* **by** (*force simp*: *facet*)
ultimately show *?thesis*

```
    by (force simp: rel)
qed

lemma rel_boundary_of_polyhedron:
  fixes S :: 'a :: euclidean_space set
  assumes polyhedron S
    shows S - rel_interior S = ⋃ {F. F facet_of S}
using facet_of_imp_subset by (fastforce simp add: rel_interior_of_polyhedron
assms)

lemma rel_frontier_of_polyhedron:
  fixes S :: 'a :: euclidean_space set
  assumes polyhedron S
    shows rel_frontier S = ⋃ {F. F facet_of S}
by (simp add: assms rel_frontier_def polyhedron_imp_closed rel_boundary_of_polyhedron)

lemma rel_frontier_of_polyhedron_alt:
  fixes S :: 'a :: euclidean_space set
  assumes polyhedron S
  shows rel_frontier S = ⋃ {F. F face_of S ∧ F ≠ S}
proof
  show rel_frontier S ⊆ ⋃ {F. F face_of S ∧ F ≠ S}
    by (force simp: rel_frontier_of_polyhedron facet_of_def assms)
qed (use face_of_subset_rel_frontier in fastforce)
```

A characterization of polyhedra as having finitely many faces

```
proposition polyhedron_eq_finite_exposed_faces:
  fixes S :: 'a :: euclidean_space set
  shows polyhedron S ⟷ closed S ∧ convex S ∧ finite {F. F exposed_face_of S}
        (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
   by (auto simp: polyhedron_imp_closed polyhedron_imp_convex finite_polyhedron_exposed_faces)
next
  assume ?rhs
  then have closed S convex S and fin: finite {F. F exposed_face_of S} by auto
  show ?lhs
  proof (cases S = {})
    case True then show ?thesis by auto
  next
    case False
    define F where F = {h. h exposed_face_of S ∧ h ≠ {} ∧ h ≠ S}
    have finite F by (simp add: fin F_def)
    have hface: h face_of S
      and ∃ a b. a ≠ 0 ∧ S ⊆ {x. a · x ≤ b} ∧ h = S ∩ {x. a · x = b}
      if h ∈ F for h
      using exposed_face_of F_def that by blast+
    then obtain a b where ab:
```

```
    ⋀h. h ∈ F ⟹ a h ≠ 0 ∧ S ⊆ {x. a h · x ≤ b h} ∧ h = S ∩ {x. a h · x = b
h}
     by metis
   have ∗: False
     if paff: p ∈ affine hull S and p ∉ S
     and pint: p ∈ ⋂{{x. a h · x ≤ b h} |h. h ∈ F} for p
   proof −
     have rel_interior S ≠ {}
       by (simp add: ‹S ≠ {}› ‹convex S› rel_interior_eq_empty)
     then obtain c where c: c ∈ rel_interior S by auto
     with rel_interior_subset have c ∈ S  by blast
     have ccp: closed_segment c p ⊆ affine hull S
       by (meson affine_affine_hull affine_imp_convex c closed_segment_subset
hull_subset paff rel_interior_subset subsetCE)
      have oS: openin (top_of_set (closed_segment c p)) (closed_segment c p ∩
rel_interior S)
     by (force simp: openin_rel_interior openin_Int intro: openin_subtopology_Int_subset
[OF _ ccp])
      obtain x where xcl: x ∈ closed_segment c p and x ∈ S and xnot: x ∉
rel_interior S
       using connected_openin [of closed_segment c p]
       apply simp
       apply (drule_tac x=closed_segment c p ∩ rel_interior S in spec)
       apply (drule mp [OF _ oS])
       apply (drule_tac x=closed_segment c p ∩ (− S) in spec)
       using rel_interior_subset ‹closed S› c ‹p ∉ S› apply blast
       done
     then obtain μ where 0 ≤ μ μ ≤ 1 and xeq: x = (1 − μ) ∗R c + μ ∗R p
       by (auto simp: in_segment)
     show False
     proof (cases μ=0 ∨ μ=1)
       case True with xeq c xnot ‹x ∈ S› ‹p ∉ S›
       show False by auto
     next
       case False
       then have xos: x ∈ open_segment c p
         using ‹x ∈ S› c open_segment_def that(2) xcl xnot by auto
       have xclo: x ∈ closure S
         using ‹x ∈ S› closure_subset by blast
       obtain d where d ≠ 0
           and dle: ⋀y. y ∈ closure S ⟹ d · x ≤ d · y
           and dless: ⋀y. y ∈ rel_interior S ⟹ d · x < d · y
         by (metis supporting_hyperplane_relative_frontier [OF ‹convex S› xclo
xnot])
       have sex: S ∩ {y. d · y = d · x} exposed_face_of S
      by (simp add: ‹closed S› dle exposed_face_of_Int_supporting_hyperplane_ge
[OF ‹convex S›])
       have sne: S ∩ {y. d · y = d · x} ≠ {}
         using ‹x ∈ S› by blast
```

```
      have sns: S ∩ {y. d · y = d · x} ≠ S
        by (metis (mono_tags) Int_Collect c subsetD dless not_le order_refl
rel_interior_subset)
      obtain h where h ∈ F x ∈ h
        using F_def ‹x ∈ S› sex sns by blast
      have abface: {y. a h · y = b h} face_of {y. a h · y ≤ b h}
        using hyperplane_face_of_halfspace_le by blast
      then have c ∈ h
        using face_ofD [OF abface xos] ‹c ∈ S› ‹h ∈ F› ab pint ‹x ∈ h› by blast
      with c have h ∩ rel_interior S ≠ {} by blast
      then show False
        using ‹h ∈ F› F_def face_of_disjoint_rel_interior hface by auto
    qed
  qed
  let ?S' = affine hull S ∩ ⋂{{x. a h · x ≤ b h} |h. h ∈ F}
  have S ⊆ ?S'
    using ab by (auto simp: hull_subset)
  moreover have ?S' ⊆ S
    using ∗ by blast
  ultimately have S = ?S' ..
  moreover have polyhedron ?S'
    by (force intro: polyhedron_affine_hull polyhedron_halfspace_le simp: ‹finite
F›)
  ultimately show ?thesis
    by auto
 qed
qed

corollary polyhedron_eq_finite_faces:
 fixes S :: 'a :: euclidean_space set
 shows polyhedron S ⟷ closed S ∧ convex S ∧ finite {F. F face_of S}
       (is ?lhs = ?rhs)
proof
 assume ?lhs
 then show ?rhs
  by (simp add: finite_polyhedron_faces polyhedron_imp_closed polyhedron_imp_convex)
next
 assume ?rhs
 then show ?lhs
    by (force simp: polyhedron_eq_finite_exposed_faces exposed_face_of intro:
finite_subset)
qed

lemma polyhedron_linear_image_eq:
 fixes h :: 'a :: euclidean_space ⇒ 'b :: euclidean_space
 assumes linear h bij h
   shows polyhedron (h ‘ S) ⟷ polyhedron S
proof −
 have [simp]: inj h using bij_is_inj assms by blast
```

```
  then have injim: inj_on ((`) h) A for A
    by (simp add: inj_on_def inj_image_eq_iff)
  { fix P
    have ⋀x. P x ⟹ x ∈ (`) h ` {f. P (h ` f)}
      using bij_is_surj [OF ‹bij h›]
      by (metis image_eqI mem_Collect_eq subset_imageE top_greatest)
    then have {f. P f} = (image h) ` {f. P (h ` f)}
      by force
  }
  then have finite {F. F face_of h ` S} =finite {F. F face_of S}
    using ‹linear h›
    by (simp add: finite_image_iff injim flip: face_of_linear_image [of h _ S])
  then show ?thesis
    using ‹linear h›
    by (simp add: polyhedron_eq_finite_faces closed_injective_linear_image_eq)
qed

lemma polyhedron_negations:
  fixes S :: 'a :: euclidean_space set
  shows   polyhedron S ⟹ polyhedron(image uminus S)
 by (subst polyhedron_linear_image_eq) (auto simp: bij_uminus intro!: linear_uminus)
```

6.34.13 Relation between polytopes and polyhedra

```
proposition polytope_eq_bounded_polyhedron:
  fixes S :: 'a :: euclidean_space set
  shows polytope S ⟷ polyhedron S ∧ bounded S
        (is ?lhs = ?rhs)
proof
  assume ?lhs
  then show ?rhs
    by (simp add: finite_polytope_faces polyhedron_eq_finite_faces
                  polytope_imp_closed polytope_imp_convex polytope_imp_bounded)
next
  assume R: ?rhs
  then have finite {v. v extreme_point_of S}
    by (simp add: finite_polyhedron_extreme_points)
  moreover have S = convex hull {v. v extreme_point_of S}
   using R by (simp add: Krein_Milman_Minkowski compact_eq_bounded_closed
polyhedron_imp_closed polyhedron_imp_convex)
  ultimately show ?lhs
    unfolding polytope_def by blast
qed

lemma polytope_Int:
  fixes S :: 'a :: euclidean_space set
  shows ⟦polytope S; polytope T⟧ ⟹ polytope(S ∩ T)
by (simp add: polytope_eq_bounded_polyhedron bounded_Int)
```

lemma *polytope_Int_polyhedron*:
fixes $S :: 'a :: euclidean\_space\ set$
shows $[\![polytope\ S; polyhedron\ T]\!] \Longrightarrow polytope(S \cap T)$
by (*simp add*: *bounded_Int polytope_eq_bounded_polyhedron*)

lemma *polyhedron_Int_polytope*:
fixes $S :: 'a :: euclidean\_space\ set$
shows $[\![polyhedron\ S; polytope\ T]\!] \Longrightarrow polytope(S \cap T)$
by (*simp add*: *bounded_Int polytope_eq_bounded_polyhedron*)

lemma *polytope_imp_polyhedron*:
fixes $S :: 'a :: euclidean\_space\ set$
shows $polytope\ S \Longrightarrow polyhedron\ S$
by (*simp add*: *polytope_eq_bounded_polyhedron*)

lemma *polytope_facet_exists*:
fixes $p :: 'a :: euclidean\_space\ set$
assumes $polytope\ p\ 0 < aff\_dim\ p$
obtains F **where** $F\ facet\_of\ p$
proof (*cases* $p = \{\}$)
case *True* **with** *assms* **show** *?thesis* **by** *auto*
next
case *False*
then obtain v **where** $v\ extreme\_point\_of\ p$
using *extreme_point_exists_convex*
by (*blast intro*: ‹*polytope p*› *polytope_imp_compact polytope_imp_convex*)
then
show *?thesis*
by (*metis face_of_polyhedron_subset_facet polytope_imp_polyhedron aff_dim_sing all_not_in_conv assms face_of_singleton less_irrefl singletonI that*)
qed

lemma *polyhedron_interval* [*iff*]: $polyhedron(cbox\ a\ b)$
by (*metis polytope_imp_polyhedron polytope_interval*)

lemma *polyhedron_convex_hull*:
fixes $S :: 'a :: euclidean\_space\ set$
shows $finite\ S \Longrightarrow polyhedron(convex\ hull\ S)$
by (*simp add*: *polytope_convex_hull polytope_imp_polyhedron*)

6.34.14 Relative and absolute frontier of a polytope

lemma *rel_boundary_of_convex_hull*:
fixes $S :: 'a::euclidean\_space\ set$
assumes $\neg\ affine\_dependent\ S$
shows $(convex\ hull\ S) - rel\_interior(convex\ hull\ S) = (\bigcup a{\in}S.\ convex\ hull\ (S - \{a\}))$
proof −

```
  have finite S by (metis assms aff_independent_finite)
  then consider card S = 0 | card S = 1 | 2 ≤ card S by arith
  then show ?thesis
  proof cases
    case 1 then have S = {} by (simp add: ‹finite S›)
    then show ?thesis by simp
  next
    case 2 show ?thesis
      by (auto intro: card_1_singletonE [OF ‹card S = 1›])
  next
    case 3
    with assms show ?thesis
    by (auto simp: polyhedron_convex_hull rel_boundary_of_polyhedron facet_of_convex_hull_affine_independen
‹finite S›)
  qed
qed

proposition frontier_of_convex_hull:
    fixes S :: 'a::euclidean_space set
    assumes card S = Suc (DIM('a))
      shows frontier(convex hull S) = ⋃ {convex hull (S - {a}) | a. a ∈ S}
proof (cases affine_dependent S)
  case True
    have [iff]: finite S
      using assms using card.infinite by force
    then have ccs: closed (convex hull S)
      by (simp add: compact_imp_closed finite_imp_compact_convex_hull)
    { fix x T
      assume int (card T) ≤ aff_dim S + 1  finite T T ⊆ S x ∈ convex hull T
      then have S ≠ T
          using True ‹finite S› aff_dim_le_card affine_independent_iff_card by
fastforce
      then obtain a where a ∈ S a ∉ T
        using ‹T ⊆ S› by blast
      then have ∃ y∈S. x ∈ convex hull (S - {y})
        using True affine_independent_iff_card [of S]
        by (metis (no_types, opaque_lifting) Diff_eq_empty_iff Diff_insert0 ‹a ∉
T› ‹T ⊆ S› ‹x ∈ convex hull T› hull_mono insert_Diff_single subsetCE)
    } note * = this
    have 1: convex hull S ⊆ (⋃ a∈S. convex hull (S - {a}))
      by (subst caratheodory_aff_dim) (blast dest: *)
    have 2: ⋃((λa. convex hull (S - {a})) ` S) ⊆ convex hull S
        by (rule Union_least) (metis (no_types, lifting)  Diff_subset hull_mono
imageE)
    show ?thesis using True
      apply (simp add: segment_convex_hull frontier_def)
      using interior_convex_hull_eq_empty [OF assms]
      apply (simp add: closure_closed [OF ccs])
      using 1 2 by auto
```

next
case *False*
then have *frontier* (*convex hull S*) = *closure* (*convex hull S*) − *interior* (*convex hull S*)
by (*simp add*: *rel_boundary_of_convex_hull frontier_def*)
also have ... = (*convex hull S*) − *rel_interior*(*convex hull S*)
by (*metis False aff_independent_finite assms closure_convex_hull finite_imp_compact_convex_hull hull_hull interior_convex_hull_eq_empty rel_interior_nonempty_interior*)
also have ... = ⋃{*convex hull* (*S* − {*a*}) |*a*. *a* ∈ *S*}
proof −
have *convex hull S* − *rel_interior* (*convex hull S*) = *rel_frontier* (*convex hull S*)
by (*simp add*: *False aff_independent_finite polyhedron_convex_hull rel_boundary_of_polyhedron rel_frontier_of_polyhedron*)
then show *?thesis*
by (*simp add*: *False rel_frontier_convex_hull_cases*)
qed
finally show *?thesis* **.**
qed

6.34.15 Special case of a triangle

proposition *frontier_of_triangle*:
fixes *a* :: *'a*::*euclidean_space*
assumes *DIM*(*'a*) = *2*
shows *frontier*(*convex hull* {*a*,*b*,*c*}) = *closed_segment a b* ∪ *closed_segment b c* ∪ *closed_segment c a*
(**is** *?lhs* = *?rhs*)
proof (*cases b* = *a* ∨ *c* = *a* ∨ *c* = *b*)
case *True* **then show** *?thesis*
by (*auto simp*: *assms segment_convex_hull frontier_def empty_interior_convex_hull insert_commute card_insert_le_m1 hull_inc insert_absorb*)
next
case *False* **then have** [*simp*]: *card* {*a*, *b*, *c*} = *Suc* (*DIM*(*'a*))
by (*simp add*: *card.insert_remove Set.insert_Diff_if assms*)
show *?thesis*
proof
show *?lhs* ⊆ *?rhs*
using *False*
by (*force simp*: *segment_convex_hull frontier_of_convex_hull insert_Diff_if insert_commute split*: *if_split_asm*)
show *?rhs* ⊆ *?lhs*
using *False*
apply (*simp add*: *frontier_of_convex_hull segment_convex_hull*)
apply (*intro conjI subsetI*)
apply (*rule_tac X*=*convex hull* {*a*,*b*} **in** *UnionI*; *force simp*: *Set.insert_Diff_if*)
apply (*rule_tac X*=*convex hull* {*b*,*c*} **in** *UnionI*; *force*)
apply (*rule_tac X*=*convex hull* {*a*,*c*} **in** *UnionI*; *force simp*: *insert_commute Set.insert_Diff_if*)

```
    done
  qed
qed

corollary inside_of_triangle:
    fixes a :: 'a::euclidean_space
    assumes DIM('a) = 2
     shows inside (closed_segment a b ∪ closed_segment b c ∪ closed_segment c
a) = interior(convex hull {a,b,c})
by (metis assms frontier_of_triangle bounded_empty bounded_insert convex_convex_hull
inside_frontier_eq_interior bounded_convex_hull)

corollary interior_of_triangle:
    fixes a :: 'a::euclidean_space
    assumes DIM('a) = 2
    shows interior(convex hull {a,b,c}) =
            convex hull {a,b,c} - (closed_segment a b ∪ closed_segment b c ∪
closed_segment c a)
  using interior_subset
  by (force simp: frontier_of_triangle [OF assms, symmetric] frontier_def Diff_Diff_Int)
```

6.34.16 Subdividing a cell complex

```
lemma subdivide_interval:
  fixes x::real
  assumes a < |x - y| 0 < a
  obtains n where n ∈ ℤ x < n * a ∧ n * a < y ∨ y <  n * a ∧ n * a < x
proof -
  consider a + x < y | a + y < x
    using assms by linarith
  then show ?thesis
  proof cases
    case 1
    let ?n = of_int (floor (x/a)) + 1
    have x: x < ?n * a
      by (meson ‹0 < a› divide_less_eq floor_eq_iff)
    have ?n * a ≤ a + x
      using ‹a>0› by (simp add: distrib_right floor_divide_lower)
    also have ... < y
      by (rule 1)
    finally have ?n * a < y .
    with x show ?thesis
      using Ints_1 Ints_add Ints_of_int that by blast
  next
    case 2
    let ?n = of_int (floor (y/a)) + 1
    have y: y < ?n * a
      by (meson ‹0 < a› divide_less_eq floor_eq_iff)
    have ?n * a ≤ a + y
```

```
    using ‹a>0› by (simp add: distrib_right floor_divide_lower)
   also have ... < x
    by (rule 2)
   finally have ?n * a < x .
   then show ?thesis
    using Ints_1 Ints_add Ints_of_int that y by blast
  qed
qed

lemma cell_subdivision_lemma:
  assumes finite ℱ
     and ⋀X. X ∈ ℱ ⟹ polytope X
     and ⋀X. X ∈ ℱ ⟹ aff_dim X ≤ d
     and ⋀X Y. ⟦X ∈ ℱ; Y ∈ ℱ⟧ ⟹ (X ∩ Y) face_of X
     and finite I
   shows ∃𝒢. ⋃𝒢 = ⋃ℱ ∧
               finite 𝒢 ∧
               (∀C ∈ 𝒢. ∃D. D ∈ ℱ ∧ C ⊆ D) ∧
               (∀C ∈ ℱ. ∀x ∈ C. ∃D. D ∈ 𝒢 ∧ x ∈ D ∧ D ⊆ C) ∧
               (∀X ∈ 𝒢. polytope X) ∧
               (∀X ∈ 𝒢. aff_dim X ≤ d) ∧
               (∀X ∈ 𝒢. ∀Y ∈ 𝒢. X ∩ Y face_of X) ∧
               (∀X ∈ 𝒢. ∀x ∈ X. ∀y ∈ X. ∀a b.
                     (a,b) ∈ I ⟶ a • x ≤ b ∧ a • y ≤ b ∨
                                 a • x ≥ b ∧ a • y ≥ b)
  using ‹finite I›
proof induction
  case empty
  then show ?case
   by (rule_tac x=ℱ in exI) (auto simp: assms)
next
  case (insert ab I)
  then obtain 𝒢 where eq: ⋃𝒢 = ⋃ℱ and finite 𝒢
                and sub1: ⋀C. C ∈ 𝒢 ⟹ ∃D. D ∈ ℱ ∧ C ⊆ D
                 and sub2: ⋀C x. C ∈ ℱ ∧ x ∈ C ⟹ ∃D. D ∈ 𝒢 ∧ x ∈ D ∧ D
⊆ C
                and poly: ⋀X. X ∈ 𝒢 ⟹ polytope X
                and aff: ⋀X. X ∈ 𝒢 ⟹ aff_dim X ≤ d
                and face: ⋀X Y. ⟦X ∈ 𝒢; Y ∈ 𝒢⟧ ⟹ X ∩ Y face_of X
                and I: ⋀X x y a b. ⟦X ∈ 𝒢; x ∈ X; y ∈ X; (a,b) ∈ I⟧ ⟹
                            a • x ≤ b ∧ a • y ≤ b ∨ a • x ≥ b ∧ a • y ≥ b
   by (auto simp: that)
  obtain a b where ab = (a,b)
   by fastforce
  let ?𝒢 = (λX. X ∩ {x. a • x ≤ b}) ‘ 𝒢 ∪ (λX. X ∩ {x. a • x ≥ b}) ‘ 𝒢
  have eqInt: (S ∩ Collect P) ∩ (T ∩ Collect Q) = (S ∩ T) ∩ (Collect P ∩ Collect
Q) for S T::'a set and P Q
   by blast
  show ?case
```

proof (*intro conjI exI*)
show $\bigcup ?\mathcal{G} = \bigcup \mathcal{F}$
by (*force simp*: *eq* [*symmetric*])
show *finite* $?\mathcal{G}$
using ‹*finite* $\mathcal{G}$› **by** *force*
show $\forall X \in ?\mathcal{G}.$ *polytope* X
by (*force simp*: *poly polytope_Int_polyhedron polyhedron_halfspace_le polyhedron_halfspace_ge*)
show $\forall X \in ?\mathcal{G}.$ *aff_dim* $X \leq d$
by (*auto*; *metis order_trans aff aff_dim_subset inf_le1*)
show $\forall X \in ?\mathcal{G}.\ \forall x \in X.\ \forall y \in X.\ \forall a\ b.$
$(a,b) \in$ *insert ab* $I \longrightarrow a \cdot x \leq b \wedge a \cdot y \leq b \vee$
$a \cdot x \geq b \wedge a \cdot y \geq b$
using ‹$ab = (a, b)$› I **by** *fastforce*
show $\forall X \in ?\mathcal{G}.\ \forall Y \in ?\mathcal{G}.\ X \cap Y$ *face_of* X
by (*auto simp*: *eqInt halfspace_Int_eq face_of_Int_Int face face_of_halfspace_le face_of_halfspace_ge*)
show $\forall C \in ?\mathcal{G}.\ \exists D.\ D \in \mathcal{F} \wedge C \subseteq D$
using *sub1* **by** *force*
show $\forall C{\in}\mathcal{F}.\ \forall x{\in}C.\ \exists D.\ D \in ?\mathcal{G} \wedge x \in D \wedge D \subseteq C$
proof (*intro ballI*)
fix $C\ z$
assume $C \in \mathcal{F}\ z \in C$
with *sub2* **obtain** D **where** D: $D \in \mathcal{G}\ z \in D\ D \subseteq C$ **by** *blast*
have $D \in \mathcal{G} \wedge z \in D \cap \{x.\ a \cdot x \leq b\} \wedge D \cap \{x.\ a \cdot x \leq b\} \subseteq C \vee$
$D \in \mathcal{G} \wedge z \in D \cap \{x.\ a \cdot x \geq b\} \wedge D \cap \{x.\ a \cdot x \geq b\} \subseteq C$
using *linorder_class.linear* [*of* $a \cdot z\ b$] D **by** *blast*
then show $\exists D.\ D \in ?\mathcal{G} \wedge z \in D \wedge D \subseteq C$
by *blast*
qed
qed
qed

proposition *cell_complex_subdivision_exists*:
fixes $\mathcal{F}$ :: *'a*::*euclidean_space set set*
assumes $0 < e$ *finite* $\mathcal{F}$
and *poly*: $\bigwedge X.\ X \in \mathcal{F} \Longrightarrow$ *polytope* X
and *aff*: $\bigwedge X.\ X \in \mathcal{F} \Longrightarrow$ *aff_dim* $X \leq d$
and *face*: $\bigwedge X\ Y.\ [\![X \in \mathcal{F};\ Y \in \mathcal{F}]\!] \Longrightarrow X \cap Y$ *face_of* X
obtains $\mathcal{F}'$ **where** *finite* $\mathcal{F}'$ $\bigcup \mathcal{F}' = \bigcup \mathcal{F}$ $\bigwedge X.\ X \in \mathcal{F}' \Longrightarrow$ *diameter* $X < e$
$\bigwedge X.\ X \in \mathcal{F}' \Longrightarrow$ *polytope* X $\bigwedge X.\ X \in \mathcal{F}' \Longrightarrow$ *aff_dim* $X \leq d$
$\bigwedge X\ Y.\ [\![X \in \mathcal{F}';\ Y \in \mathcal{F}']\!] \Longrightarrow X \cap Y$ *face_of* X
$\bigwedge C.\ C \in \mathcal{F}' \Longrightarrow \exists D.\ D \in \mathcal{F} \wedge C \subseteq D$
$\bigwedge C\ x.\ C \in \mathcal{F} \wedge x \in C \Longrightarrow \exists D.\ D \in \mathcal{F}' \wedge x \in D \wedge D \subseteq C$
proof −
have *bounded*($\bigcup \mathcal{F}$)
by (*simp add*: ‹*finite* $\mathcal{F}$› *poly bounded_Union polytope_imp_bounded*)
then obtain B **where** $B > 0$ **and** B: $\bigwedge x.\ x \in \bigcup \mathcal{F} \Longrightarrow$ *norm* $x < B$

```
    by (meson bounded_pos_less)
  define C where C ≡ {z ∈ ℤ. |z * e / 2 / real DIM('a)| ≤ B}
  define I where I ≡ ⋃i ∈ Basis. ⋃j ∈ C. { (i::'a, j * e / 2 / DIM('a)) }
  have C ⊆ {x ∈ ℤ. - B / (e / 2 / real DIM('a)) ≤ x ∧ x ≤ B / (e / 2 / real
DIM('a))}
    using ‹0 < e› by (auto simp: field_split_simps C_def)
  then have finite C
    using finite_int_segment finite_subset by blast
  then have finite I
    by (simp add: I_def)
  obtain 𝓕' where eq: ⋃𝓕' = ⋃𝓕 and finite 𝓕'
             and poly: ⋀X. X ∈ 𝓕' ⟹ polytope X
             and aff: ⋀X. X ∈ 𝓕' ⟹ aff_dim X ≤ d
             and face: ⋀X Y. ⟦X ∈ 𝓕'; Y ∈ 𝓕'⟧ ⟹ X ∩ Y face_of X
             and I: ⋀X x y a b. ⟦X ∈ 𝓕'; x ∈ X; y ∈ X; (a,b) ∈ I⟧ ⟹
                                  a • x ≤ b ∧ a • y ≤ b ∨ a • x ≥ b ∧ a • y ≥ b
             and sub1: ⋀C. C ∈ 𝓕' ⟹ ∃D. D ∈ 𝓕 ∧ C ⊆ D
            and sub2: ⋀C x. C ∈ 𝓕 ∧ x ∈ C ⟹ ∃D. D ∈ 𝓕' ∧ x ∈ D ∧ D ⊆ C
    apply (rule exE [OF cell_subdivision_lemma])
    using assms ‹finite I› by auto
  show ?thesis
  proof (rule_tac 𝓕'=𝓕' in that)
    show diameter X < e if X ∈ 𝓕' for X
    proof -
      have diameter X ≤ e/2
      proof (rule diameter_le)
        show norm (x - y) ≤ e / 2 if x ∈ X y ∈ X for x y
        proof -
          have norm x < B norm y < B
            using B ‹X ∈ 𝓕'› eq that by blast+
          have norm (x - y) ≤ (∑b∈Basis. |(x-y) • b|)
            by (rule norm_le_l1)
          also have ... ≤ of_nat (DIM('a)) * (e / 2 / DIM('a))
          proof (rule sum_bounded_above)
            fix i::'a
            assume i ∈ Basis
            then have I': ⋀z b. ⟦z ∈ C; b = z * e / (2 * real DIM('a))⟧ ⟹ i • x
≤ b ∧ i • y ≤ b ∨ i • x ≥ b ∧ i • y ≥ b
              using I[of X x y] ‹X ∈ 𝓕'› that unfolding I_def by auto
            show |(x - y) • i| ≤ e / 2 / real DIM('a)
            proof (rule ccontr)
              assume ¬ |(x - y) • i| ≤ e / 2 / real DIM('a)
              then have xyi: |i • x - i • y| > e / 2 / real DIM('a)
                by (simp add: inner_commute inner_diff_right)
              obtain n where n ∈ ℤ and n: i • x < n * (e / 2 / real DIM('a)) ∧
n * (e / 2 / real DIM('a)) < i • y ∨ i • y < n * (e / 2 / real DIM('a)) ∧ n * (e
/ 2 / real DIM('a)) < i • x
                using subdivide_interval [OF xyi] DIM_positive ‹0 < e›
                by (auto simp: zero_less_divide_iff)
```

```
          have |i · x| < B
        by (metis ‹i ∈ Basis› ‹norm x < B› inner_commute norm_bound_Basis_lt)
          have |i · y| < B
        by (metis ‹i ∈ Basis› ‹norm y < B› inner_commute norm_bound_Basis_lt)
          have *: |n * e| ≤ B * (2 * real DIM('a))
                if |ix| < B |iy| < B
                  and ix: ix * (2 * real DIM('a)) < n * e
                  and iy: n * e < iy * (2 * real DIM('a)) for ix iy
          proof (rule abs_leI)
            have iy * (2 * real DIM('a)) ≤ B * (2 * real DIM('a))
              by (rule mult_right_mono) (use ‹|iy| < B› in linarith)+
            then show n * e ≤ B * (2 * real DIM('a))
              using iy by linarith
          next
            have - ix * (2 * real DIM('a)) ≤ B * (2 * real DIM('a))
              by (rule mult_right_mono) (use ‹|ix| < B› in linarith)+
            then show - (n * e) ≤ B * (2 * real DIM('a))
              using ix by linarith
          qed
          have n ∈ C
           using ‹n ∈ ℤ› n  by (auto simp: C_def divide_simps intro: * ‹|i · x|
< B› ‹|i · y| < B›)
          show False
            using  I' [OF ‹n ∈ C› refl] n  by auto
        qed
      qed
      also have ... = e / 2
        by simp
      finally show ?thesis .
    qed
  qed (use ‹0 < e› in force)
  also have ... < e
    by (simp add: ‹0 < e›)
  finally show ?thesis .
 qed
qed (auto simp: eq poly aff face sub1 sub2 ‹finite ℱ'›)
qed
```

6.34.17 Simplexes

The notion of n-simplex for integer $-(1::'a) \leq n$

```
definition simplex :: int ⇒ 'a::euclidean_space set ⇒ bool (infix simplex 50)
  where n simplex S ≡ ∃ C. ¬ affine_dependent C ∧ int(card C) = n + 1 ∧ S =
convex hull C

lemma simplex:
    n simplex S ⟷ (∃ C. finite C ∧
                    ¬ affine_dependent C ∧
                    int(card C) = n + 1 ∧
```

```
                    S = convex hull C)
  by (auto simp add: simplex_def intro: aff_independent_finite)

lemma simplex_convex_hull:
   ¬ affine_dependent C ∧ int(card C) = n + 1 ⟹ n simplex (convex hull C)
  by (auto simp add: simplex_def)

lemma convex_simplex: n simplex S ⟹ convex S
  by (metis convex_convex_hull simplex_def)

lemma compact_simplex: n simplex S ⟹ compact S
  unfolding simplex
  using finite_imp_compact_convex_hull by blast

lemma closed_simplex: n simplex S ⟹ closed S
  by (simp add: compact_imp_closed compact_simplex)

lemma simplex_imp_polytope:
   n simplex S ⟹ polytope S
  unfolding simplex_def polytope_def
  using aff_independent_finite by blast

lemma simplex_imp_polyhedron:
   n simplex S ⟹ polyhedron S
  by (simp add: polytope_imp_polyhedron simplex_imp_polytope)

lemma simplex_dim_ge: n simplex S ⟹ -1 ≤ n
  by (metis (no_types, opaque_lifting) aff_dim_geq affine_independent_iff_card
diff_add_cancel diff_diff_eq2 simplex_def)

lemma simplex_empty [simp]: n simplex {} ⟷ n = -1
proof
  assume n simplex {}
  then show n = -1
   unfolding simplex by (metis card.empty convex_hull_eq_empty diff_0 diff_eq_eq
of_nat_0)
next
  assume n = -1 then show n simplex {}
    by (fastforce simp: simplex)
qed

lemma simplex_minus_1 [simp]: -1 simplex S ⟷ S = {}
 by (metis simplex cancel_comm_monoid_add_class.diff_cancel card_0_eq diff_minus_eq_add
of_nat_eq_0_iff simplex_empty)


lemma aff_dim_simplex:
   n simplex S ⟹ aff_dim S = n
 by (metis simplex add.commute add_diff_cancel_left' aff_dim_convex_hull affine_independent_iff_ca
```

lemma *zero_simplex_sing*: *0 simplex* {*a*}
using *affine_independent_1 simplex_convex_hull* **by** *fastforce*

lemma *simplex_sing* [*simp*]: *n simplex* {*a*} ⟷ *n = 0*
using *aff_dim_simplex aff_dim_sing zero_simplex_sing* **by** *blast*

lemma *simplex_zero*: *0 simplex S* ⟷ (∃ *a*. *S* = {*a*})
by (*metis aff_dim_eq_0 aff_dim_simplex simplex_sing*)

lemma *one_simplex_segment*: *a* ≠ *b* ⟹ *1 simplex closed_segment a b*
unfolding *simplex_def*
by (*rule_tac x*={*a*,*b*} **in** *exI*) (*auto simp*: *segment_convex_hull*)

lemma *simplex_segment_cases*:
(*if a = b then 0 else 1*) *simplex closed_segment a b*
by (*auto simp*: *one_simplex_segment*)

lemma *simplex_segment*:
∃ *n*. *n simplex closed_segment a b*
using *simplex_segment_cases* **by** *metis*

lemma *polytope_lowdim_imp_simplex*:
assumes *polytope P aff_dim P* ≤ *1*
obtains *n* **where** *n simplex P*
proof (*cases P* = {})
case *True*
then show *?thesis*
by (*simp add*: *that*)
next
case *False*
then show *?thesis*
by (*metis assms compact_convex_collinear_segment collinear_aff_dim polytope_imp_compact polytope_imp_convex simplex_segment_cases that*)
qed

lemma *simplex_insert_dimplus1*:
fixes *n*::*int*
assumes *n simplex S* **and** *a*: *a* ∉ *affine hull S*
shows (*n*+*1*) *simplex* (*convex hull* (*insert a S*))
proof −
obtain *C* **where** *C*: *finite C* ¬ *affine_dependent C int*(*card C*) = *n*+*1* **and** *S*: *S* = *convex hull C*
using *assms* **unfolding** *simplex* **by** *force*
show *?thesis*
unfolding *simplex*
proof (*intro exI conjI*)
have *aff_dim S* = *n*
using *aff_dim_simplex assms*(*1*) **by** *blast*

moreover have *a* ∉ *affine hull C*
using *S a affine_hull_convex_hull* **by** *blast*
moreover have *a* ∉ *C*
using *S a hull_inc* **by** *fastforce*
ultimately show ¬ *affine_dependent* (*insert a C*)
by (*simp add*: *C S aff_dim_convex_hull aff_dim_insert affine_independent_iff_card*)
next
have *a* ∉ *C*
using *S a hull_inc* **by** *fastforce*
then show *int* (*card* (*insert a C*)) = *n* + *1* + *1*
by (*simp add*: *C*)
next
show *convex hull insert a S* = *convex hull* (*insert a C*)
by (*simp add*: *S convex_hull_insert_segments*)
qed (*use C* **in** *auto*)
qed

6.34.18 Simplicial complexes and triangulations

definition *simplicial_complex* **where**
simplicial_complex $\mathcal{C} \equiv$
finite $\mathcal{C} \wedge$
$(\forall S \in \mathcal{C}.\ \exists n.\ n$ *simplex* $S) \wedge$
$(\forall F\ S.\ S \in \mathcal{C} \wedge F$ *face_of* $S \longrightarrow F \in \mathcal{C}) \wedge$
$(\forall S\ S'.\ S \in \mathcal{C} \wedge S' \in \mathcal{C} \longrightarrow (S \cap S')$ *face_of* $S)$

definition *triangulation* **where**
triangulation $\mathcal{T} \equiv$
finite $\mathcal{T} \wedge$
$(\forall T \in \mathcal{T}.\ \exists n.\ n$ *simplex* $T) \wedge$
$(\forall T\ T'.\ T \in \mathcal{T} \wedge T' \in \mathcal{T} \longrightarrow (T \cap T')$ *face_of* $T)$

6.34.19 Refining a cell complex to a simplicial complex

proposition *convex_hull_insert_Int_eq*:
fixes *z* :: *'a* :: *euclidean_space*
assumes *z*: *z* ∈ *rel_interior S*
and *T*: *T* ⊆ *rel_frontier S*
and *U*: *U* ⊆ *rel_frontier S*
and *convex S convex T convex U*
shows *convex hull* (*insert z T*) ∩ *convex hull* (*insert z U*) = *convex hull* (*insert z* (*T* ∩ *U*))
(**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs*
proof (*cases T*={} ∨ *U*={})
case *True* **then show** *?thesis* **by** *auto*
next
case *False*
then have *T* ≠ {} *U* ≠ {} **by** *auto*

```
  have TU: convex (T ∩ U)
    by (simp add: ‹convex T› ‹convex U› convex_Int)
  have (⋃x∈T. closed_segment z x) ∩ (⋃x∈U. closed_segment z x)
        ⊆ (if T ∩ U = {} then {z} else ⋃((closed_segment z) ` (T ∩ U))) (is _
⊆ ?IF)
  proof clarify
    fix x t u
    assume xt: x ∈ closed_segment z t
      and xu: x ∈ closed_segment z u
      and t ∈ T u ∈ U
    then have ne: t ≠ z u ≠ z
      using T U z unfolding rel_frontier_def by blast+
    show x ∈ ?IF
    proof (cases x = z)
      case True then show ?thesis by auto
    next
      case False
      have t: t ∈ closure S
        using T ‹t ∈ T› rel_frontier_def by auto
      have u: u ∈ closure S
        using U ‹u ∈ U› rel_frontier_def by auto
      show ?thesis
      proof (cases t = u)
        case True
        then show ?thesis
          using ‹t ∈ T› ‹u ∈ U› xt by auto
      next
        case False
        have tnot: t ∉ closed_segment u z
        proof -
          have t ∈ closure S - rel_interior S
            using T ‹t ∈ T› rel_frontier_def by blast
          then have t ∉ open_segment z u
            by (meson DiffD2 rel_interior_closure_convex_segment [OF ‹convex
S› z u] subsetD)
          then show ?thesis
          by (simp add: ‹t ≠ u› ‹t ≠ z› open_segment_commute open_segment_def)
        qed
        moreover have u ∉ closed_segment z t
          using rel_interior_closure_convex_segment [OF ‹convex S› z t] ‹u ∈
U› ‹u ≠ z›
            U [unfolded rel_frontier_def] tnot
          by (auto simp: closed_segment_eq_open)
        ultimately
        have ¬(between (t,u) z | between (u,z) t | between (z,t) u) if x ≠ z
          using that xt xu
          by (meson between_antisym between_mem_segment between_trans_2
ends_in_segment(2))
        then have ¬ collinear {t, z, u} if x ≠ z
```

```
          by (auto simp: that collinear_between_cases between_commute)
        moreover have collinear {t, z, x}
        by (metis closed_segment_commute collinear_2 collinear_closed_segment
collinear_triples ends_in_segment(1) insert_absorb insert_absorb2 xt)
        moreover have collinear {z, x, u}
        by (metis closed_segment_commute collinear_2 collinear_closed_segment
collinear_triples ends_in_segment(1) insert_absorb insert_absorb2 xu)
        ultimately have False
          using collinear_3_trans [of t z x u] ‹x ≠ z› by blast
        then show ?thesis by metis
      qed
    qed
  qed
  then show ?thesis
    using False ‹convex T› ‹convex U› TU
    by (simp add: convex_hull_insert_segments hull_same split: if_split_asm)
 qed
 show ?rhs ⊆ ?lhs
  by (metis inf_greatest hull_mono inf.cobounded1 inf.cobounded2 insert_mono)
qed

lemma simplicial_subdivision_aux:
 assumes finite ℳ
    and ⋀C. C ∈ ℳ ⟹ polytope C
    and ⋀C. C ∈ ℳ ⟹ aff_dim C ≤ of_nat n
    and ⋀C F. ⟦C ∈ ℳ; F face_of C⟧ ⟹ F ∈ ℳ
    and ⋀C1 C2. ⟦C1 ∈ ℳ; C2 ∈ ℳ⟧ ⟹ C1 ∩ C2 face_of C1
  shows ∃𝒯. simplicial_complex 𝒯 ∧
              (∀K ∈ 𝒯. aff_dim K ≤ of_nat n) ∧
              ⋃𝒯 = ⋃ℳ ∧
              (∀C ∈ ℳ. ∃F. finite F ∧ F ⊆ 𝒯 ∧ C = ⋃F) ∧
              (∀K ∈ 𝒯. ∃C. C ∈ ℳ ∧ K ⊆ C)
 using assms
proof (induction n arbitrary: ℳ rule: less_induct)
 case (less n)
 then have polyℳ: ⋀C. C ∈ ℳ ⟹ polytope C
   and affℳ:   ⋀C. C ∈ ℳ ⟹ aff_dim C ≤ of_nat n
   and faceℳ:  ⋀C F. ⟦C ∈ ℳ; F face_of C⟧ ⟹ F ∈ ℳ
   and intfaceℳ: ⋀C1 C2. ⟦C1 ∈ ℳ; C2 ∈ ℳ⟧ ⟹ C1 ∩ C2 face_of C1
   by metis+
 show ?case
 proof (cases n ≤ 1)
   case True
   have ⋀s. ⟦n ≤ 1; s ∈ ℳ⟧ ⟹ ∃m. m simplex s
     using polyℳ affℳ by (force intro: polytope_lowdim_imp_simplex)
   then show ?thesis
     unfolding simplicial_complex_def using True
     by (rule_tac x=ℳ in exI) (auto simp: less.prems)
 next
```

case *False*
define $\mathcal{S}$ **where** $\mathcal{S} \equiv \{C \in \mathcal{M}.\ \mathit{aff\_dim}\ C < n\}$
have *finite* $\mathcal{S}$ $\bigwedge C.\ C \in \mathcal{S} \Longrightarrow \mathit{polytope}\ C$ $\bigwedge C.\ C \in \mathcal{S} \Longrightarrow \mathit{aff\_dim}\ C \leq \mathit{int}\ (n - 1)$
$\bigwedge C1\ C2.\ [\![C1 \in \mathcal{S};\ C2 \in \mathcal{S}]\!] \Longrightarrow C1 \cap C2\ \mathit{face\_of}\ C1$
using *less.prems* **by** (*auto simp*: $\mathcal{S}$*_def*)
moreover have §: $\bigwedge C\ F.\ [\![C \in \mathcal{S};\ F\ \mathit{face\_of}\ C]\!] \Longrightarrow F \in \mathcal{S}$
using *less.prems* **unfolding** $\mathcal{S}$*_def*
by (*metis* (*no_types*, *lifting*) *mem_Collect_eq aff_dim_subset face_of_imp_subset less_le not_le*)
ultimately obtain $\mathcal{U}$ **where** *simplicial_complex* $\mathcal{U}$
and *aff_dim*$\mathcal{U}$: $\bigwedge K.\ K \in \mathcal{U} \Longrightarrow \mathit{aff\_dim}\ K \leq \mathit{int}\ (n - 1)$
and $\bigcup \mathcal{U} = \bigcup \mathcal{S}$
and *fin*$\mathcal{U}$: $\bigwedge C.\ C \in \mathcal{S} \Longrightarrow \exists F.\ \mathit{finite}\ F \wedge F \subseteq \mathcal{U} \wedge C = \bigcup F$
and *C*$\mathcal{U}$: $\bigwedge K.\ K \in \mathcal{U} \Longrightarrow \exists C.\ C \in \mathcal{S} \wedge K \subseteq C$
using *less.IH* [*of* $n-1$ $\mathcal{S}$] *False* **by** *auto*
then have *finite* $\mathcal{U}$
and *simpl*$\mathcal{U}$: $\bigwedge S.\ S \in \mathcal{U} \Longrightarrow \exists n.\ n\ \mathit{simplex}\ S$
and *face*$\mathcal{U}$: $\bigwedge F\ S.\ [\![S \in \mathcal{U};\ F\ \mathit{face\_of}\ S]\!] \Longrightarrow F \in \mathcal{U}$
and *faceI*$\mathcal{U}$: $\bigwedge S\ S'.\ [\![S \in \mathcal{U};\ S' \in \mathcal{U}]\!] \Longrightarrow (S \cap S')\ \mathit{face\_of}\ S$
by (*auto simp*: *simplicial_complex_def*)
define $\mathcal{N}$ **where** $\mathcal{N} \equiv \{C \in \mathcal{M}.\ \mathit{aff\_dim}\ C = n\}$
have *finite* $\mathcal{N}$
by (*simp add*: $\mathcal{N}$*_def less.prems*(*1*))
have *poly*$\mathcal{N}$: $\bigwedge C.\ C \in \mathcal{N} \Longrightarrow \mathit{polytope}\ C$
and *convex*$\mathcal{N}$: $\bigwedge C.\ C \in \mathcal{N} \Longrightarrow \mathit{convex}\ C$
and *closed*$\mathcal{N}$: $\bigwedge C.\ C \in \mathcal{N} \Longrightarrow \mathit{closed}\ C$
by (*auto simp*: $\mathcal{N}$*_def poly*$\mathcal{M}$ *polytope_imp_convex polytope_imp_closed*)
have *in_rel_interior*: $(\mathit{SOME}\ z.\ z \in \mathit{rel\_interior}\ C) \in \mathit{rel\_interior}\ C$ **if** $C \in \mathcal{N}$ **for** C
using *that poly*$\mathcal{M}$ *polytope_imp_convex rel_interior_aff_dim some_in_eq* **by** (*fastforce simp*: $\mathcal{N}$*_def*)
have $*$: $\exists T.\ \neg\ \mathit{affine\_dependent}\ T \wedge \mathit{card}\ T \leq n \wedge \mathit{aff\_dim}\ K < n \wedge K = \mathit{convex\ hull}\ T$
if $K \in \mathcal{U}$ **for** K
proof −
obtain r **where** r: $r\ \mathit{simplex}\ K$
using ‹$K \in \mathcal{U}$› *simpl*$\mathcal{U}$ **by** *blast*
have $r = \mathit{aff\_dim}\ K$
using ‹$r\ \mathit{simplex}\ K$› *aff_dim_simplex* **by** *blast*
with r
show *?thesis*
unfolding *simplex_def*
using *False* ‹$\bigwedge K.\ K \in \mathcal{U} \Longrightarrow \mathit{aff\_dim}\ K \leq \mathit{int}\ (n - 1)$› *that* **by** *fastforce*
qed
have *ahK_C_disjoint*: $\mathit{affine\ hull}\ K \cap \mathit{rel\_interior}\ C = \{\}$
if $C \in \mathcal{N}$ $K \in \mathcal{U}$ $K \subseteq \mathit{rel\_frontier}\ C$ **for** $C\ K$
proof −
have *convex* C *closed* C

```
      by (auto simp: convexN closedN ‹C ∈ N›)
    obtain F where F: F face_of C and F ≠ C K ⊆ F
    proof −
      obtain L where L ∈ S K ⊆ L
        using ‹K ∈ U› CU by blast
      have K ≤ rel_frontier C
        by (simp add: ‹K ⊆ rel_frontier C›)
      also have ... ≤ C
        by (simp add: ‹closed C› rel_frontier_def subset_iff)
      finally have K ⊆ C .
      have L ∩ C face_of C
            using N_def S_def ‹C ∈ N› ‹L ∈ S› intfaceM by (simp add:
inf_commute)
      moreover have L ∩ C ≠ C
        using ‹C ∈ N› ‹L ∈ S›
        by (metis (mono_tags, lifting) N_def S_def intfaceM mem_Collect_eq
not_le order_refl §)
      moreover have K ⊆ L ∩ C
       using ‹C ∈ N› ‹L ∈ S› ‹K ⊆ C› ‹K ⊆ L› by (auto simp: N_def S_def)
      ultimately show ?thesis using that by metis
    qed
    have affine hull F ∩ rel_interior C = {}
      by (rule affine_hull_face_of_disjoint_rel_interior [OF ‹convex C› F ‹F
≠ C›])
    with hull_mono [OF ‹K ⊆ F›]
    show affine hull K ∩ rel_interior C = {}
      by fastforce
  qed
  let ?T = (⋃ C ∈ N. ⋃ K ∈ U ∩ Pow (rel_frontier C).
                {convex hull (insert (SOME z. z ∈ rel_interior C) K)})
  have ∃ T. simplicial_complex T ∧
          (∀ K ∈ T. aff_dim K ≤ of_nat n) ∧
          (∀ C ∈ M. ∃ F. F ⊆ T ∧ C = ⋃ F) ∧
          (∀ K ∈ T. ∃ C. C ∈ M ∧ K ⊆ C)
  proof (rule exI, intro conjI ballI)
    show simplicial_complex (U ∪ ?T)
      unfolding simplicial_complex_def
    proof (intro conjI impI ballI allI)
      show finite (U ∪ ?T)
        using ‹finite U› ‹finite N› by simp
      show ∃ n. n simplex S if S ∈ U ∪ ?T for S
     using that ahK_C_disjoint in_rel_interior simplU simplex_insert_dimplus1
by fastforce
      show F ∈ U ∪ ?T if S: S ∈ U ∪ ?T ∧ F face_of S for F S
      proof −
        have F ∈ U if S ∈ U
          using S faceU that by blast
        moreover have F ∈ U ∪ ?T
          if F face_of S C ∈ N K ∈ U and K ⊆ rel_frontier C
```

```
          and S: S = convex hull insert (SOME z. z ∈ rel_interior C) K for C
K
        proof −
          let ?z = SOME z. z ∈ rel_interior C
          have ?z ∈ rel_interior C
            by (simp add: in_rel_interior ‹C ∈ 𝒩›)
          moreover
          obtain I where ¬ affine_dependent I card I ≤ n aff_dim K < int n K
= convex hull I
            using ∗ [OF ‹K ∈ 𝒰›] by auto
          ultimately have ?z ∉ affine hull I
            using ahK_C_disjoint affine_hull_convex_hull that by blast
          have compact I finite I
              by (auto simp: ‹¬ affine_dependent I› aff_independent_finite fi-
nite_imp_compact)
          moreover have F face_of convex hull insert ?z I
           by (metis S ‹F face_of S› ‹K = convex hull I› convex_hull_eq_empty
convex_hull_insert_segments hull_hull)
          ultimately obtain J where J: J ⊆ insert ?z I F = convex hull J
            using face_of_convex_hull_subset [of insert ?z I F] by auto
          show ?thesis
          proof (cases ?z ∈ J)
            case True
           have F ∈ (⋃ K∈𝒰 ∩ Pow (rel_frontier C). {convex hull insert ?z K})
            proof
              have convex hull (J − {?z}) face_of K
                    by (metis True ‹J ⊆ insert ?z I› ‹K = convex hull I› ‹¬
affine_dependent I› face_of_convex_hull_affine_independent subset_insert_iff)
              then have convex hull (J − {?z}) ∈ 𝒰
                by (rule face𝒰 [OF ‹K ∈ 𝒰›])
              moreover
              have ⋀x. x ∈ convex hull (J − {?z}) ⟹ x ∈ rel_frontier C
                  by (metis True ‹J ⊆ insert ?z I› ‹K = convex hull I› subsetD
hull_mono subset_insert_iff that(4))
               ultimately show convex hull (J − {?z}) ∈ 𝒰 ∩ Pow (rel_frontier
C) by auto
              let ?F = convex hull insert ?z (convex hull (J − {?z}))
              have F ⊆ ?F
                    by (simp add: ‹F = convex hull J› hull_mono hull_subset
subset_insert_iff)
              moreover have ?F ⊆ F
                   by (metis True ‹F = convex hull J› hull_insert insert_Diff
set_eq_subset)
              ultimately
              show F ∈ {?F} by auto
            qed
            with ‹C∈𝒩› show ?thesis by auto
          next
            case False
```

```
          then have F ∈ 𝒰
         using face_of_convex_hull_affine_independent [OF ‹¬ affine_dependent
I›]
            by (metis J ‹K = convex hull I› face𝒰 subset_insert ‹K ∈ 𝒰›)
          then show F ∈ 𝒰 ∪ ?𝒯
            by blast
         qed
        qed
        ultimately show ?thesis
         using that by auto
       qed
       have §: X ∩ Y face_of X ∧ X ∩ Y face_of Y
        if XY: X ∈ 𝒰 Y ∈ ?𝒯 for X Y
       proof −
        obtain C K
         where C ∈ 𝒩 K ∈ 𝒰 K ⊆ rel_frontier C
          and Y: Y = convex hull insert (SOME z. z ∈ rel_interior C) K
         using XY by blast
        have convex C
         by (simp add: ‹C ∈ 𝒩› convex𝒩)
        have K ⊆ C
         by (metis DiffE ‹C ∈ 𝒩› ‹K ⊆ rel_frontier C› closed𝒩 closure_closed
rel_frontier_def subset_iff)
        let ?z = (SOME z. z ∈ rel_interior C)
        have z: ?z ∈ rel_interior C
         using ‹C ∈ 𝒩› in_rel_interior by blast
        obtain D where D ∈ 𝒮 X ⊆ D
         using C𝒰 ‹X ∈ 𝒰› by blast
        have D ∩ rel_interior C = (C ∩ D) ∩ rel_interior C
         using rel_interior_subset by blast
        also have (C ∩ D) ∩ rel_interior C = {}
        proof (rule face_of_disjoint_rel_interior)
         show C ∩ D face_of C
          using 𝒩_def 𝒮_def ‹C ∈ 𝒩› ‹D ∈ 𝒮› intface𝒩 by blast
         show C ∩ D ≠ C
          by (metis (mono_tags, lifting) Int_lower2 𝒩_def 𝒮_def ‹C ∈ 𝒩› ‹D
∈ 𝒮› aff_dim_subset mem_Collect_eq not_le)
        qed
        finally have DC: D ∩ rel_interior C = {} .
        have eq: X ∩ convex hull (insert ?z K) = X ∩ convex hull K
         proof (rule Int_convex_hull_insert_rel_exterior [OF ‹convex C› ‹K ⊆
C› z])
         show disjnt X (rel_interior C)
          using DC by (meson ‹X ⊆ D› disjnt_def disjnt_subset1)
        qed
        obtain I where I: ¬ affine_dependent I
         and Keq: K = convex hull I and [simp]: convex hull K = K
         using ∗ ‹K ∈ 𝒰› by force
        then have ?z ∉ affine hull I
```

```
            using ahK_C_disjoint ‹C ∈ 𝒩› ‹K ∈ 𝒰› ‹K ⊆ rel_frontier C›
affine_hull_convex_hull z by blast
        have X ∩ K face_of K
          by (simp add: XY(1) ‹K ∈ 𝒰› faceI𝒰 inf_commute)
        also have ... face_of convex hull insert ?z K
            by (metis I Keq ‹?z ∉ affine hull I› aff_independent_finite con-
vex_convex_hull face_of_convex_hull_insert face_of_refl hull_insert)
        finally have X ∩ K face_of convex hull insert ?z K .
        then show ?thesis
          by (simp add: XY(1) Y ‹K ∈ 𝒰› eq faceI𝒰)
      qed

      show S ∩ S' face_of S
        if S ∈ 𝒰 ∪ ?𝒯 ∧ S' ∈ 𝒰 ∪ ?𝒯 for S S'
        using that
      proof (elim conjE UnE)
        fix X Y
        assume X ∈ 𝒰 and Y ∈ 𝒰
        then show X ∩ Y face_of X
          by (simp add: faceI𝒰)
      next
        fix X Y
        assume XY: X ∈ 𝒰 Y ∈ ?𝒯
        then show X ∩ Y face_of X Y ∩ X face_of Y
          using § [OF XY] by (auto simp: Int_commute)
      next
        fix X Y
        assume XY: X ∈ ?𝒯 Y ∈ ?𝒯
        show X ∩ Y face_of X
        proof −
          obtain C K D L
            where C ∈ 𝒩 K ∈ 𝒰 K ⊆ rel_frontier C
              and X: X = convex hull insert (SOME z. z ∈ rel_interior C) K
              and D ∈ 𝒩 L ∈ 𝒰 L ⊆ rel_frontier D
              and Y: Y = convex hull insert (SOME z. z ∈ rel_interior D) L
            using XY by blast
          let ?z = (SOME z. z ∈ rel_interior C)
          have z: ?z ∈ rel_interior C
            using ‹C ∈ 𝒩› in_rel_interior by blast
          have convex C
            by (simp add: ‹C ∈ 𝒩› convex𝒩)
          have convex K
            using ∗ ‹K ∈ 𝒰› by blast
          have convex L
            by (meson ‹L ∈ 𝒰› convex_simplex simpl𝒰)
          show ?thesis
          proof (cases D=C)
            case True
            then have L ⊆ rel_frontier C
```

```
            using ‹L ⊆ rel_frontier D› by auto
          have convex hull insert (SOME z. z ∈ rel_interior C) (K ∩ L) face_of
                convex hull insert (SOME z. z ∈ rel_interior C) K
            by (metis IntI ‹C ∈ 𝒩› ‹K ∈ 𝒰› ‹K ⊆ rel_frontier C› ‹L
∈ 𝒰› ahK_C_disjoint empty_iff faceI𝒰 face_of_polytope_insert2 simpl𝒰 sim-
plex_imp_polytope z)
          then show ?thesis
            using True X Y ‹K ⊆ rel_frontier C› ‹L ⊆ rel_frontier C› ‹convex
C› ‹convex K› ‹convex L› convex_hull_insert_Int_eq z by force
        next
          case False
          have convex D
            by (simp add: ‹D ∈ 𝒩› convex𝒩)
          have K ⊆ C
          by (metis DiffE ‹C ∈ 𝒩› ‹K ⊆ rel_frontier C› closed𝒩 closure_closed
rel_frontier_def subset_eq)
          have L ⊆ D
          by (metis DiffE ‹D ∈ 𝒩› ‹L ⊆ rel_frontier D› closed𝒩 closure_closed
rel_frontier_def subset_eq)
          let ?w = (SOME w. w ∈ rel_interior D)
          have w: ?w ∈ rel_interior D
            using ‹D ∈ 𝒩› in_rel_interior by blast
          have C ∩ rel_interior D = (D ∩ C) ∩ rel_interior D
            using rel_interior_subset by blast
          also have (D ∩ C) ∩ rel_interior D = {}
          proof (rule face_of_disjoint_rel_interior)
            show D ∩ C face_of D
              using 𝒩_def ‹C ∈ 𝒩› ‹D ∈ 𝒩› intface𝒩 by blast
            have D ∈ 𝒩 ∧ aff_dim D = int n
              using 𝒩_def ‹D ∈ 𝒩› by blast
            moreover have C ∈ 𝒩 ∧ aff_dim C = int n
              using 𝒩_def ‹C ∈ 𝒩› by blast
            ultimately show D ∩ C ≠ D
             by (metis Int_commute False face_of_aff_dim_lt inf.idem inf_le1
intface𝒩 not_le poly𝒩 polytope_imp_convex)
          qed
          finally have CD: C ∩ (rel_interior D) = {} .
          have zKC: (convex hull insert ?z K) ⊆ C
        by (metis ‹K ⊆ C› ‹convex C› in_mono insert_subsetI rel_interior_subset
subset_hull z)
           have disjnt (convex hull insert (SOME z. z ∈ rel_interior C) K)
(rel_interior D)
            using zKC CD by (force simp: disjnt_def)
          then have eq: convex hull (insert ?z K) ∩ convex hull (insert ?w L) =
                        convex hull (insert ?z K) ∩ convex hull L
            by (rule Int_convex_hull_insert_rel_exterior [OF ‹convex D› ‹L ⊆
D› w])
          have ch_id: convex hull K = K convex hull L = L
            using * ‹K ∈ 𝒰› ‹L ∈ 𝒰› hull_same by auto
```

```
          have convex C
            by (simp add: ‹C ∈ 𝒩› convex𝒩)
          have convex hull (insert ?z K) ∩ L = L ∩ convex hull (insert ?z K)
            by blast
          also have ... = convex hull K ∩ L
          proof (subst Int_convex_hull_insert_rel_exterior [OF ‹convex C› ‹K
⊆ C› z])
            have (C ∩ D) ∩ rel_interior C = {}
            proof (rule face_of_disjoint_rel_interior)
              show C ∩ D face_of C
                using 𝒩_def ‹C ∈ 𝒩› ‹D ∈ 𝒩› intface𝓜 by blast
              have D ∈ 𝓜 aff_dim D = int n
                using 𝒩_def ‹D ∈ 𝒩› by fastforce+
              moreover have C ∈ 𝓜 aff_dim C = int n
                using 𝒩_def ‹C ∈ 𝒩› by fastforce+
              ultimately have aff_dim D + - 1 * aff_dim C ≤ 0
                by fastforce
              then have ¬ C face_of D
                using False ‹convex D› face_of_aff_dim_lt by fastforce
              show C ∩ D ≠ C
                  by (metis inf_commute ‹C ∈ 𝓜› ‹D ∈ 𝓜› ‹¬ C face_of D›
intface𝓜)
            qed
            then have D ∩ rel_interior C = {}
          by (metis inf.absorb_iff2 inf_assoc inf_sup_aci(1) rel_interior_subset)
            then show disjnt L (rel_interior C)
              by (meson ‹L ⊆ D› disjnt_def disjnt_subset1)
          next
            show L ∩ convex hull K = convex hull K ∩ L
              by force
          qed
          finally have chKL: convex hull (insert ?z K) ∩ L = convex hull K ∩
L .
          have convex hull insert ?z K ∩ convex hull L face_of K
            by (simp add: ‹K ∈ 𝒰› ‹L ∈ 𝒰› ch_id chKL face𝒰)
          also have ... face_of convex hull insert ?z K
          proof -
            obtain I where I: ¬ affine_dependent I K = convex hull I
              using * [OF ‹K ∈ 𝒰›] by auto
            then have ⋀a. a ∉ rel_interior C ∨ a ∉ affine hull I
              using ahK_C_disjoint ‹C ∈ 𝒩› ‹K ∈ 𝒰› ‹K ⊆ rel_frontier C›
affine_hull_convex_hull by blast
            then show ?thesis
          by (metis I ‹convex K› aff_independent_finite face_of_convex_hull_insert_eq
face_of_refl hull_insert z)
          qed
          finally have 1: convex hull insert ?z K ∩ convex hull L face_of convex
hull insert ?z K .
          have convex hull insert ?z K ∩ convex hull L face_of L
```

by (*metis* ‹$K \in \mathcal{U}$› ‹$L \in \mathcal{U}$› *chKL ch_id faceI*$\mathcal{U}$ *inf_commute*)
also have ... *face_of convex hull insert ?w L*
proof −
obtain *I* **where** *I*: ¬ *affine_dependent I L* = *convex hull I*
using ∗ [*OF* ‹$L \in \mathcal{U}$›] **by** *auto*
then have $\bigwedge a.\ a \notin$ *rel_interior D* ∨ $a \notin$ *affine hull I*
using ‹$D \in \mathcal{N}$› ‹$L \in \mathcal{U}$› ‹$L \subseteq$ *rel_frontier D*› *affine_hull_convex_hull ahK_C_disjoint* **by** *blast*
then show *?thesis*
by (*metis I* ‹*convex L*› *aff_independent_finite face_of_convex_hull_insert face_of_refl hull_insert w*)
qed
finally have *2*: *convex hull insert ?z K* ∩ *convex hull L face_of convex hull insert ?w L* **.**
show *?thesis*
by (*simp add*: *X Y eq 1 2*)
qed
qed
qed
qed
show ∃ *F* ⊆ $\mathcal{U}$ ∪ *?*$\mathcal{T}$. *C* = ⋃ *F* **if** *C* ∈ $\mathcal{M}$ **for** *C*
proof (*cases C* ∈ $\mathcal{S}$)
case *True*
then show *?thesis*
by (*meson UnCI fin*$\mathcal{U}$ *subsetD subsetI*)
next
case *False*
then have *C* ∈ $\mathcal{N}$
by (*simp add*: $\mathcal{N}$*_def* $\mathcal{S}$*_def aff*$\mathcal{M}$ *less_le that*)
let *?z* = *SOME z*. *z* ∈ *rel_interior C*
have *z*: *?z* ∈ *rel_interior C*
using ‹*C* ∈ $\mathcal{N}$› *in_rel_interior* **by** *blast*
let *?F* = ⋃ *K* ∈ $\mathcal{U}$ ∩ *Pow* (*rel_frontier C*). {*convex hull* (*insert ?z K*)}
have *?F* ⊆ *?*$\mathcal{T}$
using ‹*C* ∈ $\mathcal{N}$› **by** *blast*
moreover have *C* ⊆ ⋃ *?F*
proof
fix *x*
assume *x* ∈ *C*
have *convex C*
using ‹*C* ∈ $\mathcal{N}$› *convex*$\mathcal{N}$ **by** *blast*
have *bounded C*
using ‹*C* ∈ $\mathcal{N}$› **by** (*simp add*: *poly*$\mathcal{M}$ *polytope_imp_bounded that*)
have *polytope C*
using ‹*C* ∈ $\mathcal{N}$› *poly*$\mathcal{N}$ **by** *auto*
have ¬ (*?z* = *x* ∧ *C* = {*?z*})
using ‹*C* ∈ $\mathcal{N}$› *aff_dim_sing* [*of ?z*] ‹¬ *n* ≤ *1*› **by** (*force simp*: $\mathcal{N}$*_def*)
then obtain *y* **where** *y*: *y* ∈ *rel_frontier C* **and** *xzy*: *x* ∈ *closed_segment ?z y*

and *sub*: *open_segment ?z y* ⊆ *rel_interior C*
by (*blast intro*: *segment_to_rel_frontier* [*OF* ‹*convex C*› ‹*bounded C*› *z* ‹*x* ∈ *C*›])
then obtain *F* **where** *y* ∈ *F F face_of C F* ≠ *C*
by (*auto simp*: *rel_frontier_of_polyhedron_alt* [*OF polytope_imp_polyhedron* [*OF* ‹*polytope C*›]])
then obtain *G* **where** *finite G G* ⊆ *U F* = ⋃*G*
by (*metis* (*mono_tags*, *lifting*) *S_def* ‹*C* ∈ *M*› ‹*convex C*› *affM faceM face_of_aff_dim_lt finU le_less_trans mem_Collect_eq not_less*)
then obtain *K* **where** *y* ∈ *K K* ∈ *G*
using ‹*y* ∈ *F*› **by** *blast*
moreover have *x*: *x* ∈ *convex hull* {*?z*,*y*}
using *segment_convex_hull xzy* **by** *auto*
moreover have *convex hull* {*?z*,*y*} ⊆ *convex hull insert ?z K*
by (*metis* (*full_types*) ‹*y* ∈ *K*› *hull_mono empty_subsetI insertCI insert_subset*)
moreover have *K* ∈ *U*
using ‹*K* ∈ *G*› ‹*G* ⊆ *U*› **by** *blast*
moreover have *K* ⊆ *rel_frontier C*
using ‹*F* = ⋃*G*› ‹*F* ≠ *C*› ‹*F face_of C*› ‹*K* ∈ *G*› *face_of_subset_rel_frontier* **by** *fastforce*
ultimately show *x* ∈ ⋃ *?F*
by *force*
qed
moreover
have *convex hull insert* (*SOME z*. *z* ∈ *rel_interior C*) *K* ⊆ *C*
if *K* ∈ *U K* ⊆ *rel_frontier C* **for** *K*
proof (*rule hull_minimal*)
show *insert* (*SOME z*. *z* ∈ *rel_interior C*) *K* ⊆ *C*
using *that* ‹*C* ∈ *N*› *in_rel_interior rel_interior_subset*
by (*force simp*: *closure_eq rel_frontier_def closedN*)
show *convex C*
by (*simp add*: ‹*C* ∈ *N*› *convexN*)
qed
then have ⋃ *?F* ⊆ *C*
by *auto*
ultimately show *?thesis*
by *blast*
qed
have (∃ *C*. *C* ∈ *M* ∧ *L* ⊆ *C*) ∧ *aff_dim L* ≤ *int n* **if** *L* ∈ *U* ∪ *?T* **for** *L*
using *that*
proof
assume *L* ∈ *U*
then show *?thesis*
using *CU S_def* ∗ **by** *fastforce*
next
assume *L* ∈ *?T*
then obtain *C K* **where** *C* ∈ *N*
and *L*: *L* = *convex hull insert* (*SOME z*. *z* ∈ *rel_interior C*) *K*

```
        and K: K ∈ 𝒰 K ⊆ rel_frontier C
        by auto
      then have convex hull C = C
        by (meson convex𝒩 convex_hull_eq)
      then have convex C
        by (metis (no_types) convex_convex_hull)
      have rel_frontier C ⊆ C
       by (metis DiffE closed𝒩 ‹C ∈ 𝒩› closure_closed rel_frontier_def subsetI)
      have K ⊆ C
        using K ‹rel_frontier C ⊆ C› by blast
      have C ∈ ℳ
        using 𝒩_def ‹C ∈ 𝒩› by auto
      moreover have L ⊆ C
        using K L ‹C ∈ 𝒩›
          by (metis ‹K ⊆ C› ‹convex hull C = C› contra_subsetD hull_mono
in_rel_interior insert_subset rel_interior_subset)
      ultimately show ?thesis
         using ‹rel_frontier C ⊆ C› ‹L ⊆ C› aff ℳ aff_dim_subset ‹C ∈ ℳ›
dual_order.trans by blast
    qed
    then show ∃ C. C ∈ ℳ ∧ L ⊆ C aff_dim L ≤ int n if L ∈ 𝒰 ∪ ?𝒯 for L
      using that by auto
  qed
  then show ?thesis
    apply (rule ex_forward, safe)
     apply (meson Union_iff subsetCE, fastforce)
    by (meson infinite_super simplicial_complex_def)
 qed
qed


lemma simplicial_subdivision_of_cell_complex_lowdim:
 assumes finite ℳ
    and poly: ⋀C. C ∈ ℳ ⟹ polytope C
    and face: ⋀C1 C2. ⟦C1 ∈ ℳ; C2 ∈ ℳ⟧ ⟹ C1 ∩ C2 face_of C1
    and aff: ⋀C. C ∈ ℳ ⟹ aff_dim C ≤ d
 obtains 𝒯 where simplicial_complex 𝒯 ⋀K. K ∈ 𝒯 ⟹ aff_dim K ≤ d
                 ⋃𝒯 = ⋃ℳ
                 ⋀C. C ∈ ℳ ⟹ ∃ F. finite F ∧ F ⊆ 𝒯 ∧ C = ⋃F
                 ⋀K. K ∈ 𝒯 ⟹ ∃ C. C ∈ ℳ ∧ K ⊆ C
proof (cases d ≥ 0)
 case True
 then obtain n where n: d = of_nat n
   using zero_le_imp_eq_int by blast
 have ∃ 𝒯. simplicial_complex 𝒯 ∧
         (∀ K∈𝒯. aff_dim K ≤ int n) ∧
         ⋃𝒯 = ⋃(⋃ C∈ℳ. {F. F face_of C}) ∧
         (∀ C∈⋃ C∈ℳ. {F. F face_of C}.
            ∃ F. finite F ∧ F ⊆ 𝒯 ∧ C = ⋃F) ∧
```

```
        (∀K∈𝒯. ∃C. C ∈ (⋃C∈ℳ. {F. F face_of C}) ∧ K ⊆ C)
  proof (rule simplicial_subdivision_aux)
    show finite (⋃C∈ℳ. {F. F face_of C})
      using ‹finite ℳ› poly polyhedron_eq_finite_faces polytope_imp_polyhedron
by fastforce
    show polytope F if F ∈ (⋃C∈ℳ. {F. F face_of C}) for F
      using poly that face_of_polytope_polytope by blast
    show aff_dim F ≤ int n if F ∈ (⋃C∈ℳ. {F. F face_of C}) for F
      using that
      by clarify (metis n aff_dim_subset aff face_of_imp_subset order_trans)
    show F ∈ (⋃C∈ℳ. {F. F face_of C})
      if G ∈ (⋃C∈ℳ. {F. F face_of C}) and F face_of G for F G
      using that face_of_trans by blast
  next
    fix F1 F2
   assume F1 ∈ (⋃C∈ℳ. {F. F face_of C}) and F2 ∈ (⋃C∈ℳ. {F. F face_of
C})
     then obtain C1 C2 where C1 ∈ ℳ C2 ∈ ℳ and F: F1 face_of C1 F2
face_of C2
      by auto
    show F1 ∩ F2 face_of F1
      using face_of_Int_subface [OF _ _ F]
      by (metis ‹C1 ∈ ℳ› ‹C2 ∈ ℳ› face inf_commute)
  qed
  moreover
  have ⋃(⋃C∈ℳ. {F. F face_of C}) = ⋃ℳ
    using face_of_imp_subset face by blast
  ultimately show ?thesis
    using face_of_imp_subset n
    by (fastforce intro!: that simp add: poly face_of_refl polytope_imp_convex)
next
  case False
  then have m1: ⋀C. C ∈ ℳ ⟹ aff_dim C = -1
   by (metis aff aff_dim_empty_eq aff_dim_negative_iff dual_order.trans not_less)
  then have faceℳ: ⋀F S. ⟦S ∈ ℳ; F face_of S⟧ ⟹ F ∈ ℳ
    by (metis aff_dim_empty face_of_empty)
  show ?thesis
  proof
    have ⋀S. S ∈ ℳ ⟹ ∃n. n simplex S
      by (metis (no_types) m1 aff_dim_empty simplex_minus_1)
    then show simplicial_complex ℳ
      by (auto simp: simplicial_complex_def ‹finite ℳ› face intro: faceℳ)
    show aff_dim K ≤ d if K ∈ ℳ for K
      by (simp add: that aff)
    show ∃F. finite F ∧ F ⊆ ℳ ∧ C = ⋃F if C ∈ ℳ for C
      using ‹C ∈ ℳ› equals0I by auto
    show ∃C. C ∈ ℳ ∧ K ⊆ C if K ∈ ℳ for K
      using ‹K ∈ ℳ› by blast
  qed auto
```

qed

proposition *simplicial_subdivision_of_cell_complex*:
assumes *finite* $\mathcal{M}$
and *poly*: $\bigwedge C.\ C \in \mathcal{M} \Longrightarrow$ *polytope* C
and *face*: $\bigwedge C1\ C2.\ [\![C1 \in \mathcal{M};\ C2 \in \mathcal{M}]\!] \Longrightarrow C1 \cap C2$ *face_of* $C1$
obtains $\mathcal{T}$ **where** *simplicial_complex* $\mathcal{T}$
$\bigcup \mathcal{T} = \bigcup \mathcal{M}$
$\bigwedge C.\ C \in \mathcal{M} \Longrightarrow \exists F.$ *finite* $F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F$
$\bigwedge K.\ K \in \mathcal{T} \Longrightarrow \exists C.\ C \in \mathcal{M} \wedge K \subseteq C$
by (*blast intro*: *simplicial_subdivision_of_cell_complex_lowdim* [*OF assms aff_dim_le_DIM*])

corollary *fine_simplicial_subdivision_of_cell_complex*:
assumes $0 < e$ *finite* $\mathcal{M}$
and *poly*: $\bigwedge C.\ C \in \mathcal{M} \Longrightarrow$ *polytope* C
and *face*: $\bigwedge C1\ C2.\ [\![C1 \in \mathcal{M};\ C2 \in \mathcal{M}]\!] \Longrightarrow C1 \cap C2$ *face_of* $C1$
obtains $\mathcal{T}$ **where** *simplicial_complex* $\mathcal{T}$
$\bigwedge K.\ K \in \mathcal{T} \Longrightarrow$ *diameter* $K < e$
$\bigcup \mathcal{T} = \bigcup \mathcal{M}$
$\bigwedge C.\ C \in \mathcal{M} \Longrightarrow \exists F.$ *finite* $F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F$
$\bigwedge K.\ K \in \mathcal{T} \Longrightarrow \exists C.\ C \in \mathcal{M} \wedge K \subseteq C$
proof −
obtain $\mathcal{N}$ **where** $\mathcal{N}$: *finite* $\mathcal{N}$ $\bigcup \mathcal{N} = \bigcup \mathcal{M}$
and *diapoly*: $\bigwedge X.\ X \in \mathcal{N} \Longrightarrow$ *diameter* $X < e$ $\bigwedge X.\ X \in \mathcal{N} \Longrightarrow$ *polytope* X
and $\bigwedge X\ Y.\ [\![X \in \mathcal{N};\ Y \in \mathcal{N}]\!] \Longrightarrow X \cap Y$ *face_of* X
and $\mathcal{N}$*covers*: $\bigwedge C\ x.\ C \in \mathcal{M} \wedge x \in C \Longrightarrow \exists D.\ D \in \mathcal{N} \wedge x \in D \wedge D \subseteq C$
and $\mathcal{N}$*covered*: $\bigwedge C.\ C \in \mathcal{N} \Longrightarrow \exists D.\ D \in \mathcal{M} \wedge C \subseteq D$
by (*blast intro*: *cell_complex_subdivision_exists* [*OF* ‹$0 < e$› ‹*finite* $\mathcal{M}$› *poly aff_dim_le_DIM face*])
then obtain $\mathcal{T}$ **where** $\mathcal{T}$: *simplicial_complex* $\mathcal{T}$ $\bigcup \mathcal{T} = \bigcup \mathcal{N}$
and $\mathcal{T}$*covers*: $\bigwedge C.\ C \in \mathcal{N} \Longrightarrow \exists F.$ *finite* $F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F$
and $\mathcal{T}$*covered*: $\bigwedge K.\ K \in \mathcal{T} \Longrightarrow \exists C.\ C \in \mathcal{N} \wedge K \subseteq C$
using *simplicial_subdivision_of_cell_complex* [*OF* ‹*finite* $\mathcal{N}$›] **by** *metis*
show *?thesis*
proof
show *simplicial_complex* $\mathcal{T}$
by (*rule* $\mathcal{T}$)
show *diameter* $K < e$ **if** $K \in \mathcal{T}$ **for** K
by (*metis le_less_trans diapoly* $\mathcal{T}$*covered diameter_subset polytope_imp_bounded that*)
show $\bigcup \mathcal{T} = \bigcup \mathcal{M}$
by (*simp add*: $\mathcal{N}(2)$ ‹$\bigcup \mathcal{T} = \bigcup \mathcal{N}$›)
show $\exists F.$ *finite* $F \wedge F \subseteq \mathcal{T} \wedge C = \bigcup F$ **if** $C \in \mathcal{M}$ **for** C
proof −
{ fix x
assume $x \in C$
then obtain D **where** $D \in \mathcal{T}$ $x \in D$ $D \subseteq C$

```
      using ℕ covers ‹C ∈ ℳ› 𝒯 covers by force
    then have ∃X∈𝒯 ∩ Pow C. x ∈ X
      using ‹D ∈ 𝒯› ‹D ⊆ C› ‹x ∈ D› by blast
   }
   moreover
   have finite (𝒯 ∩ Pow C)
    using ‹simplicial_complex 𝒯› simplicial_complex_def by auto
   ultimately show ?thesis
    by (rule_tac x=(𝒯 ∩ Pow C) in exI) auto
  qed
  show ∃C. C ∈ ℳ ∧ K ⊆ C if K ∈ 𝒯 for K
   by (meson ℕ covered 𝒯 covered order_trans that)
 qed
qed
```

6.34.20 Some results on cell division with full-dimensional cells only

```
lemma convex_Union_fulldim_cells:
 assumes finite 𝒮 and clo: ⋀C. C ∈ 𝒮 ⟹ closed C and con: ⋀C. C ∈ 𝒮 ⟹
convex C
     and eq: ⋃𝒮 = Uand  convex U
 shows ⋃{C ∈ 𝒮. aff_dim C = aff_dim U} = U  (is ?lhs = U)
proof −
 have closed U
  using ‹finite 𝒮› clo eq by blast
 have ?lhs ⊆ U
  using eq by blast
 moreover have U ⊆ ?lhs
 proof (cases ∀C ∈ 𝒮. aff_dim C = aff_dim U)
  case True
  then show ?thesis
   using eq by blast
 next
  case False
  have closed ?lhs
   by (simp add: ‹finite 𝒮› clo closed_Union)
  moreover have U ⊆ closure ?lhs
  proof −
   have U ⊆ closure(⋂{U − C |C. C ∈ 𝒮 ∧ aff_dim C < aff_dim U})
   proof (rule Baire [OF ‹closed U›])
    show countable {U − C |C. C ∈ 𝒮 ∧ aff_dim C < aff_dim U}
     using ‹finite 𝒮› uncountable_infinite by fastforce
    have ⋀C. C ∈ 𝒮 ⟹ openin (top_of_set U) (U−C)
        by (metis Sup_upper clo closed_limpt closedin_limpt eq openin_diff
openin_subtopology_self)
    then show openin (top_of_set U) T ∧ U ⊆ closure T
     if T ∈ {U − C |C. C ∈ 𝒮 ∧ aff_dim C < aff_dim U} for T
      using that dense_complement_convex_closed ‹closed U› ‹convex U› by
```

```
auto
      qed
      also have ... ⊆ closure ?lhs
      proof −
        obtain C where C ∈ 𝒮 aff_dim C < aff_dim U
          by (metis False Sup_upper aff_dim_subset eq eq_iff not_le)
        then have ∃X. X ∈ 𝒮 ∧ aff_dim X = aff_dim U ∧ x ∈ X
          if ⋀V. (∃C. V = U − C ∧ C ∈ 𝒮 ∧ aff_dim C < aff_dim U) ⟹ x ∈
V for x
            by (metis Diff_iff Sup_upper UnionE aff_dim_subset eq order_less_le
that)
        then show ?thesis
          by (auto intro!: closure_mono)
      qed
      finally show ?thesis .
    qed
    ultimately show ?thesis
      using closure_subset_eq by blast
  qed
  ultimately show ?thesis by blast
qed

proposition fine_triangular_subdivision_of_cell_complex:
  assumes 0 < e finite ℳ
      and poly: ⋀C. C ∈ ℳ ⟹ polytope C
      and aff: ⋀C. C ∈ ℳ ⟹ aff_dim C = d
      and face: ⋀C1 C2. ⟦C1 ∈ ℳ; C2 ∈ ℳ⟧ ⟹ C1 ∩ C2 face_of C1
  obtains 𝒯 where triangulation 𝒯 ⋀k. k ∈ 𝒯 ⟹ diameter k < e
                  ⋀k. k ∈ 𝒯 ⟹ aff_dim k = d ⋃𝒯 = ⋃ℳ
                  ⋀C. C ∈ ℳ ⟹ ∃f. finite f ∧ f ⊆ 𝒯 ∧ C = ⋃f
                  ⋀k. k ∈ 𝒯 ⟹ ∃C. C ∈ ℳ ∧ k ⊆ C
proof −
  obtain 𝒯 where simplicial_complex 𝒯
            and diaT: ⋀K. K ∈ 𝒯 ⟹ diameter K < e
            and ⋃𝒯 = ⋃ℳ
            and inℳ: ⋀C. C ∈ ℳ ⟹ ∃F. finite F ∧ F ⊆ 𝒯 ∧ C = ⋃F
            and in𝒯: ⋀K. K ∈ 𝒯 ⟹ ∃C. C ∈ ℳ ∧ K ⊆ C
    by (blast intro: fine_simplicial_subdivision_of_cell_complex [OF ‹e > 0›
‹finite ℳ› poly face])
  let ?𝒯 = {K ∈ 𝒯. aff_dim K = d}
  show thesis
  proof
    show triangulation ?𝒯
      using ‹simplicial_complex 𝒯› by (auto simp: triangulation_def simpli-
cial_complex_def)
    show diameter L < e if L ∈ {K ∈ 𝒯. aff_dim K = d} for L
      using that by (auto simp: diaT)
    show aff_dim L = d if L ∈ {K ∈ 𝒯. aff_dim K = d} for L
      using that by auto
```

```
    show ∃F. finite F ∧ F ⊆ {K ∈ 𝒯. aff_dim K = d} ∧ C = ⋃F if C ∈ ℳ
for C
    proof −
      obtain F where finite F F ⊆ 𝒯 C = ⋃F
        using inℳ [OF ‹C ∈ ℳ›] by auto
      show ?thesis
      proof (intro exI conjI)
        show finite {K ∈ F. aff_dim K = d}
          by (simp add: ‹finite F›)
        show {K ∈ F. aff_dim K = d} ⊆ {K ∈ 𝒯. aff_dim K = d}
          using ‹F ⊆ 𝒯› by blast
        have d = aff_dim C
          by (simp add: aff that)
        moreover have ⋀K. K ∈ F ⟹ closed K ∧ convex K
          using ‹simplicial_complex 𝒯› ‹F ⊆ 𝒯›
       unfolding simplicial_complex_def by (metis subsetCE ‹F ⊆ 𝒯› closed_simplex
convex_simplex)
        moreover have convex (⋃F)
          using ‹C = ⋃F› poly polytope_imp_convex that by blast
        ultimately show C = ⋃{K ∈ F. aff_dim K = d}
          by (simp add: convex_Union_fulldim_cells ‹C = ⋃F› ‹finite F›)
      qed
    qed
    then show ⋃{K ∈ 𝒯. aff_dim K = d} = ⋃ℳ
      by auto (meson in𝒯 subsetCE)
    show ∃C. C ∈ ℳ ∧ L ⊆ C
      if L ∈ {K ∈ 𝒯. aff_dim K = d} for L
      using that by (auto simp: in𝒯)
  qed
qed

end
```

6.35 Absolute Retracts, Absolute Neighbourhood Retracts and Euclidean Neighbourhood Retracts

```
theory Retracts
imports
  Brouwer_Fixpoint
  Continuous_Extension
begin
```

Absolute retracts (AR), absolute neighbourhood retracts (ANR) and also Euclidean neighbourhood retracts (ENR). We define AR and ANR by specializing the standard definitions for a set to embedding in spaces of higher dimension.

John Harrison writes: "This turns out to be sufficient (since any set in $\mathbb{R}^n$ can be embedded as a closed subset of a convex subset of $\mathbb{R}^{n+1}$) to derive the usual definitions, but we need to split them into two implications because of the lack of type quantifiers. Then ENR turns out to be equivalent to ANR plus local compactness."

definition *AR* :: *'a::topological_space set* ⇒ *bool* **where**
AR S ≡ ∀ *U*. ∀ *S'*::(*'a* ∗ *real*) *set*.
S homeomorphic S' ∧ *closedin* (*top_of_set U*) *S'* ⟶ *S' retract_of U*

definition *ANR* :: *'a::topological_space set* ⇒ *bool* **where**
ANR S ≡ ∀ *U*. ∀ *S'*::(*'a* ∗ *real*) *set*.
S homeomorphic S' ∧ *closedin* (*top_of_set U*) *S'*
⟶ (∃ *T*. *openin* (*top_of_set U*) *T* ∧ *S' retract_of T*)

definition *ENR* :: *'a::topological_space set* ⇒ *bool* **where**
ENR S ≡ ∃ *U*. *open U* ∧ *S retract_of U*

First, show that we do indeed get the "usual" properties of ARs and ANRs.

lemma *AR_imp_absolute_extensor*:
fixes *f* :: *'a::euclidean_space* ⇒ *'b::euclidean_space*
assumes *AR S* **and** *contf*: *continuous_on T f* **and** *f* ‘ *T* ⊆ *S*
and *cloUT*: *closedin* (*top_of_set U*) *T*
obtains *g* **where** *continuous_on U g g* ‘ *U* ⊆ *S* ⋀*x*. *x* ∈ *T* ⟹ *g x* = *f x*
proof −
have *aff_dim S* < *int* (*DIM*(*'b* × *real*))
using *aff_dim_le_DIM* [*of S*] **by** *simp*
then obtain *C* **and** *S'* :: (*'b* ∗ *real*) *set*
where *C*: *convex C C* ≠ {}
and *cloCS*: *closedin* (*top_of_set C*) *S'*
and *hom*: *S homeomorphic S'*
by (*metis that homeomorphic_closedin_convex*)
then have *S' retract_of C*
using ‹*AR S*› **by** (*simp add*: *AR_def*)
then obtain *r* **where** *S'* ⊆ *C* **and** *contr*: *continuous_on C r*
and *r* ‘ *C* ⊆ *S'* **and** *rid*: ⋀*x*. *x*∈*S'* ⟹ *r x* = *x*
by (*auto simp*: *retraction_def retract_of_def*)
obtain *g h* **where** *homeomorphism S S' g h*
using *hom* **by** (*force simp*: *homeomorphic_def*)
then have *continuous_on* (*f* ‘ *T*) *g*
by (*meson* ‹*f* ‘ *T* ⊆ *S*› *continuous_on_subset homeomorphism_def*)
then have *contgf*: *continuous_on T* (*g* ∘ *f*)
by (*metis continuous_on_compose contf*)
have *gfTC*: (*g* ∘ *f*) ‘ *T* ⊆ *C*
proof −
have *g* ‘ *S* = *S'*
by (*metis* (*no_types*) ‹*homeomorphism S S' g h*› *homeomorphism_def*)
with ‹*S'* ⊆ *C*› ‹*f* ‘ *T* ⊆ *S*› **show** *?thesis* **by** *force*
qed

```
  obtain f' where f': continuous_on U f'  f' ` U ⊆ C
                    ⋀x. x ∈ T ⟹ f' x = (g ∘ f) x
    by (metis Dugundji [OF C cloUT contgf gfTC])
  show ?thesis
  proof (rule_tac g = h ∘ r ∘ f' in that)
    show continuous_on U (h ∘ r ∘ f')
    proof (intro continuous_on_compose f')
      show continuous_on (f' ` U) r
        using continuous_on_subset contr f' by blast
      show continuous_on (r ` f' ` U) h
        using ‹homeomorphism S S' g h› ‹f' ` U ⊆ C›
        unfolding homeomorphism_def
        by (metis ‹r ` C ⊆ S'› continuous_on_subset image_mono)
    qed
    show (h ∘ r ∘ f') ` U ⊆ S
      using ‹homeomorphism S S' g h› ‹r ` C ⊆ S'› ‹f' ` U ⊆ C›
      by (fastforce simp: homeomorphism_def)
    show ⋀x. x ∈ T ⟹ (h ∘ r ∘ f') x = f x
      using ‹homeomorphism S S' g h› ‹f ` T ⊆ S› f'
      by (auto simp: rid homeomorphism_def)
  qed
qed

lemma AR_imp_absolute_retract:
  fixes S :: 'a::euclidean_space set and S' :: 'b::euclidean_space set
  assumes AR S S homeomorphic S'
      and clo: closedin (top_of_set U) S'
    shows S' retract_of U
proof −
  obtain g h where hom: homeomorphism S S' g h
    using assms by (force simp: homeomorphic_def)
  obtain h: continuous_on S' h  h ` S' ⊆ S
    using hom homeomorphism_def by blast
  obtain h' where h': continuous_on U h' h' ` U ⊆ S
            and h'h: ⋀x. x ∈ S' ⟹ h' x = h x
    by (blast intro: AR_imp_absolute_extensor [OF ‹AR S› h clo])
  have [simp]: S' ⊆ U using clo closedin_limpt by blast
  show ?thesis
  proof (simp add: retraction_def retract_of_def, intro exI conjI)
    show continuous_on U (g ∘ h')
      by (meson continuous_on_compose continuous_on_subset h' hom homeomorphism_cont1)
    show (g ∘ h') ∈ U → S'
      using h'  by clarsimp (metis hom subsetD homeomorphism_def imageI)
    show ∀x∈S'. (g ∘ h') x = x
      by clarsimp (metis h'h hom homeomorphism_def)
  qed
qed
```

lemma *AR_imp_absolute_retract_UNIV*:
fixes *S* :: *'a*::*euclidean_space set* **and** *S'* :: *'b*::*euclidean_space set*
assumes *AR S S homeomorphic S' closed S'*
shows *S' retract_of UNIV*
using *AR_imp_absolute_retract assms* **by** *fastforce*

lemma *absolute_extensor_imp_AR*:
fixes *S* :: *'a*::*euclidean_space set*
assumes $\bigwedge f$ :: *'a* ∗ *real* ⇒ *'a*.
$\bigwedge U\ T.\ [\![$*continuous_on T f*; *f* ‘ *T* ⊆ *S*;
closedin (*top_of_set U*) *T*$]\!]$
⟹ ∃ *g*. *continuous_on U g* ∧ *g* ‘ *U* ⊆ *S* ∧ (∀ *x* ∈ *T*. *g x* = *f x*)
shows *AR S*
proof (*clarsimp simp*: *AR_def*)
fix *U* **and** *T* :: (*'a* ∗ *real*) *set*
assume *S homeomorphic T* **and** *clo*: *closedin* (*top_of_set U*) *T*
then obtain *g h* **where** *hom*: *homeomorphism S T g h*
by (*force simp*: *homeomorphic_def*)
obtain *h*: *continuous_on T h* *h* ‘ *T* ⊆ *S*
using *hom homeomorphism_def* **by** *blast*
obtain *h'* **where** *h'*: *continuous_on U h' h'* ‘ *U* ⊆ *S*
and *h'h*: ∀ *x*∈*T*. *h' x* = *h x*
using *assms* [*OF h clo*] **by** *blast*
have [*simp*]: *T* ⊆ *U*
using *clo closedin_imp_subset* **by** *auto*
show *T retract_of U*
proof (*simp add*: *retraction_def retract_of_def*, *intro exI conjI*)
show *continuous_on U* (*g* ∘ *h'*)
by (*meson continuous_on_compose continuous_on_subset h' hom homeomorphism_cont1*)
show (*g* ∘ *h'*) ∈ *U* → *T*
using *h'* **by** *clarsimp* (*metis hom subsetD homeomorphism_def imageI*)
show ∀ *x*∈*T*. (*g* ∘ *h'*) *x* = *x*
by *clarsimp* (*metis h'h hom homeomorphism_def*)
qed
qed

lemma *AR_eq_absolute_extensor*:
fixes *S* :: *'a*::*euclidean_space set*
shows *AR S* ⟷
(∀ *f* :: *'a* ∗ *real* ⇒ *'a*.
∀ *U T*. *continuous_on T f* ⟶ *f* ‘ *T* ⊆ *S* ⟶
closedin (*top_of_set U*) *T* ⟶
(∃ *g*. *continuous_on U g* ∧ *g* ‘ *U* ⊆ *S* ∧ (∀ *x* ∈ *T*. *g x* = *f x*)))
by (*metis* (*mono_tags*, *opaque_lifting*) *AR_imp_absolute_extensor absolute_extensor_imp_AR*)

lemma *AR_imp_retract*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *AR S* ∧ *closedin* (*top_of_set U*) *S*

shows *S retract_of U*
using *AR_imp_absolute_retract assms homeomorphic_refl* **by** *blast*

lemma *AR_homeomorphic_AR*:
fixes *S* :: *'a*::*euclidean_space set* **and** *T* :: *'b*::*euclidean_space set*
assumes *AR T S homeomorphic T*
shows *AR S*
unfolding *AR_def*
by (*metis assms AR_imp_absolute_retract homeomorphic_trans* [*of* _ *S*] *homeomorphic_sym*)

lemma *homeomorphic_AR_iff_AR*:
fixes *S* :: *'a*::*euclidean_space set* **and** *T* :: *'b*::*euclidean_space set*
shows *S homeomorphic T* $\Longrightarrow$ *AR S* $\longleftrightarrow$ *AR T*
by (*metis AR_homeomorphic_AR homeomorphic_sym*)

lemma *ANR_imp_absolute_neighbourhood_extensor*:
fixes *f* :: *'a*::*euclidean_space* $\Rightarrow$ *'b*::*euclidean_space*
assumes *ANR S* **and** *contf*: *continuous_on T f* **and** *f* $\in$ *T* $\to$ *S*
and *cloUT*: *closedin* (*top_of_set U*) *T*
obtains *V g* **where** *T* $\subseteq$ *V openin* (*top_of_set U*) *V*
continuous_on V g
g $\in$ *V* $\to$ *S* $\bigwedge x.$ *x* $\in$ *T* $\Longrightarrow$ *g x* = *f x*
proof −
have *aff_dim S* < *int* (*DIM*(*'b* × *real*))
using *aff_dim_le_DIM* [*of S*] **by** *simp*
then obtain *C* **and** *S'* :: (*'b* ∗ *real*) *set*
where *C*: *convex C C* ≠ {}
and *cloCS*: *closedin* (*top_of_set C*) *S'*
and *hom*: *S homeomorphic S'*
by (*metis that homeomorphic_closedin_convex*)
then obtain *D* **where** *opD*: *openin* (*top_of_set C*) *D* **and** *S' retract_of D*
using ‹*ANR S*› **by** (*auto simp*: *ANR_def*)
then obtain *r* **where** *S'* ⊆ *D* **and** *contr*: *continuous_on D r*
and *r* ‘ *D* ⊆ *S'* **and** *rid*: ⋀*x*. *x* ∈ *S'* ⟹ *r x* = *x*
by (*auto simp*: *retraction_def retract_of_def*)
obtain *g h* **where** *homgh*: *homeomorphism S S' g h*
using *hom* **by** (*force simp*: *homeomorphic_def*)
have *continuous_on* (*f* ‘ *T*) *g*
by (*metis PiE assms*(*3*) *continuous_on_subset homeomorphism_cont1 homgh image_subset_iff*)
then have *contgf*: *continuous_on T* (*g* ∘ *f*)
by (*intro continuous_on_compose contf*)
have *gfTC*: (*g* ∘ *f*) ‘ *T* ⊆ *C*
proof −
have *g* ‘ *S* = *S'*
by (*metis* (*no_types*) *homeomorphism_def homgh*)
then show *?thesis*

by (*metis PiE assms*(3) *cloCS closedin_def image_comp image_mono image_subset_iff order.trans topspace_euclidean_subtopology*)
qed
obtain f' **where** *contf'*: *continuous_on* U f'
and f' ‘ $U \subseteq C$
and *eq*: $\bigwedge x.\ x \in T \Longrightarrow f'\ x = (g \circ f)\ x$
by (*metis Dugundji* [*OF C cloUT contgf gfTC*])
show *?thesis*
proof (*rule_tac* $V = U \cap f' -‘ D$ **and** $g = h \circ r \circ f'$ **in** *that*)
show $T \subseteq U \cap f' -‘ D$
using *cloUT closedin_imp_subset* ‹$S' \subseteq D$› ‹$f \in T \to S$› *eq homeomorphism_image1 homgh*
by *fastforce*
show *ope*: *openin* (*top_of_set U*) ($U \cap f' -‘ D$)
by (*meson* ‹f' ‘ $U \subseteq C$› *contf' continuous_openin_preimage image_subset_iff_funcset opD*)
have *conth*: *continuous_on* (r ‘ f' ‘ ($U \cap f' -‘ D$)) h
proof (*rule continuous_on_subset* [*of S'*])
show *continuous_on* S' h
using *homeomorphism_def homgh* **by** *blast*
qed (*use* ‹r ‘ $D \subseteq S'$› **in** *blast*)
show *continuous_on* ($U \cap f' -‘ D$) ($h \circ r \circ f'$)
by (*blast intro*: *continuous_on_compose conth continuous_on_subset* [*OF contr*] *continuous_on_subset* [*OF contf'*])
show ($h \circ r \circ f'$) $\in$ ($U \cap f' -‘ D$) $\to$ S
using ‹*homeomorphism S S' g h*› ‹f' ‘ $U \subseteq C$› ‹r ‘ $D \subseteq S'$›
by (*auto simp*: *homeomorphism_def*)
show $\bigwedge x.\ x \in T \Longrightarrow (h \circ r \circ f')\ x = f\ x$
using ‹*homeomorphism S S' g h*› ‹$f \in T \to S$› *eq*
by (*metis PiE comp_apply homeomorphism_def image_iff rid*)
qed
qed

corollary *ANR_imp_absolute_neighbourhood_retract*:
fixes S :: *'a*::*euclidean_space set* **and** S' :: *'b*::*euclidean_space set*
assumes *ANR S S homeomorphic S'*
and *clo*: *closedin* (*top_of_set U*) S'
obtains V **where** *openin* (*top_of_set U*) V S' *retract_of* V
proof −
obtain g h **where** *hom*: *homeomorphism S S' g h*
using *assms* **by** (*force simp*: *homeomorphic_def*)
obtain h: *continuous_on* S' h $h \in S' \to S$
using *hom homeomorphism_def* **by** *blast*
from *ANR_imp_absolute_neighbourhood_extensor* [*OF* ‹*ANR S*› h *clo*]
obtain V h' **where** $S' \subseteq V$ **and** *opUV*: *openin* (*top_of_set U*) V
and h': *continuous_on* V h' h' ‘ $V \subseteq S$
and *h'h*:$\bigwedge x.\ x \in S' \Longrightarrow h'\ x = h\ x$
by (*blast intro*: *ANR_imp_absolute_neighbourhood_extensor* [*OF* ‹*ANR S*› h

```
clo])
  have S' retract_of V
  proof (simp add: retraction_def retract_of_def, intro exI conjI ‹S' ⊆ V›)
    show continuous_on V (g ∘ h')
      by (meson continuous_on_compose continuous_on_subset h'(1) h'(2) hom
homeomorphism_cont1)
    show (g ∘ h') ∈ V → S'
      using h' by clarsimp (metis hom subsetD homeomorphism_def imageI)
    show ∀x∈S'. (g ∘ h') x = x
      by clarsimp (metis h'h hom homeomorphism_def)
  qed
  then show ?thesis
    by (rule that [OF opUV])
qed

corollary ANR_imp_absolute_neighbourhood_retract_UNIV:
  fixes S :: 'a::euclidean_space set and S' :: 'b::euclidean_space set
  assumes ANR S and hom: S homeomorphic S' and clo: closed S'
  obtains V where open V S' retract_of V
  using ANR_imp_absolute_neighbourhood_retract [OF ‹ANR S› hom]
by (metis clo closed_closedin open_openin subtopology_UNIV)

corollary neighbourhood_extension_into_ANR:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes contf: continuous_on S f and fim: f ∈ S → T and ANR T closed S
  obtains V g where S ⊆ V open V continuous_on V g
                    g ∈ V → T ⋀x. x ∈ S ⟹ g x = f x
  using ANR_imp_absolute_neighbourhood_extensor [OF ‹ANR T› contf fim]
  by (metis ‹closed S› closed_closedin open_openin subtopology_UNIV)

lemma absolute_neighbourhood_extensor_imp_ANR:
  fixes S :: 'a::euclidean_space set
  assumes ⋀f :: 'a * real ⇒ 'a.
          ⋀U T. ⟦continuous_on T f;  f ∈ T → S;
                 closedin (top_of_set U) T⟧
                ⟹ ∃V g. T ⊆ V ∧ openin (top_of_set U) V ∧
                     continuous_on V g ∧ g ∈ V → S ∧ (∀x ∈ T. g x = f x)
  shows ANR S
proof (clarsimp simp: ANR_def)
  fix U and T :: ('a * real) set
  assume S homeomorphic T and clo: closedin (top_of_set U) T
  then obtain g h where hom: homeomorphism S T g h
    by (force simp: homeomorphic_def)
  obtain h: continuous_on T h  h ∈ T → S
    using hom homeomorphism_def by blast
  obtain V h' where T ⊆ V and opV: openin (top_of_set U) V
              and h': continuous_on V h' h' ∈ V → S
             and h'h: ∀x∈T. h' x = h x
    using assms [OF h clo] by blast
```

```
  have [simp]: T ⊆ U
    using clo closedin_imp_subset by auto
  have T retract_of V
  proof (simp add: retraction_def retract_of_def, intro exI conjI ‹T ⊆ V›)
    show continuous_on V (g ∘ h')
      by (meson continuous_on_compose continuous_on_subset h' hom homeo-
morphism_def image_subset_iff_funcset)
    show (g ∘ h') ∈ V → T
      using h' hom homeomorphism_image1 by fastforce
    show ∀ x∈T. (g ∘ h') x = x
      by clarsimp (metis h'h hom homeomorphism_def)
  qed
  then show ∃ V. openin (top_of_set U) V ∧ T retract_of V
    using opV by blast
qed

lemma ANR_eq_absolute_neighbourhood_extensor:
  fixes S :: 'a::euclidean_space set
  shows ANR S ⟷
        (∀ f :: 'a * real ⇒ 'a.
         ∀ U T. continuous_on T f ⟶ f ∈ T → S ⟶
                closedin (top_of_set U) T ⟶
                (∃ V g. T ⊆ V ∧ openin (top_of_set U) V ∧
                        continuous_on V g ∧ g ∈ V → S ∧ (∀ x ∈ T. g x = f x))) (is
_ = ?rhs)
proof
  assume ANR S then show ?rhs
    by (metis ANR_imp_absolute_neighbourhood_extensor)
qed (simp add: absolute_neighbourhood_extensor_imp_ANR)

lemma ANR_imp_neighbourhood_retract:
  fixes S :: 'a::euclidean_space set
  assumes ANR S closedin (top_of_set U) S
  obtains V where openin (top_of_set U) V S retract_of V
using ANR_imp_absolute_neighbourhood_retract assms homeomorphic_refl by
blast

lemma ANR_imp_absolute_closed_neighbourhood_retract:
  fixes S :: 'a::euclidean_space set and S' :: 'b::euclidean_space set
  assumes ANR S S homeomorphic S' and US': closedin (top_of_set U) S'
  obtains V W
    where openin (top_of_set U) V
          closedin (top_of_set U) W
          S' ⊆ V V ⊆ W S' retract_of W
proof −
  obtain Z where openin (top_of_set U) Z and S'Z: S' retract_of Z
    by (blast intro: assms ANR_imp_absolute_neighbourhood_retract)
  then have UUZ: closedin (top_of_set U) (U − Z)
    by auto
```

```
  have S' ∩ (U − Z) = {}
    using ‹S' retract_of Z› closedin_retract closedin_subtopology by fastforce
  then obtain V W
      where openin (top_of_set U) V
        and openin (top_of_set U) W
        and S' ⊆ V U − Z ⊆ W V ∩ W = {}
      using separation_normal_local [OF US' UUZ]  by auto
  moreover have S' retract_of U − W
  proof (rule retract_of_subset [OF S'Z])
    show S' ⊆ U − W
      using US' ‹S' ⊆ V› ‹V ∩ W = {}› closedin_subset by fastforce
    show U − W ⊆ Z
      using Diff_subset_conv ‹U − Z ⊆ W› by blast
  qed
  ultimately show ?thesis
   by (metis Diff_subset_conv Diff_triv Int_Diff_Un Int_absorb1 openin_closedin_eq
that topspace_euclidean_subtopology)
qed

lemma ANR_imp_closed_neighbourhood_retract:
  fixes S :: 'a::euclidean_space set
  assumes ANR S closedin (top_of_set U) S
  obtains V W where openin (top_of_set U) V
                    closedin (top_of_set U) W
                    S ⊆ V V ⊆ W S retract_of W
by (meson ANR_imp_absolute_closed_neighbourhood_retract assms homeomor-
phic_refl)

lemma ANR_homeomorphic_ANR:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes ANR T S homeomorphic T
    shows ANR S
unfolding ANR_def
by (metis assms ANR_imp_absolute_neighbourhood_retract homeomorphic_trans
[of _ S] homeomorphic_sym)

lemma homeomorphic_ANR_iff_ANR:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  shows S homeomorphic T ⟹ ANR S ⟷ ANR T
by (metis ANR_homeomorphic_ANR homeomorphic_sym)
```

6.35.1 Analogous properties of ENRs

```
lemma ENR_imp_absolute_neighbourhood_retract:
  fixes S :: 'a::euclidean_space set and S' :: 'b::euclidean_space set
  assumes ENR S and hom: S homeomorphic S'
     and S' ⊆ U
  obtains V where openin (top_of_set U) V S' retract_of V
proof −
```

```
  obtain X where open X S retract_of X
    using ‹ENR S› by (auto simp: ENR_def)
  then obtain r where retraction X S r
    by (auto simp: retract_of_def)
  have locally compact S'
    using retract_of_locally_compact open_imp_locally_compact
        homeomorphic_local_compactness ‹S retract_of X› ‹open X› hom by blast
  then obtain W where UW: openin (top_of_set U) W
                  and WS': closedin (top_of_set W) S'
    apply (rule locally_compact_closedin_open)
     by (meson Int_lower2 assms(3) closedin_imp_subset closedin_subset_trans
le_inf_iff openin_open)
  obtain f g where hom: homeomorphism S S' f g
    using assms by (force simp: homeomorphic_def)
  have contg: continuous_on S' g
    using hom homeomorphism_def by blast
  moreover have g ‘ S' ⊆ S by (metis hom equalityE homeomorphism_def)
  ultimately obtain h where conth: continuous_on W h and hg: ⋀x. x ∈ S'
⟹ h x = g x
    using Tietze_unbounded [of S' g W] WS' by blast
  have W ⊆ U using UW openin_open by auto
  have S' ⊆ W using WS' closedin_closed by auto
  have him: ⋀x. x ∈ S' ⟹ h x ∈ X
   by (metis (no_types) ‹S retract_of X› hg hom homeomorphism_def image_insert
insert_absorb insert_iff retract_of_imp_subset subset_eq)
  have S' retract_of (W ∩ h −‘ X)
  proof (simp add: retraction_def retract_of_def, intro exI conjI)
    show S' ⊆ W S' ⊆ h −‘ X
      using him WS' closedin_imp_subset by blast+
    show continuous_on (W ∩ h −‘ X) (f ∘ r ∘ h)
    proof (intro continuous_on_compose)
      show continuous_on (W ∩ h −‘ X) h
        by (meson conth continuous_on_subset inf_le1)
      show continuous_on (h ‘ (W ∩ h −‘ X)) r
      proof −
        have h ‘ (W ∩ h −‘ X) ⊆ X
          by blast
        then show continuous_on (h ‘ (W ∩ h −‘ X)) r
          by (meson ‹retraction X S r› continuous_on_subset retraction)
      qed
      show continuous_on (r ‘ h ‘ (W ∩ h −‘ X)) f
      proof (rule continuous_on_subset [of S])
        show continuous_on S f
          using hom homeomorphism_def by blast
        show r ‘ h ‘ (W ∩ h −‘ X) ⊆ S
        by (metis ‹retraction X S r› image_mono image_subset_iff_subset_vimage
inf_le2 retraction)
      qed
    qed
```

show $(f \circ r \circ h) \in (W \cap h -` X) \to S'$
using ‹*retraction X S r*› *hom*
by (*auto simp*: *retraction_def homeomorphism_def*)
show $\forall x \in S'.\ (f \circ r \circ h)\ x = x$
using ‹*retraction X S r*› *hom* **by** (*auto simp*: *retraction_def homeomorphism_def hg*)
qed
then show *?thesis*
using *UW* ‹*open X*› *conth continuous_openin_preimage_eq openin_trans that*
by *blast*
qed

corollary *ENR_imp_absolute_neighbourhood_retract_UNIV*:
fixes $S :: 'a::euclidean\_space\ set$ **and** $S' :: 'b::euclidean\_space\ set$
assumes *ENR S S homeomorphic S'*
obtains T' **where** *open T' S' retract_of T'*
by (*metis ENR_imp_absolute_neighbourhood_retract UNIV_I assms*(*1*) *assms*(*2*) *open_openin subsetI subtopology_UNIV*)

lemma *ENR_homeomorphic_ENR*:
fixes $S :: 'a::euclidean\_space\ set$ **and** $T :: 'b::euclidean\_space\ set$
assumes *ENR T S homeomorphic T*
shows *ENR S*
unfolding *ENR_def*
by (*meson ENR_imp_absolute_neighbourhood_retract_UNIV assms homeomorphic_sym*)

lemma *homeomorphic_ENR_iff_ENR*:
fixes $S :: 'a::euclidean\_space\ set$ **and** $T :: 'b::euclidean\_space\ set$
assumes *S homeomorphic T*
shows *ENR S* $\longleftrightarrow$ *ENR T*
by (*meson ENR_homeomorphic_ENR assms homeomorphic_sym*)

lemma *ENR_translation*:
fixes $S :: 'a::euclidean\_space\ set$
shows *ENR*(*image* $(\lambda x.\ a + x)$ *S*) $\longleftrightarrow$ *ENR S*
by (*meson homeomorphic_sym homeomorphic_translation homeomorphic_ENR_iff_ENR*)

lemma *ENR_linear_image_eq*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes *linear f inj f*
shows *ENR* (*image f S*) $\longleftrightarrow$ *ENR S*
by (*meson assms homeomorphic_ENR_iff_ENR linear_homeomorphic_image*)

Some relations among the concepts. We also relate AR to being a retract of UNIV, which is often a more convenient proxy in the closed case.

lemma *AR_imp_ANR*: *AR S* $\Longrightarrow$ *ANR S*
using *ANR_def AR_def* **by** *fastforce*

lemma *ENR_imp_ANR*:
fixes S :: *'a*::*euclidean_space set*
shows *ENR S* $\Longrightarrow$ *ANR S*
by (*meson ANR_def ENR_imp_absolute_neighbourhood_retract closedin_imp_subset*)

lemma *ENR_ANR*:
fixes S :: *'a*::*euclidean_space set*
shows *ENR S* $\longleftrightarrow$ *ANR S* $\wedge$ *locally compact S*
proof
assume *ENR S*
then have *locally compact S*
using *ENR_def open_imp_locally_compact retract_of_locally_compact* **by**
auto
then show *ANR S* $\wedge$ *locally compact S*
using *ENR_imp_ANR* ‹*ENR S*› **by** *blast*
next
assume *ANR S* $\wedge$ *locally compact S*
then have *ANR S locally compact S* **by** *auto*
then obtain T :: (*'a* $*$ *real*) *set* **where** *closed T S homeomorphic T*
using *locally_compact_homeomorphic_closed*
by (*metis DIM_prod DIM_real Suc_eq_plus1 lessI*)
then show *ENR S*
using ‹*ANR S*›
by (*meson ANR_imp_absolute_neighbourhood_retract_UNIV ENR_def ENR_homeomorphic_ENR*)
qed

lemma *AR_ANR*:
fixes S :: *'a*::*euclidean_space set*
shows *AR S* $\longleftrightarrow$ *ANR S* $\wedge$ *contractible S* $\wedge$ $S \neq \{\}$
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
have *aff_dim S* $<$ *int DIM*(*'a* $\times$ *real*)
using *aff_dim_le_DIM* [*of S*] **by** *auto*
then obtain C **and** S' :: (*'a* $*$ *real*) *set*
where *convex C C* $\neq \{\}$ *closedin* (*top_of_set C*) S' *S homeomorphic S'*
using *homeomorphic_closedin_convex* **by** *blast*
with ‹*AR S*› **have** *contractible S*
by (*meson AR_def convex_imp_contractible homeomorphic_contractible_eq*
retract_of_contractible)
with ‹*AR S*› **show** *?rhs*
using *AR_imp_ANR AR_imp_retract* **by** *fastforce*
next
assume *?rhs*
then obtain a **and** h:: *real* $\times$ *'a* $\Rightarrow$ *'a*
where *conth*: *continuous_on* ($\{0..1\} \times S$) h
and *hS*: h ‘ ($\{0..1\} \times S$) $\subseteq S$
and [*simp*]: $\bigwedge x.\ h(0, x) = x$

```
      and [simp]: ⋀x. h(1, x) = a
      and ANR S S ≠ {}
  by (auto simp: contractible_def homotopic_with_def)
 then have a ∈ S
  by (metis all_not_in_conv atLeastAtMost_iff image_subset_iff mem_Sigma_iff
order_refl zero_le_one)
 have ∃g. continuous_on W g ∧ g ∈ W → S ∧ (∀x∈T. g x = f x)
       if      f: continuous_on T f f ∈ T → S
         and WT: closedin (top_of_set W) T
       for W T and f :: 'a × real ⇒ 'a
 proof −
   obtain U g
     where T ⊆ U and WU: openin (top_of_set W) U
       and contg: continuous_on U g
       and g ∈ U → S and gf: ⋀x. x ∈ T ⟹ g x = f x
        using iffD1 [OF ANR_eq_absolute_neighbourhood_extensor ‹ANR S›,
rule_format, OF f WT]
     by auto
   have WWU: closedin (top_of_set W) (W − U)
     using WU closedin_diff by fastforce
   moreover have (W − U) ∩ T = {}
     using ‹T ⊆ U› by auto
   ultimately obtain V V'
     where WV': openin (top_of_set W) V'
       and WV: openin (top_of_set W) V
       and W − U ⊆ V' T ⊆ V V' ∩ V = {}
     using separation_normal_local [of W W−U T] WT by blast
   then have WVT: T ∩ (W − V) = {}
     by auto
   have WWV: closedin (top_of_set W) (W − V)
     using WV closedin_diff by fastforce
   obtain j :: 'a × real ⇒ real
     where contj: continuous_on W j
       and j: ⋀x. x ∈ W ⟹ j x ∈ {0..1}
       and j0: ⋀x. x ∈ W − V ⟹ j x = 1
       and j1: ⋀x. x ∈ T ⟹ j x = 0
       by (rule Urysohn_local [OF WT WWV WVT, of 0 1::real]) (auto simp:
in_segment)
   have Weq: W = (W − V) ∪ (W − V')
     using ‹V' ∩ V = {}› by force
   show ?thesis
   proof (intro conjI exI)
     have ∗: continuous_on (W − V') (λx. h (j x, g x))
     proof (rule continuous_on_compose2 [OF conth continuous_on_Pair])
       show continuous_on (W − V') j
         by (rule continuous_on_subset [OF contj Diff_subset])
       show continuous_on (W − V') g
         by (metis Diff_subset_conv ‹W − U ⊆ V'› contg continuous_on_subset
Un_commute)
```

```
      show (λx. (j x, g x)) ‘ (W − V′) ⊆ {0..1} × S
        using j ‹g ∈ U → S› ‹W − U ⊆ V′› by fastforce
    qed
    show continuous_on W (λx. if x ∈ W − V then a else h (j x, g x))
    proof (subst Weq, rule continuous_on_cases_local)
      show continuous_on (W − V′) (λx. h (j x, g x))
        using ∗ by blast
    qed (use WWV WV′ Weq j0 j1 in auto)
  next
    have h (j (x, y), g (x, y)) ∈ S if (x, y) ∈ W (x, y) ∈ V for x y
    proof −
      have j(x, y) ∈ {0..1}
        using j that by blast
      moreover have g(x, y) ∈ S
        using ‹V′ ∩ V = {}› ‹W − U ⊆ V′› ‹g ∈ U → S› that by fastforce
      ultimately show ?thesis
        using hS by blast
    qed
    with ‹a ∈ S› ‹g ∈ U → S›
    show (λx. if x ∈ W − V then a else h (j x, g x)) ∈ W → S
      by auto
  next
    show ∀x∈T. (if x ∈ W − V then a else h (j x, g x)) = f x
      using ‹T ⊆ V› by (auto simp: j0 j1 gf)
  qed
 qed
 then show ?lhs
  by (simp add: AR_eq_absolute_extensor image_subset_iff_funcset)
qed


lemma ANR_retract_of_ANR:
 fixes S :: 'a::euclidean_space set
 assumes ANR T and ST: S retract_of T
 shows ANR S
proof (clarsimp simp add: ANR_eq_absolute_neighbourhood_extensor)
 fix f::'a × real ⇒ 'a and U W
 assume W: continuous_on W f f ∈ W → S closedin (top_of_set U) W
 then obtain r where S ⊆ T and r: continuous_on T r r ∈ T → S ∀x∈S. r
x = x continuous_on W f f ∈ W → S
                                closedin (top_of_set U) W
  by (metis ST retract_of_def retraction_def)
 then have f ‘ W ⊆ T
  by blast
 with W obtain V g where V: W ⊆ V openin (top_of_set U) V continuous_on
V g g ∈ V → T ∀x∈W. g x = f x
   by (smt (verit) ANR_imp_absolute_neighbourhood_extensor Pi_I assms(1)
funcset_mem image_subset_iff_funcset)
 with r have continuous_on V (r ∘ g) ∧ (r ∘ g) ∈ V → S ∧ (∀x∈W. (r ∘ g) x
```

$= f\ x$)
by (*smt* (*verit*, *del_insts*) *Pi_iff comp_apply continuous_on_compose continuous_on_subset image_subset_iff_funcset*)
then show $\exists\, V.\ W \subseteq V \wedge$ *openin* (*top_of_set U*) $V \wedge (\exists\, g.$ *continuous_on V* $g \wedge g \in V \to S \wedge (\forall\, x{\in}W.\ g\ x = f\ x))$
by (*meson V*)
qed

lemma *AR_retract_of_AR*:
fixes S :: *'a*::*euclidean_space set*
shows $[\![$*AR T*; *S retract_of T*$]\!] \Longrightarrow$ *AR S*
using *ANR_retract_of_ANR AR_ANR retract_of_contractible* **by** *fastforce*

lemma *ENR_retract_of_ENR*:
$[\![$*ENR T*; *S retract_of T*$]\!] \Longrightarrow$ *ENR S*
by (*meson ENR_def retract_of_trans*)

lemma *retract_of_UNIV*:
fixes S :: *'a*::*euclidean_space set*
shows *S retract_of UNIV* $\longleftrightarrow$ *AR S* $\wedge$ *closed S*
by (*metis AR_ANR AR_imp_retract ENR_def ENR_imp_ANR closed_UNIV closed_closedin contractible_UNIV empty_not_UNIV open_UNIV retract_of_closed retract_of_contractible retract_of_empty*(*1*) *subtopology_UNIV*)

lemma *compact_AR*:
fixes S :: *'a*::*euclidean_space set*
shows *compact S* $\wedge$ *AR S* $\longleftrightarrow$ *compact S* $\wedge$ *S retract_of UNIV*
using *compact_imp_closed retract_of_UNIV* **by** *blast*

More properties of ARs, ANRs and ENRs

lemma *not_AR_empty* [*simp*]: $\neg$ *AR*({})
by (*auto simp*: *AR_def*)

lemma *ENR_empty* [*simp*]: *ENR* {}
by (*simp add*: *ENR_def*)

lemma *ANR_empty* [*simp*]: *ANR* ({} :: *'a*::*euclidean_space set*)
by (*simp add*: *ENR_imp_ANR*)

lemma *convex_imp_AR*:
fixes S :: *'a*::*euclidean_space set*
shows $[\![$*convex S*; $S \neq \{\}]\!] \Longrightarrow$ *AR S*
by (*metis* (*mono_tags*, *lifting*) *Dugundji absolute_extensor_imp_AR*)

lemma *convex_imp_ANR*:
fixes S :: *'a*::*euclidean_space set*
shows *convex S* $\Longrightarrow$ *ANR S*
using *ANR_empty AR_imp_ANR convex_imp_AR* **by** *blast*

lemma *ENR_convex_closed*:
fixes $S :: 'a::euclidean\_space\ set$
shows $[\![closed\ S;\ convex\ S]\!] \Longrightarrow ENR\ S$
using *ENR_def ENR_empty convex_imp_AR retract_of_UNIV* **by** *blast*

lemma *AR_UNIV* [*simp*]: $AR\ (UNIV :: 'a::euclidean\_space\ set)$
using *retract_of_UNIV* **by** *auto*

lemma *ANR_UNIV* [*simp*]: $ANR\ (UNIV :: 'a::euclidean\_space\ set)$
by (*simp add*: *AR_imp_ANR*)

lemma *ENR_UNIV* [*simp*]:*ENR UNIV*
using *ENR_def* **by** *blast*

lemma *AR_singleton*:
fixes $a :: 'a::euclidean\_space$
shows $AR\ \{a\}$
using *retract_of_UNIV* **by** *blast*

lemma *ANR_singleton*:
fixes $a :: 'a::euclidean\_space$
shows $ANR\ \{a\}$
by (*simp add*: *AR_imp_ANR AR_singleton*)

lemma *ENR_singleton*: $ENR\ \{a\}$
using *ENR_def* **by** *blast*

ARs closed under union

lemma *AR_closed_Un_local_aux*:
fixes $U :: 'a::euclidean\_space\ set$
assumes *closedin* (*top_of_set U*) *S*
closedin (*top_of_set U*) *T*
AR S AR T AR$(S \cap T)$
shows $(S \cup T)\ retract\_of\ U$
proof −
have $S \cap T \neq \{\}$
using *assms AR_def* **by** *fastforce*
have $S \subseteq U\ T \subseteq U$
using *assms* **by** (*auto simp*: *closedin_imp_subset*)
define S' **where** $S' \equiv \{x \in U.\ setdist\ \{x\}\ S \leq setdist\ \{x\}\ T\}$
define T' **where** $T' \equiv \{x \in U.\ setdist\ \{x\}\ T \leq setdist\ \{x\}\ S\}$
define W **where** $W \equiv \{x \in U.\ setdist\ \{x\}\ S = setdist\ \{x\}\ T\}$
have *US'*: *closedin* (*top_of_set U*) S'
using *continuous_closedin_preimage* [*of U* $\lambda x.\ setdist\ \{x\}\ S - setdist\ \{x\}\ T$ *{..0}*]
by (*simp add*: *S'_def vimage_def Collect_conj_eq continuous_on_diff continuous_on_setdist*)
have *UT'*: *closedin* (*top_of_set U*) T'
using *continuous_closedin_preimage* [*of U* $\lambda x.\ setdist\ \{x\}\ T - setdist\ \{x\}\ S$

```
{..0}]
    by (simp add: T'_def vimage_def Collect_conj_eq continuous_on_diff con-
tinuous_on_setdist)
  have S ⊆ S'
    using S'_def ‹S ⊆ U› setdist_sing_in_set by fastforce
  have T ⊆ T'
    using T'_def ‹T ⊆ U› setdist_sing_in_set by fastforce
  have S ∩ T ⊆ W W ⊆ U
    using ‹S ⊆ U› by (auto simp: W_def setdist_sing_in_set)
  have (S ∩ T) retract_of W
  proof (rule AR_imp_absolute_retract [OF ‹AR(S ∩ T)›])
    show S ∩ T homeomorphic S ∩ T
      by (simp add: homeomorphic_refl)
    show closedin (top_of_set W) (S ∩ T)
     by (meson ‹S ∩ T ⊆ W› ‹W ⊆ U› assms closedin_Int closedin_subset_trans)
  qed
  then obtain r0
    where S ∩ T ⊆ W and contr0: continuous_on W r0
      and r0 ` W ⊆ S ∩ T
      and r0 [simp]: ⋀x. x ∈ S ∩ T ⟹ r0 x = x
      by (auto simp: retract_of_def retraction_def)
  have ST: x ∈ W ⟹ x ∈ S ⟷ x ∈ T for x
    using setdist_eq_0_closedin ‹S ∩ T ≠ {}› assms
    by (force simp: W_def setdist_sing_in_set)
  have S' ∩ T' = W
    by (auto simp: S'_def T'_def W_def)
  then have cloUW: closedin (top_of_set U) W
    using closedin_Int US' UT' by blast
  define r where r ≡ λx. if x ∈ W then r0 x else x
  have contr: continuous_on (W ∪ (S ∪ T)) r
  unfolding r_def
  proof (rule continuous_on_cases_local [OF _ _ contr0 continuous_on_id])
    show closedin (top_of_set (W ∪ (S ∪ T))) W
    using ‹S ⊆ U› ‹T ⊆ U› ‹W ⊆ U› ‹closedin (top_of_set U) W› closedin_subset_trans
by fastforce
    show closedin (top_of_set (W ∪ (S ∪ T))) (S ∪ T)
    by (meson ‹S ⊆ U› ‹T ⊆ U› ‹W ⊆ U› assms closedin_Un closedin_subset_trans
sup.bounded_iff sup.cobounded2)
    show ⋀x. x ∈ W ∧ x ∉ W ∨ x ∈ S ∪ T ∧ x ∈ W ⟹ r0 x = x
      by (auto simp: ST)
  qed
  have rim: r ` (W ∪ S) ⊆ S r ` (W ∪ T) ⊆ T
    using ‹r0 ` W ⊆ S ∩ T› r_def by auto
  have cloUWS: closedin (top_of_set U) (W ∪ S)
    by (simp add: cloUW assms closedin_Un)
  obtain g where contg: continuous_on U g
    and g ` U ⊆ S and geqr: ⋀x. x ∈ W ∪ S ⟹ g x = r x
  proof (rule AR_imp_absolute_extensor [OF ‹AR S› _ _ cloUWS])
    show continuous_on (W ∪ S) r
```

```
      using continuous_on_subset contr sup_assoc by blast
  qed (use rim in auto)
  have cloUWT: closedin (top_of_set U) (W ∪ T)
    by (simp add: cloUW assms closedin_Un)
  obtain h where conth: continuous_on U h
            and h ‘ U ⊆ T and heqr: ⋀x. x ∈ W ∪ T ⟹ h x = r x
  proof (rule AR_imp_absolute_extensor [OF ‹AR T› _ _ cloUWT])
    show continuous_on (W ∪ T) r
      using continuous_on_subset contr sup_assoc by blast
  qed (use rim in auto)
  have U: U = S' ∪ T'
    by (force simp: S'_def T'_def)
  have cont: continuous_on U (λx. if x ∈ S' then g x else h x)
    unfolding U
    apply (rule continuous_on_cases_local)
    using US' UT' ‹S' ∩ T' = W› ‹U = S' ∪ T'›
          contg conth continuous_on_subset geqr heqr by auto
  have UST: (λx. if x ∈ S' then g x else h x) ‘ U ⊆ S ∪ T
    using ‹g ‘ U ⊆ S› ‹h ‘ U ⊆ T› by auto
  show ?thesis
    apply (simp add: retract_of_def retraction_def ‹S ⊆ U› ‹T ⊆ U›)
    apply (rule_tac x=λx. if x ∈ S' then g x else h x in exI)
    using ST UST ‹S ⊆ S'› ‹S' ∩ T' = W› ‹T ⊆ T'› cont geqr heqr r_def
    by (smt (verit, del_insts) IntI Pi_I Un_iff image_subset_iff r0 subsetD)
qed


lemma AR_closed_Un_local:
  fixes S :: 'a::euclidean_space set
  assumes STS: closedin (top_of_set (S ∪ T)) S
      and STT: closedin (top_of_set (S ∪ T)) T
      and AR S AR T AR(S ∩ T)
    shows AR(S ∪ T)
proof −
  have C retract_of U
       if hom: S ∪ T homeomorphic C and UC: closedin (top_of_set U) C
       for U and C :: ('a * real) set
  proof −
    obtain f g where hom: homeomorphism (S ∪ T) C f g
      using hom by (force simp: homeomorphic_def)
    have US: closedin (top_of_set U) (C ∩ g −‘ S)
    by (metis STS continuous_on_imp_closedin hom homeomorphism_def closedin_trans
[OF _ UC])
    have UT: closedin (top_of_set U) (C ∩ g −‘ T)
    by (metis STT continuous_on_closed hom homeomorphism_def closedin_trans
[OF _ UC])
    have homeomorphism (C ∩ g −‘ S) S g f
      using hom
      apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
```

```
    apply (rule_tac x=f x in image_eqI, auto)
    done
  then have ARS: AR (C ∩ g -‘ S)
    using ‹AR S› homeomorphic_AR_iff_AR homeomorphic_def by blast
  have homeomorphism (C ∩ g -‘ T) T g f
    using hom
    apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
    apply (rule_tac x=f x in image_eqI, auto)
    done
  then have ART: AR (C ∩ g -‘ T)
    using ‹AR T› homeomorphic_AR_iff_AR homeomorphic_def by blast
  have homeomorphism (C ∩ g -‘ S ∩ (C ∩ g -‘ T)) (S ∩ T) g f
    using hom
    apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
    apply (rule_tac x=f x in image_eqI, auto)
    done
  then have ARI: AR ((C ∩ g -‘ S) ∩ (C ∩ g -‘ T))
   using ‹AR (S ∩ T)› homeomorphic_AR_iff_AR homeomorphic_def by blast
  have C = (C ∩ g -‘ S) ∪ (C ∩ g -‘ T)
    using hom by (auto simp: homeomorphism_def)
  then show ?thesis
    by (metis AR_closed_Un_local_aux [OF US UT ARS ART ARI])
 qed
 then show ?thesis
  by (force simp: AR_def)
qed

corollary AR_closed_Un:
 fixes S :: 'a::euclidean_space set
 shows ⟦closed S; closed T; AR S; AR T; AR (S ∩ T)⟧ ⟹ AR (S ∪ T)
by (metis AR_closed_Un_local_aux closed_closedin retract_of_UNIV subtopology_UNIV)
```

ANRs closed under union

```
lemma ANR_closed_Un_local_aux:
 fixes U :: 'a::euclidean_space set
 assumes US: closedin (top_of_set U) S
    and UT: closedin (top_of_set U) T
    and ANR S ANR T ANR(S ∩ T)
 obtains V where openin (top_of_set U) V (S ∪ T) retract_of V
proof (cases S = {} ∨ T = {})
 case True with assms that show ?thesis
  by (metis ANR_imp_neighbourhood_retract Un_commute inf_bot_right sup_inf_absorb)
next
 case False
 then have [simp]: S ≠ {} T ≠ {} by auto
 have S ⊆ U T ⊆ U
  using assms by (auto simp: closedin_imp_subset)
 define S' where S' ≡ {x ∈ U. setdist {x} S ≤ setdist {x} T}
```

define T' **where** $T' \equiv \{x \in U.\ setdist\ \{x\}\ T \leq setdist\ \{x\}\ S\}$
define W **where** $W \equiv \{x \in U.\ setdist\ \{x\}\ S = setdist\ \{x\}\ T\}$
have *cloUS'*: *closedin* (*top_of_set U*) S'
using *continuous_closedin_preimage* [*of U* $\lambda x.$ *setdist* $\{x\}$ S − *setdist* $\{x\}$ T {..*0*}]
by (*simp add*: *S'_def vimage_def Collect_conj_eq continuous_on_diff continuous_on_setdist*)
have *cloUT'*: *closedin* (*top_of_set U*) T'
using *continuous_closedin_preimage* [*of U* $\lambda x.$ *setdist* $\{x\}$ T − *setdist* $\{x\}$ S {..*0*}]
by (*simp add*: *T'_def vimage_def Collect_conj_eq continuous_on_diff continuous_on_setdist*)
have $S \subseteq S'$
using *S'_def* ‹$S \subseteq U$› *setdist_sing_in_set* **by** *fastforce*
have $T \subseteq T'$
using *T'_def* ‹$T \subseteq U$› *setdist_sing_in_set* **by** *fastforce*
have $S' \cup T' = U$
by (*auto simp*: *S'_def T'_def*)
have $W \subseteq S'$
by (*simp add*: *Collect_mono S'_def W_def*)
have $W \subseteq T'$
by (*simp add*: *Collect_mono T'_def W_def*)
have *ST_W*: $S \cap T \subseteq W$ **and** $W \subseteq U$
using ‹$S \subseteq U$› **by** (*force simp*: *W_def setdist_sing_in_set*)+
have $S' \cap T' = W$
by (*auto simp*: *S'_def T'_def W_def*)
then have *cloUW*: *closedin* (*top_of_set U*) W
using *closedin_Int cloUS' cloUT'* **by** *blast*
obtain *W' W0* **where** *openin* (*top_of_set W*) *W'*
and *cloWW0*: *closedin* (*top_of_set W*) *W0*
and $S \cap T \subseteq W'$ $W' \subseteq W0$
and *ret*: ($S \cap T$) *retract_of W0*
by (*meson ANR_imp_closed_neighbourhood_retract ST_W US UT* ‹$W \subseteq U$› ‹*ANR*($S \cap T$)› *closedin_Int closedin_subset_trans*)
then obtain *U0* **where** *opeUU0*: *openin* (*top_of_set U*) *U0*
and *U0*: $S \cap T \subseteq U0$ $U0 \cap W \subseteq W0$
unfolding *openin_open* **using** ‹$W \subseteq U$› **by** *blast*
have $W0 \subseteq U$
using ‹$W \subseteq U$› *cloWW0 closedin_subset* **by** *fastforce*
obtain *r0*
where $S \cap T \subseteq W0$ **and** *contr0*: *continuous_on W0 r0* **and** $r0 \in W0 \to S \cap T$
and *r0* [*simp*]: $\bigwedge x.\ x \in S \cap T \Longrightarrow r0\ x = x$
using *ret* **by** (*force simp*: *retract_of_def retraction_def*)
have *ST*: $x \in W \Longrightarrow x \in S \longleftrightarrow x \in T$ **for** x
using *assms* **by** (*auto simp*: *W_def setdist_sing_in_set dest*!: *setdist_eq_0_closedin*)
define r **where** $r \equiv \lambda x.$ *if* $x \in W0$ *then r0 x else x*
have r ‘ ($W0 \cup S$) $\subseteq S$ r ‘ ($W0 \cup T$) $\subseteq T$
using ‹*r0* $\in W0 \to S \cap T$› *r_def* **by** *auto*

have *contr*: *continuous_on* (*W0* ∪ (*S* ∪ *T*)) *r*
unfolding *r_def*
proof (*rule continuous_on_cases_local* [*OF* _ _ *contr0 continuous_on_id*])
show *closedin* (*top_of_set* (*W0* ∪ (*S* ∪ *T*))) *W0*
using *closedin_subset_trans* [*of U*]
by (*metis le_sup_iff order_refl cloWW0 cloUW closedin_trans* ‹*W0* ⊆ *U*› ‹*S* ⊆ *U*› ‹*T* ⊆ *U*›)
show *closedin* (*top_of_set* (*W0* ∪ (*S* ∪ *T*))) (*S* ∪ *T*)
by (*meson* ‹*S* ⊆ *U*› ‹*T* ⊆ *U*› ‹*W0* ⊆ *U*› *assms closedin_Un closedin_subset_trans sup.bounded_iff sup.cobounded2*)
show ⋀*x*. *x* ∈ *W0* ∧ *x* ∉ *W0* ∨ *x* ∈ *S* ∪ *T* ∧ *x* ∈ *W0* ⟹ *r0 x* = *x*
using *ST cloWW0 closedin_subset* **by** *fastforce*
qed
have *cloS'WS*: *closedin* (*top_of_set S'*) (*W0* ∪ *S*)
by (*meson closedin_subset_trans US cloUS'* ‹*S* ⊆ *S'*› ‹*W* ⊆ *S'*› *cloUW cloWW0*
closedin_Un closedin_imp_subset closedin_trans)
obtain *W1 g* **where** *W0* ∪ *S* ⊆ *W1* **and** *contg*: *continuous_on W1 g*
and *opeSW1*: *openin* (*top_of_set S'*) *W1*
and *g* ∈ *W1* → *S* **and** *geqr*: ⋀*x*. *x* ∈ *W0* ∪ *S* ⟹ *g x* = *r x*
proof (*rule ANR_imp_absolute_neighbourhood_extensor* [*OF* ‹*ANR S*› _ _ *cloS'WS*])
show *continuous_on* (*W0* ∪ *S*) *r*
using *continuous_on_subset contr sup_assoc* **by** *blast*
qed (*use* ‹*r* ‘ (*W0* ∪ *S*) ⊆ *S*› **in** *auto*)
have *cloT'WT*: *closedin* (*top_of_set T'*) (*W0* ∪ *T*)
by (*meson closedin_subset_trans UT cloUT'* ‹*T* ⊆ *T'*› ‹*W* ⊆ *T'*› *cloUW cloWW0*
closedin_Un closedin_imp_subset closedin_trans)
obtain *W2 h* **where** *W0* ∪ *T* ⊆ *W2* **and** *conth*: *continuous_on W2 h*
and *opeSW2*: *openin* (*top_of_set T'*) *W2*
and *h* ‘ *W2* ⊆ *T* **and** *heqr*: ⋀*x*. *x* ∈ *W0* ∪ *T* ⟹ *h x* = *r x*
proof (*rule ANR_imp_absolute_neighbourhood_extensor* [*OF* ‹*ANR T*› _ _ *cloT'WT*])
show *continuous_on* (*W0* ∪ *T*) *r*
using *continuous_on_subset contr sup_assoc* **by** *blast*
qed (*use* ‹*r* ‘ (*W0* ∪ *T*) ⊆ *T*› **in** *auto*)
have *S'* ∩ *T'* = *W*
by (*force simp*: *S'_def T'_def W_def*)
obtain *O1 O2* **where** *O12*: *open O1 W1* = *S'* ∩ *O1 open O2 W2* = *T'* ∩ *O2*
using *opeSW1 opeSW2* **by** (*force simp*: *openin_open*)
show *?thesis*
proof
have *eq*: *W1* − (*W* − *U0*) ∪ (*W2* − (*W* − *U0*))
= ((*U* − *T'*) ∩ *O1* ∪ (*U* − *S'*) ∩ *O2* ∪ *U* ∩ *O1* ∩ *O2*) − (*W* − *U0*)
(**is** *?WW1* ∪ *?WW2* = *?rhs*)
using ‹*U0* ∩ *W* ⊆ *W0*› ‹*W0* ∪ *S* ⊆ *W1*› ‹*W0* ∪ *T* ⊆ *W2*›
by (*auto simp*: ‹*S'* ∪ *T'* = *U*› [*symmetric*] ‹*S'* ∩ *T'* = *W*› [*symmetric*] ‹*W1* = *S'* ∩ *O1*› ‹*W2* = *T'* ∩ *O2*›)

show *openin* (*top_of_set U*) (*?WW1* ∪ *?WW2*)
by (*simp add*: *eq* ‹*open O1*› ‹*open O2*› *cloUS′ cloUT′ cloUW closedin_diff opeUU0 openin_Int_open openin_Un openin_diff*)
obtain *SU′* **where** *closed SU′ S′* = *U* ∩ *SU′*
using *cloUS′* **by** (*auto simp add*: *closedin_closed*)
moreover have *?WW1* = (*?WW1* ∪ *?WW2*) ∩ *SU′*
using ‹*S′* = *U* ∩ *SU′*› ‹*W1* = *S′* ∩ *O1*› ‹*S′* ∪ *T′* = *U*› ‹*W2* = *T′* ∩ *O2*› ‹*S′* ∩ *T′* = *W*› ‹*W0* ∪ *S* ⊆ *W1*› *U0*
by *auto*
ultimately have *cloW1*: *closedin* (*top_of_set* (*W1* − (*W* − *U0*) ∪ (*W2* − (*W* − *U0*)))) (*W1* − (*W* − *U0*))
by (*metis closedin_closed_Int*)
obtain *TU′* **where** *closed TU′ T′* = *U* ∩ *TU′*
using *cloUT′* **by** (*auto simp add*: *closedin_closed*)
moreover have *?WW2* = (*?WW1* ∪ *?WW2*) ∩ *TU′*
using ‹*T′* = *U* ∩ *TU′*› ‹*W1* = *S′* ∩ *O1*› ‹*S′* ∪ *T′* = *U*› ‹*W2* = *T′* ∩ *O2*› ‹*S′* ∩ *T′* = *W*› ‹*W0* ∪ *T* ⊆ *W2*› *U0*
by *auto*
ultimately have *cloW2*: *closedin* (*top_of_set* (*?WW1* ∪ *?WW2*)) *?WW2*
by (*metis closedin_closed_Int*)
let *?gh* = λ*x*. *if x* ∈ *S′ then g x else h x*
have ∃*r*. *continuous_on* (*?WW1* ∪ *?WW2*) *r* ∧ *r* ‘ (*?WW1* ∪ *?WW2*) ⊆ *S* ∪ *T* ∧ (∀ *x*∈*S* ∪ *T*. *r x* = *x*)
proof (*intro exI conjI*)
show ∀ *x*∈*S* ∪ *T*. *?gh x* = *x*
using *ST* ‹*S′* ∩ *T′* = *W*› *geqr heqr O12*
by (*metis Int_iff Un_iff* ‹*W0* ∪ *S* ⊆ *W1*› ‹*W0* ∪ *T* ⊆ *W2*› *r0 r_def sup.order_iff*)
have ⋀*x*. *x* ∈ *?WW1* ∧ *x* ∉ *S′* ∨ *x* ∈ *?WW2* ∧ *x* ∈ *S′* ⟹ *g x* = *h x*
using *O12*
by (*metis* (*full_types*) *DiffD1 DiffD2 DiffI IntE IntI U0*(*2*) *UnCI* ‹*S′* ∩ *T′* = *W*› *geqr heqr in_mono*)
then show *continuous_on* (*?WW1* ∪ *?WW2*) *?gh*
using *continuous_on_cases_local* [*OF cloW1 cloW2 continuous_on_subset* [*OF contg*] *continuous_on_subset* [*OF conth*]]
by *simp*
show *?gh* ‘ (*?WW1* ∪ *?WW2*) ⊆ *S* ∪ *T*
using ‹*W1* = *S′* ∩ *O1*› ‹*W2* = *T′* ∩ *O2*› ‹*S′* ∩ *T′* = *W*› ‹*g* ∈ *W1* → *S*› ‹*h* ‘ *W2* ⊆ *T*› ‹*U0* ∩ *W* ⊆ *W0*› ‹*W0* ∪ *S* ⊆ *W1*›
by (*auto simp add*: *image_subset_iff*)
qed
then show *S* ∪ *T retract_of ?WW1* ∪ *?WW2*
using ‹*W0* ∪ *S* ⊆ *W1*› ‹*W0* ∪ *T* ⊆ *W2*› *ST opeUU0 U0*
by (*auto simp*: *retract_of_def retraction_def image_subset_iff_funcset*)
qed
qed

lemma *ANR_closed_Un_local*:

```
  fixes S :: 'a::euclidean_space set
  assumes STS: closedin (top_of_set (S ∪ T)) S
      and STT: closedin (top_of_set (S ∪ T)) T
      and ANR S ANR T ANR(S ∩ T)
    shows ANR(S ∪ T)
proof −
  have ∃ T. openin (top_of_set U) T ∧ C retract_of T
       if hom: S ∪ T homeomorphic C and UC: closedin (top_of_set U) C
       for U and C :: ('a * real) set
  proof −
    obtain f g where hom: homeomorphism (S ∪ T) C f g
      using hom by (force simp: homeomorphic_def)
    have US: closedin (top_of_set U) (C ∩ g −‘ S)
      by (metis STS UC closedin_trans continuous_on_imp_closedin hom homeomorphism_def)
    have UT: closedin (top_of_set U) (C ∩ g −‘ T)
      by (metis STT UC closedin_trans continuous_on_imp_closedin hom homeomorphism_def)
    have homeomorphism (C ∩ g −‘ S) S g f
      using hom
      apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
      by (rule_tac x=f x in image_eqI, auto)
    then have ANRS: ANR (C ∩ g −‘ S)
      using ‹ANR S› homeomorphic_ANR_iff_ANR homeomorphic_def by blast
    have homeomorphism (C ∩ g −‘ T) T g f
    using hom apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
      by (rule_tac x=f x in image_eqI, auto)
    then have ANRT: ANR (C ∩ g −‘ T)
     using ‹ANR T› homeomorphic_ANR_iff_ANR homeomorphic_def by blast
    have homeomorphism (C ∩ g −‘ S ∩ (C ∩ g −‘ T)) (S ∩ T) g f
      using hom
      apply (auto simp: homeomorphism_def elim!: continuous_on_subset)
      by (rule_tac x=f x in image_eqI, auto)
    then have ANRI: ANR ((C ∩ g −‘ S) ∩ (C ∩ g −‘ T))
       using ‹ANR (S ∩ T)› homeomorphic_ANR_iff_ANR homeomorphic_def
by blast
    have C = (C ∩ g −‘ S) ∪ (C ∩ g −‘ T)
      using hom by (auto simp: homeomorphism_def)
    then show ?thesis
      by (metis ANR_closed_Un_local_aux [OF US UT ANRS ANRT ANRI])
  qed
  then show ?thesis
    by (auto simp: ANR_def)
qed

corollary ANR_closed_Un:
  fixes S :: 'a::euclidean_space set
  shows ⟦closed S; closed T; ANR S; ANR T; ANR (S ∩ T)⟧ ⟹ ANR (S ∪ T)
by (simp add: ANR_closed_Un_local closedin_def diff_eq open_Compl openin_open_Int)
```

lemma *ANR_openin*:
fixes $S :: 'a::euclidean\_space\ set$
assumes *ANR T* **and** *opeTS*: *openin* (*top_of_set T*) *S*
shows *ANR S*
proof (*clarsimp simp only*: *ANR_eq_absolute_neighbourhood_extensor*)
fix $f :: 'a \times real \Rightarrow 'a$ **and** *U C*
assume *contf*: *continuous_on C f* **and** *fim*: $f \in C \to S$
and *cloUC*: *closedin* (*top_of_set U*) *C*
have $f \in C \to T$
using *fim opeTS openin_imp_subset* **by** *blast*
obtain *W g* **where** $C \subseteq W$
and *UW*: *openin* (*top_of_set U*) *W*
and *contg*: *continuous_on W g*
and *gim*: $g \in W \to T$
and *geq*: $\bigwedge x.\ x \in C \Longrightarrow g\ x = f\ x$
using *ANR_imp_absolute_neighbourhood_extensor* [*OF* ‹*ANR T*› *contf* ‹$f \in C \to T$› *cloUC*] *fim* **by** *auto*
show $\exists V\ g.\ C \subseteq V \wedge$ *openin* (*top_of_set U*) *V* $\wedge$ *continuous_on V g* $\wedge\ g \in V \to S \wedge (\forall x{\in}C.\ g\ x = f\ x)$
proof (*intro exI conjI*)
show $C \subseteq W \cap g -` S$
using ‹$C \subseteq W$› *fim geq* **by** *blast*
show *openin* (*top_of_set U*) $(W \cap g -` S)$
by (*metis* (*mono_tags*, *lifting*) *UW contg continuous_openin_preimage gim opeTS openin_trans*)
show *continuous_on* $(W \cap g -` S)\ g$
by (*blast intro*: *continuous_on_subset* [*OF contg*])
show $g \in (W \cap g -` S) \to S$
using *gim* **by** *blast*
show $\forall x{\in}C.\ g\ x = f\ x$
using *geq* **by** *blast*
qed
qed

lemma *ENR_openin*:
fixes $S :: 'a::euclidean\_space\ set$
assumes *ENR T openin* (*top_of_set T*) *S*
shows *ENR S*
by (*meson ANR_openin ENR_ANR assms locally_open_subset*)

lemma *ANR_neighborhood_retract*:
fixes $S :: 'a::euclidean\_space\ set$
assumes *ANR U S retract_of T openin* (*top_of_set U*) *T*
shows *ANR S*
using *ANR_openin ANR_retract_of_ANR assms* **by** *blast*

lemma *ENR_neighborhood_retract*:
fixes $S :: 'a::euclidean\_space\ set$

assumes *ENR U S retract_of T openin* (*top_of_set U*) *T*
shows *ENR S*
using *ENR_openin ENR_retract_of_ENR assms* **by** *blast*

lemma *ANR_rel_interior*:
fixes *S* :: *'a*::*euclidean_space set*
shows *ANR S* ⟹ *ANR*(*rel_interior S*)
by (*blast intro*: *ANR_openin openin_set_rel_interior*)

lemma *ANR_delete*:
fixes *S* :: *'a*::*euclidean_space set*
shows *ANR S* ⟹ *ANR*(*S* − {*a*})
by (*blast intro*: *ANR_openin openin_delete openin_subtopology_self*)

lemma *ENR_rel_interior*:
fixes *S* :: *'a*::*euclidean_space set*
shows *ENR S* ⟹ *ENR*(*rel_interior S*)
by (*blast intro*: *ENR_openin openin_set_rel_interior*)

lemma *ENR_delete*:
fixes *S* :: *'a*::*euclidean_space set*
shows *ENR S* ⟹ *ENR*(*S* − {*a*})
by (*blast intro*: *ENR_openin openin_delete openin_subtopology_self*)

lemma *open_imp_ENR*: *open S* ⟹ *ENR S*
using *ENR_def* **by** *blast*

lemma *open_imp_ANR*:
fixes *S* :: *'a*::*euclidean_space set*
shows *open S* ⟹ *ANR S*
by (*simp add*: *ENR_imp_ANR open_imp_ENR*)

lemma *ANR_ball* [*iff*]:
fixes *a* :: *'a*::*euclidean_space*
shows *ANR*(*ball a r*)
by (*simp add*: *convex_imp_ANR*)

lemma *ENR_ball* [*iff*]: *ENR*(*ball a r*)
by (*simp add*: *open_imp_ENR*)

lemma *AR_ball* [*simp*]:
fixes *a* :: *'a*::*euclidean_space*
shows *AR*(*ball a r*) ⟷ *0* < *r*
by (*auto simp*: *AR_ANR convex_imp_contractible*)

lemma *ANR_cball* [*iff*]:
fixes *a* :: *'a*::*euclidean_space*
shows *ANR*(*cball a r*)
by (*simp add*: *convex_imp_ANR*)

lemma *ENR_cball*:
  fixes *a* :: *'a*::*euclidean_space*
  shows *ENR*(*cball a r*)
 using *ENR_convex_closed* **by** *blast*

lemma *AR_cball* [*simp*]:
  fixes *a* :: *'a*::*euclidean_space*
  shows *AR*(*cball a r*) ⟷ *0* ≤ *r*
 by (*auto simp*: *AR_ANR convex_imp_contractible*)

lemma *ANR_box* [*iff*]:
  fixes *a* :: *'a*::*euclidean_space*
  shows *ANR*(*cbox a b*) *ANR*(*box a b*)
 by (*auto simp*: *convex_imp_ANR open_imp_ANR*)

lemma *ENR_box* [*iff*]:
  fixes *a* :: *'a*::*euclidean_space*
  shows *ENR*(*cbox a b*) *ENR*(*box a b*)
 by (*simp_all add*: *ENR_convex_closed closed_cbox open_box open_imp_ENR*)

lemma *AR_box* [*simp*]:
  AR(*cbox a b*) ⟷ *cbox a b* ≠ {} *AR*(*box a b*) ⟷ *box a b* ≠ {}
 by (*auto simp*: *AR_ANR convex_imp_contractible*)

lemma *ANR_interior*:
  fixes *S* :: *'a*::*euclidean_space set*
  shows *ANR*(*interior S*)
 by (*simp add*: *open_imp_ANR*)

lemma *ENR_interior*:
  fixes *S* :: *'a*::*euclidean_space set*
  shows *ENR*(*interior S*)
 by (*simp add*: *open_imp_ENR*)

lemma *AR_imp_contractible*:
  fixes *S* :: *'a*::*euclidean_space set*
  shows *AR S* ⟹ *contractible S*
 by (*simp add*: *AR_ANR*)

lemma *ENR_imp_locally_compact*:
  fixes *S* :: *'a*::*euclidean_space set*
  shows *ENR S* ⟹ *locally compact S*
 by (*simp add*: *ENR_ANR*)

lemma *ANR_imp_locally_path_connected*:
 fixes *S* :: *'a*::*euclidean_space set*
 assumes *ANR S*
  shows *locally path_connected S*

proof −
obtain *U* **and** *T* :: (*'a* × *real*) *set*
where *convex U U* ≠ {}
and *UT*: *closedin* (*top_of_set U*) *T* **and** *S homeomorphic T*
proof (*rule homeomorphic_closedin_convex*)
show *aff_dim S* < *int DIM*(*'a* × *real*)
using *aff_dim_le_DIM* [*of S*] **by** *auto*
qed *auto*
then have *locally path_connected T*
by (*meson ANR_imp_absolute_neighbourhood_retract*
assms convex_imp_locally_path_connected locally_open_subset retract_of_locally_path_connected)
then have *S*: *locally path_connected S*
if *openin* (*top_of_set U*) *V T retract_of V U* ≠ {} **for** *V*
using ‹*S homeomorphic T*› *homeomorphic_locally homeomorphic_path_connectedness*
by *blast*
obtain *Ta* **where** (*openin* (*top_of_set U*) *Ta* ∧ *T retract_of Ta*)
using *ANR_def UT* ‹*S homeomorphic T*› *assms* **by** *moura*
then show *?thesis*
using *S* ‹*U* ≠ {}› **by** *blast*
qed

lemma *ANR_imp_locally_connected*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *ANR S*
shows *locally connected S*
using *locally_path_connected_imp_locally_connected ANR_imp_locally_path_connected*
assms **by** *auto*

lemma *AR_imp_locally_path_connected*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *AR S*
shows *locally path_connected S*
by (*simp add*: *ANR_imp_locally_path_connected AR_imp_ANR assms*)

lemma *AR_imp_locally_connected*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *AR S*
shows *locally connected S*
using *ANR_imp_locally_connected AR_ANR assms* **by** *blast*

lemma *ENR_imp_locally_path_connected*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *ENR S*
shows *locally path_connected S*
by (*simp add*: *ANR_imp_locally_path_connected ENR_imp_ANR assms*)

lemma *ENR_imp_locally_connected*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *ENR S*

```
    shows locally connected S
using ANR_imp_locally_connected ENR_ANR assms by blast

lemma ANR_Times:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes ANR S ANR T shows ANR(S × T)
proof (clarsimp simp only: ANR_eq_absolute_neighbourhood_extensor)
  fix f :: ('a × 'b) × real ⇒ 'a × 'b and U C
  assume continuous_on C f and fim: f ∈ C → S × T
     and cloUC: closedin (top_of_set U) C
  have contf1: continuous_on C (fst ∘ f)
    by (simp add: ‹continuous_on C f› continuous_on_fst)
  obtain W1 g where C ⊆ W1
              and UW1: openin (top_of_set U) W1
              and contg: continuous_on W1 g
              and gim: g ‘ W1 ⊆ S
              and geq: ⋀x. x ∈ C ⟹ g x = (fst ∘ f) x
  proof (rule ANR_imp_absolute_neighbourhood_extensor [OF ‹ANR S› contf1
_ cloUC])
    show (fst ∘ f) ∈ C → S
      using fim by force
  qed auto
  have contf2: continuous_on C (snd ∘ f)
    by (simp add: ‹continuous_on C f› continuous_on_snd)
  obtain W2 h where C ⊆ W2
              and UW2: openin (top_of_set U) W2
              and conth: continuous_on W2 h
              and him: h ∈ W2 → T
              and heq: ⋀x. x ∈ C ⟹ h x = (snd ∘ f) x
  proof (rule ANR_imp_absolute_neighbourhood_extensor [OF ‹ANR T› contf2
_ cloUC])
    show (snd ∘ f) ∈ C → T
      using fim by force
  qed auto
  show ∃ V g. C ⊆ V ∧
              openin (top_of_set U) V ∧
              continuous_on V g ∧ g ∈ V → S × T ∧ (∀ x∈C. g x = f x)
  proof (intro exI conjI)
    show C ⊆ W1 ∩ W2
      by (simp add: ‹C ⊆ W1› ‹C ⊆ W2›)
    show openin (top_of_set U) (W1 ∩ W2)
      by (simp add: UW1 UW2 openin_Int)
    show  continuous_on (W1 ∩ W2) (λx. (g x, h x))
     by (metis (no_types) contg conth continuous_on_Pair continuous_on_subset
inf_commute inf_le1)
    show  (λx. (g x, h x)) ∈ (W1 ∩ W2) → S × T
      using gim him by blast
    show  (∀ x∈C. (g x, h x) = f x)
      using geq heq by auto
```

```
  qed
qed

lemma AR_Times:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes AR S AR T shows AR(S × T)
  using assms by (simp add: AR_ANR ANR_Times contractible_Times)
```

6.35.2 More advanced properties of ANRs and ENRs

```
lemma ENR_rel_frontier_convex:
  fixes S :: 'a::euclidean_space set
  assumes bounded S convex S
    shows ENR(rel_frontier S)
proof (cases S = {})
  case True then show ?thesis
    by simp
next
  case False
  with assms have rel_interior S ≠ {}
    by (simp add: rel_interior_eq_empty)
  then obtain a where a: a ∈ rel_interior S
    by auto
  have ahS: affine hull S − {a} ⊆ {x. closest_point (affine hull S) x ≠ a}
    by (auto simp: closest_point_self)
  have rel_frontier S retract_of affine hull S − {a}
    by (simp add: assms a rel_frontier_retract_of_punctured_affine_hull)
  also have ... retract_of {x. closest_point (affine hull S) x ≠ a}
    unfolding retract_of_def retraction_def ahS
    apply (rule_tac x=closest_point (affine hull S) in exI)
   apply (auto simp: False closest_point_self affine_imp_convex closest_point_in_set
continuous_on_closest_point)
    done
  finally have rel_frontier S retract_of {x. closest_point (affine hull S) x ≠ a} .
  moreover have openin (top_of_set UNIV) (UNIV ∩ closest_point (affine hull
S) -` (- {a}))
   by (intro continuous_openin_preimage_gen) (auto simp: False affine_imp_convex
continuous_on_closest_point)
  ultimately show ?thesis
   by (meson ENR_convex_closed ENR_delete ENR_retract_of_ENR ‹rel_frontier
S retract_of affine hull S − {a}›
            closed_affine_hull convex_affine_hull)
qed

lemma ANR_rel_frontier_convex:
                fixes S :: 'a::euclidean_space set
  assumes bounded S convex S
    shows ANR(rel_frontier S)
by (simp add: ENR_imp_ANR ENR_rel_frontier_convex assms)
```

lemma *ENR_closedin_Un_local*:
fixes $S :: 'a::euclidean\_space\ set$
shows $[\![ENR\ S;\ ENR\ T;\ ENR(S \cap T);$
$closedin\ (top\_of\_set\ (S \cup T))\ S;\ closedin\ (top\_of\_set\ (S \cup T))\ T]\!] \Longrightarrow ENR(S \cup T)$
by (*simp add*: *ENR_ANR ANR_closed_Un_local locally_compact_closedin_Un*)

lemma *ENR_closed_Un*:
fixes $S :: 'a::euclidean\_space\ set$
shows $[\![closed\ S;\ closed\ T;\ ENR\ S;\ ENR\ T;\ ENR(S \cap T)]\!] \Longrightarrow ENR(S \cup T)$
by (*auto simp*: *closed_subset ENR_closedin_Un_local*)

lemma *absolute_retract_Un*:
fixes $S :: 'a::euclidean\_space\ set$
shows $[\![S\ retract\_of\ UNIV;\ T\ retract\_of\ UNIV;\ (S \cap T)\ retract\_of\ UNIV]\!]$
$\Longrightarrow (S \cup T)\ retract\_of\ UNIV$
by (*meson AR_closed_Un_local_aux closed_subset retract_of_UNIV retract_of_imp_subset*)

lemma *retract_from_Un_Int*:
fixes $S :: 'a::euclidean\_space\ set$
assumes *clS*: $closedin\ (top\_of\_set\ (S \cup T))\ S$
and *clT*: $closedin\ (top\_of\_set\ (S \cup T))\ T$
and *Un*: $(S \cup T)\ retract\_of\ U$ **and** *Int*: $(S \cap T)\ retract\_of\ T$
shows $S\ retract\_of\ U$
proof −
obtain r **where** r: $continuous\_on\ T\ r\ r\ `\ T \subseteq S \cap T\ \forall x{\in}S \cap T.\ r\ x = x$
using *Int* **by** (*auto simp*: *retraction_def retract_of_def*)
have $S\ retract\_of\ S \cup T$
unfolding *retraction_def retract_of_def*
proof (*intro exI conjI*)
show $continuous\_on\ (S \cup T)\ (\lambda x.\ if\ x \in S\ then\ x\ else\ r\ x)$
using r **by** (*intro continuous_on_cases_local* [*OF clS clT*]) *auto*
qed (*use r* **in** *auto*)
also have ... *retract_of U*
by (*rule Un*)
finally show *?thesis* **.**
qed

lemma *AR_from_Un_Int_local*:
fixes $S :: 'a::euclidean\_space\ set$
assumes *clS*: $closedin\ (top\_of\_set\ (S \cup T))\ S$
and *clT*: $closedin\ (top\_of\_set\ (S \cup T))\ T$
and *Un*: $AR(S \cup T)$ **and** *Int*: $AR(S \cap T)$
shows $AR\ S$
by (*meson AR_imp_retract AR_retract_of_AR Un assms closedin_closed_subset local.Int*
retract_from_Un_Int retract_of_refl sup_ge2)

lemma *AR_from_Un_Int_local'*:
fixes S :: *'a::euclidean_space set*
assumes *closedin* (*top_of_set* ($S \cup T$)) S
and *closedin* (*top_of_set* ($S \cup T$)) T
and $AR(S \cup T)$ $AR(S \cap T)$
shows $AR\ T$
using *AR_from_Un_Int_local* [*of* T S] *assms* **by** (*simp add*: *Un_commute Int_commute*)

lemma *AR_from_Un_Int*:
fixes S :: *'a::euclidean_space set*
assumes *clo*: *closed* S *closed* T **and** *Un*: $AR(S \cup T)$ **and** *Int*: $AR(S \cap T)$
shows $AR\ S$
by (*metis AR_from_Un_Int_local* [*OF _ _ Un Int*] *Un_commute clo closed_closedin closedin_closed_subset inf_sup_absorb subtopology_UNIV top_greatest*)

lemma *ANR_from_Un_Int_local*:
fixes S :: *'a::euclidean_space set*
assumes *clS*: *closedin* (*top_of_set* ($S \cup T$)) S
and *clT*: *closedin* (*top_of_set* ($S \cup T$)) T
and *Un*: $ANR(S \cup T)$ **and** *Int*: $ANR(S \cap T)$
shows $ANR\ S$
proof −
obtain V **where** *clo*: *closedin* (*top_of_set* ($S \cup T$)) ($S \cap T$)
and *ope*: *openin* (*top_of_set* ($S \cup T$)) V
and *ret*: $S \cap T$ *retract_of* V
using *ANR_imp_neighbourhood_retract* [*OF Int*] **by** (*metis clS clT closedin_Int*)
then obtain r **where** r: *continuous_on* V r **and** *rim*: r ' $V \subseteq S \cap T$ **and** *req*: $\forall x{\in}S \cap T.\ r\ x = x$
by (*auto simp*: *retraction_def retract_of_def*)
have *Vsub*: $V \subseteq S \cup T$
by (*meson ope openin_contains_cball*)
have *Vsup*: $S \cap T \subseteq V$
by (*simp add*: *retract_of_imp_subset ret*)
then have *eq*: $S \cup V = ((S \cup T) - T) \cup V$
by *auto*
have *eq'*: $S \cup V = S \cup (V \cap T)$
using *Vsub* **by** *blast*
have *continuous_on* ($S \cup V \cap T$) ($\lambda x.$ *if* $x \in S$ *then* x *else* $r\ x$)
proof (*rule continuous_on_cases_local*)
show *closedin* (*top_of_set* ($S \cup V \cap T$)) S
using *clS closedin_subset_trans inf.boundedE* **by** *blast*
show *closedin* (*top_of_set* ($S \cup V \cap T$)) ($V \cap T$)
using *clT Vsup* **by** (*auto simp*: *closedin_closed*)
show *continuous_on* ($V \cap T$) r
by (*meson Int_lower1 continuous_on_subset r*)
qed (*use req continuous_on_id* **in** *auto*)
with *rim* **have** S *retract_of* $S \cup V$
unfolding *retraction_def retract_of_def* **using** *eq'* **by** *fastforce*

```
  then show ?thesis
    using ANR_neighborhood_retract [OF Un]
    using ‹S ∪ V = S ∪ T − T ∪ V› clT ope by fastforce
qed

lemma ANR_from_Un_Int:
  fixes S :: 'a::euclidean_space set
  assumes clo: closed S closed T and Un: ANR(S ∪ T) and Int: ANR(S ∩ T)
  shows ANR S
 by (metis ANR_from_Un_Int_local [OF _ _ Un Int] Un_commute clo closed_closedin
closedin_closed_subset inf_sup_absorb subtopology_UNIV top_greatest)

lemma ANR_finite_Union_convex_closed:
  fixes T :: 'a::euclidean_space set set
  assumes T: finite T and clo: ⋀C. C ∈ T ⟹ closed C and con: ⋀C. C ∈ T
⟹ convex C
  shows ANR(⋃T)
proof −
  have ANR(⋃T) if card T < n for n
  using assms that
  proof (induction n arbitrary: T)
    case 0 then show ?case by simp
  next
    case (Suc n)
    have ANR(⋃U) if finite U U ⊆ T for U
      using that
    proof (induction U)
      case empty
      then show ?case  by simp
    next
      case (insert C U)
      have ANR (C ∪ ⋃U)
      proof (rule ANR_closed_Un)
        show ANR (C ∩ ⋃U)
          unfolding Int_Union
        proof (rule Suc)
          show finite ((∩) C ‘ U)
            by (simp add: insert.hyps(1))
          show ⋀Ca. Ca ∈ (∩) C ‘ U ⟹ closed Ca
             by (metis (no_types, opaque_lifting) Suc.prems(2) closed_Int subsetD
imageE insert.prems insertI1 insertI2)
          show ⋀Ca. Ca ∈ (∩) C ‘ U ⟹ convex Ca
              by (metis (mono_tags, lifting) Suc.prems(3) convex_Int imageE in-
sert.prems insert_subset subsetCE)
          show card ((∩) C ‘ U) < n
          proof −
            have card T ≤ n
              by (meson Suc.prems(4) not_less not_less_eq)
            then show ?thesis
```

by (*metis Suc.prems*(*1*) *card_image_le card_seteq insert.hyps insert.prems insert_subset le_trans not_less*)
qed
qed
show *closed* (⋃ *U*)
using *Suc.prems*(*2*) *insert.hyps*(*1*) *insert.prems* **by** *blast*
qed (*use Suc.prems convex_imp_ANR insert.prems insert.IH* **in** *auto*)
then show *?case*
by *simp*
qed
then show *?case*
using *Suc.prems*(*1*) **by** *blast*
qed
then show *?thesis*
by *blast*
qed

lemma *finite_imp_ANR*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *finite S*
shows *ANR S*
proof −
have *ANR*(⋃ *x* ∈ *S*. {*x*})
by (*blast intro*: *ANR_finite_Union_convex_closed assms*)
then show *?thesis*
by *simp*
qed

lemma *ANR_insert*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *ANR S closed S*
shows *ANR*(*insert a S*)
by (*metis ANR_closed_Un ANR_empty ANR_singleton Diff_disjoint Diff_insert_absorb assms closed_singleton insert_absorb insert_is_Un*)

lemma *ANR_path_component_ANR*:
fixes *S* :: *'a*::*euclidean_space set*
shows *ANR S* ⟹ *ANR*(*path_component_set S x*)
using *ANR_imp_locally_path_connected ANR_openin openin_path_component_locally_path_connected*
by *blast*

lemma *ANR_connected_component_ANR*:
fixes *S* :: *'a*::*euclidean_space set*
shows *ANR S* ⟹ *ANR*(*connected_component_set S x*)
by (*metis ANR_openin openin_connected_component_locally_connected ANR_imp_locally_connected*)

lemma *ANR_component_ANR*:
fixes *S* :: *'a*::*euclidean_space set*

assumes *ANR S c* ∈ *components S*
shows *ANR c*
by (*metis ANR_connected_component_ANR assms componentsE*)

6.35.3 Original ANR material, now for ENRs

lemma *ENR_bounded*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *bounded S*
shows *ENR S* ⟷ (∃ *U*. *open U* ∧ *bounded U* ∧ *S retract_of U*)
(**is** *?lhs* = *?rhs*)
proof
obtain *r* **where** *0* < *r* **and** *r*: *S* ⊆ *ball 0 r*
using *bounded_subset_ballD assms* **by** *blast*
assume *?lhs*
then show *?rhs*
by (*meson ENR_def Elementary_Metric_Spaces.open_ball bounded_Int bounded_ball inf_le2 le_inf_iff*
open_Int r retract_of_imp_subset retract_of_subset)
next
assume *?rhs*
then show *?lhs*
using *ENR_def* **by** *blast*
qed

lemma *absolute_retract_imp_AR_gen*:
fixes *S* :: *'a*::*euclidean_space set* **and** *S'* :: *'b*::*euclidean_space set*
assumes *S retract_of T convex T T* ≠ {} *S homeomorphic S' closedin* (*top_of_set U*) *S'*
shows *S' retract_of U*
proof −
have *AR T*
by (*simp add*: *assms convex_imp_AR*)
then have *AR S*
using *AR_retract_of_AR assms* **by** *auto*
then show *?thesis*
using *assms AR_imp_absolute_retract* **by** *metis*
qed

lemma *absolute_retract_imp_AR*:
fixes *S* :: *'a*::*euclidean_space set* **and** *S'* :: *'b*::*euclidean_space set*
assumes *S retract_of UNIV S homeomorphic S' closed S'*
shows *S' retract_of UNIV*
using *AR_imp_absolute_retract_UNIV assms retract_of_UNIV* **by** *blast*

lemma *homeomorphic_compact_arness*:
fixes *S* :: *'a*::*euclidean_space set* **and** *S'* :: *'b*::*euclidean_space set*
assumes *S homeomorphic S'*
shows *compact S* ∧ *S retract_of UNIV* ⟷ *compact S'* ∧ *S' retract_of UNIV*

using *assms homeomorphic_compactness*
by (*metis compact_AR homeomorphic_AR_iff_AR*)

lemma *absolute_retract_from_Un_Int*:
fixes S :: *'a::euclidean_space set*
assumes $(S \cup T)$ *retract_of UNIV* $(S \cap T)$ *retract_of UNIV closed S closed T*
shows S *retract_of UNIV*
using *AR_from_Un_Int assms retract_of_UNIV* **by** *auto*

lemma *ENR_from_Un_Int_gen*:
fixes S :: *'a::euclidean_space set*
assumes *closedin* (*top_of_set* $(S \cup T)$) S *closedin* (*top_of_set* $(S \cup T)$) T
ENR$(S \cup T)$ *ENR*$(S \cap T)$
shows *ENR S*
by (*meson ANR_from_Un_Int_local ANR_imp_neighbourhood_retract ENR_ANR*
ENR_neighborhood_retract assms)

lemma *ENR_from_Un_Int*:
fixes S :: *'a::euclidean_space set*
assumes *closed S closed T ENR*$(S \cup T)$ *ENR*$(S \cap T)$
shows *ENR S*
by (*meson ENR_from_Un_Int_gen assms closed_subset sup_ge1 sup_ge2*)

lemma *ENR_finite_Union_convex_closed*:
fixes $\mathcal{T}$ :: *'a::euclidean_space set set*
assumes $\mathcal{T}$: *finite* $\mathcal{T}$ **and** *clo*: $\bigwedge C.\ C \in \mathcal{T} \Longrightarrow$ *closed* C **and** *con*: $\bigwedge C.\ C \in \mathcal{T}$
$\Longrightarrow$ *convex* C
shows *ENR*$(\bigcup\ \mathcal{T})$
by (*simp add*: *ENR_ANR ANR_finite_Union_convex_closed* $\mathcal{T}$ *clo closed_Union*
closed_imp_locally_compact con)

lemma *finite_imp_ENR*:
fixes S :: *'a::euclidean_space set*
shows *finite* $S \Longrightarrow$ *ENR S*
by (*simp add*: *ENR_ANR finite_imp_ANR finite_imp_closed closed_imp_locally_compact*)

lemma *ENR_insert*:
fixes S :: *'a::euclidean_space set*
assumes *closed S ENR S*
shows *ENR*(*insert a S*)
proof −
have *ENR* $(\{a\} \cup S)$
by (*metis ANR_insert ENR_ANR Un_commute Un_insert_right assms closed_imp_locally_compact*
closed_insert sup_bot_right)
then show *?thesis*
by *auto*
qed

lemma *ENR_path_component_ENR*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *ENR S*
shows *ENR*(*path_component_set S x*)
by (*metis ANR_imp_locally_path_connected ENR_empty ENR_imp_ANR ENR_openin assms*
locally_path_connected_2 openin_subtopology_self path_component_eq_empty)

6.35.4 Finally, spheres are ANRs and ENRs

lemma *absolute_retract_homeomorphic_convex_compact*:
fixes *S* :: *'a*::*euclidean_space set* **and** *U* :: *'b*::*euclidean_space set*
assumes *S homeomorphic U S* ≠ {} *S* ⊆ *T convex U compact U*
shows *S retract_of T*
by (*metis UNIV_I assms compact_AR convex_imp_AR homeomorphic_AR_iff_AR homeomorphic_compactness homeomorphic_empty*(*1*) *retract_of_subset subsetI*)

lemma *frontier_retract_of_punctured_universe*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *convex S bounded S a* ∈ *interior S*
shows (*frontier S*) *retract_of* (− {*a*})
using *rel_frontier_retract_of_punctured_affine_hull*
by (*metis Compl_eq_Diff_UNIV affine_hull_nonempty_interior assms empty_iff rel_frontier_frontier rel_interior_nonempty_interior*)

lemma *sphere_retract_of_punctured_universe_gen*:
fixes *a* :: *'a*::*euclidean_space*
assumes *b* ∈ *ball a r*
shows *sphere a r retract_of* (− {*b*})
proof −
have *frontier* (*cball a r*) *retract_of* (− {*b*})
using *assms frontier_retract_of_punctured_universe interior_cball* **by** *blast*
then show *?thesis*
by *simp*
qed

lemma *sphere_retract_of_punctured_universe*:
fixes *a* :: *'a*::*euclidean_space*
assumes *0* < *r*
shows *sphere a r retract_of* (− {*a*})
by (*simp add*: *assms sphere_retract_of_punctured_universe_gen*)

lemma *ENR_sphere*:
fixes *a* :: *'a*::*euclidean_space*
shows *ENR*(*sphere a r*)
proof (*cases 0* < *r*)
case *True*
then have *sphere a r retract_of* −{*a*}

by (*simp add*: *sphere_retract_of_punctured_universe*)
with *open_delete* **show** *?thesis*
by (*auto simp*: *ENR_def*)
next
case *False*
then show *?thesis*
using *finite_imp_ENR*
by (*metis finite_insert infinite_imp_nonempty less_linear sphere_eq_empty sphere_trivial*)
qed

corollary *ANR_sphere*:
fixes *a* :: *'a*::*euclidean_space*
shows *ANR*(*sphere a r*)
by (*simp add*: *ENR_imp_ANR ENR_sphere*)

6.35.5 Spheres are connected, etc

lemma *locally_path_connected_sphere_gen*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *bounded S* **and** *convex S*
shows *locally path_connected* (*rel_frontier S*)
proof (*cases rel_interior S* = {})
case *True*
with *assms* **show** *?thesis*
by (*simp add*: *rel_interior_eq_empty*)
next
case *False*
then obtain *a* **where** *a*: *a* ∈ *rel_interior S*
by *blast*
show *?thesis*
proof (*rule retract_of_locally_path_connected*)
show *locally path_connected* (*affine hull S* − {*a*})
by (*meson convex_affine_hull convex_imp_locally_path_connected locally_open_subset openin_delete openin_subtopology_self*)
show *rel_frontier S retract_of affine hull S* − {*a*}
using *a assms rel_frontier_retract_of_punctured_affine_hull* **by** *blast*
qed
qed

lemma *locally_connected_sphere_gen*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *bounded S* **and** *convex S*
shows *locally connected* (*rel_frontier S*)
by (*simp add*: *ANR_imp_locally_connected ANR_rel_frontier_convex assms*)

lemma *locally_path_connected_sphere*:
fixes *a* :: *'a*::*euclidean_space*
shows *locally path_connected* (*sphere a r*)

using *ENR_imp_locally_path_connected ENR_sphere* **by** *blast*

lemma *locally_connected_sphere*:
fixes *a* :: *'a*::*euclidean_space*
shows *locally connected*(*sphere a r*)
using *ANR_imp_locally_connected ANR_sphere* **by** *blast*

6.35.6 Borsuk homotopy extension theorem

It's only this late so we can use the concept of retraction, saying that the domain sets or range set are ENRs.

theorem *Borsuk_homotopy_extension_homotopic*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *cloTS*: *closedin* (*top_of_set T*) *S*
and *anr*: (*ANR S* ∧ *ANR T*) ∨ *ANR U*
and *contf*: *continuous_on T f*
and *f* ∈ *T* → *U*
and *homotopic_with_canon* (λ*x*. *True*) *S U f g*
obtains *g'* **where** *homotopic_with_canon* (λ*x*. *True*) *T U f g'*
continuous_on T g' image g' T ⊆ *U*
⋀*x*. *x* ∈ *S* ⟹ *g' x* = *g x*
proof −
have *S* ⊆ *T* **using** *assms closedin_imp_subset* **by** *blast*
obtain *h* **where** *conth*: *continuous_on* ({*0..1*} × *S*) *h*
and *him*: *h* ∈ ({*0..1*} × *S*) → *U*
and [*simp*]: ⋀*x*. *h*(*0*, *x*) = *f x* ⋀*x*. *h*(*1*::*real*, *x*) = *g x*
using *assms* **by** (*fastforce simp*: *homotopic_with_def*)
define *h'* **where** *h'* ≡ λ*z*. *if snd z* ∈ *S then h z else* (*f* ∘ *snd*) *z*
define *B* **where** *B* ≡ {*0*::*real*} × *T* ∪ {*0..1*} × *S*
have *clo0T*: *closedin* (*top_of_set* ({*0..1*} × *T*)) ({*0*::*real*} × *T*)
by (*simp add*: *Abstract_Topology.closedin_Times*)
moreover have *cloT1S*: *closedin* (*top_of_set* ({*0..1*} × *T*)) ({*0..1*} × *S*)
by (*simp add*: *Abstract_Topology.closedin_Times assms*)
ultimately have *clo0TB*:*closedin* (*top_of_set* ({*0..1*} × *T*)) *B*
by (*auto simp*: *B_def*)
have *cloBS*: *closedin* (*top_of_set B*) ({*0..1*} × *S*)
by (*metis* (*no_types*) *Un_subset_iff B_def closedin_subset_trans* [*OF cloT1S*] *clo0TB closedin_imp_subset closedin_self*)
moreover have *cloBT*: *closedin* (*top_of_set B*) ({*0*} × *T*)
using ‹*S* ⊆ *T*› *closedin_subset_trans* [*OF clo0T*]
by (*metis B_def Un_upper1 clo0TB closedin_closed inf_le1*)
moreover have *continuous_on* ({*0*} × *T*) (*f* ∘ *snd*)
proof (*rule continuous_intros*)+
show *continuous_on* (*snd* ' ({*0*} × *T*)) *f*
by (*simp add*: *contf*)
qed
ultimately have *continuous_on* ({*0..1*} × *S* ∪ {*0*} × *T*) (λ*x*. *if snd x* ∈ *S then h x else* (*f* ∘ *snd*) *x*)
by (*auto intro*!: *continuous_on_cases_local conth simp*: *B_def Un_commute*

```
[of {0} × T])
  then have conth': continuous_on B h'
    by (simp add: h'_def B_def Un_commute [of {0} × T])
  have image h' B ⊆ U
    using ‹f ∈ T → U› him by (auto simp: h'_def B_def)
  obtain V k where B ⊆ V and opeTV: openin (top_of_set ({0..1} × T)) V
              and contk: continuous_on V k and kim: k ∈ V → U
              and keq: ⋀x. x ∈ B ⟹ k x = h' x
  using anr
  proof
    assume ST: ANR S ∧ ANR T
    have eq: ({0} × T ∩ {0..1} × S) = {0::real} × S
      using ‹S ⊆ T› by auto
    have ANR B
      unfolding B_def
    proof (rule ANR_closed_Un_local)
      show closedin (top_of_set ({0} × T ∪ {0..1} × S)) ({0::real} × T)
        by (metis cloBT B_def)
      show closedin (top_of_set ({0} × T ∪ {0..1} × S)) ({0..1::real} × S)
        by (metis Un_commute cloBS B_def)
    qed (simp_all add: ANR_Times convex_imp_ANR ANR_singleton ST eq)
    note Vk = that
    have ∗: thesis if openin (top_of_set ({0..1::real} × T)) V
                      retraction V B r for V r
    proof −
      have continuous_on V (h' ∘ r)
        using conth' continuous_on_compose retractionE that(2) by blast
      moreover have (h' ∘ r) ‘ V ⊆ U
        by (metis ‹h' ‘ B ⊆ U› image_comp retractionE that(2))
      ultimately show ?thesis
      using Vk [of V h' ∘ r] by (metis comp_apply retraction image_subset_iff_funcset
that)
    qed
    show thesis
    by (meson ∗ ANR_imp_neighbourhood_retract ‹ANR B› clo0TB retract_of_def)
  next
    assume ANR U
   with ANR_imp_absolute_neighbourhood_extensor ‹h' ‘ B ⊆ U› clo0TB conth'
image_subset_iff_funcset that
    show ?thesis
      by (smt (verit) Pi_I funcset_mem)
  qed
  define S' where S' ≡ {x. ∃u::real. u ∈ {0..1} ∧ (u, x::'a) ∈ {0..1} × T − V}
  have closedin (top_of_set T) S'
    unfolding S'_def using closedin_self opeTV
    by (blast intro: closedin_compact_projection)
  have S'_def: S' = {x. ∃u::real.  (u, x::'a) ∈ {0..1} × T − V}
    by (auto simp: S'_def)
  have cloTS': closedin (top_of_set T) S'
```

```
    using S'_def ‹closedin (top_of_set T) S'› by blast
  have S ∩ S' = {}
    using S'_def B_def ‹B ⊆ V› by force
  obtain a :: 'a ⇒ real where conta: continuous_on T a
      and ⋀x. x ∈ T ⟹ a x ∈ closed_segment 1 0
      and a1: ⋀x. x ∈ S ⟹ a x = 1
      and a0: ⋀x. x ∈ S' ⟹ a x = 0
    by (rule Urysohn_local [OF cloTS cloTS' ‹S ∩ S' = {}›, of 1 0], blast)
  then have ain: ⋀x. x ∈ T ⟹ a x ∈ {0..1}
    using closed_segment_eq_real_ivl by auto
  have inV: (u * a t, t) ∈ V if t ∈ T 0 ≤ u u ≤ 1 for t u
  proof (rule ccontr)
    assume (u * a t, t) ∉ V
    with ain [OF ‹t ∈ T›] have a t = 0
      apply simp
       by (metis (no_types, lifting) a0 DiffI S'_def SigmaI atLeastAtMost_iff
mem_Collect_eq mult_le_one mult_nonneg_nonneg that)
    show False
      using B_def ‹(u * a t, t) ∉ V› ‹B ⊆ V› ‹a t = 0› that by auto
  qed
  show ?thesis
  proof
    show hom: homotopic_with_canon (λx. True) T U f (λx. k (a x, x))
    proof (simp add: homotopic_with, intro exI conjI)
      show continuous_on ({0..1} × T) (k ∘ (λz. (fst z *R (a ∘ snd) z, snd z)))
        apply (intro continuous_on_compose continuous_intros)
      apply (force intro: inV continuous_on_subset [OF contk] continuous_on_subset
[OF conta])+
        done
      show (k ∘ (λz. (fst z *R (a ∘ snd) z, snd z))) ‘ ({0..1} × T) ⊆ U
        using inV kim by auto
      show ∀x∈T. (k ∘ (λz. (fst z *R (a ∘ snd) z, snd z))) (0, x) = f x
        by (simp add: B_def h'_def keq)
      show ∀x∈T. (k ∘ (λz. (fst z *R (a ∘ snd) z, snd z))) (1, x) = k (a x, x)
        by auto
    qed
  show continuous_on T (λx. k (a x, x))
    using homotopic_with_imp_continuous_maps [OF hom] by auto
  show (λx. k (a x, x)) ‘ T ⊆ U
  proof clarify
    fix t
    assume t ∈ T
    show k (a t, t) ∈ U
    by (metis ‹t ∈ T› image_subset_iff inV kim not_one_le_zero linear mult_cancel_right1
image_subset_iff_funcset)
  qed
  show ⋀x. x ∈ S ⟹ k (a x, x) = g x
    by (simp add: B_def a1 h'_def keq)
  qed
```

qed

corollary *nullhomotopic_into_ANR_extension*:
fixes f :: *'a::euclidean_space* ⇒ *'b::euclidean_space*
assumes *closed S*
and *contf*: *continuous_on S f*
and *ANR T*
and *fim*: f ‘ $S \subseteq T$
and $S \neq \{\}$
shows ($\exists$ *c*. *homotopic_with_canon* (λx. *True*) *S T f* (λx. *c*)) ⟷
($\exists$ *g*. *continuous_on UNIV g* ∧ *range g* ⊆ *T* ∧ ($\forall$ *x* ∈ *S*. *g x* = *f x*))
(**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then obtain *c* **where** *c*: *homotopic_with_canon* (λx. *True*) *S T* (λx. *c*) *f*
by (*blast intro*: *homotopic_with_symD*)
have *closedin* (*top_of_set UNIV*) *S*
using ‹*closed S*› *closed_closedin* **by** *fastforce*
then obtain *g* **where** *continuous_on UNIV g range g* ⊆ *T*
⋀*x*. *x* ∈ *S* ⟹ *g x* = *f x*
proof (*rule Borsuk_homotopy_extension_homotopic*)
show (λx. *c*) ∈ *UNIV* → *T*
using ‹*S* ≠ {}› *c homotopic_with_imp_subset1* **by** *fastforce*
qed (*use assms c* **in** *auto*)
then show *?rhs* **by** *blast*
next
assume *?rhs*
then obtain *g* **where** *continuous_on UNIV g range g* ⊆ *T* ⋀*x*. *x*∈*S* ⟹ *g x* = *f x*
by *blast*
then obtain *c* **where** *homotopic_with_canon* (λh. *True*) *UNIV T g* (λx. *c*)
using *nullhomotopic_from_contractible* [*of UNIV g T*] *contractible_UNIV* **by** *blast*
then have *homotopic_with_canon* (λx. *True*) *S T g* (λx. *c*)
by (*simp add*: *homotopic_from_subtopology*)
then show *?lhs*
by (*force elim*: *homotopic_with_eq* [*of* _ _ _ *g* λx. *c*] *simp*: ‹⋀*x*. *x* ∈ *S* ⟹ *g x* = *f x*›)
qed

corollary *nullhomotopic_into_rel_frontier_extension*:
fixes f :: *'a::euclidean_space* ⇒ *'b::euclidean_space*
assumes *closed S*
and *contf*: *continuous_on S f*
and *convex T bounded T*
and *fim*: f ‘ $S \subseteq$ *rel_frontier T*
and $S \neq \{\}$
shows ($\exists$ *c*. *homotopic_with_canon* (λx. *True*) *S* (*rel_frontier T*) *f* (λx. *c*))

```
⟷
        (∃g. continuous_on UNIV g ∧ range g ⊆ rel_frontier T ∧ (∀x∈S. g x
= f x))
by (simp add: nullhomotopic_into_ANR_extension assms ANR_rel_frontier_convex)

corollary nullhomotopic_into_sphere_extension:
  fixes f :: 'a::euclidean_space ⇒ 'b :: euclidean_space
  assumes closed S and contf: continuous_on S f
      and S ≠ {} and fim: f ` S ⊆ sphere a r
    shows ((∃c. homotopic_with_canon (λx. True) S (sphere a r) f (λx. c)) ⟷
        (∃g. continuous_on UNIV g ∧ range g ⊆ sphere a r ∧ (∀x∈S. g x = f
x)))
        (is ?lhs = ?rhs)
proof (cases r = 0)
  case True with fim show ?thesis
   by (metis ANR_sphere ‹closed S› ‹S ≠ {}› contf nullhomotopic_into_ANR_extension)
next
  case False
  then have eq: sphere a r = rel_frontier (cball a r) by simp
  show ?thesis
     using fim nullhomotopic_into_rel_frontier_extension [OF ‹closed S› contf
convex_cball bounded_cball]
    by (simp add: ‹S ≠ {}› eq)
qed

proposition Borsuk_map_essential_bounded_component:
  fixes a :: 'a :: euclidean_space
  assumes compact S and a ∉ S
   shows bounded (connected_component_set (− S) a) ⟷
        ¬(∃c. homotopic_with_canon (λx. True) S (sphere 0 1)
                                   (λx. inverse(norm(x − a)) *R (x − a)) (λx. c))
   (is ?lhs = ?rhs)
proof (cases S = {})
  case True then show ?thesis
    by (simp add: homotopic_on_emptyI)
next
  case False
  have closed S bounded S
    using ‹compact S› compact_eq_bounded_closed by auto
  have s01: (λx. (x − a) /R norm (x − a)) ` S ⊆ sphere 0 1
     using ‹a ∉ S›  by clarsimp (metis dist_eq_0_iff dist_norm mult.commute
right_inverse)
  have aincc: a ∈ connected_component_set (− S) a
    by (simp add: ‹a ∉ S›)
  obtain r where r>0 and r: S ⊆ ball 0 r
    using bounded_subset_ballD ‹bounded S› by blast
  have ¬ ?rhs ⟷ ¬ ?lhs
  proof
    assume notr: ¬ ?rhs
```

```
    have nog: ∄g. continuous_on (S ∪ connected_component_set (− S) a) g ∧
                  g ‘ (S ∪ connected_component_set (− S) a) ⊆ sphere 0 1 ∧
                  (∀x∈S. g x = (x − a) /R norm (x − a))
      if bounded (connected_component_set (− S) a)
      using non_extensible_Borsuk_map [OF ‹compact S› componentsI _ aincc]
‹a ∉ S› that by auto
    obtain g where range g ⊆ sphere 0 1 continuous_on UNIV g
                        ⋀x. x ∈ S ⟹ g x = (x − a) /R norm (x − a)
      using notr
      by (auto simp: nullhomotopic_into_sphere_extension
            [OF ‹closed S› continuous_on_Borsuk_map [OF ‹a ∉ S›] False s01])
    with ‹a ∉ S› show ¬ ?lhs
      by (metis UNIV_I continuous_on_subset image_subset_iff nog subsetI)
  next
    assume ¬ ?lhs
   then obtain b where b: b ∈ connected_component_set (− S) a and r ≤ norm
b
      using bounded_iff linear by blast
    then have bnot: b ∉ ball 0 r
      by simp
    have homotopic_with_canon (λx. True) S (sphere 0 1) (λx. (x − a) /R norm
(x − a))
                                                        (λx. (x − b) /R norm (x − b))
    proof −
      have path_component (− S) a b
      by (metis (full_types) ‹closed S› b mem_Collect_eq open_Compl open_path_connected_component)
      then show ?thesis
        using Borsuk_maps_homotopic_in_path_component by blast
    qed
    moreover
    obtain c where homotopic_with_canon (λx. True) (ball 0 r) (sphere 0 1)
                                        (λx. inverse (norm (x − b)) ∗R (x − b)) (λx. c)
    proof (rule nullhomotopic_from_contractible)
      show contractible (ball (0::'a) r)
        by (metis convex_imp_contractible convex_ball)
      show continuous_on (ball 0 r) (λx. inverse(norm (x − b)) ∗R (x − b))
        by (rule continuous_on_Borsuk_map [OF bnot])
      show (λx. (x − b) /R norm (x − b)) ∈ ball 0 r → sphere 0 1
        using bnot Borsuk_map_into_sphere by blast
    qed blast
    ultimately have homotopic_with_canon (λx. True) S (sphere 0 1) (λx. (x −
a) /R norm (x − a)) (λx. c)
      by (meson homotopic_with_subset_left homotopic_with_trans r)
    then show ¬ ?rhs
      by blast
  qed
  then show ?thesis by blast
qed
```

lemma *homotopic_Borsuk_maps_in_bounded_component*:
fixes $a :: 'a :: euclidean\_space$
assumes *compact S* **and** $a \notin S$**and** $b \notin S$
and *boc*: *bounded* (*connected_component_set* $(-S)$ *a*)
and *hom*: *homotopic_with_canon* $(\lambda x.\ True)$ *S* (*sphere 0 1*)
$(\lambda x.\ (x - a) /_R norm\ (x - a))$
$(\lambda x.\ (x - b) /_R norm\ (x - b))$
shows *connected_component* $(-S)$ *a b*
proof (*rule ccontr*)
assume *notcc*: $\neg$ *connected_component* $(-S)$ *a b*
let *?T* $= S \cup$ *connected_component_set* $(-S)$ *a*
have $\nexists g.$ *continuous_on* $(S \cup$ *connected_component_set* $(-S)$ *a*$)$ *g* $\wedge$
$g \in (S \cup$ *connected_component_set* $(-S)$ *a*$) \to$ *sphere 0 1* $\wedge$
$(\forall x{\in}S.\ g\ x = (x - a) /_R norm\ (x - a))$
using *non_extensible_Borsuk_map* [*OF* ‹*compact S*› _ *boc*] ‹*a* $\notin$ *S*›
by (*simp add*: *componentsI*)
moreover obtain *g* **where** *continuous_on* $(S \cup$ *connected_component_set* $(-S)$ *a*$)$ *g*
g ‘ $(S \cup$ *connected_component_set* $(-S)$ *a*$) \subseteq$ *sphere 0 1*
$\bigwedge x.\ x \in S \Longrightarrow g\ x = (x - a) /_R norm\ (x - a)$
proof (*rule Borsuk_homotopy_extension_homotopic*)
show *closedin* (*top_of_set ?T*) *S*
by (*simp add*: ‹*compact S*› *closed_subset compact_imp_closed*)
show *continuous_on ?T* $(\lambda x.\ (x - b) /_R norm\ (x - b))$
by (*simp add*: ‹*b* $\notin$ *S*› *notcc continuous_on_Borsuk_map*)
show $(\lambda x.\ (x - b) /_R norm\ (x - b)) \in$ *?T* $\to$ *sphere 0 1*
by (*simp add*: ‹*b* $\notin$ *S*› *notcc Borsuk_map_into_sphere*)
show *homotopic_with_canon* $(\lambda x.\ True)$ *S* (*sphere 0 1*)
$(\lambda x.\ (x - b) /_R norm\ (x - b))$ $(\lambda x.\ (x - a) /_R norm\ (x - a))$
by (*simp add*: *hom homotopic_with_symD*)
qed (*auto simp*: *ANR_sphere intro*: *that*)
ultimately show *False* **by** *blast*
qed

lemma *Borsuk_maps_homotopic_in_connected_component_eq*:
fixes $a :: 'a :: euclidean\_space$
assumes *S*: *compact S a* $\notin S$ *b* $\notin S$ **and** *2*: $2 \le DIM('a)$
shows (*homotopic_with_canon* $(\lambda x.\ True)$ *S* (*sphere 0 1*)
$(\lambda x.\ (x - a) /_R norm\ (x - a))$
$(\lambda x.\ (x - b) /_R norm\ (x - b)) \longleftrightarrow$
connected_component $(-S)$ *a b*)
(**is** *?lhs* $=$ *?rhs*)
proof
assume *L*: *?lhs*
show *?rhs*
proof (*cases bounded*(*connected_component_set* $(-S)$ *a*))
case *True*
show *?thesis*

```
    by (rule homotopic_Borsuk_maps_in_bounded_component [OF S True L])
  next
    case not_bo_a: False
    show ?thesis
    proof (cases bounded(connected_component_set (- S) b))
      case True
      show ?thesis
        using homotopic_Borsuk_maps_in_bounded_component [OF S]
      by (simp add: L True assms connected_component_sym homotopic_Borsuk_maps_in_bounded_component
homotopic_with_sym)
    next
      case False
      then show ?thesis
        using cobounded_unique_unbounded_component [of -S a b] ‹compact S›
not_bo_a
      by (auto simp: compact_eq_bounded_closed assms connected_component_eq_eq)
    qed
  qed
next
  assume R: ?rhs
  then have path_component (- S) a b
   using assms(1) compact_eq_bounded_closed open_Compl open_path_connected_component_set
by fastforce
  then show ?lhs
    by (simp add: Borsuk_maps_homotopic_in_path_component)
qed
```

6.35.7 More extension theorems

```
lemma extension_from_clopen:
  assumes ope: openin (top_of_set S) T
      and clo: closedin (top_of_set S) T
      and contf: continuous_on T f and fim: f ‘ T ⊆ U and null: U = {} ⟹ S
= {}
 obtains g where continuous_on S g g ‘ S ⊆ U ⋀x. x ∈ T ⟹ g x = f x
proof (cases U = {})
  case True
  then show ?thesis
    by (simp add: null that)
next
  case False
  then obtain a where a ∈ U
    by auto
  let ?g = λx. if x ∈ T then f x else a
  have Seq: S = T ∪ (S - T)
    using clo closedin_imp_subset by fastforce
  show ?thesis
  proof
    have continuous_on (T ∪ (S - T)) ?g
```

```
    using Seq clo ope  by (intro continuous_on_cases_local) (auto simp: contf)
   with Seq show continuous_on S ?g
    by metis
   show ?g ` S ⊆ U
    using ‹a ∈ U› fim by auto
   show ⋀x. x ∈ T ⟹ ?g x = f x
    by auto
 qed
qed


lemma extension_from_component:
 fixes f :: 'a :: euclidean_space ⇒ 'b :: euclidean_space
 assumes S: locally connected S ∨ compact S and ANR U
   and C: C ∈ components S and contf: continuous_on C f and fim: f ∈ C →
U
 obtains g where continuous_on S g g ∈ S → U ⋀x. x ∈ C ⟹ g x = f x
proof −
 obtain T g where ope: openin (top_of_set S) T
             and clo: closedin (top_of_set S) T
             and C ⊆ T and contg: continuous_on T g and gim: g ∈ T → U
             and gf: ⋀x. x ∈ C ⟹ g x = f x
   using S
 proof
   assume locally connected S
   show ?thesis
     by (metis C ‹locally connected S› openin_components_locally_connected
closedin_component contf fim order_refl that)
 next
   assume compact S
   then obtain W g where C ⊆ W and opeW: openin (top_of_set S) W
              and contg: continuous_on W g
              and gim: g ∈ W → U and gf: ⋀x. x ∈ C ⟹ g x = f x
    using ANR_imp_absolute_neighbourhood_extensor [of U C f S] C ‹ANR U›
closedin_component contf fim by blast
   then obtain V where open V and V: W = S ∩ V
    by (auto simp: openin_open)
   moreover have locally compact S
   by (simp add: ‹compact S› closed_imp_locally_compact compact_imp_closed)
   ultimately obtain K where opeK: openin (top_of_set S) K and compact K
C ⊆ K K ⊆ V
     by (metis C Int_subset_iff ‹C ⊆ W› ‹compact S› compact_components
Sura_Bura_clopen_subset)
   show ?thesis
   proof
     show closedin (top_of_set S) K
     by (meson ‹compact K› ‹compact S› closedin_compact_eq opeK openin_imp_subset)
     show continuous_on K g
       by (metis Int_subset_iff V ‹K ⊆ V› contg continuous_on_subset opeK
```

openin_subtopology subset_eq)
show $g \in K \to U$
using V ‹$K \subseteq V$› *gim opeK openin_imp_subset* **by** *fastforce*
qed (*use opeK gf* ‹$C \subseteq K$› **in** *auto*)
qed
obtain h **where** *continuous_on S h* $h \in S \to U$ $\bigwedge x.\ x \in T \Longrightarrow h\ x = g\ x$
using *extension_from_clopen*
by (*metis C bot.extremum_uniqueI clo contg gim fim image_is_empty in_components_nonempty ope image_subset_iff_funcset*)
then show *?thesis*
by (*metis* ‹$C \subseteq T$› *gf subset_eq that*)
qed

lemma *tube_lemma*:
fixes S :: *'a::euclidean_space set* **and** T :: *'b::euclidean_space set*
assumes *compact S* **and** S: $S \neq \{\}$ $(\lambda x.\ (x,a))$ ‘ $S \subseteq U$
and *ope*: *openin* (*top_of_set* ($S \times T$)) U
obtains V **where** *openin* (*top_of_set T*) V $a \in V$ $S \times V \subseteq U$
proof −
let *?W* = $\{y.\ \exists x.\ x \in S \wedge (x,\ y) \in (S \times T - U)\}$
have $U \subseteq S \times T$ *closedin* (*top_of_set* ($S \times T$)) ($S \times T - U$)
using *ope* **by** (*auto simp*: *openin_closedin_eq*)
then have *closedin* (*top_of_set T*) *?W*
using ‹*compact S*› *closedin_compact_projection* **by** *blast*
moreover have $a \in T - ?W$
using ‹$U \subseteq S \times T$› S **by** *auto*
moreover have $S \times (T - ?W) \subseteq U$
by *auto*
ultimately show *?thesis*
by (*metis* (*no_types*, *lifting*) *Sigma_cong closedin_def that topspace_euclidean_subtopology*)
qed

lemma *tube_lemma_gen*:
fixes S :: *'a::euclidean_space set* **and** T :: *'b::euclidean_space set*
assumes *compact S* $S \neq \{\}$ $T \subseteq T'$ $S \times T \subseteq U$
and *ope*: *openin* (*top_of_set* ($S \times T'$)) U
obtains V **where** *openin* (*top_of_set T'*) V $T \subseteq V$ $S \times V \subseteq U$
proof −
have $\bigwedge x.\ x \in T \Longrightarrow \exists V.$ *openin* (*top_of_set T'*) $V \wedge x \in V \wedge S \times V \subseteq U$
using *assms* **by** (*auto intro*: *tube_lemma* [*OF* ‹*compact S*›])
then obtain F **where** F: $\bigwedge x.\ x \in T \Longrightarrow$ *openin* (*top_of_set T'*) ($F\ x$) $\wedge\ x \in F\ x \wedge S \times F\ x \subseteq U$
by *metis*
show *?thesis*
proof
show *openin* (*top_of_set T'*) ($\bigcup$(F ‘ T))
using F **by** *blast*
show $T \subseteq \bigcup(F$ ‘ $T)$

```
    using F by blast
  show S × ⋃(F ‘ T) ⊆ U
    using F by auto
 qed
qed

proposition homotopic_neighbourhood_extension:
 fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
 assumes contf: continuous_on S f and fim: f ‘ S ⊆ U
    and contg: continuous_on S g and gim: g ‘ S ⊆ U
    and clo: closedin (top_of_set S) T
    and ANR U and hom: homotopic_with_canon (λx. True) T U f g
  obtains V where T ⊆ V openin (top_of_set S) V
                  homotopic_with_canon (λx. True) V U f g
proof −
 have T ⊆ S
   using clo closedin_imp_subset by blast
 obtain h where conth: continuous_on ({0..1::real} × T) h
          and him: h ‘ ({0..1} × T) ⊆ U
          and h0: ⋀x. h(0, x) = f x and h1: ⋀x. h(1, x) = g x
   using hom by (auto simp: homotopic_with_def)
 define h' where h' ≡ λz. if fst z ∈ {0} then f(snd z)
                          else if fst z ∈ {1} then g(snd z)
                          else h z
 let ?S0 = {0::real} × S and ?S1 = {1::real} × S
 have continuous_on(?S0 ∪ (?S1 ∪ {0..1} × T)) h'
   unfolding h'_def
 proof (intro continuous_on_cases_local)
   show closedin (top_of_set (?S0 ∪ (?S1 ∪ {0..1} × T))) ?S0
        closedin (top_of_set (?S1 ∪ {0..1} × T)) ?S1
     using ‹T ⊆ S› by (force intro: closedin_Times closedin_subset_trans [of
{0..1} × S])+
   show closedin (top_of_set (?S0 ∪ (?S1 ∪ {0..1} × T))) (?S1 ∪ {0..1} × T)
        closedin (top_of_set (?S1 ∪ {0..1} × T)) ({0..1} × T)
     using ‹T ⊆ S› by (force intro: clo closedin_Times closedin_subset_trans [of
{0..1} × S])+
   show continuous_on (?S0) (λx. f (snd x))
     by (intro continuous_intros continuous_on_compose2 [OF contf]) auto
   show continuous_on (?S1) (λx. g (snd x))
     by (intro continuous_intros continuous_on_compose2 [OF contg]) auto
 qed (use h0 h1 conth in auto)
 then have continuous_on ({0,1} × S ∪ ({0..1} × T)) h'
   by (metis Sigma_Un_distrib1 Un_assoc insert_is_Un)
 moreover have h' ‘ ({0,1} × S ∪ {0..1} × T) ⊆ U
   using fim gim him ‹T ⊆ S› unfolding h'_def by force
 moreover have closedin (top_of_set ({0..1::real} × S)) ({0,1} × S ∪ {0..1::real}
× T)
   by (intro closedin_Times closedin_Un clo) (simp_all add: closed_subset)
 ultimately
```

```
  obtain W k where W: ({0,1} × S) ∪ ({0..1} × T) ⊆ W
              and opeW: openin (top_of_set ({0..1} × S)) W
              and contk: continuous_on W k
              and kim: k ∈ W → U
              and kh': ⋀x. x ∈ ({0,1} × S) ∪ ({0..1} × T) ⟹ k x = h' x
      by (metis ANR_imp_absolute_neighbourhood_extensor [OF ‹ANR U›, of
({0,1} × S) ∪ ({0..1} × T) h' {0..1} × S] image_subset_iff_funcset)
  obtain T' where opeT': openin (top_of_set S) T'
             and T ⊆ T' and TW: {0..1} × T' ⊆ W
    using tube_lemma_gen [of {0..1::real} T S W] W ‹T ⊆ S› opeW by auto
  moreover have homotopic_with_canon (λx. True) T' U f g
  proof (simp add: homotopic_with, intro exI conjI)
    show continuous_on ({0..1} × T') k
      using TW continuous_on_subset contk by auto
    show k ‘ ({0..1} × T') ⊆ U
      using TW kim by fastforce
    have T' ⊆ S
      by (meson opeT' subsetD openin_imp_subset)
    then show ∀x∈T'. k (0, x) = f x ∀x∈T'. k (1, x) = g x
      by (auto simp: kh' h'_def)
  qed
  ultimately show ?thesis
    by (blast intro: that)
qed
```

Homotopy on a union of closed-open sets.

```
proposition homotopic_on_clopen_Union:
  fixes ℱ :: 'a::euclidean_space set set
  assumes ⋀S. S ∈ ℱ ⟹ closedin (top_of_set (⋃ℱ)) S
      and ⋀S. S ∈ ℱ ⟹ openin (top_of_set (⋃ℱ)) S
      and ⋀S. S ∈ ℱ ⟹ homotopic_with_canon (λx. True) S T f g
  shows homotopic_with_canon (λx. True) (⋃ℱ) T f g
proof −
  obtain 𝒱 where 𝒱 ⊆ ℱ countable 𝒱 and eqU: ⋃𝒱 = ⋃ℱ
    using Lindelof_openin assms by blast
  show ?thesis
  proof (cases 𝒱 = {})
    case True
    then show ?thesis
      by (metis Union_empty eqU homotopic_with_canon_on_empty)
  next
    case False
    then obtain V :: nat ⇒ 'a set where V: range V = 𝒱
      using range_from_nat_into ‹countable 𝒱› by metis
    with ‹𝒱 ⊆ ℱ› have clo: ⋀n. closedin (top_of_set (⋃ℱ)) (V n)
                  and ope: ⋀n. openin (top_of_set (⋃ℱ)) (V n)
                  and hom: ⋀n. homotopic_with_canon (λx. True) (V n) T f g
      using assms by auto
    then obtain h where conth: ⋀n. continuous_on ({0..1::real} × V n) (h n)
```

```
                and him: ⋀n. h n ` ({0..1} × V n) ⊆ T
                and h0: ⋀n. ⋀x. x ∈ V n ⟹ h n (0, x) = f x
                and h1: ⋀n. ⋀x. x ∈ V n ⟹ h n (1, x) = g x
      by (simp add: homotopic_with) metis
    have wop: b ∈ V x ⟹ ∃k. b ∈ V k ∧ (∀j<k. b ∉ V j) for b x
      using nat_less_induct [where P = λi. b ∉ V i] by meson
    obtain ζ where cont: continuous_on ({0..1} × ⋃(V ` UNIV)) ζ
            and eq: ⋀x i. ⟦x ∈ {0..1} × ⋃(V ` UNIV) ∩
                                {0..1} × (V i − (⋃m<i. V m))⟧ ⟹ ζ x = h i x
    proof (rule pasting_lemma_exists)
      let ?X = top_of_set ({0..1::real} × ⋃(range V))
      show topspace ?X ⊆ (⋃i. {0..1::real} × (V i − (⋃m<i. V m)))
        by (force simp: Ball_def dest: wop)
      show openin (top_of_set ({0..1} × ⋃(V ` UNIV)))
                  ({0..1::real} × (V i − (⋃m<i. V m))) for i
      proof (intro openin_Times openin_subtopology_self openin_diff)
        show openin (top_of_set (⋃(V ` UNIV))) (V i)
          using ope V eqU by auto
        show closedin (top_of_set (⋃(V ` UNIV))) (⋃m<i. V m)
          using V clo eqU by (force intro: closedin_Union)
      qed
      show continuous_map (subtopology ?X ({0..1} × (V i − ⋃ (V ` {..<i})))))
euclidean (h i) for i
        by (auto simp add: subtopology_subtopology intro!: continuous_on_subset
[OF conth])
      show ⋀i j x. x ∈ topspace ?X ∩ {0..1} × (V i − (⋃m<i. V m)) ∩ {0..1}
× (V j − (⋃m<j. V m))
                  ⟹ h i x = h j x
        by clarsimp (metis lessThan_iff linorder_neqE_nat)
    qed auto
    show ?thesis
    proof (simp add: homotopic_with eqU [symmetric], intro exI conjI ballI)
      show continuous_on ({0..1} × ⋃V) ζ
        using V eqU by (blast intro!: continuous_on_subset [OF cont])
      show ζ ` ({0..1} × ⋃V) ⊆ T
      proof clarsimp
        fix t :: real and y :: 'a and X :: 'a set
        assume y ∈ X X ∈ V and t: 0 ≤ t t ≤ 1
        then obtain k where y ∈ V k and j: ∀j<k. y ∉ V j
          by (metis image_iff V wop)
        with him t show ζ(t, y) ∈ T
          by (subst eq) force+
      qed
      fix X y
      assume X ∈ V y ∈ X
      then obtain k where y ∈ V k and j: ∀j<k. y ∉ V j
        by (metis image_iff V wop)
      then show ζ(0, y) = f y and ζ(1, y) = g y
        by (subst eq [where i=k]; force simp: h0 h1)+
```

```
    qed
  qed
qed

lemma homotopic_on_components_eq:
  fixes S :: 'a :: euclidean_space set and T :: 'b :: euclidean_space set
  assumes S: locally connected S ∨ compact S and ANR T
  shows homotopic_with_canon (λx. True) S T f g ⟷
        (continuous_on S f ∧ f ‘ S ⊆ T ∧ continuous_on S g ∧ g ‘ S ⊆ T) ∧
        (∀ C ∈ components S. homotopic_with_canon (λx. True) C T f g)
    (is ?lhs ⟷ ?C ∧ ?rhs)
proof −
  have continuous_on S f f ‘ S ⊆ T continuous_on S g g ‘ S ⊆ T if ?lhs
    using homotopic_with_imp_continuous homotopic_with_imp_subset1 homo-
topic_with_imp_subset2 that by blast+
  moreover have ?lhs ⟷ ?rhs
    if contf: continuous_on S f and fim: f ‘ S ⊆ T and contg: continuous_on S g
and gim: g ‘ S ⊆ T
  proof
    assume ?lhs
    with that show ?rhs
      by (simp add: homotopic_with_subset_left in_components_subset)
  next
    assume R: ?rhs
    have ∃ U. C ⊆ U ∧ closedin (top_of_set S) U ∧
              openin (top_of_set S) U ∧
               homotopic_with_canon (λx. True) U T f g if C: C ∈ components S
for C
    proof −
      have C ⊆ S
        by (simp add: in_components_subset that)
      show ?thesis
        using S
      proof
        assume locally connected S
        show ?thesis
        proof (intro exI conjI)
          show closedin (top_of_set S) C
            by (simp add: closedin_component that)
          show openin (top_of_set S) C
          by (simp add: ‹locally connected S› openin_components_locally_connected
that)
          show homotopic_with_canon (λx. True) C T f g
            by (simp add: R that)
        qed auto
      next
        assume compact S
        have hom: homotopic_with_canon (λx. True) C T f g
          using R that by blast
```

```
      obtain U where C ⊆ U and openU: openin (top_of_set S) U
              and hom: homotopic_with_canon (λx. True) U T f g
        using homotopic_neighbourhood_extension [OF contf fim contg gim _
‹ANR T› hom]
        ‹C ∈ components S› closedin_component by blast
      then obtain V where open V and V: U = S ∩ V
       by (auto simp: openin_open)
      moreover have locally compact S
     by (simp add: ‹compact S› closed_imp_locally_compact compact_imp_closed)
     ultimately obtain K where openK: openin (top_of_set S) K and compact
K C ⊆ K K ⊆ V
       by (metis C Int_subset_iff Sura_Bura_clopen_subset ‹C ⊆ U› ‹compact
S› compact_components)
      show ?thesis
      proof (intro exI conjI)
       show closedin (top_of_set S) K
             by (meson ‹compact K› ‹compact S› closedin_compact_eq openK
openin_imp_subset)
        show homotopic_with_canon (λx. True) K T f g
       using V ‹K ⊆ V› hom homotopic_with_subset_left openK openin_imp_subset
by fastforce
      qed (use openK ‹C ⊆ K› in auto)
    qed
  qed
  then obtain φ where φ: ⋀C. C ∈ components S ⟹ C ⊆ φ C
              and cloφ: ⋀C. C ∈ components S ⟹ closedin (top_of_set S) (φ
C)
               and opeφ: ⋀C. C ∈ components S ⟹ openin (top_of_set S) (φ
C)
                and homφ: ⋀C. C ∈ components S ⟹ homotopic_with_canon
(λx. True) (φ C) T f g
    by metis
  have Seq: S = ⋃ (φ ` components S)
  proof
    show S ⊆ ⋃ (φ ` components S)
     by (metis Sup_mono Union_components φ imageI)
    show ⋃ (φ ` components S) ⊆ S
     using opeφ openin_imp_subset by fastforce
  qed
  show ?lhs
    apply (subst Seq)
    using Seq cloφ opeφ homφ by (intro homotopic_on_clopen_Union) auto
 qed
 ultimately show ?thesis by blast
qed


lemma cohomotopically_trivial_on_components:
 fixes S :: 'a :: euclidean_space set and T :: 'b :: euclidean_space set
```

assumes *S*: *locally connected S* ∨ *compact S* **and** *ANR T*
shows
(∀ *f g*. *continuous_on S f* ⟶ *f* ∈ *S* → *T* ⟶ *continuous_on S g* ⟶ *g* ∈ *S* → *T* ⟶
homotopic_with_canon (λ*x*. *True*) *S T f g*)
⟷
(∀ *C*∈*components S*.
∀ *f g*. *continuous_on C f* ⟶ *f* ∈ *C* → *T* ⟶ *continuous_on C g* ⟶ *g* ∈ *C* → *T* ⟶
homotopic_with_canon (λ*x*. *True*) *C T f g*)
(**is** *?lhs* = *?rhs*)
proof
assume *L*[*rule_format*]: *?lhs*
show *?rhs*
proof *clarify*
fix *C f g*
assume *contf*: *continuous_on C f* **and** *fim*: *f* ∈ *C* → *T*
and *contg*: *continuous_on C g* **and** *gim*: *g* ∈ *C* → *T* **and** *C*: *C* ∈ *components S*
obtain *f′* **where** *contf′*: *continuous_on S f′* **and** *f′im*: *f′* ∈ *S* → *T* **and** *f′f*: ⋀*x*. *x* ∈ *C* ⟹ *f′ x* = *f x*
using *extension_from_component* [*OF S* ‹*ANR T*› *C contf fim*] **by** *metis*
obtain *g′* **where** *contg′*: *continuous_on S g′* **and** *g′im*: *g′* ∈ *S* → *T* **and** *g′g*: ⋀*x*. *x* ∈ *C* ⟹ *g′ x* = *g x*
using *extension_from_component* [*OF S* ‹*ANR T*› *C contg gim*] **by** *metis*
have *homotopic_with_canon* (λ*x*. *True*) *C T f′ g′*
using *L* [*OF contf′ f′im contg′ g′im*] *homotopic_with_subset_left C in_components_subset*
by *fastforce*
then show *homotopic_with_canon* (λ*x*. *True*) *C T f g*
using *f′f g′g homotopic_with_eq* **by** *force*
qed
next
assume *R* [*rule_format*]: *?rhs*
show *?lhs*
proof *clarify*
fix *f g*
assume *contf*: *continuous_on S f* **and** *fim*: *f* ∈ *S* → *T*
and *contg*: *continuous_on S g* **and** *gim*: *g* ∈ *S* → *T*
moreover have *homotopic_with_canon* (λ*x*. *True*) *C T f g* **if** *C* ∈ *components S* **for** *C*
using *R* [*OF that*] *contf contg continuous_on_subset fim gim in_components_subset*
by (*smt* (*verit*, *del_insts*) *Pi_anti_mono subsetD that*)
ultimately show *homotopic_with_canon* (λ*x*. *True*) *S T f g*
by (*subst homotopic_on_components_eq* [*OF S* ‹*ANR T*›]) *auto*
qed
qed

6.35.8 The complement of a set and path-connectedness

Complement in dimension N > 1 of set homeomorphic to any interval in any dimension is (path-)connected. This naively generalizes the argument in Ryuji Maehara's paper "The Jordan curve theorem via the Brouwer fixed point theorem", American Mathematical Monthly 1984.

lemma *unbounded_components_complement_absolute_retract*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *C*: *C* ∈ *components*(− *S*) **and** *S*: *compact S AR S*
shows ¬ *bounded C*
proof −
obtain *y* **where** *y*: *C* = *connected_component_set* (− *S*) *y* **and** *y* ∉ *S*
using *C* **by** (*auto simp*: *components_def*)
have *open*(− *S*)
using *S* **by** (*simp add*: *closed_open compact_eq_bounded_closed*)
have *S retract_of UNIV*
using *S compact_AR* **by** *blast*
then obtain *r* **where** *contr*: *continuous_on UNIV r* **and** *ontor*: *range r* ⊆ *S*
and *r*: ⋀*x*. *x* ∈ *S* ⟹ *r x* = *x*
by (*auto simp*: *retract_of_def retraction_def*)
show *?thesis*
proof
assume *bounded C*
have *connected_component_set* (− *S*) *y* ⊆ *S*
proof (*rule frontier_subset_retraction*)
show *bounded* (*connected_component_set* (− *S*) *y*)
using ‹*bounded C*› *y* **by** *blast*
show *frontier* (*connected_component_set* (− *S*) *y*) ⊆ *S*
using *C* ‹*compact S*› *compact_eq_bounded_closed frontier_of_components_closed_complement*
y **by** *blast*
show *continuous_on* (*closure* (*connected_component_set* (− *S*) *y*)) *r*
by (*blast intro*: *continuous_on_subset* [*OF contr*])
qed (*use ontor r* **in** *auto*)
with ‹*y* ∉ *S*› **show** *False* **by** *force*
qed
qed

lemma *connected_complement_absolute_retract*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *S*: *compact S AR S* **and** *2*: *2* ≤ *DIM*(*'a*)
shows *connected*(− *S*)
proof −
have *S retract_of UNIV*
using *S compact_AR* **by** *blast*
show *?thesis*
proof (*clarsimp simp*: *connected_iff_connected_component_eq*)
have ¬ *bounded* (*connected_component_set* (− *S*) *x*) **if** *x* ∉ *S* **for** *x*
by (*meson Compl_iff assms componentsI that unbounded_components_complement_absolute_retract*
then show *connected_component_set* (− *S*) *x* = *connected_component_set*

```
(− S) y
      if x ∉ S y ∉ S for x y
      using cobounded_unique_unbounded_component [OF _ 2]
      by (metis ‹compact S› compact_imp_bounded double_compl that)
  qed
qed

lemma path_connected_complement_absolute_retract:
  fixes S :: 'a::euclidean_space set
  assumes compact S AR S 2 ≤ DIM('a)
    shows path_connected(− S)
  using connected_complement_absolute_retract [OF assms]
 using ‹compact S› compact_eq_bounded_closed connected_open_path_connected
by blast

theorem connected_complement_homeomorphic_convex_compact:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
   assumes hom: S homeomorphic T and T: convex T compact T and 2: 2 ≤
DIM('a)
    shows connected(− S)
proof (cases S = {})
  case True
  then show ?thesis
    by (simp add: connected_UNIV)
next
  case False
  show ?thesis
  proof (rule connected_complement_absolute_retract)
    show compact S
      using ‹compact T› hom homeomorphic_compactness by auto
    show AR S
     by (meson AR_ANR False ‹convex T› convex_imp_ANR convex_imp_contractible
hom homeomorphic_ANR_iff_ANR homeomorphic_contractible_eq)
  qed (rule 2)
qed

corollary path_connected_complement_homeomorphic_convex_compact:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes hom: S homeomorphic T convex T compact T 2 ≤ DIM('a)
    shows path_connected(− S)
  using connected_complement_homeomorphic_convex_compact [OF assms]
 using ‹compact T› compact_eq_bounded_closed connected_open_path_connected
hom homeomorphic_compactness by blast

lemma path_connected_complement_homeomorphic_interval:
  fixes S :: 'a::euclidean_space set
  assumes S homeomorphic cbox a b 2 ≤ DIM('a)
  shows path_connected(−S)
 using assms compact_cbox convex_box(1) path_connected_complement_homeomorphic_convex_compact
```

by *blast*

lemma *connected_complement_homeomorphic_interval*:
fixes S :: *'a*::*euclidean_space set*
assumes *S homeomorphic cbox a b 2* $\leq$ *DIM*(*'a*)
shows *connected*(−*S*)
using *assms path_connected_complement_homeomorphic_interval path_connected_imp_connected*
by *blast*

end

6.36 Extending Continous Maps, Invariance of Domain, etc

Ported from HOL Light (moretop.ml) by L C Paulson

theory *Further_Topology*
imports *Weierstrass_Theorems Polytope Complex_Transcendental Equivalence_Lebesgue_Henstock_I*
Retracts
begin

6.36.1 A map from a sphere to a higher dimensional sphere is nullhomotopic

lemma *spheremap_lemma1*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'a*::*euclidean_space*
assumes *subspace S subspace T* **and** *dimST*: *dim S* < *dim T*
and $S \subseteq T$
and *diff_f*: *f differentiable_on sphere 0 1* ∩ *S*
shows *f* ‘ (*sphere 0 1* ∩ *S*) ≠ *sphere 0 1* ∩ *T*
proof
assume *fim*: *f* ‘ (*sphere 0 1* ∩ *S*) = *sphere 0 1* ∩ *T*
have *inS*: ⋀*x*. ⟦*x* ∈ *S*; *x* ≠ *0*⟧ ⟹ (*x* /*R norm x*) ∈ *S*
using *subspace_mul* ‹*subspace S*› **by** *blast*
have *subS01*: (λ*x*. *x* /*R norm x*) ‘ (*S* − {*0*}) ⊆ *sphere 0 1* ∩ *S*
using ‹*subspace S*› *subspace_mul* **by** *fastforce*
then have *diff_f'*: *f differentiable_on* (λ*x*. *x* /*R norm x*) ‘ (*S* − {*0*})
by (*rule differentiable_on_subset* [*OF diff_f*])
define *g* **where** *g* ≡ λ*x*. *norm x* ∗*R f*(*inverse*(*norm x*) ∗*R x*)
have *gdiff*: *g differentiable_on S* − {*0*}
unfolding *g_def*
by (*rule diff_f' derivative_intros differentiable_on_compose* [**where** *f*=*f*] |
force)+
have *geq*: *g* ‘ (*S* − {*0*}) = *T* − {*0*}
proof
have ⋀*u*. ⟦*u* ∈ *S*; *norm u* ∗*R f* (*u* /*R norm u*) ∉ *T*⟧ ⟹ *u* = *0*
by (*metis* (*mono_tags*, *lifting*) *DiffI subS01 subspace_mul* [*OF* ‹*subspace T*›]
fim image_subset_iff inf_le2 singletonD)
then have *g* ‘ (*S* − {*0*}) ⊆ *T*

```
    using g_def by blast
  moreover have g ‘ (S − {0}) ⊆ UNIV − {0}
  proof (clarsimp simp: g_def)
    fix y
    assume y ∈ S and f0: f (y /R norm y) = 0
    then have y ≠ 0 ⟹ y /R norm y ∈ sphere 0 1 ∩ S
      by (auto simp: subspace_mul [OF ‹subspace S›])
    then show y = 0
    by (metis fim f0 Int_iff image_iff mem_sphere_0 norm_eq_zero zero_neq_one)
  qed
  ultimately show g ‘ (S − {0}) ⊆ T − {0}
    by auto
next
  have ∗: sphere 0 1 ∩ T ⊆ f ‘ (sphere 0 1 ∩ S)
    using fim by (simp add: image_subset_iff)
  have x ∈ (λx. norm x ∗R f (x /R norm x)) ‘ (S − {0})
       if x ∈ T x ≠ 0 for x
  proof −
    have x /R norm x ∈ T
      using ‹subspace T› subspace_mul that by blast
    then obtain u where u: f u ∈ T x /R norm x = f u norm u = 1 u ∈ S
      using ∗ [THEN subsetD, of x /R norm x] ‹x ≠ 0› by auto
    with that have [simp]: norm x ∗R f u = x
      by (metis divideR_right norm_eq_zero)
    moreover have norm x ∗R u ∈ S − {0}
      using ‹subspace S› subspace_scale that(2) u by auto
    with u show ?thesis
      by (simp add: image_eqI [where x=norm x ∗R u])
  qed
  then have T − {0} ⊆ (λx. norm x ∗R f (x /R norm x)) ‘ (S − {0})
    by force
  then show T − {0} ⊆ g ‘ (S − {0})
    by (simp add: g_def)
qed
define T′ where T′ ≡ {y. ∀x ∈ T. orthogonal x y}
have subspace T′
  by (simp add: subspace_orthogonal_to_vectors T′_def)
have dim_eq: dim T′ + dim T = DIM('a)
  using dim_subspace_orthogonal_to_vectors [of T UNIV] ‹subspace T›
  by (simp add: T′_def)
have ∃v1 v2. v1 ∈ span T ∧ (∀w ∈ span T. orthogonal v2 w) ∧ x = v1 + v2
for x
  by (force intro: orthogonal_subspace_decomp_exists [of T x])
then obtain p1 p2 where p1span: p1 x ∈ span T
                    and ⋀w. w ∈ span T ⟹ orthogonal (p2 x) w
                    and eq: p1 x + p2 x = x for x
  by metis
then have p1: ⋀z. p1 z ∈ T and ortho: ⋀w. w ∈ T ⟹ orthogonal (p2 x) w
for x
```

```
    using span_eq_iff ‹subspace T› by blast+
  then have p2: ⋀z. p2 z ∈ T'
    by (simp add: T'_def orthogonal_commute)
  have p12_eq: ⋀x y. ⟦x ∈ T; y ∈ T'⟧ ⟹ p1(x + y) = x ∧ p2(x + y) = y
  proof (rule orthogonal_subspace_decomp_unique [OF eq p1span, where T=T'])
    show ⋀x y. ⟦x ∈ T; y ∈ T'⟧ ⟹ p2 (x + y) ∈ span T'
      using span_eq_iff p2 ‹subspace T'› by blast
    show ⋀a b. ⟦a ∈ T; b ∈ T'⟧ ⟹ orthogonal a b
      using T'_def by blast
  qed (auto simp: span_base)
  then have ⋀c x. p1 (c *R x) = c *R p1 x ∧ p2 (c *R x) = c *R p2 x
  proof −
    fix c :: real and x :: 'a
    have f1: c *R x = c *R p1 x + c *R p2 x
      by (metis eq pth_6)
    have f2: c *R p2 x ∈ T'
      by (simp add: ‹subspace T'› p2 subspace_scale)
    have c *R p1 x ∈ T
      by (metis (full_types) assms(2) p1span span_eq_iff subspace_scale)
    then show p1 (c *R x) = c *R p1 x ∧ p2 (c *R x) = c *R p2 x
      using f2 f1 p12_eq by presburger
  qed
  moreover have lin_add: ⋀x y. p1 (x + y) = p1 x + p1 y ∧ p2 (x + y) = p2
x + p2 y
  proof (rule orthogonal_subspace_decomp_unique [OF _ p1span, where T=T'])
    show ⋀x y. p1 (x + y) + p2 (x + y) = p1 x + p1 y + (p2 x + p2 y)
      by (simp add: add.assoc add.left_commute eq)
    show  ⋀a b. ⟦a ∈ T; b ∈ T'⟧ ⟹ orthogonal a b
      using T'_def by blast
  qed (auto simp: p1span p2 span_base span_add)
  ultimately have linear p1 linear p2
    by unfold_locales auto
  have g differentiable_on p1 ` {x + y |x y. x ∈ S − {0} ∧ y ∈ T'}
    using p12_eq ‹S ⊆ T›  by (force intro: differentiable_on_subset [OF gdiff])
  then have (λz. g (p1 z)) differentiable_on {x + y |x y. x ∈ S − {0} ∧ y ∈ T'}
     by (rule differentiable_on_compose [OF linear_imp_differentiable_on [OF
‹linear p1›]])
  then have diff: (λx. g (p1 x) + p2 x) differentiable_on {x + y |x y. x ∈ S −
{0} ∧ y ∈ T'}
    by (intro derivative_intros linear_imp_differentiable_on [OF ‹linear p2›])
  have dim {x + y |x y. x ∈ S − {0} ∧ y ∈ T'} ≤ dim {x + y |x y. x ∈ S ∧ y
∈ T'}
    by (blast intro: dim_subset)
  also have ... = dim S + dim T' − dim (S ∩ T')
    using dim_sums_Int [OF ‹subspace S› ‹subspace T'›]
    by (simp add: algebra_simps)
  also have ... < DIM('a)
    using dimST dim_eq by auto
  finally have neg: negligible {x + y |x y. x ∈ S − {0} ∧ y ∈ T'}
```

```
    by (rule negligible_lowdim)
  have negligible ((λx. g (p1 x) + p2 x) ‘ {x + y |x y. x ∈ S − {0} ∧ y ∈ T′})
    by (rule negligible_differentiable_image_negligible [OF order_refl neg diff])
  then have negligible {x + y |x y. x ∈ g ‘ (S − {0}) ∧ y ∈ T′}
  proof (rule negligible_subset)
    have ⟦t′ ∈ T′; s ∈ S; s ≠ 0⟧
         ⟹ g s + t′ ∈ (λx. g (p1 x) + p2 x) ‘
                       {x + t′ |x t′. x ∈ S ∧ x ≠ 0 ∧ t′ ∈ T′} for t′ s
      using ‹S ⊆ T› p12_eq  by (rule_tac x=s + t′ in image_eqI) auto
    then show {x + y |x y. x ∈ g ‘ (S − {0}) ∧ y ∈ T′}
         ⊆ (λx. g (p1 x) + p2 x) ‘ {x + y |x y. x ∈ S − {0} ∧ y ∈ T′}
      by auto
  qed
  moreover have − T′ ⊆ {x + y |x y. x ∈ g ‘ (S − {0}) ∧ y ∈ T′}
  proof clarsimp
    fix z assume z ∉ T′
    show ∃x y. z = x + y ∧ x ∈ g ‘ (S − {0}) ∧ y ∈ T′
      by (metis Diff_iff ‹z ∉ T′› add.left_neutral eq geq p1 p2 singletonD)
  qed
  ultimately have negligible (−T′)
    using negligible_subset by blast
  moreover have negligible T′
    using negligible_lowdim
     by (metis add.commute assms(3) diff_add_inverse2 diff_self_eq_0 dim_eq
le_add1 le_antisym linordered_semidom_class.add_diff_inverse not_less0)
  ultimately have  negligible (−T′ ∪ T′)
    by (metis negligible_Un_eq)
  then show False
    using negligible_Un_eq non_negligible_UNIV by simp
qed


lemma spheremap_lemma2:
  fixes f :: ′a::euclidean_space ⇒ ′a::euclidean_space
  assumes ST: subspace S subspace T dim S < dim T
      and S ⊆ T
      and contf: continuous_on (sphere 0 1 ∩ S) f
      and fim: f ‘ (sphere 0 1 ∩ S) ⊆ sphere 0 1 ∩ T
    shows ∃c. homotopic_with_canon (λx. True) (sphere 0 1 ∩ S) (sphere 0 1 ∩
T) f (λx. c)
proof −
  have [simp]: ⋀x. ⟦norm x = 1; x ∈ S⟧ ⟹ norm (f x) = 1
    using fim by (simp add: image_subset_iff)
  have compact (sphere 0 1 ∩ S)
    by (simp add: ‹subspace S› closed_subspace compact_Int_closed)
  then obtain g where pfg: polynomial_function g and gim: g ‘ (sphere 0 1 ∩
S) ⊆ T
                and g12: ⋀x. x ∈ sphere 0 1 ∩ S ⟹ norm(f x − g x) < 1/2
    using Stone_Weierstrass_polynomial_function_subspace [OF _ contf _ ‹sub-
```

```
space T›, of 1/2] fim by auto
  have gnz: g x ≠ 0 if x ∈ sphere 0 1 ∩ S for x
  proof −
    have norm (f x) = 1
      using fim that by (simp add: image_subset_iff)
    then show ?thesis
      using g12 [OF that] by auto
  qed
  have diffg: g differentiable_on sphere 0 1 ∩ S
    by (metis pfg differentiable_on_polynomial_function)
  define h where h ≡ λx. inverse(norm(g x)) *_R g x
  have h: x ∈ sphere 0 1 ∩ S ⟹ h x ∈ sphere 0 1 ∩ T for x
    unfolding h_def
    using gnz [of x]
    by (auto simp: subspace_mul [OF ‹subspace T›] subsetD [OF gim])
  have diffh: h differentiable_on sphere 0 1 ∩ S
    unfolding h_def using gnz
   by (fastforce intro: derivative_intros diffg differentiable_on_compose [OF diffg])
  have homfg: homotopic_with_canon (λz. True) (sphere 0 1 ∩ S) (T − {0}) f g
  proof (rule homotopic_with_linear [OF contf])
    show continuous_on (sphere 0 1 ∩ S) g
      using pfg by (simp add: differentiable_imp_continuous_on diffg)
  next
    have non0fg: 0 ∉ closed_segment (f x) (g x) if norm x = 1 x ∈ S for x
    proof −
      have f x ∈ sphere 0 1
        using fim that by (simp add: image_subset_iff)
      moreover have norm(f x − g x) < 1/2
        using g12 that by auto
      ultimately show ?thesis
        by (auto simp: norm_minus_commute dest: segment_bound)
    qed
    show closed_segment (f x) (g x) ⊆ T − {0} if x ∈ sphere 0 1 ∩ S for x
    proof −
      have convex T
        by (simp add: ‹subspace T› subspace_imp_convex)
      then have convex hull {f x, g x} ⊆ T
          by (metis IntD2 closed_segment_subset fim gim image_subset_iff seg-
ment_convex_hull that)
      then show ?thesis
        using that non0fg segment_convex_hull by fastforce
    qed
  qed
  obtain d where d: d ∈ (sphere 0 1 ∩ T) − h ‘ (sphere 0 1 ∩ S)
    using h spheremap_lemma1 [OF ST ‹S ⊆ T› diffh] by force
  then have non0hd: 0 ∉ closed_segment (h x) (− d) if norm x = 1 x ∈ S for x
    using midpoint_between [of 0 h x −d] that h [of x]
    by (auto simp: between_mem_segment midpoint_def)
  have conth: continuous_on (sphere 0 1 ∩ S) h
```

```
    using differentiable_imp_continuous_on diffh by blast
  have hom_hd: homotopic_with_canon (λz. True) (sphere 0 1 ∩ S) (T − {0})
h (λx. −d)
  proof (rule homotopic_with_linear [OF conth continuous_on_const])
    fix x
    assume x: x ∈ sphere 0 1 ∩ S
    have convex hull {h x, − d} ⊆ T
    proof (rule hull_minimal)
      show {h x, − d} ⊆ T
        using h d x by (force simp: subspace_neg [OF ‹subspace T›])
    qed (simp add: subspace_imp_convex [OF ‹subspace T›])
    with x segment_convex_hull show closed_segment (h x) (− d) ⊆ T − {0}
      by (auto simp add: subset_Diff_insert non0hd)
  qed
  have conT0: continuous_on (T − {0}) (λy. inverse(norm y) *R y)
    by (intro continuous_intros) auto
  have sub0T: (λy. y /R norm y) ∈ (T − {0}) → sphere 0 1 ∩ T
    by (fastforce simp: assms(2) subspace_mul)
  obtain c where homhc: homotopic_with_canon (λz. True) (sphere 0 1 ∩ S)
(sphere 0 1 ∩ T) h (λx. c)
  proof
    show homotopic_with_canon (λz. True) (sphere 0 1 ∩ S) (sphere 0 1 ∩ T) h
(λx. − d)
      using d
      by (force simp: h_def
          intro: homotopic_with_eq homotopic_with_compose_continuous_left [OF
hom_hd conT0 sub0T])
  qed
  have homotopic_with_canon (λx. True) (sphere 0 1 ∩ S) (sphere 0 1 ∩ T) f h
    by (force simp: h_def
            intro:  homotopic_with_eq homotopic_with_compose_continuous_left
[OF homfg conT0 sub0T])
  then show ?thesis
    by (metis homotopic_with_trans [OF _ homhc])
qed


lemma spheremap_lemma3:
  assumes bounded S convex S subspace U and affSU: aff_dim S ≤ dim U
  obtains T where subspace T T ⊆ U S ≠ {} ⟹ aff_dim T = aff_dim S
                (rel_frontier S) homeomorphic (sphere 0 1 ∩ T)
proof (cases S = {})
  case True
  with ‹subspace U› subspace_0 show ?thesis
    by (rule_tac T = {0} in that) auto
next
  case False
  then obtain a where a ∈ S
    by auto
```

```
  then have affS: aff_dim S = int (dim ((λx. −a+x) ‘ S))
    by (metis hull_inc aff_dim_eq_dim)
  with affSU have dim ((λx. −a+x) ‘ S) ≤ dim U
    by linarith
  with choose_subspace_of_subspace
  obtain T where subspace T T ⊆ span U and dimT: dim T = dim ((λx. −a+x)
‘ S) .
  show ?thesis
  proof (rule that [OF ‹subspace T›])
    show T ⊆ U
      using span_eq_iff ‹subspace U› ‹T ⊆ span U› by blast
    show aff_dim T = aff_dim S
      using dimT ‹subspace T› affS aff_dim_subspace by fastforce
    show rel_frontier S homeomorphic sphere 0 1 ∩ T
    proof −
      have aff_dim (ball 0 1 ∩ T) = aff_dim (T)
      by (metis IntI interior_ball ‹subspace T› aff_dim_convex_Int_nonempty_interior
centre_in_ball empty_iff inf_commute subspace_0 subspace_imp_convex zero_less_one)
      then have affS_eq: aff_dim S = aff_dim (ball 0 1 ∩ T)
        using ‹aff_dim T = aff_dim S› by simp
      have rel_frontier S homeomorphic rel_frontier(ball 0 1 ∩ T)
      proof (rule homeomorphic_rel_frontiers_convex_bounded_sets [OF ‹convex
S› ‹bounded S›])
        show convex (ball 0 1 ∩ T)
          by (simp add: ‹subspace T› convex_Int subspace_imp_convex)
        show bounded (ball 0 1 ∩ T)
          by (simp add: bounded_Int)
        show aff_dim S = aff_dim (ball 0 1 ∩ T)
          by (rule affS_eq)
      qed
      also have ... = frontier (ball 0 1) ∩ T
      proof (rule convex_affine_rel_frontier_Int [OF convex_ball])
        show affine T
          by (simp add: ‹subspace T› subspace_imp_affine)
        show interior (ball 0 1) ∩ T ≠ {}
          using ‹subspace T› subspace_0 by force
      qed
      also have ... = sphere 0 1 ∩ T
        by auto
      finally show ?thesis .
    qed
  qed
qed


proposition inessential_spheremap_lowdim_gen:
  fixes f :: 'M::euclidean_space ⇒ 'a::euclidean_space
  assumes convex S bounded S convex T bounded T
      and affST: aff_dim S < aff_dim T
```

```
      and contf: continuous_on (rel_frontier S) f
      and fim: f ∈ (rel_frontier S) → rel_frontier T
  obtains c where homotopic_with_canon (λz. True) (rel_frontier S) (rel_frontier
T) f (λx. c)
proof (cases S = {})
  case True
  then show ?thesis
    using homotopic_with_canon_on_empty that by auto
next
  case False
  then show ?thesis
  proof (cases T = {})
    case True
    then have rel_frontier S = {} rel_frontier T = {}
      using fim by fastforce+
    then show ?thesis
      using that by (simp add: homotopic_on_emptyI)
  next
    case False
    obtain T'::'a set
      where subspace T' and affT': aff_dim T' = aff_dim T
        and homT: rel_frontier T homeomorphic sphere 0 1 ∩ T'
        using ‹T ≠ {}› spheremap_lemma3 [OF ‹bounded T› ‹convex T› sub-
space_UNIV, where 'b='a]
      by (force simp add: aff_dim_le_DIM)
    with homeomorphic_imp_homotopy_eqv
    have relT: sphere 0 1 ∩ T' homotopy_eqv rel_frontier T
      using homotopy_equivalent_space_sym by blast
    have aff_dim S ≤ int (dim T')
      using affT' ‹subspace T'› affST aff_dim_subspace by force
    with spheremap_lemma3 [OF ‹bounded S› ‹convex S› ‹subspace T'›] ‹S ≠ {}›
    obtain S'::'a set where subspace S' S' ⊆ T'
       and affS': aff_dim S' = aff_dim S
       and homT: rel_frontier S homeomorphic sphere 0 1 ∩ S'
       by metis
    with homeomorphic_imp_homotopy_eqv
    have relS: sphere 0 1 ∩ S' homotopy_eqv rel_frontier S
      using homotopy_equivalent_space_sym by blast
    have dimST': dim S' < dim T'
      by (metis ‹S' ⊆ T'› ‹subspace S'› ‹subspace T'› affS' affST affT' less_irrefl
not_le subspace_dim_equal)
    have ∃ c. homotopic_with_canon (λz. True) (rel_frontier S) (rel_frontier T)
f (λx. c)
       using spheremap_lemma2 homotopy_eqv_cohomotopic_triviality_null[OF
relS]
      using homotopy_eqv_homotopic_triviality_null_imp [OF relT contf fim]
     by (metis ‹S' ⊆ T'› ‹subspace S'› ‹subspace T'› dimST' image_subset_iff_funcset)
    with that show ?thesis by blast
  qed
```

qed

lemma *inessential_spheremap_lowdim*:
fixes $f :: 'M::euclidean\_space \Rightarrow 'a::euclidean\_space$
assumes
$DIM('M) < DIM('a)$ **and** f: *continuous_on* (*sphere a r*) $f\, f \in$ (*sphere a r*) $\to$ (*sphere b s*)
obtains *c* **where** *homotopic_with_canon* ($\lambda z.\ True$) (*sphere a r*) (*sphere b s*) f ($\lambda x.\ c$)
proof (*cases* $s \leq 0$)
case *True* **then show** *?thesis*
by (*meson nullhomotopic_into_contractible f contractible_sphere that*)
next
case *False*
show *?thesis*
proof (*cases* $r \leq 0$)
case *True* **then show** *?thesis*
by (*meson f nullhomotopic_from_contractible contractible_sphere that*)
next
case *False*
with ‹$\neg\ s \leq 0$› **have** $r > 0\ s > 0$ **by** *auto*
show *thesis*
using *inessential_spheremap_lowdim_gen* [*of cball a r cball b s f*]
using ‹$0 < r$› ‹$0 < s$› *assms*(*1*) *that* **by** (*auto simp add*: *f aff_dim_cball*)
qed
qed

6.36.2 Some technical lemmas about extending maps from cell complexes

lemma *extending_maps_Union_aux*:
assumes *fin*: *finite* $\mathcal{F}$
and $\bigwedge S.\ S \in \mathcal{F} \Longrightarrow$ *closed S*
and $\bigwedge S\ T.\ [\![S \in \mathcal{F};\ T \in \mathcal{F};\ S \neq T]\!] \Longrightarrow S \cap T \subseteq K$
and $\bigwedge S.\ S \in \mathcal{F} \Longrightarrow \exists g.$ *continuous_on S g* $\wedge$ *g ‘ S* $\subseteq T \wedge (\forall x \in S \cap K.\ g\ x = h\ x)$
shows $\exists g.$ *continuous_on* $(\bigcup \mathcal{F})$ *g* $\wedge$ *g ‘* $(\bigcup \mathcal{F}) \subseteq T \wedge (\forall x \in \bigcup \mathcal{F} \cap K.\ g\ x = h\ x)$
using *assms*
proof (*induction* $\mathcal{F}$)
case *empty* **show** *?case* **by** *simp*
next
case (*insert S* $\mathcal{F}$)
then obtain *f* **where** *contf*: *continuous_on* (*S*) *f* **and** *fim*: *f ‘ S* $\subseteq T$ **and** *feq*: $\forall x \in S \cap K.\ f\ x = h\ x$
by (*meson insertI1*)
obtain *g* **where** *contg*: *continuous_on* $(\bigcup \mathcal{F})$ *g* **and** *gim*: *g ‘* $\bigcup \mathcal{F} \subseteq T$ **and** *geq*: $\forall x \in \bigcup \mathcal{F} \cap K.\ g\ x = h\ x$
using *insert* **by** *auto*

have *fg*: $f\ x = g\ x$ **if** $x \in T\ T \in \mathcal{F}\ x \in S$ **for** $x\ T$
proof −
have $T \cap S \subseteq K \vee S = T$
using *that* **by** (*metis* (*no_types*) *insert.prems*(*2*) *insertCI*)
then show *?thesis*
using *UnionI feq geq* ‹$S \notin \mathcal{F}$› *subsetD that* **by** *fastforce*
qed
moreover have *continuous_on* $(S \cup \bigcup\ \mathcal{F})$ $(\lambda x.\ \textit{if}\ x \in S\ \textit{then}\ f\ x\ \textit{else}\ g\ x)$
by (*auto simp*: *insert closed_Union contf contg* *intro*: *fg continuous_on_cases*)
ultimately show *?case*
by (*smt* (*verit*, *del_insts*) *Int_iff UnE complete_lattice_class.Sup_insert feq fim geq gim image_subset_iff*)
qed

lemma *extending_maps_Union*:
assumes *fin*: *finite* $\mathcal{F}$
and $\bigwedge S.\ S \in \mathcal{F} \Longrightarrow \exists g.\ \textit{continuous\_on}\ S\ g \wedge g\ `\ S \subseteq T \wedge (\forall x \in S \cap K.\ g\ x = h\ x)$
and $\bigwedge S.\ S \in \mathcal{F} \Longrightarrow \textit{closed}\ S$
and K: $\bigwedge X\ Y.\ [\![X \in \mathcal{F};\ Y \in \mathcal{F};\ \neg\ X \subseteq Y;\ \neg\ Y \subseteq X]\!] \Longrightarrow X \cap Y \subseteq K$
shows $\exists g.\ \textit{continuous\_on}\ (\bigcup \mathcal{F})\ g \wedge g\ `\ (\bigcup \mathcal{F}) \subseteq T \wedge (\forall x \in \bigcup \mathcal{F} \cap K.\ g\ x = h\ x)$
apply (*simp flip*: *Union_maximal_sets* [*OF fin*])
apply (*rule extending_maps_Union_aux*)
apply (*simp_all add*: *Union_maximal_sets* [*OF fin*] *assms*)
by (*metis K psubsetI*)

lemma *extend_map_lemma*:
assumes *finite* $\mathcal{F}\ \mathcal{G} \subseteq \mathcal{F}$ *convex T bounded T*
and *poly*: $\bigwedge X.\ X \in \mathcal{F} \Longrightarrow \textit{polytope}\ X$
and *aff*: $\bigwedge X.\ X \in \mathcal{F} - \mathcal{G} \Longrightarrow \textit{aff\_dim}\ X < \textit{aff\_dim}\ T$
and *face*: $\bigwedge S\ T.\ [\![S \in \mathcal{F};\ T \in \mathcal{F}]\!] \Longrightarrow (S \cap T)\ \textit{face\_of}\ S$
and *contf*: *continuous_on* $(\bigcup \mathcal{G})\ f$ **and** *fim*: $f\ `\ (\bigcup \mathcal{G}) \subseteq \textit{rel\_frontier}\ T$
obtains g **where** *continuous_on* $(\bigcup \mathcal{F})\ g\ g\ `\ (\bigcup \mathcal{F}) \subseteq \textit{rel\_frontier}\ T\ \bigwedge x.\ x \in \bigcup \mathcal{G} \Longrightarrow g\ x = f\ x$
proof (*cases* $\mathcal{F} - \mathcal{G} = \{\}$)
case *True*
show *?thesis*
proof
show *continuous_on* $(\bigcup\ \mathcal{F})\ f$
using *True* ‹$\mathcal{G} \subseteq \mathcal{F}$› *contf* **by** *auto*
show $f\ `\ \bigcup\ \mathcal{F} \subseteq \textit{rel\_frontier}\ T$
using *True fim* **by** *auto*
qed *auto*
next
case *False*
then have $0 \leq \textit{aff\_dim}\ T$
by (*metis aff aff_dim_empty aff_dim_geq aff_dim_negative_iff all_not_in_conv*

```
not_less)
  then obtain i::nat where i: int i = aff_dim T
    by (metis nonneg_eq_int)
  have Union_empty_eq: ⋃{D. D = {} ∧ P D} = {} for P :: 'a set ⇒ bool
    by auto
  have face': ⋀S T. ⟦S ∈ ℱ; T ∈ ℱ⟧ ⟹ (S ∩ T) face_of S ∧ (S ∩ T) face_of T
    by (metis face inf_commute)
  have extendf: ∃g. continuous_on (⋃(𝒢 ∪ {D. ∃C ∈ ℱ. D face_of C ∧ aff_dim
D < i})) g ∧
                    g ` (⋃ (𝒢 ∪ {D. ∃C ∈ ℱ. D face_of C ∧ aff_dim D < i})) ⊆
rel_frontier T ∧
                    (∀x ∈ ⋃𝒢. g x = f x)
      if i ≤ aff_dim T for i::nat
  using that
  proof (induction i)
    case 0
    show ?case
      using 0 contf fim by (auto simp add: Union_empty_eq)
  next
    case (Suc p)
    with ‹bounded T› have rel_frontier T ≠ {}
      by (auto simp: rel_frontier_eq_empty affine_bounded_eq_lowdim [of T])
    then obtain t where t: t ∈ rel_frontier T by auto
    have ple: int p ≤ aff_dim T using Suc.prems by force
    obtain h where conth: continuous_on (⋃(𝒢 ∪ {D. ∃C ∈ ℱ. D face_of C ∧
aff_dim D < p})) h
              and him: h ` (⋃ (𝒢 ∪ {D. ∃C ∈ ℱ. D face_of C ∧ aff_dim D < p}))
                       ⊆ rel_frontier T
              and heq: ⋀x. x ∈ ⋃𝒢 ⟹ h x = f x
      using Suc.IH [OF ple] by auto
    let ?Faces = {D. ∃C ∈ ℱ. D face_of C ∧ aff_dim D ≤ p}
    have extendh: ∃g. continuous_on D g ∧
                      g ` D ⊆ rel_frontier T ∧
                      (∀x ∈ D ∩ ⋃(𝒢 ∪ {D. ∃C ∈ ℱ. D face_of C ∧ aff_dim D <
p}). g x = h x)
      if D: D ∈ 𝒢 ∪ ?Faces for D
    proof (cases D ⊆ ⋃(𝒢 ∪ {D. ∃C ∈ ℱ. D face_of C ∧ aff_dim D < p}))
      case True
      have continuous_on D h
        using True conth continuous_on_subset by blast
      moreover have h ` D ⊆ rel_frontier T
        using True him by blast
      ultimately show ?thesis
        by blast
    next
      case False
      note notDsub = False
      show ?thesis
      proof (cases ∃a. D = {a})
```

case *True*
then obtain *a* **where** $D = \{a\}$ **by** *auto*
with *notDsub t* **show** *?thesis*
by (*rule_tac x*=$\lambda x.\ t$ **in** *exI*) *simp*
next
case *False*
have $D \neq \{\}$ **using** *notDsub* **by** *auto*
have *Dnotin*: $D \notin \mathcal{G} \cup \{D.\ \exists\, C \in \mathcal{F}.\ D$ *face_of* $C \wedge$ *aff_dim* $D < p\}$
using *notDsub* **by** *auto*
then have $D \notin \mathcal{G}$ **by** *simp*
have $D \in$ *?Faces* $- \{D.\ \exists\, C \in \mathcal{F}.\ D$ *face_of* $C \wedge$ *aff_dim* $D < p\}$
using *Dnotin that* **by** *auto*
then obtain *C* **where** $C \in \mathcal{F}$ D *face_of* C **and** *affD*: *aff_dim* $D =$ *int* p
by *auto*
then have *bounded D*
using *face_of_polytope_polytope poly polytope_imp_bounded* **by** *blast*
then have [*simp*]: $\neg$ *affine D*
using *affine_bounded_eq_trivial False* ‹$D \neq \{\}$› ‹*bounded D*› **by** *blast*
have $\{F.\ F$ *facet_of* $D\} \subseteq \{E.\ E$ *face_of* $C \wedge$ *aff_dim* $E <$ *int* $p\}$
by *clarify* (*metis* ‹*D face_of C*› *affD eq_iff face_of_trans facet_of_def zle_diff1_eq*)
moreover have *polyhedron D*
using ‹$C \in \mathcal{F}$› ‹*D face_of C*› *face_of_polytope_polytope poly polytope_imp_polyhedron* **by** *auto*
ultimately have *relf_sub*: *rel_frontier* $D \subseteq \bigcup \{E.\ E$ *face_of* $C \wedge$ *aff_dim* $E < p\}$
by (*simp add*: *rel_frontier_of_polyhedron Union_mono*)
then have *him_relf*: $h \in$ *rel_frontier* $D \to$ *rel_frontier* T
using ‹$C \in \mathcal{F}$› *him* **by** *blast*
have *convex D*
by (*simp add*: ‹*polyhedron D*› *polyhedron_imp_convex*)
have *affD_lessT*: *aff_dim* $D <$ *aff_dim* T
using *Suc.prems affD* **by** *linarith*
have *contDh*: *continuous_on* (*rel_frontier D*) h
using ‹$C \in \mathcal{F}$› *relf_sub* **by** (*blast intro*: *continuous_on_subset* [*OF conth*])
then have $*$: $(\exists\, c.$ *homotopic_with_canon* $(\lambda x.\ True)$ (*rel_frontier D*) (*rel_frontier T*) h $(\lambda x.\ c)) =$
$(\exists\, g.$ *continuous_on UNIV* $g \wedge$ *range* $g \subseteq$ *rel_frontier* $T \wedge$
$(\forall\, x \in$*rel_frontier D*. $g\ x = h\ x))$
by (*simp add*: *assms image_subset_iff_funcset rel_frontier_eq_empty him_relf nullhomotopic_into_rel_frontier_extension* [*OF closed_rel_frontier*])
have $\exists\, c.$ *homotopic_with_canon* $(\lambda x.\ True)$ (*rel_frontier D*) (*rel_frontier T*) h $(\lambda x.\ c)$
by (*metis inessential_spheremap_lowdim_gen*
[*OF* ‹*convex D*› ‹*bounded D*› ‹*convex T*› ‹*bounded T*› *affD_lessT contDh him_relf*])
then obtain g **where** *contg*: *continuous_on UNIV* g
and *gim*: *range* $g \subseteq$ *rel_frontier* T
and *gh*: $\bigwedge x.\ x \in$ *rel_frontier* $D \Longrightarrow g\ x = h\ x$

```
        by (metis *)
      have D ∩ E ⊆ rel_frontier D
          if E ∈ 𝒢 ∪ {D. Bex ℱ ((face_of) D) ∧ aff_dim D < int p} for E
      proof (rule face_of_subset_rel_frontier)
        show D ∩ E face_of D
          using that
        proof safe
          assume E ∈ 𝒢
          then show D ∩ E face_of D
           by (meson ‹C ∈ ℱ› ‹D face_of C› assms(2) face' face_of_Int_subface
face_of_refl_eq poly polytope_imp_convex subsetD)
        next
          fix x
          assume aff_dim E < int p x ∈ ℱ E face_of x
          then show D ∩ E face_of D
            by (meson ‹C ∈ ℱ› ‹D face_of C› face' face_of_Int_subface that)
        qed
        show D ∩ E ≠ D
          using that notDsub by auto
      qed
      moreover have continuous_on D g
        using contg continuous_on_subset by blast
      ultimately show ?thesis
        by (rule_tac x=g in exI) (use gh gim in fastforce)
    qed
  qed
  have intle: i < 1 + int j ⟷ i ≤ int j for i j
    by auto
  have finite 𝒢
    using ‹finite ℱ› ‹𝒢 ⊆ ℱ› rev_finite_subset by blast
  moreover have finite (?Faces)
  proof −
    have §: finite (⋃ {{D. D face_of C} |C. C ∈ ℱ})
      by (auto simp: ‹finite ℱ› finite_polytope_faces poly)
    show ?thesis
      by (auto intro: finite_subset [OF _ §])
  qed
  ultimately have fin: finite (𝒢 ∪ ?Faces)
    by simp
  have clo: closed S if S ∈ 𝒢 ∪ ?Faces for S
    using that ‹𝒢 ⊆ ℱ› face_of_polytope_polytope poly polytope_imp_closed by
blast
  have K: X ∩ Y ⊆ ⋃(𝒢 ∪ {D. ∃ C∈ℱ. D face_of C ∧ aff_dim D < int p})
            if X ∈ 𝒢 ∪ ?Faces Y ∈ 𝒢 ∪ ?Faces ¬ Y ⊆ X for X Y
  proof −
    have ff: X ∩ Y face_of X ∧ X ∩ Y face_of Y
      if XY: X face_of D Y face_of E and DE: D ∈ ℱ E ∈ ℱ for D E
      by (rule face_of_Int_subface [OF _ _ XY]) (auto simp: face' DE)
   show ?thesis
```

using *that*
apply *clarsimp*
by (*smt* (*verit*, *ccfv_SIG*) *IntI face_of_aff_dim_lt face_of_imp_convex* [*of X*] *face_of_imp_convex* [*of Y*] *face_of_trans ff*)
qed
obtain *g* **where** *continuous_on* ($\bigcup(\mathcal{G} \cup$ *?Faces*)) *g*
g ' $\bigcup(\mathcal{G} \cup$ *?Faces*$) \subseteq$ *rel_frontier T*
$(\forall x \in \bigcup(\mathcal{G} \cup$ *?Faces*$) \cap$
$\bigcup(\mathcal{G} \cup \{D.\ \exists C{\in}\mathcal{F}.\ D$ *face_of* $C \wedge$ *aff_dim* $D < p\}).\ g\ x = h\ x$)
by (*rule exE* [*OF extending_maps_Union* [*OF fin extendh clo K*]], *blast*+)
then show *?case*
by (*simp add*: *intle local.heq* [*symmetric*], *blast*)
qed
have *eq*: $\bigcup(\mathcal{G} \cup \{D.\ \exists C \in \mathcal{F}.\ D$ *face_of* $C \wedge$ *aff_dim* $D < i\}) = \bigcup\mathcal{F}$
proof
show $\bigcup(\mathcal{G} \cup \{D.\ \exists C{\in}\mathcal{F}.\ D$ *face_of* $C \wedge$ *aff_dim* $D <$ *int* $i\}) \subseteq \bigcup\mathcal{F}$
using ‹$\mathcal{G} \subseteq \mathcal{F}$› *face_of_imp_subset* **by** *fastforce*
show $\bigcup\mathcal{F} \subseteq \bigcup(\mathcal{G} \cup \{D.\ \exists C{\in}\mathcal{F}.\ D$ *face_of* $C \wedge$ *aff_dim* $D < i\})$
proof (*rule Union_mono*)
show $\mathcal{F} \subseteq \mathcal{G} \cup \{D.\ \exists C{\in}\mathcal{F}.\ D$ *face_of* $C \wedge$ *aff_dim* $D <$ *int* $i\}$
using *face* **by** (*fastforce simp*: *aff i*)
qed
qed
have *int* $i \leq$ *aff_dim* T **by** (*simp add*: *i*)
then show *?thesis*
using *extendf* [*of i*] **unfolding** *eq* **by** (*metis that*)
qed

lemma *extend_map_lemma_cofinite0*:
assumes *finite* $\mathcal{F}$
and *pairwise* $(\lambda S\ T.\ S \cap T \subseteq K)\ \mathcal{F}$
and $\bigwedge S.\ S \in \mathcal{F} \Longrightarrow \exists a\ g.\ a \notin U \wedge$ *continuous_on* $(S - \{a\})\ g \wedge g$ ' $(S - \{a\}) \subseteq T \wedge (\forall x \in S \cap K.\ g\ x = h\ x)$
and $\bigwedge S.\ S \in \mathcal{F} \Longrightarrow$ *closed* S
shows $\exists C\ g.$ *finite* $C \wedge$ *disjnt* $C\ U \wedge$ *card* $C \leq$ *card* $\mathcal{F} \wedge$
continuous_on $(\bigcup\mathcal{F} - C)\ g \wedge g$ ' $(\bigcup\mathcal{F} - C) \subseteq T$
$\wedge\ (\forall x \in (\bigcup\mathcal{F} - C) \cap K.\ g\ x = h\ x)$
using *assms*
proof *induction*
case *empty* **then show** *?case*
by *force*
next
case (*insert X* $\mathcal{F}$)
then have *closed X* **and** *clo*: $\bigwedge X.\ X \in \mathcal{F} \Longrightarrow$ *closed X*
and $\mathcal{F}$: $\bigwedge S.\ S \in \mathcal{F} \Longrightarrow \exists a\ g.\ a \notin U \wedge$ *continuous_on* $(S - \{a\})\ g \wedge g$ ' $(S - \{a\}) \subseteq T \wedge (\forall x \in S \cap K.\ g\ x = h\ x)$
and *pwX*: $\bigwedge Y.\ Y \in \mathcal{F} \wedge Y \neq X \longrightarrow X \cap Y \subseteq K \wedge Y \cap X \subseteq K$
and *pwF*: *pairwise* $(\lambda\ S\ T.\ S \cap T \subseteq K)\ \mathcal{F}$

```
    by (simp_all add: pairwise_insert)
  obtain C g where C: finite C disjnt C U card C ≤ card ℱ
              and contg: continuous_on (⋃ℱ − C) g
              and gim: g ‘ (⋃ℱ − C) ⊆ T
              and gh: ⋀x. x ∈ (⋃ℱ − C) ∩ K ⟹ g x = h x
    using insert.IH [OF pwF ℱ clo] by auto
  obtain a f where a ∉ U
              and contf: continuous_on (X − {a}) f
              and fim: f ‘ (X − {a}) ⊆ T
              and fh: (∀x ∈ X ∩ K. f x = h x)
    using insert.prems by (meson insertI1)
  show ?case
  proof (intro exI conjI)
    show finite (insert a C)
      by (simp add: C)
    show disjnt (insert a C) U
      using C ‹a ∉ U› by simp
    show card (insert a C) ≤ card (insert X ℱ)
      by (simp add: C card_insert_if insert.hyps le_SucI)
    have closed (⋃ℱ)
      using clo insert.hyps by blast
    have continuous_on (X − insert a C) f
      using contf by (force simp: elim: continuous_on_subset)
    moreover have continuous_on (⋃ ℱ − insert a C) g
      using contg by (force simp: elim: continuous_on_subset)
    ultimately
     have continuous_on (X − insert a C ∪ (⋃ℱ − insert a C)) (λx. if x ∈ X
then f x else g x)
      apply (intro continuous_on_cases_local; simp add: closedin_closed)
        using ‹closed X› apply blast
        using ‹closed (⋃ℱ)› apply blast
        using fh gh insert.hyps pwX by fastforce
    then show continuous_on (⋃(insert X ℱ) − insert a C) (λa. if a ∈ X then f
a else g a)
      by (blast intro: continuous_on_subset)
    show ∀x∈(⋃(insert X ℱ) − insert a C) ∩ K. (if x ∈ X then f x else g x) = h
x
      using gh by (auto simp: fh)
    show (λa. if a ∈ X then f a else g a) ‘ (⋃(insert X ℱ) − insert a C) ⊆ T
      using fim gim by auto force
  qed
qed


lemma extend_map_lemma_cofinite1:
assumes finite ℱ
    and ℱ: ⋀X. X ∈ ℱ ⟹ ∃a g. a ∉ U ∧ continuous_on (X − {a}) g ∧ g ‘ (X
− {a}) ⊆ T ∧ (∀x ∈ X ∩ K. g x = h x)
    and clo: ⋀X. X ∈ ℱ ⟹ closed X
```

and K: $\bigwedge X\ Y.\ [\![X \in \mathcal{F};\ Y \in \mathcal{F};\ \neg\ X \subseteq Y;\ \neg\ Y \subseteq X]\!] \Longrightarrow X \cap Y \subseteq K$
obtains $C\ g$ **where** *finite C disjnt C U card C* $\leq$ *card* $\mathcal{F}$ *continuous_on* $(\bigcup \mathcal{F} - C)\ g$
$g\ `\ (\bigcup \mathcal{F} - C) \subseteq T$
$\bigwedge x.\ x \in (\bigcup \mathcal{F} - C) \cap K \Longrightarrow g\ x = h\ x$
proof −
let $?\mathcal{F} = \{X \in \mathcal{F}.\ \forall\ Y{\in}\mathcal{F}.\ \neg\ X \subset Y\}$
have [*simp*]: $\bigcup ?\mathcal{F} = \bigcup \mathcal{F}$
by (*simp add*: *Union_maximal_sets assms*)
have *fin*: *finite* $?\mathcal{F}$
by (*force intro*: *finite_subset* [*OF _* ‹*finite* $\mathcal{F}$›])
have *pw*: *pairwise* $(\lambda\ S\ T.\ S \cap T \subseteq K)$ $?\mathcal{F}$
by (*simp add*: *pairwise_def*) (*metis K psubsetI*)
have *card* $\{X \in \mathcal{F}.\ \forall\ Y{\in}\mathcal{F}.\ \neg\ X \subset Y\} \leq$ *card* $\mathcal{F}$
by (*simp add*: ‹*finite* $\mathcal{F}$› *card_mono*)
moreover
obtain $C\ g$ **where** *finite C* $\wedge$ *disjnt C U* $\wedge$ *card C* $\leq$ *card* $?\mathcal{F}$ $\wedge$
continuous_on $(\bigcup ?\mathcal{F} - C)\ g \wedge g\ `\ (\bigcup ?\mathcal{F} - C) \subseteq T$
$\wedge\ (\forall\, x \in (\bigcup ?\mathcal{F} - C) \cap K.\ g\ x = h\ x)$
using *extend_map_lemma_cofinite0* [*OF fin pw*, *of U T h*] **by** (*fastforce intro*!: *clo* $\mathcal{F}$)
ultimately show *?thesis*
by (*rule_tac C*=*C* **and** *g*=*g* **in** *that*) *auto*
qed

lemma *extend_map_lemma_cofinite*:
assumes *finite* $\mathcal{F}$ $\mathcal{G} \subseteq \mathcal{F}$ **and** T: *convex T bounded T*
and *poly*: $\bigwedge X.\ X \in \mathcal{F} \Longrightarrow$ *polytope X*
and *contf*: *continuous_on* $(\bigcup \mathcal{G})\ f$ **and** *fim*: $f\ `\ (\bigcup \mathcal{G}) \subseteq$ *rel_frontier T*
and *face*: $\bigwedge X\ Y.\ [\![X \in \mathcal{F};\ Y \in \mathcal{F}]\!] \Longrightarrow (X \cap Y)$ *face_of X*
and *aff*: $\bigwedge X.\ X \in \mathcal{F} - \mathcal{G} \Longrightarrow$ *aff_dim X* $\leq$ *aff_dim T*
obtains $C\ g$ **where**
finite C disjnt C $(\bigcup \mathcal{G})$ *card C* $\leq$ *card* $\mathcal{F}$ *continuous_on* $(\bigcup \mathcal{F} - C)\ g$
$g\ `\ (\bigcup\ \mathcal{F} - C) \subseteq$ *rel_frontier T* $\bigwedge x.\ x \in \bigcup \mathcal{G} \Longrightarrow g\ x = f\ x$
proof −
define $\mathcal{H}$ **where** $\mathcal{H} \equiv \mathcal{G} \cup \{D.\ \exists\, C \in \mathcal{F} - \mathcal{G}.\ D$ *face_of C* $\wedge$ *aff_dim D* $<$ *aff_dim T*$\}$
have *finite* $\mathcal{G}$
using *assms finite_subset* **by** *blast*
have $*$: *finite* $(\bigcup \{\{D.\ D$ *face_of C*$\}\ |C.\ C \in \mathcal{F}\})$
using *finite_polytope_faces poly* ‹*finite* $\mathcal{F}$› **by** *force*
then have *finite* $\mathcal{H}$
by (*auto simp*: $\mathcal{H}$*_def* ‹*finite* $\mathcal{G}$› *intro*: *finite_subset* [*OF _* $*$])
have *face*′: $\bigwedge S\ T.\ [\![S \in \mathcal{F};\ T \in \mathcal{F}]\!] \Longrightarrow (S \cap T)$ *face_of S* $\wedge$ $(S \cap T)$ *face_of T*
by (*metis face inf_commute*)
have $*$: $\bigwedge X\ Y.\ [\![X \in \mathcal{H};\ Y \in \mathcal{H}]\!] \Longrightarrow X \cap Y$ *face_of X*
by (*simp add*: $\mathcal{H}$*_def*) (*smt* (*verit*) ‹$\mathcal{G} \subseteq \mathcal{F}$› *DiffE face*′ *face_of_Int_subface in_mono inf.idem*)

obtain h **where** *conth*: *continuous_on* $(\bigcup \mathcal{H})$ h **and** *him*: h ' $(\bigcup \mathcal{H}) \subseteq$ *rel_frontier* T
and *hf*: $\bigwedge x.\ x \in \bigcup \mathcal{G} \Longrightarrow h\ x = f\ x$
proof (*rule extend_map_lemma* [*OF* ‹*finite* $\mathcal{H}$›] [*unfolded* $\mathcal{H}$*_def*] *Un_upper1* T])
show $\bigwedge X.\ [\![X \in \mathcal{G} \cup \{D.\ \exists\, C{\in}\mathcal{F} - \mathcal{G}.\ D$ *face_of* $C \wedge$ *aff_dim* $D <$ *aff_dim* $T\}]\!] \Longrightarrow$ *polytope* X
using ‹$\mathcal{G} \subseteq \mathcal{F}$› *face_of_polytope_polytope poly* **by** *fastforce*
qed (*use* $*$ $\mathcal{H}$*_def contf fim* **in** *auto*)
have *bounded* $(\bigcup \mathcal{G})$
using ‹*finite* $\mathcal{G}$› ‹$\mathcal{G} \subseteq \mathcal{F}$› *poly polytope_imp_bounded* **by** *blast*
then have $\bigcup \mathcal{G} \neq$ *UNIV*
by *auto*
then obtain a **where** a: $a \notin \bigcup \mathcal{G}$
by *blast*
have $\mathcal{F}$: $\exists\, a\ g.\ a \notin \bigcup \mathcal{G} \wedge$ *continuous_on* $(D - \{a\})\ g \wedge$
g ' $(D - \{a\}) \subseteq$ *rel_frontier* $T \wedge (\forall\, x \in D \cap \bigcup \mathcal{H}.\ g\ x = h\ x)$
if $D \in \mathcal{F}$ **for** D
proof (*cases* $D \subseteq \bigcup \mathcal{H}$)
case *True*
then have h ' $(D - \{a\}) \subseteq$ *rel_frontier* T *continuous_on* $(D - \{a\})\ h$
using *him* **by** (*blast intro*!: ‹$a \notin \bigcup \mathcal{G}$› *continuous_on_subset* [*OF conth*])+
then show *?thesis*
using a **by** *blast*
next
case *False*
note *D_not_subset* = *False*
show *?thesis*
proof (*cases* $D \in \mathcal{G}$)
case *True*
with *D_not_subset* **show** *?thesis*
by (*auto simp*: $\mathcal{H}$*_def*)
next
case *False*
then have *affD*: *aff_dim* $D \leq$ *aff_dim* T
by (*simp add*: ‹$D \in \mathcal{F}$› *aff*)
show *?thesis*
proof (*cases rel_interior* $D = \{\}$)
case *True*
with ‹$D \in \mathcal{F}$› *poly* a **show** *?thesis*
by (*force simp*: *rel_interior_eq_empty polytope_imp_convex*)
next
case *False*
then obtain b **where** *brelD*: $b \in$ *rel_interior* D
by *blast*
have *polyhedron* D
by (*simp add*: *poly polytope_imp_polyhedron that*)
have *rel_frontier* D *retract_of affine hull* $D - \{b\}$
by (*simp add*: *rel_frontier_retract_of_punctured_affine_hull poly poly-*

```
tope_imp_bounded polytope_imp_convex that brelD)
      then obtain r where relfD: rel_frontier D ⊆ affine hull D − {b}
                      and contr: continuous_on (affine hull D − {b}) r
                      and rim: r ‘ (affine hull D − {b}) ⊆ rel_frontier D
                      and rid: ⋀x. x ∈ rel_frontier D ⟹ r x = x
        by (auto simp: retract_of_def retraction_def)
      show ?thesis
      proof (intro exI conjI ballI)
        show b ∉ ⋃𝒢
        proof clarify
          fix E
          assume b ∈ E E ∈ 𝒢
          then have E ∩ D face_of E ∧ E ∩ D face_of D
            using ‹𝒢 ⊆ ℱ› face' that by auto
       with face_of_subset_rel_frontier ‹E ∈ 𝒢› ‹b ∈ E› brelD rel_interior_subset
[of D]
              D_not_subset rel_frontier_def ℋ_def
          show False
            by blast
        qed
        have r ‘ (D − {b}) ⊆ r ‘ (affine hull D − {b})
          by (simp add: Diff_mono hull_subset image_mono)
        also have ... ⊆ rel_frontier D
          by (rule rim)
        also have ... ⊆ ⋃{E. E face_of D ∧ aff_dim E < aff_dim T}
          using affD
           by (force simp: rel_frontier_of_polyhedron [OF ‹polyhedron D›]
facet_of_def)
        also have ... ⊆ ⋃(ℋ)
          using D_not_subset ℋ_def that by fastforce
        finally have rsub: r ‘ (D − {b}) ⊆ ⋃(ℋ) .
        show continuous_on (D − {b}) (h ∘ r)
        proof (rule continuous_on_compose)
          show continuous_on (D − {b}) r
            by (meson Diff_mono continuous_on_subset contr hull_subset or-
der_refl)
          show continuous_on (r ‘ (D − {b})) h
           by (simp add: Diff_mono hull_subset continuous_on_subset [OF conth
rsub])
        qed
        show (h ∘ r) ‘ (D − {b}) ⊆ rel_frontier T
          using brelD him rsub by fastforce
        show (h ∘ r) x = h x if x: x ∈ D ∩ ⋃ℋ for x
        proof −
          consider A where x ∈ D A ∈ 𝒢 x ∈ A
            | A B where x ∈ D A face_of B B ∈ ℱ B ∉ 𝒢 aff_dim A < aff_dim
T x ∈ A
            using x by (auto simp: ℋ_def)
          then have xrel: x ∈ rel_frontier D
```

```
          proof cases
            case 1 show ?thesis
            proof (rule face_of_subset_rel_frontier [THEN subsetD])
              show D ∩ A face_of D
                using ‹A ∈ 𝒢› ‹𝒢 ⊆ ℱ› face ‹D ∈ ℱ› by blast
              show D ∩ A ≠ D
                using ‹A ∈ 𝒢› D_not_subset ℋ_def by blast
            qed (auto simp: 1)
          next
            case 2 show ?thesis
            proof (rule face_of_subset_rel_frontier [THEN subsetD])
              have D face_of D
               by (simp add: ‹polyhedron D› polyhedron_imp_convex face_of_refl)
              then show D ∩ A face_of D
               by (meson 2(2) 2(3) ‹D ∈ ℱ› face' face_of_Int_Int face_of_face)
              show D ∩ A ≠ D
                using 2 D_not_subset ℋ_def by blast
            qed (auto simp: 2)
          qed
          show ?thesis
            by (simp add: rid xrel)
        qed
      qed
    qed
  qed
  have clo: ⋀S. S ∈ ℱ ⟹ closed S
    by (simp add: poly polytope_imp_closed)
  obtain C g where finite C disjnt C (⋃𝒢) card C ≤ card ℱ continuous_on (⋃ℱ
− C) g
                   g ‘ (⋃ℱ − C) ⊆ rel_frontier T
              and gh: ⋀x. x ∈ (⋃ℱ − C) ∩ ⋃ℋ ⟹ g x = h x
  proof (rule extend_map_lemma_cofinite1 [OF ‹finite ℱ› ℱ clo])
    show X ∩ Y ⊆ ⋃ℋ if XY: X ∈ ℱ Y ∈ ℱ and ¬ X ⊆ Y ¬ Y ⊆ X for X Y
    proof (cases X ∈ 𝒢)
      case True
      then show ?thesis
        by (auto simp: ℋ_def)
    next
      case False
      have X ∩ Y ≠ X
        using ‹¬ X ⊆ Y› by blast
      with XY
      show ?thesis
        by (clarsimp simp: ℋ_def)
           (metis Diff_iff Int_iff aff antisym_conv face face_of_aff_dim_lt
face_of_refl
            not_le poly polytope_imp_convex)
    qed
```

qed (*blast*)+
with ‹$\mathcal{G} \subseteq \mathcal{F}$› **show** *?thesis*
by (*rule_tac C=C* **and** *g=g* **in** *that*) (*auto simp*: *disjnt_def hf* [*symmetric*] $\mathcal{H}$*_def intro*!: *gh*)
qed

The next two proofs are similar

theorem *extend_map_cell_complex_to_sphere*:
assumes *finite* $\mathcal{F}$ **and** *S*: $S \subseteq \bigcup \mathcal{F}$ *closed S* **and** *T*: *convex T bounded T*
and *poly*: $\bigwedge X.\ X \in \mathcal{F} \Longrightarrow$ *polytope X*
and *aff*: $\bigwedge X.\ X \in \mathcal{F} \Longrightarrow$ *aff_dim X* < *aff_dim T*
and *face*: $\bigwedge X\ Y.\ [\![X \in \mathcal{F};\ Y \in \mathcal{F}]\!] \Longrightarrow (X \cap Y)$ *face_of X*
and *contf*: *continuous_on S f* **and** *fim*: $f \in S \to$ *rel_frontier T*
obtains *g* **where** *continuous_on* $(\bigcup \mathcal{F})$ *g*
g ‘ $(\bigcup \mathcal{F}) \subseteq$ *rel_frontier T* $\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
proof −
obtain *V g* **where** $S \subseteq V$ *open V continuous_on V g* **and** *gim*: $g \in V \to$ *rel_frontier T* **and** *gf*: $\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
using *neighbourhood_extension_into_ANR* [*OF contf fim* _ ‹*closed S*›] *ANR_rel_frontier_convex T* **by** *blast*
have *compact S*
by (*meson assms compact_Union poly polytope_imp_compact seq_compact_closed_subset seq_compact_eq_compact*)
then obtain *d* **where** $d > 0$ **and** *d*: $\bigwedge x\ y.\ [\![x \in S;\ y \in -\ V]\!] \Longrightarrow d \le$ *dist x y*
using *separate_compact_closed* [*of S* $-V$] ‹*open V*› ‹$S \subseteq V$› **by** *force*
obtain $\mathcal{G}$ **where** *finite* $\mathcal{G}$ $\bigcup \mathcal{G} = \bigcup \mathcal{F}$
and *diaG*: $\bigwedge X.\ X \in \mathcal{G} \Longrightarrow$ *diameter X* < *d*
and *polyG*: $\bigwedge X.\ X \in \mathcal{G} \Longrightarrow$ *polytope X*
and *affG*: $\bigwedge X.\ X \in \mathcal{G} \Longrightarrow$ *aff_dim X* ≤ *aff_dim T* − *1*
and *faceG*: $\bigwedge X\ Y.\ [\![X \in \mathcal{G};\ Y \in \mathcal{G}]\!] \Longrightarrow X \cap Y$ *face_of X*
proof (*rule cell_complex_subdivision_exists* [*OF* ‹*d*>*0*› ‹*finite* $\mathcal{F}$› *poly* _ *face*])
show $\bigwedge X.\ X \in \mathcal{F} \Longrightarrow$ *aff_dim X* ≤ *aff_dim T* − *1*
by (*simp add*: *aff*)
qed *auto*
obtain *h* **where** *conth*: *continuous_on* $(\bigcup \mathcal{G})$ *h* **and** *him*: *h* ‘ $\bigcup \mathcal{G} \subseteq$ *rel_frontier T* **and** *hg*: $\bigwedge x.\ x \in \bigcup(\mathcal{G} \cap$ *Pow V*$) \Longrightarrow h\ x = g\ x$
proof (*rule extend_map_lemma* [*of* $\mathcal{G}$ $\mathcal{G} \cap$ *Pow V T g*])
show *continuous_on* $(\bigcup(\mathcal{G} \cap$ *Pow V*$))$ *g*
by (*metis Union_Int_subset Union_Pow_eq* ‹*continuous_on V g*› *continuous_on_subset le_inf_iff*)
qed (*use* ‹*finite* $\mathcal{G}$› *T polyG affG faceG gim* **in** *fastforce*)+
show *?thesis*
proof
show *continuous_on* $(\bigcup \mathcal{F})$ *h*
using ‹$\bigcup \mathcal{G} = \bigcup \mathcal{F}$› *conth* **by** *auto*
show *h* ‘ $\bigcup \mathcal{F} \subseteq$ *rel_frontier T*
using ‹$\bigcup \mathcal{G} = \bigcup \mathcal{F}$› *him* **by** *auto*
show $h\ x = f\ x$ **if** $x \in S$ **for** *x*
proof −

```
      have x ∈ ⋃𝒢
        using ‹⋃𝒢 = ⋃ℱ› ‹S ⊆ ⋃ℱ› that by auto
      then obtain X where x ∈ X X ∈ 𝒢 by blast
      then have diameter X < d bounded X
        by (auto simp: diaG ‹X ∈ 𝒢› polyG polytope_imp_bounded)
       then have X ⊆ V using d [OF ‹x ∈ S›] diameter_bounded_bound [OF
‹bounded X› ‹x ∈ X›]
        by fastforce
      have h x = g x
        using ‹X ∈ 𝒢› ‹X ⊆ V› ‹x ∈ X› hg by auto
      also have ... = f x
        by (simp add: gf that)
      finally show h x = f x .
    qed
  qed
qed


theorem extend_map_cell_complex_to_sphere_cofinite:
  assumes finite ℱ and S: S ⊆ ⋃ℱ closed S and T: convex T bounded T
      and poly: ⋀X. X ∈ ℱ ⟹ polytope X
      and aff: ⋀X. X ∈ ℱ ⟹ aff_dim X ≤ aff_dim T
      and face: ⋀X Y. ⟦X ∈ ℱ; Y ∈ ℱ⟧ ⟹ (X ∩ Y) face_of X
      and contf: continuous_on S f and fim: f ∈ S → rel_frontier T
  obtains C g where finite C disjnt C S continuous_on (⋃ℱ − C) g
     g ‘ (⋃ℱ − C) ⊆ rel_frontier T ⋀x. x ∈ S ⟹ g x = f x
proof −
  obtain V g where S ⊆ V open V continuous_on V g and gim: g ∈ V →
rel_frontier T and gf: ⋀x. x ∈ S ⟹ g x = f x
   using neighbourhood_extension_into_ANR [OF contf fim _ ‹closed S›] ANR_rel_frontier_convex
T by blast
  have compact S
   by (meson assms compact_Union poly polytope_imp_compact seq_compact_closed_subset
seq_compact_eq_compact)
  then obtain d where d > 0 and d: ⋀x y. ⟦x ∈ S; y ∈ − V⟧ ⟹ d ≤ dist x y
    using separate_compact_closed [of S − V] ‹open V› ‹S ⊆ V› by force
  obtain 𝒢 where finite 𝒢 ⋃𝒢 = ⋃ℱ
            and diaG: ⋀X. X ∈ 𝒢 ⟹ diameter X < d
            and polyG: ⋀X. X ∈ 𝒢 ⟹ polytope X
            and affG: ⋀X. X ∈ 𝒢 ⟹ aff_dim X ≤ aff_dim T
            and faceG: ⋀X Y. ⟦X ∈ 𝒢; Y ∈ 𝒢⟧ ⟹ X ∩ Y face_of X
    by (rule cell_complex_subdivision_exists [OF ‹d>0› ‹finite ℱ› poly aff face])
auto
  obtain C h where finite C and dis: disjnt C (⋃(𝒢 ∩ Pow V))
              and card: card C ≤ card 𝒢 and conth: continuous_on (⋃𝒢 − C) h
              and him: h ‘ (⋃𝒢 − C) ⊆ rel_frontier T
              and hg: ⋀x. x ∈ ⋃(𝒢 ∩ Pow V) ⟹ h x = g x
  proof (rule extend_map_lemma_cofinite [of 𝒢 𝒢 ∩ Pow V T g])
    show continuous_on (⋃(𝒢 ∩ Pow V)) g
```

by (*metis Union_Int_subset Union_Pow_eq* ‹*continuous_on V g*› *continuous_on_subset le_inf_iff*)
show *g* ‘ $\bigcup(\mathcal{G} \cap Pow\ V) \subseteq$ *rel_frontier T*
using *gim* **by** *force*
qed (*auto intro*: ‹*finite* $\mathcal{G}$› *T polyG affG dest*: *faceG*)
have *Ssub*: $S \subseteq \bigcup(\mathcal{G} \cap Pow\ V)$
proof
fix *x*
assume $x \in S$
then have $x \in \bigcup\mathcal{G}$
using ‹$\bigcup\mathcal{G} = \bigcup\mathcal{F}$› ‹$S \subseteq \bigcup\mathcal{F}$› **by** *auto*
then obtain *X* **where** $x \in X\ X \in \mathcal{G}$ **by** *blast*
then have *diameter X* < *d bounded X*
by (*auto simp*: *diaG* ‹$X \in \mathcal{G}$› *polyG polytope_imp_bounded*)
then have $X \subseteq V$ **using** *d* [*OF* ‹$x \in S$›] *diameter_bounded_bound* [*OF* ‹*bounded X*› ‹$x \in X$›]
by *fastforce*
then show $x \in \bigcup(\mathcal{G} \cap Pow\ V)$
using ‹$X \in \mathcal{G}$› ‹$x \in X$› **by** *blast*
qed
show *?thesis*
proof
show *continuous_on* ($\bigcup\mathcal{F}-C$) *h*
using ‹$\bigcup\mathcal{G} = \bigcup\mathcal{F}$› *conth* **by** *auto*
show *h* ‘ ($\bigcup\mathcal{F}$ − *C*) ⊆ *rel_frontier T*
using ‹$\bigcup\mathcal{G} = \bigcup\mathcal{F}$› *him* **by** *auto*
show *h x* = *f x* **if** $x \in S$ **for** *x*
proof −
have *h x* = *g x*
using *Ssub hg that* **by** *blast*
also have ... = *f x*
by (*simp add*: *gf that*)
finally show *h x* = *f x* **.**
qed
show *disjnt C S*
using *dis Ssub* **by** (*meson disjnt_iff subset_eq*)
qed (*intro* ‹*finite C*›)
qed

6.36.3 Special cases and corollaries involving spheres

proposition *extend_map_affine_to_sphere_cofinite_simple*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *compact S convex U bounded U*
and *aff*: *aff_dim T* ≤ *aff_dim U*
and *S* ⊆ *T* **and** *contf*: *continuous_on S f*
and *fim*: *f* ∈ *S* → *rel_frontier U*
obtains *K g* **where** *finite K K* ⊆ *T disjnt K S continuous_on* (*T* − *K*) *g*
g ∈ (*T* − *K*) → *rel_frontier U*

```
                    ⋀x. x ∈ S ⟹ g x = f x
proof −
  have ∃K g. finite K ∧ disjnt K S ∧ continuous_on (T − K) g ∧
             g ` (T − K) → rel_frontier U ∧ (∀x ∈ S. g x = f x)
       if affine T S ⊆ T and aff: aff_dim T ≤ aff_dim U  for T
  proof (cases S = {})
    case True
    show ?thesis
    proof (cases rel_frontier U = {})
      case True
      with ‹bounded U› have aff_dim U ≤ 0
        using affine_bounded_eq_lowdim rel_frontier_eq_empty by auto
      with aff have aff_dim T ≤ 0 by auto
      then obtain a where T ⊆ {a}
        using ‹affine T› affine_bounded_eq_lowdim affine_bounded_eq_trivial by
auto
      then show ?thesis
        using ‹S = {}› fim
          by (metis Diff_cancel contf disjnt_empty2 finite.emptyI finite_insert fi-
nite_subset)
    next
      case False
      then obtain a where a ∈ rel_frontier U
        by auto
      then show ?thesis
        using continuous_on_const [of _ a] ‹S = {}› by force
    qed
  next
    case False
    have bounded S
      by (simp add: ‹compact S› compact_imp_bounded)
    then obtain b where b: S ⊆ cbox (−b) b
      using bounded_subset_cbox_symmetric by blast
    define bbox where bbox ≡ cbox (−(b+One)) (b+One)
    have cbox (−b) b ⊆ bbox
      by (auto simp: bbox_def algebra_simps intro!: subset_box_imp)
    with b ‹S ⊆ T› have S ⊆ bbox ∩ T
      by auto
    then have Ssub: S ⊆ ⋃{bbox ∩ T}
      by auto
    then have aff_dim (bbox ∩ T) ≤ aff_dim U
      by (metis aff aff_dim_subset inf_commute inf_le1 order_trans)
    obtain K g where K: finite K disjnt K S
                  and contg: continuous_on (⋃{bbox ∩ T} − K) g
                  and gim: g ` (⋃{bbox ∩ T} − K) ⊆ rel_frontier U
                  and gf: ⋀x. x ∈ S ⟹ g x = f x
    proof (rule extend_map_cell_complex_to_sphere_cofinite
            [OF _ Ssub _ ‹convex U› ‹bounded U› _ _ _ contf fim])
      show closed S
```

```
      using ‹compact S› compact_eq_bounded_closed by auto
    show poly: ⋀X. X ∈ {bbox ∩ T} ⟹ polytope X
    by (simp add: polytope_Int_polyhedron bbox_def polytope_interval affine_imp_polyhedron
‹affine T›)
    show ⋀X Y. ⟦X ∈ {bbox ∩ T}; Y ∈ {bbox ∩ T}⟧ ⟹ X ∩ Y face_of X
      by (simp add:poly face_of_refl polytope_imp_convex)
    show ⋀X. X ∈ {bbox ∩ T} ⟹ aff_dim X ≤ aff_dim U
      by (simp add: ‹aff_dim (bbox ∩ T) ≤ aff_dim U›)
  qed auto
  define fro where fro ≡ λd. frontier(cbox (−(b + d *R One)) (b + d *R One))
  obtain d where d12: 1/2 ≤ d d ≤ 1 and dd: disjnt K (fro d)
  proof (rule disjoint_family_elem_disjnt [OF _ ‹finite K›])
    show infinite {1/2..1::real}
      by (simp add: infinite_Icc)
    have dis1: disjnt (fro x) (fro y) if x<y for x y
     by (auto simp: algebra_simps that subset_box_imp disjnt_Diff1 frontier_def
fro_def)
    then show disjoint_family_on fro {1/2..1}
      by (auto simp: disjoint_family_on_def disjnt_def neq_iff)
  qed auto
  define c where c ≡ b + d *R One
  have cbsub: cbox (−b) b ⊆ box (−c) c cbox (−b) b ⊆ cbox (−c) c  cbox (−c)
c ⊆ bbox
    using d12 by (auto simp: algebra_simps subset_box_imp c_def bbox_def)
  have clo_cbT: closed (cbox (− c) c ∩ T)
    by (simp add: affine_closed closed_Int closed_cbox ‹affine T›)
  have cpT_ne: cbox (− c) c ∩ T ≠ {}
    using ‹S ≠ {}› b cbsub(2) ‹S ⊆ T› by fastforce
  have closest_point (cbox (− c) c ∩ T) x ∉ K if x ∈ T x ∉ K for x
  proof (cases x ∈ cbox (−c) c)
    case True with that show ?thesis
      by (simp add: closest_point_self)
  next
    case False
    have int_ne: interior (cbox (−c) c) ∩ T ≠ {}
      using ‹S ≠ {}› ‹S ⊆ T› b ‹cbox (− b) b ⊆ box (− c) c› by force
    have convex T
      by (meson ‹affine T› affine_imp_convex)
    then have x ∈ affine hull (cbox (− c) c ∩ T)
        by (metis Int_commute Int_iff ‹S ≠ {}› ‹S ⊆ T› cbsub(1) ‹x ∈ T›
affine_hull_convex_Int_nonempty_interior all_not_in_conv b hull_inc inf.orderE
interior_cbox)
    then have x ∈ affine hull (cbox (− c) c ∩ T) − rel_interior (cbox (− c) c
∩ T)
      by (meson DiffI False Int_iff rel_interior_subset subsetCE)
    then have closest_point (cbox (− c) c ∩ T) x ∈ rel_frontier (cbox (− c) c
∩ T)
      by (rule closest_point_in_rel_frontier [OF clo_cbT cpT_ne])
    moreover have (rel_frontier (cbox (− c) c ∩ T)) ⊆ fro d
```

```
      by (subst convex_affine_rel_frontier_Int [OF _ ‹affine T› int_ne]) (auto
simp: fro_def c_def)
     ultimately show ?thesis
      using dd  by (force simp: disjnt_def)
   qed
   then have cpt_subset: closest_point (cbox (− c) c ∩ T) ‘ (T − K) ⊆ ⋃{bbox
∩ T} − K
     using closest_point_in_set [OF clo_cbT cpT_ne] cbsub(3) by force
   show ?thesis
   proof (intro conjI ballI exI)
     have continuous_on (T − K) (closest_point (cbox (− c) c ∩ T))
     proof (rule continuous_on_closest_point)
       show convex (cbox (− c) c ∩ T)
         by (simp add: affine_imp_convex convex_Int ‹affine T›)
       show closed (cbox (− c) c ∩ T)
         using clo_cbT by blast
       show cbox (− c) c ∩ T ≠ {}
         using ‹S ≠ {}› cbsub(2) b that by auto
     qed
     then show continuous_on (T − K) (g ∘ closest_point (cbox (− c) c ∩ T))
     by (metis continuous_on_compose continuous_on_subset [OF contg cpt_subset])
     have (g ∘ closest_point (cbox (− c) c ∩ T)) ‘ (T − K) ⊆ g ‘ (⋃{bbox ∩ T}
− K)
       by (metis image_comp image_mono cpt_subset)
     also have ... ⊆ rel_frontier U
       by (rule gim)
    finally show (g ∘ closest_point (cbox (− c) c ∩ T)) ∈ (T − K) → rel_frontier
U
       by blast
     show (g ∘ closest_point (cbox (− c) c ∩ T)) x = f x if x ∈ S for x
     proof −
       have (g ∘ closest_point (cbox (− c) c ∩ T)) x = g x
         unfolding o_def
         by (metis IntI ‹S ⊆ T› b cbsub(2) closest_point_self subset_eq that)
       also have ... = f x
         by (simp add: that gf)
       finally show ?thesis .
     qed
   qed (auto simp: K)
 qed
 then obtain K g where finite K disjnt K S
             and contg: continuous_on (affine hull T − K) g
             and gim:  g ∈ (affine hull T − K) → rel_frontier U
             and gf:   ⋀x. x ∈ S ⟹ g x = f x
   by (metis aff affine_affine_hull aff_dim_affine_hull
          order_trans [OF ‹S ⊆ T› hull_subset [of T affine]])
 then obtain K g where finite K disjnt K S
             and contg: continuous_on (T − K) g
             and gim:  g ‘ (T − K) ⊆ rel_frontier U
```

and *gf*: $\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
by (*rule_tac K=K* **and** *g=g* **in** *that*) (*auto simp*: *hull_inc elim*: *continuous_on_subset*)
then show *?thesis*
by (*rule_tac K=K ∩ T* **and** *g=g* **in** *that*) (*auto simp*: *disjnt_iff Diff_Int contg*)
qed

6.36.4 Extending maps to spheres

lemma *extend_map_affine_to_sphere1*:
fixes *f* :: *'a::euclidean_space* ⇒ *'b::topological_space*
assumes *finite K affine U* **and** *contf*: *continuous_on* (*U* − *K*) *f*
and *fim*: $f \in (U - K) \to T$
and *comps*: $\bigwedge C.\ [\![C \in components(U - S);\ C \cap K \neq \{\}]\!] \Longrightarrow C \cap L \neq \{\}$
and *clo*: *closedin* (*top_of_set U*) *S* **and** *K*: *disjnt K S K* ⊆ *U*
obtains *g* **where** *continuous_on* (*U* − *L*) *g g* ∈ (*U* − *L*) → *T* $\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
proof (*cases K* = {})
case *True*
then show *?thesis*
by (*metis DiffD1 Diff_empty Diff_subset PiE Pi_I contf continuous_on_subset fim that*)
next
case *False*
have *S* ⊆ *U*
using *clo closedin_limpt* **by** *blast*
then have (*U* − *S*) ∩ *K* ≠ {}
by (*metis Diff_triv False Int_Diff K disjnt_def inf.absorb_iff2 inf_commute*)
then have $\bigcup$(*components* (*U* − *S*)) ∩ *K* ≠ {}
using *Union_components* **by** *simp*
then obtain *C0* **where** *C0*: *C0* ∈ *components* (*U* − *S*) *C0* ∩ *K* ≠ {}
by *blast*
have *convex U*
by (*simp add*: *affine_imp_convex* ‹*affine U*›)
then have *locally connected U*
by (*rule convex_imp_locally_connected*)
have ∃ *a g*. *a* ∈ *C* ∧ *a* ∈ *L* ∧ *continuous_on* (*S* ∪ (*C* − {*a*})) *g* ∧
g ‘ (*S* ∪ (*C* − {*a*})) ⊆ *T* ∧ (∀ *x* ∈ *S*. *g x* = *f x*)
if *C*: *C* ∈ *components* (*U* − *S*) **and** *CK*: *C* ∩ *K* ≠ {} **for** *C*
proof −
have *C* ⊆ *U*−*S C* ∩ *L* ≠ {}
by (*simp_all add*: *in_components_subset comps that*)
then obtain *a* **where** *a*: *a* ∈ *C a* ∈ *L* **by** *auto*
have *opeUC*: *openin* (*top_of_set U*) *C*
by (*metis C* ‹*locally connected U*› *clo closedin_def locally_connected_open_component topspace_euclidean_subtopology*)
then obtain *d* **where** *C* ⊆ *U* 0 < *d* **and** *d*: *cball a d* ∩ *U* ⊆ *C*
using *openin_contains_cball* **by** (*metis* ‹*a* ∈ *C*›)

then have *ball a d ∩ U ⊆ C*
by *auto*
obtain *h k* **where** *homhk*: *homeomorphism* (*S ∪ C*) (*S ∪ C*) *h k*
and *subC*: {*x*. (¬ (*h x* = *x* ∧ *k x* = *x*))} ⊆ *C*
and *bou*: *bounded* {*x*. (¬ (*h x* = *x* ∧ *k x* = *x*))}
and *hin*: ⋀*x*. *x* ∈ *C* ∩ *K* ⟹ *h x* ∈ *ball a d* ∩ *U*
proof (*rule homeomorphism_grouping_points_exists_gen* [*of C ball a d* ∩ *U C* ∩ *K S* ∪ *C*])
show *openin* (*top_of_set C*) (*ball a d* ∩ *U*)
by (*metis open_ball* ‹*C* ⊆ *U*› ‹*ball a d* ∩ *U* ⊆ *C*› *inf.absorb_iff2 inf.orderE inf_assoc open_openin openin_subtopology*)
show *openin* (*top_of_set* (*affine hull C*)) *C*
by (*metis* ‹*a* ∈ *C*› ‹*openin* (*top_of_set U*) *C*› *affine_hull_eq affine_hull_openin all_not_in_conv* ‹*affine U*›)
show *ball a d* ∩ *U* ≠ {}
using ‹0 < *d*› ‹*C* ⊆ *U*› ‹*a* ∈ *C*› **by** *force*
show *finite* (*C* ∩ *K*)
by (*simp add*: ‹*finite K*›)
show *S* ∪ *C* ⊆ *affine hull C*
by (*metis* ‹*S* ⊆ *U*› ‹*a* ∈ *C*› *affine_hull_eq affine_hull_openin assms*(2) *empty_iff hull_subset le_sup_iff opeUC*)
show *connected C*
by (*metis C in_components_connected*)
qed *auto*
have *a_BU*: *a* ∈ *ball a d* ∩ *U*
using ‹0 < *d*› ‹*C* ⊆ *U*› ‹*a* ∈ *C*› **by** *auto*
have *rel_frontier* (*cball a d* ∩ *U*) *retract_of* (*affine hull* (*cball a d* ∩ *U*) − {*a*})
proof (*rule rel_frontier_retract_of_punctured_affine_hull*)
show *bounded* (*cball a d* ∩ *U*) *convex* (*cball a d* ∩ *U*)
by (*auto simp*: ‹*convex U*› *convex_Int*)
show *a* ∈ *rel_interior* (*cball a d* ∩ *U*)
by (*metis* ‹*affine U*› *convex_cball empty_iff interior_cball a_BU rel_interior_convex_Int_affine*)
qed
moreover have *rel_frontier* (*cball a d* ∩ *U*) = *frontier* (*cball a d*) ∩ *U*
by (*metis a_BU* ‹*affine U*› *convex_affine_rel_frontier_Int convex_cball equals0D interior_cball*)
moreover have *affine hull* (*cball a d* ∩ *U*) = *U*
by (*metis* ‹*convex U*› *a_BU affine_hull_convex_Int_nonempty_interior affine_hull_eq* ‹*affine U*› *equals0D inf.commute interior_cball*)
ultimately have *frontier* (*cball a d*) ∩ *U retract_of* (*U* − {*a*})
by *metis*
then obtain *r* **where** *contr*: *continuous_on* (*U* − {*a*}) *r*
and *rim*: *r* ‘ (*U* − {*a*}) ⊆ *sphere a d* *r* ‘ (*U* − {*a*}) ⊆ *U*
and *req*: ⋀*x*. *x* ∈ *sphere a d* ∩ *U* ⟹ *r x* = *x*
using ‹*affine U*› **by** (*force simp*: *retract_of_def retraction_def hull_same*)
define *j* **where** *j* ≡ λ*x*. *if x* ∈ *ball a d then r x else x*
have *kj*: ⋀*x*. *x* ∈ *S* ⟹ *k* (*j x*) = *x*
using ‹*C* ⊆ *U* − *S*› ‹*S* ⊆ *U*› ‹*ball a d* ∩ *U* ⊆ *C*› *j_def subC* **by** *auto*

```
    have Uaeq: U − {a} = (cball a d − {a}) ∩ U ∪ (U − ball a d)
      using ‹0 < d› by auto
    have jim: j ‘ (S ∪ (C − {a})) ⊆ (S ∪ C) − ball a d
    proof clarify
      fix y  assume y ∈ S ∪ (C − {a})
      then have y ∈ U − {a}
        using ‹C ⊆ U − S› ‹S ⊆ U› ‹a ∈ C› by auto
      then have r y ∈ sphere a d
        using rim by auto
      then show j y ∈ S ∪ C − ball a d
        unfolding j_def
        using ‹r y ∈ sphere a d› ‹y ∈ U − {a}› ‹y ∈ S ∪ (C − {a})› d rim
      by (metis Diff_iff Int_iff Un_iff subsetD cball_diff_eq_sphere image_subset_iff)
    qed
    have contj: continuous_on (U − {a}) j
      unfolding j_def Uaeq
    proof (intro continuous_on_cases_local continuous_on_id, simp_all add: req
closedin_closed Uaeq [symmetric])
      show ∃ T. closed T ∧ (cball a d − {a}) ∩ U = (U − {a}) ∩ T
       using affine_closed ‹affine U› by (rule_tac x=(cball a d) ∩ U in exI) blast
      show ∃ T. closed T ∧ U − ball a d = (U − {a}) ∩ T
        using ‹0 < d› ‹affine U›
        by (rule_tac x=U − ball a d in exI) (force simp: affine_closed)
      show continuous_on ((cball a d − {a}) ∩ U) r
        by (force intro: continuous_on_subset [OF contr])
    qed
    have fT: x ∈ U − K ⟹ f x ∈ T for x
      using fim by blast
    show ?thesis
    proof (intro conjI exI)
      show continuous_on (S ∪ (C − {a})) (f ∘ k ∘ j)
      proof (intro continuous_on_compose)
        have S ∪ (C − {a}) ⊆ U − {a}
          using ‹C ⊆ U − S› ‹S ⊆ U› ‹a ∈ C› by force
        then show continuous_on (S ∪ (C − {a})) j
          by (rule continuous_on_subset [OF contj])
        have j ‘ (S ∪ (C − {a})) ⊆ S ∪ C
          using jim ‹C ⊆ U − S› ‹S ⊆ U› ‹ball a d ∩ U ⊆ C› j_def by blast
        then show continuous_on (j ‘ (S ∪ (C − {a}))) k
         by (rule continuous_on_subset [OF homeomorphism_cont2 [OF homhk]])
        show continuous_on (k ‘ j ‘ (S ∪ (C − {a}))) f
        proof (clarify intro!: continuous_on_subset [OF contf])
          fix y  assume y ∈ S ∪ (C − {a})
          have ky: k y ∈ S ∪ C
            using homeomorphism_image2 [OF homhk] ‹y ∈ S ∪ (C − {a})› by
blast
          have jy: j y ∈ S ∪ C − ball a d
            using Un_iff ‹y ∈ S ∪ (C − {a})› jim by auto
          have k (j y) ∈ U
```

```
          using ‹C ⊆ U› ‹S ⊆ U›  homeomorphism_image2 [OF homhk] jy by
blast
        moreover have k (j y) ∉ K
          using K unfolding disjnt_iff
        by (metis DiffE Int_iff Un_iff hin homeomorphism_def homhk image_eqI
jy)
        ultimately show k (j y) ∈ U − K
          by blast
      qed
    qed
    have ST: ⋀x. x ∈ S ⟹ (f ∘ k ∘ j) x ∈ T
    proof (simp add: kj)
      show ⋀x. x ∈ S ⟹ f x ∈ T
        using K ‹S ⊆ U› fT unfolding disjnt_iff by auto
    qed
    moreover have (f ∘ k ∘ j) x ∈ T if x ∈ C x ≠ a x ∉ S for x
    proof −
      have rx: r x ∈ sphere a d
        using ‹C ⊆ U› rim that by fastforce
      have jj: j x ∈ S ∪ C − ball a d
        using jim that by blast
      have k (j x) = j x ⟶ k (j x) ∈ C ∨ j x ∈ C
      by (metis Diff_iff Int_iff Un_iff ‹S ⊆ U› subsetD d j_def jj rx sphere_cball
that(1))
      then have kj: k (j x) ∈ C
        using homeomorphism_apply2 [OF homhk, of j x]  ‹C ⊆ U› ‹S ⊆ U› a
rx
        by (metis (mono_tags, lifting) Diff_iff subsetD jj mem_Collect_eq subC)
      then show ?thesis
        by (metis DiffE DiffI IntD1 IntI ‹C ⊆ U› comp_apply fT hin homeomor-
phism_apply2 homhk jj kj subset_eq)
    qed
    ultimately show (f ∘ k ∘ j) ‘ (S ∪ (C − {a})) ⊆ T
      by force
    show ∀x∈S. (f ∘ k ∘ j) x = f x using kj by simp
  qed (auto simp: a)
qed
then obtain a h where
  ah: ⋀C. ⟦C ∈ components (U − S); C ∩ K ≠ {}⟧
        ⟹ a C ∈ C ∧ a C ∈ L ∧ continuous_on (S ∪ (C − {a C})) (h C) ∧
           h C ‘ (S ∪ (C − {a C})) ⊆ T ∧ (∀x ∈ S. h C x = f x)
  using that by metis
define F where F ≡ {C ∈ components (U − S). C ∩ K ≠ {}}
define G where G ≡ {C ∈ components (U − S). C ∩ K = {}}
define UF where UF ≡ (⋃C∈F. C − {a C})
have C0 ∈ F
  by (auto simp: F_def C0)
have finite F
proof (subst finite_image_iff [of λC. C ∩ K F, symmetric])
```

```
    show inj_on (λC. C ∩ K) F
      unfolding F_def inj_on_def
      using components_nonoverlap by blast
    show finite ((λC. C ∩ K) ` F)
      unfolding F_def
      by (rule finite_subset [of _ Pow K]) (auto simp: ‹finite K›)
  qed
  obtain g where contg: continuous_on (S ∪ UF) g
            and gh: ⋀x i. ⟦i ∈ F; x ∈ (S ∪ UF) ∩ (S ∪ (i − {a i}))⟧
                          ⟹ g x = h i x
  proof (rule pasting_lemma_exists_closed [OF ‹finite F›])
    let ?X = top_of_set (S ∪ UF)
    show topspace ?X ⊆ (⋃C∈F. S ∪ (C − {a C}))
      using ‹C0 ∈ F› by (force simp: UF_def)
    show closedin (top_of_set (S ∪ UF)) (S ∪ (C − {a C}))
        if C ∈ F for C
    proof (rule closedin_closed_subset [of U S ∪ C])
      have C ∈ components (U − S)
        using F_def that by blast
      then show closedin (top_of_set U) (S ∪ C)
        by (rule closedin_Un_complement_component [OF ‹locally connected U›
clo])
    next
      have x = a C' if C' ∈ F  x ∈ C' x ∉ U for x C'
      proof −
        have ∀A. x ∈ ⋃A ∨ C' ∉ A
          using ‹x ∈ C'› by blast
        with that show x = a C'
          by (metis (lifting) DiffD1 F_def Union_components mem_Collect_eq)
      qed
      then show S ∪ UF ⊆ U
        using ‹S ⊆ U› by (force simp: UF_def)
    next
      show S ∪ (C − {a C}) = (S ∪ C) ∩ (S ∪ UF)
        using F_def UF_def components_nonoverlap that by auto
    qed
    show continuous_map (subtopology ?X (S ∪ (C' − {a C'}))) euclidean (h C')
if C' ∈ F for C'
    proof −
      have C': C' ∈ components (U − S) C' ∩ K ≠ {}
        using F_def that by blast+
      show ?thesis
        using ah [OF C'] by (auto simp: F_def subtopology_subtopology intro:
continuous_on_subset)
    qed
    show ⋀i j x. ⟦i ∈ F; j ∈ F;
                  x ∈ topspace ?X ∩ (S ∪ (i − {a i})) ∩ (S ∪ (j − {a j}))⟧
                 ⟹ h i x = h j x
      using components_eq by (fastforce simp: components_eq F_def ah)
```

qed *auto*
have *SU'*: $S \cup \bigcup G \cup (S \cup UF) \subseteq U$
using ‹$S \subseteq U$› *in_components_subset* **by** (*auto simp*: *F_def G_def UF_def*)
have *clo1*: *closedin* (*top_of_set* ($S \cup \bigcup G \cup (S \cup UF)$)) ($S \cup \bigcup G$)
proof (*rule closedin_closed_subset* [*OF _ SU'*])
have ∗: $\bigwedge C.\ C \in F \Longrightarrow$ *openin* (*top_of_set U*) *C*
unfolding *F_def*
by (*metis* (*no_types*, *lifting*) ‹*locally connected U*› *clo closedin_def locally_connected_open_component mem_Collect_eq topspace_euclidean_subtopology*)
show *closedin* (*top_of_set U*) ($U - UF$)
unfolding *UF_def* **by** (*force intro*: *openin_delete* ∗)
have $(\bigcup x{\in}F.\ x - \{a\ x\}) \cap S = \{\} \bigcup G \subseteq U$
using *in_components_subset* **by** (*auto simp*: *F_def G_def*)
moreover have $\bigcup G \cap UF = \{\}$
using *components_nonoverlap* **by** (*fastforce simp*: *F_def G_def UF_def*)
ultimately show $S \cup \bigcup G = (U - UF) \cap (S \cup \bigcup G \cup (S \cup UF))$
using *UF_def* ‹$S \subseteq U$› **by** *auto*
qed
have *clo2*: *closedin* (*top_of_set* ($S \cup \bigcup G \cup (S \cup UF)$)) ($S \cup UF$)
proof (*rule closedin_closed_subset* [*OF _ SU'*])
show *closedin* (*top_of_set U*) ($\bigcup C{\in}F.\ S \cup C$)
proof (*rule closedin_Union*)
show $\bigwedge T.\ T \in (\cup)\ S$ ‘ $F \Longrightarrow$ *closedin* (*top_of_set U*) *T*
using *F_def* ‹*locally connected U*› *clo closedin_Un_complement_component* **by** *blast*
qed (*simp add*: ‹*finite F*›)
show $S \cup UF = (\bigcup C{\in}F.\ S \cup C) \cap (S \cup \bigcup G \cup (S \cup UF))$
proof
show $\bigcup\ ((\cup)\ S$ ‘ $F) \cap (S \cup \bigcup\ G \cup (S \cup UF)) \subseteq S \cup UF$
using *components_eq* [*of _ U−S*]
by (*auto simp add*: *F_def G_def UF_def disjoint_iff_not_equal*)
qed (*use UF_def* ‹*C0* ∈ *F*› **in** *blast*)
qed
have *SUG*: $S \cup \bigcup G \subseteq U - K$
using ‹$S \subseteq U$› *K in_components_subset*[*of _ U−S*] **by** (*force simp*: *G_def disjnt_iff*)
then have *contf'*: *continuous_on* ($S \cup \bigcup G$) *f*
by (*rule continuous_on_subset* [*OF contf*])
have *contg'*: *continuous_on* ($S \cup UF$) *g*
by (*simp add*: *contg*)
have $\bigwedge x.\ [\![S \subseteq U;\ x \in S]\!] \Longrightarrow f\ x = g\ x$
by (*subst gh*) (*auto simp*: *ah C0 intro*: ‹*C0* ∈ *F*›)
then have *f_eq_g*: $\bigwedge x.\ x \in S \cup UF \wedge x \in S \cup \bigcup G \Longrightarrow f\ x = g\ x$
using ‹$S \subseteq U$› *components_eq* [*of _ U−S*] **by** (*fastforce simp add*: *F_def G_def UF_def*)
have *cont*: *continuous_on* ($S \cup \bigcup G \cup (S \cup UF)$) ($\lambda x.$ *if* $x \in S \cup \bigcup G$ *then f x else g x*)
by (*blast intro*: *continuous_on_cases_local* [*OF clo1 clo2 contf' contg' f_eq_g*, *of* $\lambda x.\ x \in S \cup \bigcup G$])

```
  show ?thesis
  proof
    have UF: ⋃F − L ⊆ UF
      unfolding F_def UF_def using ah by blast
    have U − S − L = ⋃(components (U − S)) − L
      by simp
    also have ... = ⋃F ∪ ⋃G − L
      unfolding F_def G_def by blast
    also have ... ⊆ UF ∪ ⋃G
      using UF by blast
    finally have U − L ⊆ S ∪ ⋃G ∪ (S ∪ UF)
      by blast
    then show continuous_on (U − L) (λx. if x ∈ S ∪ ⋃G then f x else g x)
      by (rule continuous_on_subset [OF cont])
    have ((U − L) ∩ {x. x ∉ S ∧ (∀xa∈G. x ∉ xa)}) ⊆ ((U − L) ∩ (−S ∩ UF))
      using ‹U − L ⊆ S ∪ ⋃G ∪ (S ∪ UF)› by auto
    moreover have g ‘ ((U − L) ∩ (−S ∩ UF)) ⊆ T
    proof −
      have g x ∈ T if x ∈ U x ∉ L x ∉ S C ∈ F x ∈ C x ≠ a C for x C
      proof (subst gh)
        show x ∈ (S ∪ UF) ∩ (S ∪ (C − {a C}))
          using that by (auto simp: UF_def)
        show h C x ∈ T
          using ah that by (fastforce simp add: F_def)
      qed (rule that)
      then show ?thesis
        by (force simp: UF_def)
    qed
    ultimately have g ‘ ((U − L) ∩ {x. x ∉ S ∧ (∀xa∈G. x ∉ xa)}) ⊆ T
      using image_mono order_trans by blast
    moreover have f ‘ ((U − L) ∩ (S ∪ ⋃G)) ⊆ T
      using fim SUG by blast
    ultimately show (λx. if x ∈ S ∪ ⋃G then f x else g x) ∈ (U − L) → T
      by force
    show ⋀x. x ∈ S ⟹ (if x ∈ S ∪ ⋃G then f x else g x) = f x
      by (simp add: F_def G_def)
  qed
qed


lemma extend_map_affine_to_sphere2:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes compact S convex U bounded U affine T S ⊆ T
    and affTU: aff_dim T ≤ aff_dim U
    and contf: continuous_on S f
    and fim: f ∈ S → rel_frontier U
    and ovlap: ⋀C. C ∈ components(T − S) ⟹ C ∩ L ≠ {}
  obtains K g where finite K K ⊆ L K ⊆ T disjnt K S
                    continuous_on (T − K) g g ∈ (T − K) → rel_frontier U
```

$$\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$$

proof −
obtain $K\ g$ **where** K: *finite* K $K \subseteq T$ *disjnt* $K\ S$
and *contg*: *continuous_on* $(T - K)\ g$
and *gim*: $g \in (T - K) \to$ *rel_frontier* U
and *gf*: $\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
using *assms extend_map_affine_to_sphere_cofinite_simple* **by** *metis*
have $(\exists y\ C.\ C \in$ *components* $(T - S) \wedge x \in C \wedge y \in C \wedge y \in L)$ **if** $x \in K$ **for** x
proof −
have $x \in T - S$
using ‹$K \subseteq T$› ‹*disjnt* $K\ S$› *disjnt_def that* **by** *fastforce*
then obtain C **where** $C \in$ *components*$(T - S)\ x \in C$
by (*metis UnionE Union_components*)
with *ovlap* [*of* C] **show** *?thesis*
by *blast*
qed
then obtain ξ **where** ξ: $\bigwedge x.\ x \in K \Longrightarrow \exists C.\ C \in$ *components* $(T - S) \wedge x \in C \wedge \xi\ x \in C \wedge \xi\ x \in L$
by *metis*
obtain h **where** *conth*: *continuous_on* $(T - \xi\ `\ K)\ h$
and *him*: $h \in (T - \xi\ `\ K) \to$ *rel_frontier* U
and *hg*: $\bigwedge x.\ x \in S \Longrightarrow h\ x = g\ x$
proof (*rule extend_map_affine_to_sphere1* [*OF* ‹*finite* K› ‹*affine* T› *contg gim*, *of* $S\ \xi\ `\ K$])
show *cloTS*: *closedin* (*top_of_set* T) S
by (*simp add*: ‹*compact* S› ‹$S \subseteq T$› *closed_subset compact_imp_closed*)
show $\bigwedge C.\ [\![C \in$ *components* $(T - S);\ C \cap K \neq \{\}]\!] \Longrightarrow C \cap \xi\ `\ K \neq \{\}$
using ξ *components_eq* **by** *blast*
qed (*use* K **in** *auto*)
show *?thesis*
proof
show $*$: $\xi\ `\ K \subseteq L$
using ξ **by** *blast*
show *finite* $(\xi\ `\ K)$
by (*simp add*: K)
show $\xi\ `\ K \subseteq T$
by *clarify* (*meson* ξ *Diff_iff contra_subsetD in_components_subset*)
show *continuous_on* $(T - \xi\ `\ K)\ h$
by (*rule conth*)
show *disjnt* $(\xi\ `\ K)\ S$
using $K\ \xi$ *in_components_subset* **by** (*fastforce simp*: *disjnt_def*)
qed (*simp_all add*: *him hg gf*)
qed

proposition *extend_map_affine_to_sphere_cofinite_gen*:
fixes $f :: 'a{::}$*euclidean_space* $\Rightarrow$ $'b{::}$*euclidean_space*
assumes *SUT*: *compact* S *convex* U *bounded* U *affine* $T\ S \subseteq T$

and *aff*: *aff_dim T* ≤ *aff_dim U*
and *contf*: *continuous_on S f*
and *fim*: *f* ∈ *S* → *rel_frontier U*
and *dis*: ⋀*C*. ⟦*C* ∈ *components*(*T* − *S*); *bounded C*⟧ ⟹ *C* ∩ *L* ≠ {}
obtains *K g* **where** *finite K K* ⊆ *L K* ⊆ *T disjnt K S continuous_on* (*T* − *K*) *g*
g ∈ (*T* − *K*) → *rel_frontier U*
⋀*x*. *x* ∈ *S* ⟹ *g x* = *f x*
proof (*cases S* = {})
case *True*
show *?thesis*
proof (*cases rel_frontier U* = {})
case *True*
with *aff* **have** *aff_dim T* ≤ *0*
using *affine_bounded_eq_lowdim* ‹*bounded U*› *order_trans*
by (*auto simp add*: *rel_frontier_eq_empty*)
with *aff_dim_geq* [*of T*] **consider** *aff_dim T* = −*1* | *aff_dim T* = *0*
by *linarith*
then show *?thesis*
proof *cases*
assume *aff_dim T* = −*1*
then have *T* = {}
by (*simp add*: *aff_dim_empty*)
then show *?thesis*
by (*rule_tac K*={} **in** *that*) *auto*
next
assume *aff_dim T* = *0*
then obtain *a* **where** *T* = {*a*}
using *aff_dim_eq_0* **by** *blast*
then have *a* ∈ *L*
using *dis* [*of* {*a*}] ‹*S* = {}› **by** (*auto simp*: *in_components_self*)
with ‹*S* = {}› ‹*T* = {*a*}› **show** *?thesis*
by (*rule_tac K*={*a*} **and** *g*=*f* **in** *that*) *auto*
qed
next
case *False*
then obtain *y* **where** *y* ∈ *rel_frontier U*
by *auto*
with ‹*S* = {}› **show** *?thesis*
by (*rule_tac K*={} **and** *g*=λ*x*. *y* **in** *that*) (*auto*)
qed
next
case *False*
have *bounded S*
by (*simp add*: *assms compact_imp_bounded*)
then obtain *b* **where** *b*: *S* ⊆ *cbox* (−*b*) *b*
using *bounded_subset_cbox_symmetric* **by** *blast*
define *LU* **where** *LU* ≡ *L* ∪ (⋃ {*C* ∈ *components* (*T* − *S*). ¬*bounded C*} − *cbox* (−(*b*+*One*)) (*b*+*One*))

```
  obtain K g where finite K K ⊆ LU K ⊆ T disjnt K S
              and contg: continuous_on (T − K) g
              and gim: g ‘ (T − K) ⊆ rel_frontier U
              and gf:  ⋀x. x ∈ S ⟹ g x = f x
  proof (rule extend_map_affine_to_sphere2 [OF SUT aff contf fim])
    show C ∩ LU ≠ {} if C ∈ components (T − S) for C
    proof (cases bounded C)
      case True
      with dis that show ?thesis
        unfolding LU_def by fastforce
    next
      case False
      then have ¬ bounded (⋃{C ∈ components (T − S). ¬ bounded C})
        by (metis (no_types, lifting) Sup_upper bounded_subset mem_Collect_eq
that)
      then show ?thesis
      by (simp add: LU_def disjoint_iff) (meson False bounded_cbox bounded_subset
subset_iff that)
    qed
  qed blast
  have ∗: False if x ∈ cbox (− b − m ∗R One) (b + m ∗R One)
                   x ∉ box (− b − n ∗R One) (b + n ∗R One)
                   0 ≤ m m < n n ≤ 1 for m n x
    using that by (auto simp: mem_box algebra_simps)
  have disjoint_family_on (λd. frontier (cbox (− b − d ∗R One) (b + d ∗R One)))
{1 / 2..1}
    by (auto simp: disjoint_family_on_def neq_iff frontier_def dest: ∗)
  then obtain d where d12: 1/2 ≤ d d ≤ 1
                   and ddis: disjnt K (frontier (cbox (−(b + d ∗R One)) (b + d ∗R
One)))
      using disjoint_family_elem_disjnt [of {1/2..1::real} K λd. frontier (cbox
(−(b + d ∗R One)) (b + d ∗R One))]
    by (auto simp: ‹finite K›)
  define c where c ≡ b + d ∗R One
  have cbsub: cbox (−b) b ⊆ box (−c) c
              cbox (−b) b ⊆ cbox (−c) c
              cbox (−c) c ⊆ cbox (−(b+One)) (b+One)
   using d12 by (simp_all add: subset_box c_def inner_diff_left inner_left_distrib)
  have clo_cT: closed (cbox (− c) c ∩ T)
    using affine_closed ‹affine T› by blast
  have cT_ne: cbox (− c) c ∩ T ≠ {}
    using ‹S ≠ {}› ‹S ⊆ T› b cbsub by fastforce
  have S_sub_cc: S ⊆ cbox (− c) c
    using ‹cbox (− b) b ⊆ cbox (− c) c› b by auto
  show ?thesis
  proof
    show finite (K ∩ cbox (−(b+One)) (b+One))
      using ‹finite K› by blast
    show K ∩ cbox (− (b + One)) (b + One) ⊆ L
```

```
    using ‹K ⊆ LU› by (auto simp: LU_def)
  show K ∩ cbox (− (b + One)) (b + One) ⊆ T
    using ‹K ⊆ T› by auto
  show disjnt (K ∩ cbox (− (b + One)) (b + One)) S
    using ‹disjnt K S›  by (simp add: disjnt_def disjoint_eq_subset_Compl
inf.coboundedI1)
  have cloTK: closest_point (cbox (− c) c ∩ T) x ∈ T − K
        if x ∈ T and Knot: x ∈ K ⟶ x ∉ cbox (− b − One) (b + One) for
x
  proof (cases x ∈ cbox (− c) c)
    case True
    with ‹x ∈ T› show ?thesis
      using cbsub(3) Knot  by (force simp: closest_point_self)
  next
    case False
    have clo_in_rf: closest_point (cbox (− c) c ∩ T) x ∈ rel_frontier (cbox (−
c) c ∩ T)
    proof (intro closest_point_in_rel_frontier [OF clo_cT cT_ne] DiffI notI)
      have T ∩ interior (cbox (− c) c) ≠ {}
        using ‹S ≠ {}› ‹S ⊆ T› b cbsub(1) by fastforce
      then show x ∈ affine hull (cbox (− c) c ∩ T)
        by (simp add: Int_commute affine_hull_affine_Int_nonempty_interior
‹affine T› hull_inc that(1))
    next
      show False if x ∈ rel_interior (cbox (− c) c ∩ T)
      proof −
        have interior (cbox (− c) c) ∩ T ≠ {}
          using ‹S ≠ {}› ‹S ⊆ T› b cbsub(1) by fastforce
        then have affine hull (T ∩ cbox (− c) c) = T
          using affine_hull_convex_Int_nonempty_interior [of T cbox (− c) c]
          by (simp add: affine_imp_convex ‹affine T› inf_commute)
        then show ?thesis
          by (meson subsetD le_inf_iff rel_interior_subset that False)
      qed
    qed
    have closest_point (cbox (− c) c ∩ T) x ∉ K
    proof
      assume inK: closest_point (cbox (− c) c ∩ T) x ∈ K
       have ⋀x. x ∈ K ⟹ x ∉ frontier (cbox (− (b + d *R One)) (b + d *R
One))
        by (metis ddis disjnt_iff)
      then show False
        by (metis DiffI Int_iff ‹affine T› cT_ne c_def clo_cT clo_in_rf clos-
est_point_in_set
                  convex_affine_rel_frontier_Int convex_box(1) empty_iff fron-
tier_cbox inK interior_cbox)
    qed
    then show ?thesis
      using cT_ne clo_cT closest_point_in_set by blast
```

```
    qed
    have convex (cbox (− c) c ∩ T)
      by (simp add: affine_imp_convex assms(4) convex_Int)
    then show continuous_on (T − K ∩ cbox (− (b + One)) (b + One)) (g ∘
closest_point (cbox (−c) c ∩ T))
      using cloTK clo_cT cT_ne
        by (intro continuous_on_compose continuous_on_closest_point continu-
ous_on_subset [OF contg]; force)
    have g (closest_point (cbox (− c) c ∩ T) x) ∈ rel_frontier U
        if x ∈ T x ∈ K ⟶ x ∉ cbox (− b − One) (b + One) for x
      using gim [THEN subsetD] that cloTK by blast
    then show (g ∘ closest_point (cbox (− c) c ∩ T)) ∈ (T − K ∩ cbox (− (b +
One)) (b + One))
              → rel_frontier U
      by force
    show ⋀x. x ∈ S ⟹ (g ∘ closest_point (cbox (− c) c ∩ T)) x = f x
      by simp (metis (mono_tags, lifting) IntI ‹S ⊆ T› cT_ne clo_cT clos-
est_point_refl gf subsetD S_sub_cc)
  qed
qed


corollary extend_map_affine_to_sphere_cofinite:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes SUT: compact S affine T S ⊆ T
      and aff: aff_dim T ≤ DIM('b) and 0 ≤ r
      and contf: continuous_on S f
      and fim: f ∈ S → sphere a r
      and dis: ⋀C. ⟦C ∈ components(T − S); bounded C⟧ ⟹ C ∩ L ≠ {}
  obtains K g where finite K K ⊆ L K ⊆ T disjnt K S continuous_on (T − K)
g
                    g ∈ (T − K) → sphere a r ⋀x. x ∈ S ⟹ g x = f x
proof (cases r = 0)
  case True
  with fim show ?thesis
    by (rule_tac K={} and g = λx. a in that) (auto)
next
  case False
  show thesis
  proof (rule  extend_map_affine_to_sphere_cofinite_gen
      [OF ‹compact S› convex_cball bounded_cball ‹affine T› ‹S ⊆ T› _ contf])
    have 0 < r
      using assms False by auto
    then show aff_dim T ≤ aff_dim (cball a r)
      by (simp add: aff aff_dim_cball)
    show f ∈ S → rel_frontier (cball a r)
      by (simp add: False fim)
  qed (use dis False that in auto)
qed
```

corollary *extend_map_UNIV_to_sphere_cofinite*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes $DIM('a) \le DIM('b)$ **and** $0 \le r$
and *compact S*
and *continuous_on S f*
and $f \in S \to sphere\ a\ r$
and $\bigwedge C.\ [\![C \in components(-\ S);\ bounded\ C]\!] \Longrightarrow C \cap L \neq \{\}$
obtains *K g* **where** *finite K* $K \subseteq L$ *disjnt K S continuous_on* $(-\ K)\ g$
$g \in (-\ K) \to sphere\ a\ r\ \bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
using *extend_map_affine_to_sphere_cofinite*
$[OF\ \langle compact\ S\rangle\ affine\_UNIV\ subset\_UNIV]\ assms$
by (*metis Compl_eq_Diff_UNIV aff_dim_UNIV of_nat_le_iff*)

corollary *extend_map_UNIV_to_sphere_no_bounded_component*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes *aff*: $DIM('a) \le DIM('b)$ **and** $0 \le r$
and *SUT*: *compact S*
and *contf*: *continuous_on S f*
and *fim*: $f \in S \to sphere\ a\ r$
and *dis*: $\bigwedge C.\ C \in components(-\ S) \Longrightarrow \neg\ bounded\ C$
obtains *g* **where** *continuous_on UNIV g* $g \in UNIV \to sphere\ a\ r\ \bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
using *extend_map_UNIV_to_sphere_cofinite* [*OF aff* $\langle 0 \le r\rangle$ $\langle compact\ S\rangle$ *contf fim*, *of* {}]
by (*metis Compl_empty_eq dis subset_empty*)

theorem *Borsuk_separation_theorem_gen*:
fixes $S :: 'a::euclidean\_space\ set$
assumes *compact S*
shows $(\forall c \in components(-\ S).\ \neg bounded\ c) \longleftrightarrow$
$(\forall f.\ continuous\_on\ S\ f \wedge f \in S \to sphere\ (0::'a)\ 1$
$\longrightarrow (\exists c.\ homotopic\_with\_canon\ (\lambda x.\ True)\ S\ (sphere\ 0\ 1)\ f\ (\lambda x.\ c)))$
(**is** *?lhs* = *?rhs*)
proof
assume *L* [*rule_format*]: *?lhs*
show *?rhs*
proof *clarify*
fix $f :: 'a \Rightarrow 'a$
assume *contf*: *continuous_on S f* **and** *fim*: $f \in S \to sphere\ 0\ 1$
obtain *g* **where** *contg*: *continuous_on UNIV g* **and** *gim*: $g \in UNIV \to sphere\ 0\ 1$
and *gf*: $\bigwedge x.\ x \in S \Longrightarrow g\ x = f\ x$
by (*rule extend_map_UNIV_to_sphere_no_bounded_component* [*OF _ _* $\langle compact\ S\rangle$ *contf fim L*]) *auto*
then obtain *c* **where** *c*: *homotopic_with_canon* $(\lambda h.\ True)\ UNIV\ (sphere\ 0\ 1)\ g\ (\lambda x.\ c)$

by (*metis contractible_UNIV nullhomotopic_from_contractible*)
then show ∃ *c*. *homotopic_with_canon* (λ*x*. *True*) *S* (*sphere 0 1*) *f* (λ*x*. *c*)
by (*metis assms compact_imp_closed contf contg contractible_empty fim gf gim nullhomotopic_from_contractible nullhomotopic_into_sphere_extension image_subset_iff_funcset*)
qed
next
assume *R* [*rule_format*]: *?rhs*
show *?lhs*
unfolding *components_def*
proof *clarify*
fix *a*
assume *a* ∉ *S* **and** *a*: *bounded* (*connected_component_set* (− *S*) *a*)
have ∀ *x*∈*S*. *norm* (*x* − *a*) ≠ *0*
using ‹*a* ∉ *S*› **by** *auto*
then have *cont*: *continuous_on S* (λ*x*. *inverse*(*norm*(*x* − *a*)) ∗*R* (*x* − *a*))
by (*intro continuous_intros*)
have *im*: (λ*x*. *inverse*(*norm*(*x* − *a*)) ∗*R* (*x* − *a*)) ‘ *S* ⊆ *sphere 0 1*
by *clarsimp* (*metis* ‹*a* ∉ *S*› *eq_iff_diff_eq_0 left_inverse norm_eq_zero*)
show *False*
using *R cont im Borsuk_map_essential_bounded_component* [*OF* ‹*compact S*› ‹*a* ∉ *S*›] *a* **by** *blast*
qed
qed

corollary *Borsuk_separation_theorem*:
fixes *S* :: *'a*::*euclidean_space set*
assumes *compact S* **and** *2*: *2* ≤ *DIM*(*'a*)
shows *connected*(− *S*) ⟷
(∀ *f*. *continuous_on S f* ∧ *f* ∈ *S* → *sphere* (*0*::*'a*) *1*
⟶ (∃ *c*. *homotopic_with_canon* (λ*x*. *True*) *S* (*sphere 0 1*) *f* (λ*x*. *c*)))
(**is** *?lhs* = *?rhs*)
proof
assume *L*: *?lhs*
show *?rhs*
proof (*cases S* = {})
case *True*
then show *?thesis*
using *homotopic_with_canon_on_empty* **by** *blast*
next
case *False*
then have (∀ *c*∈*components* (− *S*). ¬ *bounded c*)
by (*metis L assms*(*1*) *bounded_empty cobounded_imp_unbounded compact_imp_bounded in_components_maximal order_refl*)
then show *?thesis*
by (*simp add*: *Borsuk_separation_theorem_gen* [*OF* ‹*compact S*›])
qed

next
assume *R*: *?rhs*
then have ∀ *c*∈*components* (− *S*). ¬ *bounded c* ⟹ *connected* (− *S*)
by (*metis 2 assms*(*1*) *cobounded_has_bounded_component compact_imp_bounded double_complement*)
with *R* **show** *?lhs*
by (*simp add*: *Borsuk_separation_theorem_gen* [*OF* ‹*compact S*›])
qed

lemma *homotopy_eqv_separation*:
fixes *S* :: *'a*::*euclidean_space set* **and** *T* :: *'a set*
assumes *S homotopy_eqv T* **and** *compact S* **and** *compact T*
shows *connected*(− *S*) ⟷ *connected*(− *T*)
proof −
consider *DIM*(*'a*) = *1* | *2* ≤ *DIM*(*'a*)
by (*metis DIM_ge_Suc0 One_nat_def Suc_1 dual_order.antisym not_less_eq_eq*)
then show *?thesis*
proof *cases*
case *1*
then show *?thesis*
using *bounded_connected_Compl_1 compact_imp_bounded homotopy_eqv_empty1 homotopy_eqv_empty2 assms* **by** *metis*
next
case *2*
with *assms* **show** *?thesis*
by (*simp add*: *Borsuk_separation_theorem homotopy_eqv_cohomotopic_triviality_null*)
qed
qed

proposition *Jordan_Brouwer_separation*:
fixes *S* :: *'a*::*euclidean_space set* **and** *a*::*'a*
assumes *hom*: *S homeomorphic sphere a r* **and** *0* < *r*
shows ¬ *connected*(− *S*)
proof −
have − *sphere a r* ∩ *ball a r* ≠ {}
using ‹*0* < *r*› **by** (*simp add*: *Int_absorb1 subset_eq*)
moreover
have *eq*: − *sphere a r* − *ball a r* = − *cball a r*
by *auto*
have − *cball a r* ≠ {}
proof −
have *frontier* (*cball a r*) ≠ {}
using ‹*0* < *r*› **by** *auto*
then show *?thesis*
by (*metis frontier_complement frontier_empty*)
qed
with *eq* **have** − *sphere a r* − *ball a r* ≠ {}
by *auto*

```
  moreover
  have connected (− S) = connected (− sphere a r)
  proof (rule homotopy_eqv_separation)
    show S homotopy_eqv sphere a r
      using hom homeomorphic_imp_homotopy_eqv by blast
    show compact (sphere a r)
      by simp
    then show  compact S
      using hom homeomorphic_compactness by blast
  qed
  ultimately show ?thesis
    using connected_Int_frontier [of − sphere a r ball a r] by (auto simp: ‹0 <
r›)
qed


proposition Jordan_Brouwer_frontier:
  fixes S :: 'a::euclidean_space set and a::'a
  assumes S: S homeomorphic sphere a r and T: T ∈ components(− S) and 2:
2 ≤ DIM('a)
    shows frontier T = S
proof (cases r rule: linorder_cases)
  assume r < 0
  with S T show ?thesis by auto
next
  assume r = 0
  with S T card_eq_SucD obtain b where S = {b}
    by (auto simp: homeomorphic_finite [of {a} S])
  have components (− {b}) = { −{b}}
   using T ‹S = {b}› by (auto simp: components_eq_sing_iff connected_punctured_universe
2)
  with T show ?thesis
   by (metis ‹S = {b}› cball_trivial frontier_cball frontier_complement singletonD
sphere_trivial)
next
  assume r > 0
  have compact S
    using homeomorphic_compactness compact_sphere S by blast
  show ?thesis
  proof (rule frontier_minimal_separating_closed)
    show closed S
      using ‹compact S› compact_eq_bounded_closed by blast
    show ¬ connected (− S)
      using Jordan_Brouwer_separation S ‹0 < r› by blast
    obtain f g where hom: homeomorphism S (sphere a r) f g
      using S by (auto simp: homeomorphic_def)
    show connected (− T) if closed T T ⊂ S for T
    proof −
      have f ‘ T ⊆ sphere a r
```

```
      using ‹T ⊂ S› hom homeomorphism_image1 by blast
    moreover have f ‘ T ≠ sphere a r
      using ‹T ⊂ S› hom
      by (metis homeomorphism_image2 homeomorphism_of_subsets order_refl psubsetE)
    ultimately have f ‘ T ⊂ sphere a r by blast
    then have connected (− f ‘ T)
      by (rule psubset_sphere_Compl_connected [OF _ ‹0 < r› 2])
    moreover have compact T
      using ‹compact S› bounded_subset compact_eq_bounded_closed that by blast
    moreover then have compact (f ‘ T)
      by (meson compact_continuous_image continuous_on_subset hom homeomorphism_def psubsetE ‹T ⊂ S›)
    moreover have T homotopy_eqv f ‘ T
     by (meson ‹f ‘ T ⊆ sphere a r› dual_order.strict_implies_order hom homeomorphic_def homeomorphic_imp_homotopy_eqv homeomorphism_of_subsets ‹T ⊂ S›)
    ultimately show ?thesis
      using homotopy_eqv_separation [of T f‘T] by blast
  qed
 qed (rule T)
qed

proposition Jordan_Brouwer_nonseparation:
 fixes S :: 'a::euclidean_space set and a::'a
 assumes S: S homeomorphic sphere a r and T ⊂ S and 2: 2 ≤ DIM('a)
  shows connected(− T)
proof −
 have ∗: connected(C ∪ (S − T)) if C ∈ components(− S) for C
 proof (rule connected_intermediate_closure)
  show connected C
    using in_components_connected that by auto
  have S = frontier C
    using 2 Jordan_Brouwer_frontier S that by blast
  with closure_subset show C ∪ (S − T) ⊆ closure C
    by (auto simp: frontier_def)
 qed auto
 have components(− S) ≠ {}
  by (metis S bounded_empty cobounded_imp_unbounded compact_eq_bounded_closed compact_sphere
            components_eq_empty homeomorphic_compactness)
 then have − T = (⋃ C ∈ components(− S). C ∪ (S − T))
  using Union_components [of −S] ‹T ⊂ S› by auto
 moreover have connected ...
  using ‹T ⊂ S› by (intro connected_Union) (auto simp: ∗)
 ultimately show ?thesis
  by simp
qed
```

6.36.5 Invariance of domain and corollaries

lemma *invariance_of_domain_ball*:
fixes $f :: 'a \Rightarrow 'a$::*euclidean_space*
assumes *contf*: *continuous_on* (*cball a r*) *f* **and** $0 < r$
and *inj*: *inj_on f* (*cball a r*)
shows *open*(*f ‘ ball a r*)
proof (*cases DIM*('*a*) = *1*)
case *True*
obtain *h*::'*a*⇒*real* **and** *k*
where *linear h linear k h ‘ UNIV = UNIV k ‘ UNIV = UNIV*
$\bigwedge x.$ *norm*(*h x*) = *norm x* $\bigwedge x.$ *norm*(*k x*) = *norm x*
and *kh*: $\bigwedge x.$ *k*(*h x*) = *x* **and** $\bigwedge x.$ *h*(*k x*) = *x*
proof (*rule isomorphisms_UNIV_UNIV*)
show *DIM*('*a*) = *DIM*(*real*)
using *True* **by** *force*
qed (*metis UNIV_I UNIV_eq_I imageI*)
have *cont*: *continuous_on S h continuous_on T k* **for** *S T*
by (*simp_all add*: ‹*linear h*› ‹*linear k*› *linear_continuous_on linear_linear*)
have *continuous_on* (*h ‘ cball a r*) (*h* ∘ *f* ∘ *k*)
by (*intro continuous_on_compose cont continuous_on_subset* [*OF contf*])
(*auto simp*: *kh*)
moreover have *is_interval* (*h ‘ cball a r*)
by (*simp add*: *is_interval_connected_1* ‹*linear h*› *linear_continuous_on*
linear_linear connected_continuous_image)
moreover have *inj_on* (*h* ∘ *f* ∘ *k*) (*h ‘ cball a r*)
using *inj* **by** (*simp add*: *inj_on_def*) (*metis* ‹$\bigwedge x.$ *k* (*h x*) = *x*›)
ultimately have ∗: $\bigwedge T.$ ⟦*open T*; *T* ⊆ *h ‘ cball a r*⟧ ⟹ *open* ((*h* ∘ *f* ∘ *k*) ‘
T)
using *injective_eq_1d_open_map_UNIV* **by** *blast*
have *open* ((*h* ∘ *f* ∘ *k*) ‘ (*h ‘ ball a r*))
by (*rule* ∗) (*auto simp*: ‹*linear h*› ‹*range h = UNIV*› *open_surjective_linear_image*)
then have *open* ((*h* ∘ *f*) ‘ *ball a r*)
by (*simp add*: *image_comp* ‹$\bigwedge x.$ *k* (*h x*) = *x*› *cong*: *image_cong*)
then show *?thesis*
unfolding *image_comp* [*symmetric*]
by (*metis open_bijective_linear_image_eq* ‹*linear h*› *kh* ‹*range h = UNIV*›
bijI inj_on_def)
next
case *False*
then have *2*: *DIM*('*a*) ≥ *2*
by (*metis DIM_ge_Suc0 One_nat_def Suc_1 antisym not_less_eq_eq*)
have *fimsub*: *f ‘ ball a r* ⊆ − *f ‘ sphere a r*
using *inj* **by** *clarsimp* (*metis inj_onD less_eq_real_def mem_cball order_less_irrefl*)
have *hom*: *f ‘ sphere a r homeomorphic sphere a r*
by (*meson compact_sphere contf continuous_on_subset homeomorphic_compact*
homeomorphic_sym inj inj_on_subset sphere_cball)
then have *nconn*: ¬ *connected* (− *f ‘ sphere a r*)
by (*rule Jordan_Brouwer_separation*) (*auto simp*: ‹*0 < r*›)
have *bounded* (*f ‘ sphere a r*)

```
  by (meson compact_imp_bounded compact_continuous_image_eq compact_sphere
contf inj sphere_cball)
 then obtain C where C: C ∈ components (− f ‘ sphere a r) and bounded C
   using cobounded_has_bounded_component [OF _ nconn] 2 by auto
 moreover have f ‘ (ball a r) = C
 proof
   have C ≠ {}
     by (rule in_components_nonempty [OF C])
   show C ⊆ f ‘ ball a r
   proof (rule ccontr)
     assume nonsub: ¬ C ⊆ f ‘ ball a r
     have − f ‘ cball a r ⊆ C
     proof (rule components_maximal [OF C])
       have f ‘ cball a r homeomorphic cball a r
         using compact_cball contf homeomorphic_compact homeomorphic_sym
inj by blast
       then show connected (− f ‘ cball a r)
         by (auto intro: connected_complement_homeomorphic_convex_compact
2)
       show − f ‘ cball a r ⊆ − f ‘ sphere a r
         by auto
       then show C ∩ − f ‘ cball a r ≠ {}
         using ‹C ≠ {}› in_components_subset [OF C] nonsub
         using image_iff by fastforce
     qed
     then have bounded (− f ‘ cball a r)
       using bounded_subset ‹bounded C› by auto
     then have ¬ bounded (f ‘ cball a r)
       using cobounded_imp_unbounded by blast
     then show False
     using compact_continuous_image [OF contf] compact_cball compact_imp_bounded
by blast
   qed
   with ‹C ≠ {}› have C ∩ f ‘ ball a r ≠ {}
     by (simp add: inf.absorb_iff1)
   then show f ‘ ball a r ⊆ C
    by (metis components_maximal [OF C _ fimsub] connected_continuous_image
ball_subset_cball connected_ball contf continuous_on_subset)
 qed
 moreover have open (− f ‘ sphere a r)
  using hom compact_eq_bounded_closed compact_sphere homeomorphic_compactness
by blast
 ultimately show ?thesis
   using open_components by blast
qed
```

Proved by L. E. J. Brouwer (1912)

```
theorem invariance_of_domain:
 fixes f :: 'a ⇒ 'a::euclidean_space
```

```
  assumes continuous_on S f open S inj_on f S
    shows open(f ‘ S)
  unfolding open_subopen [of f‘S]
proof clarify
  fix a
  assume a ∈ S
  obtain δ where δ > 0 and δ: cball a δ ⊆ S
    using ‹open S› ‹a ∈ S› open_contains_cball_eq by blast
  show ∃ T. open T ∧ f a ∈ T ∧ T ⊆ f ‘ S
  proof (intro exI conjI)
    show open (f ‘ (ball a δ))
      by (meson δ ‹0 < δ› assms continuous_on_subset inj_on_subset invari-
ance_of_domain_ball)
    show f a ∈ f ‘ ball a δ
      by (simp add: ‹0 < δ›)
    show f ‘ ball a δ ⊆ f ‘ S
      using δ ball_subset_cball by blast
  qed
qed

lemma inv_of_domain_ss0:
  fixes f :: 'a ⇒ 'a::euclidean_space
  assumes contf: continuous_on U f and injf: inj_on f U and fim: f ‘ U ⊆ S
      and subspace S and dimS: dim S = DIM('b::euclidean_space)
      and ope: openin (top_of_set S) U
    shows openin (top_of_set S) (f ‘ U)
proof −
  have U ⊆ S
    using ope openin_imp_subset by blast
  have (UNIV::'b set) homeomorphic S
    by (simp add: ‹subspace S› dimS homeomorphic_subspaces)
  then obtain h k where homhk: homeomorphism (UNIV::'b set) S h k
    using homeomorphic_def by blast
  have homkh: homeomorphism S (k ‘ S) k h
    using homhk homeomorphism_image2 homeomorphism_sym by fastforce
  have open ((k ∘ f ∘ h) ‘ k ‘ U)
  proof (rule invariance_of_domain)
    show continuous_on (k ‘ U) (k ∘ f ∘ h)
    proof (intro continuous_intros)
      show continuous_on (k ‘ U) h
       by (meson continuous_on_subset [OF homeomorphism_cont1 [OF homhk]]
top_greatest)
      have h ‘ k ‘ U ⊆ U
        by (metis ‹U ⊆ S› dual_order.eq_iff homeomorphism_image2 homeomor-
phism_of_subsets homkh)
      then show continuous_on (h ‘ k ‘ U) f
        by (rule continuous_on_subset [OF contf])
      have f ‘ h ‘ k ‘ U ⊆ S
        using ‹h ‘ k ‘ U ⊆ U› fim by blast
```

```
    then show continuous_on (f ‘ h ‘ k ‘ U) k
      by (rule continuous_on_subset [OF homeomorphism_cont2 [OF homhk]])
  qed
  have ope_iff: ⋀T. open T ⟷ openin (top_of_set (k ‘ S)) T
    using homhk homeomorphism_image2 open_openin by fastforce
  show open (k ‘ U)
    by (simp add: ope_iff homeomorphism_imp_open_map [OF homkh ope])
  show inj_on (k ∘ f ∘ h) (k ‘ U)
   by (smt (verit) comp_apply inj_on_def ‹U ⊆ S› fim homeomorphism_apply2
homhk image_iff injf subsetD)
 qed
 moreover
 have eq: f ‘ U = h ‘ (k ∘ f ∘ h ∘ k) ‘ U
  unfolding image_comp [symmetric] using ‹U ⊆ S› fim
   by (metis homeomorphism_image2 homeomorphism_of_subsets homkh sub-
set_image_iff)
 ultimately show ?thesis
  by (metis (no_types, opaque_lifting) homeomorphism_imp_open_map homhk
image_comp open_openin subtopology_UNIV)
qed

lemma inv_of_domain_ss1:
 fixes f :: 'a ⇒ 'a::euclidean_space
 assumes contf: continuous_on U f and injf: inj_on f U and fim: f ‘ U ⊆ S
    and subspace S
    and ope: openin (top_of_set S) U
  shows openin (top_of_set S) (f ‘ U)
proof −
 define S' where S' ≡ {y. ∀x ∈ S. orthogonal x y}
 have subspace S'
  by (simp add: S'_def subspace_orthogonal_to_vectors)
 define g where g ≡ λz::'a*'a. ((f ∘ fst)z, snd z)
 have openin (top_of_set (S × S')) (g ‘ (U × S'))
 proof (rule inv_of_domain_ss0)
  show continuous_on (U × S') g
    unfolding g_def
    by (auto intro!: continuous_intros continuous_on_compose2 [OF contf con-
tinuous_on_fst])
  show g ‘ (U × S') ⊆ S × S'
    using fim by (auto simp: g_def)
  show inj_on g (U × S')
    using injf by (auto simp: g_def inj_on_def)
  show subspace (S × S')
    by (simp add: ‹subspace S'› ‹subspace S› subspace_Times)
  show openin (top_of_set (S × S')) (U × S')
    by (simp add: openin_Times [OF ope])
  have dim (S × S') = dim S + dim S'
    by (simp add: ‹subspace S'› ‹subspace S› dim_Times)
  also have ... = DIM('a)
```

```
    using dim_subspace_orthogonal_to_vectors [OF ‹subspace S› subspace_UNIV]
      by (simp add: add.commute S'_def)
    finally show dim (S × S') = DIM('a) .
  qed
  moreover have g ‘ (U × S') = f ‘ U × S'
    by (auto simp: g_def image_iff)
  moreover have 0 ∈ S'
    using ‹subspace S'› subspace_affine by blast
  ultimately show ?thesis
    by (auto simp: openin_Times_eq)
qed


corollary invariance_of_domain_subspaces:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes ope: openin (top_of_set U) S
      and subspace U subspace V and VU: dim V ≤ dim U
      and contf: continuous_on S f and fim: f ‘ S ⊆ V
      and injf: inj_on f S
    shows openin (top_of_set V) (f ‘ S)
proof −
  obtain V' where subspace V' V' ⊆ U dim V' = dim V
    using choose_subspace_of_subspace [OF VU]
    by (metis span_eq_iff ‹subspace U›)
  then have V homeomorphic V'
    by (simp add: ‹subspace V› homeomorphic_subspaces)
  then obtain h k where homhk: homeomorphism V V' h k
    using homeomorphic_def by blast
  have eq: f ‘ S = k ‘ (h ∘ f) ‘ S
  proof −
    have k ‘ h ‘ f ‘ S = f ‘ S
      by (meson fim homeomorphism_def homeomorphism_of_subsets homhk sub-
set_refl)
    then show ?thesis
      by (simp add: image_comp)
  qed
  show ?thesis
    unfolding eq
  proof (rule homeomorphism_imp_open_map)
    show homkh: homeomorphism V' V k h
      by (simp add: homeomorphism_symD homhk)
    have hfV': (h ∘ f) ‘ S ⊆ V'
      using fim homeomorphism_image1 homhk by fastforce
    moreover have openin (top_of_set U) ((h ∘ f) ‘ S)
    proof (rule inv_of_domain_ss1)
      show continuous_on S (h ∘ f)
        by (meson contf continuous_on_compose continuous_on_subset fim home-
omorphism_cont1 homhk)
      show inj_on (h ∘ f) S
```

```
    by (smt (verit, ccfv_SIG) comp_apply fim inj_on_def homeomorphism_apply2
[OF homkh] image_subset_iff injf)
    show (h ∘ f) ‘ S ⊆ U
      using ‹V' ⊆ U› hfV' by auto
    qed (auto simp: assms)
  ultimately show openin (top_of_set V') ((h ∘ f) ‘ S)
    using openin_subset_trans ‹V' ⊆ U› by force
 qed
qed

corollary invariance_of_dimension_subspaces:
 fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
 assumes ope: openin (top_of_set U) S
    and subspace U subspace V
    and contf: continuous_on S f and fim: f ‘ S ⊆ V
    and injf: inj_on f S and S ≠ {}
  shows dim U ≤ dim V
proof −
 have False if dim V < dim U
 proof −
  obtain T where subspace T T ⊆ U dim T = dim V
    using choose_subspace_of_subspace [of dim V U]
   by (metis ‹dim V < dim U› assms(2) order.strict_implies_order span_eq_iff)
  then have V homeomorphic T
    by (simp add: ‹subspace V› homeomorphic_subspaces)
  then obtain h k where homhk: homeomorphism V T h k
    using homeomorphic_def by blast
  have continuous_on S (h ∘ f)
    by (meson contf continuous_on_compose continuous_on_subset fim homeo-
morphism_cont1 homhk)
  moreover have (h ∘ f) ‘ S ⊆ U
    using ‹T ⊆ U› fim homeomorphism_image1 homhk by fastforce
  moreover have inj_on (h ∘ f) S
     by (smt (verit, best) comp_apply inj_on_def fim homeomorphism_apply1
homhk image_subset_iff injf)
  ultimately have ope_hf: openin (top_of_set U) ((h ∘ f) ‘ S)
    using invariance_of_domain_subspaces [OF ope ‹subspace U› ‹subspace U›]
by blast
  have (h ∘ f) ‘ S ⊆ T
    using fim homeomorphism_image1 homhk by fastforce
  then have dim ((h ∘ f) ‘ S) ≤ dim T
    by (rule dim_subset)
  also have dim ((h ∘ f) ‘ S) = dim U
    using ‹S ≠ {}› ‹subspace U›
    by (blast intro: dim_openin ope_hf)
  finally show False
    using ‹dim V < dim U› ‹dim T = dim V› by simp
 qed
 then show ?thesis
```

```
    using not_less by blast
qed

corollary invariance_of_domain_affine_sets:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes ope: openin (top_of_set U) S
      and aff: affine U affine V aff_dim V ≤ aff_dim U
      and contf: continuous_on S f and fim: f ‘ S ⊆ V
      and injf: inj_on f S
    shows openin (top_of_set V) (f ‘ S)
proof (cases S = {})
  case True
  then show ?thesis by auto
next
  case False
  obtain a b where a ∈ S a ∈ U b ∈ V
    using False fim ope openin_contains_cball by fastforce
  have openin (top_of_set ((+) (− b) ‘ V)) (((+) (− b) ∘ f ∘ (+) a) ‘ (+) (− a)
‘ S)
  proof (rule invariance_of_domain_subspaces)
    show openin (top_of_set ((+) (− a) ‘ U)) ((+) (− a) ‘ S)
      by (metis ope homeomorphism_imp_open_map homeomorphism_translation
translation_galois)
    show subspace ((+) (− a) ‘ U)
        by (simp add: ‹a ∈ U› affine_diffs_subspace_subtract ‹affine U› cong:
image_cong_simp)
    show subspace ((+) (− b) ‘ V)
        by (simp add: ‹b ∈ V› affine_diffs_subspace_subtract ‹affine V› cong:
image_cong_simp)
    show dim ((+) (− b) ‘ V) ≤ dim ((+) (− a) ‘ U)
     by (metis ‹a ∈ U› ‹b ∈ V› aff_dim_eq_dim affine_hull_eq aff of_nat_le_iff)
    show continuous_on ((+) (− a) ‘ S) ((+) (− b) ∘ f ∘ (+) a)
      by (metis contf continuous_on_compose homeomorphism_cont2 homeomor-
phism_translation translation_galois)
    show ((+) (− b) ∘ f ∘ (+) a) ‘ (+) (− a) ‘ S ⊆ (+) (− b) ‘ V
      using fim by auto
    show inj_on ((+) (− b) ∘ f ∘ (+) a) ((+) (− a) ‘ S)
      by (auto simp: inj_on_def) (meson inj_onD injf)
  qed
  then show ?thesis
    by (metis (no_types, lifting) homeomorphism_imp_open_map homeomor-
phism_translation image_comp translation_galois)
qed

corollary invariance_of_dimension_affine_sets:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes ope: openin (top_of_set U) S
      and aff: affine U affine V
      and contf: continuous_on S f and fim: f ‘ S ⊆ V
```

```
    and injf: inj_on f S and S ≠ {}
  shows aff_dim U ≤ aff_dim V
proof −
 obtain a b where a ∈ S a ∈ U b ∈ V
  using ‹S ≠ {}› fim ope openin_contains_cball by fastforce
 have dim ((+) (− a) ‘ U) ≤ dim ((+) (− b) ‘ V)
 proof (rule invariance_of_dimension_subspaces)
  show openin (top_of_set ((+) (− a) ‘ U)) ((+) (− a) ‘ S)
   by (metis ope homeomorphism_imp_open_map homeomorphism_translation
translation_galois)
  show subspace ((+) (− a) ‘ U)
    by (simp add: ‹a ∈ U› affine_diffs_subspace_subtract ‹affine U› cong:
image_cong_simp)
  show subspace ((+) (− b) ‘ V)
    by (simp add: ‹b ∈ V› affine_diffs_subspace_subtract ‹affine V› cong:
image_cong_simp)
  show continuous_on ((+) (− a) ‘ S) ((+) (− b) ∘ f ∘ (+) a)
   by (metis contf continuous_on_compose homeomorphism_cont2 homeomor-
phism_translation translation_galois)
  show ((+) (− b) ∘ f ∘ (+) a) ‘ (+) (− a) ‘ S ⊆ (+) (− b) ‘ V
   using fim by auto
  show inj_on ((+) (− b) ∘ f ∘ (+) a) ((+) (− a) ‘ S)
   by (auto simp: inj_on_def) (meson inj_onD injf)
 qed (use ‹S ≠ {}› in auto)
 then show ?thesis
  by (metis ‹a ∈ U› ‹b ∈ V› aff_dim_eq_dim affine_hull_eq aff of_nat_le_iff)
qed

corollary invariance_of_dimension:
 fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
 assumes contf: continuous_on S f and open S
    and injf: inj_on f S and S ≠ {}
  shows DIM('a) ≤ DIM('b)
 using invariance_of_dimension_subspaces [of UNIV S UNIV f] assms
 by auto


corollary continuous_injective_image_subspace_dim_le:
 fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
 assumes subspace S subspace T
    and contf: continuous_on S f and fim: f ‘ S ⊆ T
    and injf: inj_on f S
  shows dim S ≤ dim T
 using invariance_of_dimension_subspaces [of S S _ f] assms by (auto simp:
subspace_affine)

lemma invariance_of_dimension_convex_domain:
 fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
 assumes convex S
```

```
    and contf: continuous_on S f and fim: f ‘ S ⊆ affine hull T
    and injf: inj_on f S
  shows aff_dim S ≤ aff_dim T
proof (cases S = {})
 case True
 then show ?thesis by (simp add: aff_dim_geq)
next
 case False
 have aff_dim (affine hull S) ≤ aff_dim (affine hull T)
 proof (rule invariance_of_dimension_affine_sets)
  show openin (top_of_set (affine hull S)) (rel_interior S)
   by (simp add: openin_rel_interior)
  show continuous_on (rel_interior S) f
   using contf continuous_on_subset rel_interior_subset by blast
  show f ‘ rel_interior S ⊆ affine hull T
   using fim rel_interior_subset by blast
  show inj_on f (rel_interior S)
   using inj_on_subset injf rel_interior_subset by blast
  show rel_interior S ≠ {}
   by (simp add: False ‹convex S› rel_interior_eq_empty)
 qed auto
 then show ?thesis
  by simp
qed


lemma homeomorphic_convex_sets_le:
 assumes convex S S homeomorphic T
 shows aff_dim S ≤ aff_dim T
proof −
 obtain h k where homhk: homeomorphism S T h k
  using homeomorphic_def assms  by blast
 show ?thesis
 proof (rule invariance_of_dimension_convex_domain [OF ‹convex S›])
  show continuous_on S h
   using homeomorphism_def homhk by blast
  show h ‘ S ⊆ affine hull T
   by (metis homeomorphism_def homhk hull_subset)
  show inj_on h S
   by (meson homeomorphism_apply1 homhk inj_on_inverseI)
 qed
qed

lemma homeomorphic_convex_sets:
 assumes convex S convex T S homeomorphic T
 shows aff_dim S = aff_dim T
 by (meson assms dual_order.antisym homeomorphic_convex_sets_le homeomor-
phic_sym)
```

lemma *homeomorphic_convex_compact_sets_eq*:
assumes *convex S compact S convex T compact T*
shows *S homeomorphic T* $\longleftrightarrow$ *aff_dim S = aff_dim T*
by (*meson assms homeomorphic_convex_compact_sets homeomorphic_convex_sets*)

lemma *invariance_of_domain_gen*:
fixes *f* :: *'a*::*euclidean_space* $\Rightarrow$ *'b*::*euclidean_space*
assumes *open S continuous_on S f inj_on f S DIM('b)* $\leq$ *DIM('a)*
shows *open*(*f* ‘ *S*)
using *invariance_of_domain_subspaces* [*of UNIV S UNIV f*] *assms* **by** *auto*

lemma *injective_into_1d_imp_open_map_UNIV*:
fixes *f* :: *'a*::*euclidean_space* $\Rightarrow$ *real*
assumes *open T continuous_on S f inj_on f S T* $\subseteq$ *S*
shows *open* (*f* ‘ *T*)
proof −
have *DIM*(*real*) $\leq$ *DIM*(*'a*)
by *simp*
then show *?thesis*
using *invariance_of_domain_gen assms continuous_on_subset subset_inj_on*
by *metis*
qed

lemma *continuous_on_inverse_open*:
fixes *f* :: *'a*::*euclidean_space* $\Rightarrow$ *'b*::*euclidean_space*
assumes *open S continuous_on S f DIM('b)* $\leq$ *DIM('a)* **and** *gf*: $\bigwedge x.\ x \in S \Longrightarrow$ *g*(*f x*) = *x*
shows *continuous_on* (*f* ‘ *S*) *g*
proof (*clarsimp simp add*: *continuous_openin_preimage_eq*)
fix *T* :: *'a set*
assume *open T*
have *eq*: *f* ‘ *S* ∩ *g* −‘ *T* = *f* ‘ (*S* ∩ *T*)
by (*auto simp*: *gf*)
have *open* (*f* ‘ *S*)
by (*rule invariance_of_domain_gen*) (*use assms inj_on_inverseI* **in** *auto*)
moreover have *open* (*f* ‘ (*S* ∩ *T*))
using *assms*
by (*metis* ‹*open T*› *continuous_on_subset inj_onI inj_on_subset invariance_of_domain_gen openin_open openin_open_eq*)
ultimately show *openin* (*top_of_set* (*f* ‘ *S*)) (*f* ‘ *S* ∩ *g* −‘ *T*)
unfolding *eq* **by** (*auto intro*: *open_openin_trans*)
qed

lemma *invariance_of_domain_homeomorphism*:
fixes *f* :: *'a*::*euclidean_space* $\Rightarrow$ *'b*::*euclidean_space*
assumes *open S continuous_on S f DIM('b)* $\leq$ *DIM('a) inj_on f S*
obtains *g* **where** *homeomorphism S* (*f* ‘ *S*) *f g*
proof
show *homeomorphism S* (*f* ‘ *S*) *f* (*inv_into S f*)

```
    by (simp add: assms continuous_on_inverse_open homeomorphism_def)
qed

corollary invariance_of_domain_homeomorphic:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes open S continuous_on S f DIM('b) ≤ DIM('a) inj_on f S
  shows S homeomorphic (f ‘ S)
  using invariance_of_domain_homeomorphism [OF assms]
  by (meson homeomorphic_def)

lemma continuous_image_subset_interior:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes continuous_on S f inj_on f S DIM('b) ≤ DIM('a)
  shows f ‘ (interior S) ⊆ interior(f ‘ S)
proof −
  have open (f ‘ interior S)
    using assms
   by (intro invariance_of_domain_gen) (auto simp: subset_inj_on interior_subset
continuous_on_subset)
  then show ?thesis
    by (simp add: image_mono interior_maximal interior_subset)
qed

lemma homeomorphic_interiors_same_dimension:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T and dimeq: DIM('a) = DIM('b)
  shows (interior S) homeomorphic (interior T)
  using assms [unfolded homeomorphic_minimal]
  unfolding homeomorphic_def
proof (clarify elim!: ex_forward)
  fix f g
  assume S: ∀ x∈S. f x ∈ T ∧ g (f x) = x and T: ∀ y∈T. g y ∈ S ∧ f (g y) = y
     and contf: continuous_on S f and contg: continuous_on T g
  then have fST: f ‘ S = T and gTS: g ‘ T = S and inj_on f S inj_on g T
    by (auto simp: inj_on_def intro: rev_image_eqI) metis+
  have fim: f ‘ interior S ⊆ interior T
    using continuous_image_subset_interior [OF contf ‹inj_on f S›] dimeq fST
by simp
  have gim: g ‘ interior T ⊆ interior S
    using continuous_image_subset_interior [OF contg ‹inj_on g T›] dimeq gTS
by simp
  show homeomorphism (interior S) (interior T) f g
    unfolding homeomorphism_def
  proof (intro conjI ballI)
    show ⋀x. x ∈ interior S ⟹ g (f x) = x
      by (meson ‹∀ x∈S. f x ∈ T ∧ g (f x) = x› subsetD interior_subset)
    have interior T ⊆ f ‘ interior S
    proof
      fix x assume x ∈ interior T
```

```
      then have g x ∈ interior S
        using gim by blast
      then show x ∈ f ‘ interior S
        by (metis T ‹x ∈ interior T› image_iff interior_subset subsetCE)
    qed
    then show f ‘ interior S = interior T
      using fim by blast
    show continuous_on (interior S) f
      by (metis interior_subset continuous_on_subset contf)
    show ⋀y. y ∈ interior T ⟹ f (g y) = y
      by (meson T subsetD interior_subset)
    have interior S ⊆ g ‘ interior T
    proof
      fix x assume x ∈ interior S
      then have f x ∈ interior T
        using fim by blast
      then show x ∈ g ‘ interior T
        by (metis S ‹x ∈ interior S› image_iff interior_subset subsetCE)
    qed
    then show g ‘ interior T = interior S
      using gim by blast
    show continuous_on (interior T) g
      by (metis interior_subset continuous_on_subset contg)
  qed
qed

lemma homeomorphic_open_imp_same_dimension:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T open S S ≠ {} open T T ≠ {}
  shows DIM('a) = DIM('b)
    using assms
    apply (simp add: homeomorphic_minimal)
    apply (rule order_antisym; metis inj_onI invariance_of_dimension)
    done

proposition homeomorphic_interiors:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T interior S = {} ⟷ interior T = {}
    shows (interior S) homeomorphic (interior T)
proof (cases interior T = {})
  case False
  then have DIM('a) = DIM('b)
    using assms
    apply (simp add: homeomorphic_minimal)
   apply (rule order_antisym; metis continuous_on_subset inj_onI inj_on_subset
interior_subset invariance_of_dimension open_interior)
    done
  then show ?thesis
   by (rule homeomorphic_interiors_same_dimension [OF ‹S homeomorphic T›])
```

qed (*use assms* **in** *auto*)

lemma *homeomorphic_frontiers_same_dimension*:
fixes S :: *'a*::*euclidean_space set* **and** T :: *'b*::*euclidean_space set*
assumes S *homeomorphic* T *closed* S *closed* T **and** *dimeq*: $DIM('a) = DIM('b)$
shows (*frontier* S) *homeomorphic* (*frontier* T)
using *assms* [*unfolded homeomorphic_minimal*]
unfolding *homeomorphic_def*
proof (*clarify elim*!: *ex_forward*)
fix f g
assume S: $\forall x{\in}S.\ f\ x \in T \wedge g\ (f\ x) = x$ **and** T: $\forall y{\in}T.\ g\ y \in S \wedge f\ (g\ y) = y$
and *contf*: *continuous_on* S f **and** *contg*: *continuous_on* T g
then have *fST*: $f\ `\ S = T$ **and** *gTS*: $g\ `\ T = S$ **and** *inj_on f S inj_on g T*
by (*auto simp*: *inj_on_def intro*: *rev_image_eqI*) *metis*+
have $g\ `$ *interior* $T \subseteq$ *interior* S
using *continuous_image_subset_interior* [*OF contg* ‹*inj_on g T*›] *dimeq gTS*
by *simp*
then have *fim*: $f\ `$ *frontier* $S \subseteq$ *frontier* T
unfolding *frontier_def*
using *continuous_image_subset_interior assms*(*2*) *assms*(*3*) S **by** *auto*
have $f\ `$ *interior* $S \subseteq$ *interior* T
using *continuous_image_subset_interior* [*OF contf* ‹*inj_on f S*›] *dimeq fST*
by *simp*
then have *gim*: $g\ `$ *frontier* $T \subseteq$ *frontier* S
unfolding *frontier_def*
using *continuous_image_subset_interior* T *assms*(*2*) *assms*(*3*) **by** *auto*
show *homeomorphism* (*frontier* S) (*frontier* T) f g
unfolding *homeomorphism_def*
proof (*intro conjI ballI*)
show *gf*: $\bigwedge x.\ x \in$ *frontier* $S \Longrightarrow g\ (f\ x) = x$
by (*simp add*: S *assms*(*2*) *frontier_def*)
show *fg*: $\bigwedge y.\ y \in$ *frontier* $T \Longrightarrow f\ (g\ y) = y$
by (*simp add*: T *assms*(*3*) *frontier_def*)
have *frontier* $T \subseteq f\ `$ *frontier* S
proof
fix x **assume** $x \in$ *frontier* T
then have $g\ x \in$ *frontier* S
using *gim* **by** *blast*
then show $x \in f\ `$ *frontier* S
by (*metis fg* ‹$x \in$ *frontier* T› *imageI*)
qed
then show $f\ `$ *frontier* S = *frontier* T
using *fim* **by** *blast*
show *continuous_on* (*frontier* S) f
by (*metis Diff_subset assms*(*2*) *closure_eq contf continuous_on_subset frontier_def*)
have *frontier* $S \subseteq g\ `$ *frontier* T
proof
fix x **assume** $x \in$ *frontier* S

```
      then have f x ∈ frontier T
        using fim by blast
      then show x ∈ g ‘ frontier T
        by (metis gf ‹x ∈ frontier S› imageI)
    qed
    then show g ‘ frontier T = frontier S
      using gim by blast
    show continuous_on (frontier T) g
      by (metis Diff_subset assms(3) closure_closed contg continuous_on_subset
frontier_def)
  qed
qed

lemma homeomorphic_frontiers:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T closed S closed T
          interior S = {} ⟷ interior T = {}
    shows (frontier S) homeomorphic (frontier T)
proof (cases interior T = {})
  case True
  then show ?thesis
    by (metis Diff_empty assms closure_eq frontier_def)
next
  case False
  then have DIM('a) = DIM('b)
   using assms homeomorphic_interiors homeomorphic_open_imp_same_dimension
by blast
  then show ?thesis
    using assms homeomorphic_frontiers_same_dimension by blast
qed

lemma continuous_image_subset_rel_interior:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes contf: continuous_on S f and injf: inj_on f S and fim: f ‘ S ⊆ T
      and TS: aff_dim T ≤ aff_dim S
  shows f ‘ (rel_interior S) ⊆ rel_interior(f ‘ S)
proof (rule rel_interior_maximal)
  show f ‘ rel_interior S ⊆ f ‘ S
    by(simp add: image_mono rel_interior_subset)
  show openin (top_of_set (affine hull f ‘ S)) (f ‘ rel_interior S)
  proof (rule invariance_of_domain_affine_sets)
    show openin (top_of_set (affine hull S)) (rel_interior S)
      by (simp add: openin_rel_interior)
    show aff_dim (affine hull f ‘ S) ≤ aff_dim (affine hull S)
      by (metis aff_dim_affine_hull aff_dim_subset fim TS order_trans)
    show f ‘ rel_interior S ⊆ affine hull f ‘ S
      by (meson ‹f ‘ rel_interior S ⊆ f ‘ S› hull_subset order_trans)
    show continuous_on (rel_interior S) f
      using contf continuous_on_subset rel_interior_subset by blast
```

```
    show inj_on f (rel_interior S)
      using inj_on_subset injf rel_interior_subset by blast
  qed auto
qed

lemma homeomorphic_rel_interiors_same_dimension:
  fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
  assumes S homeomorphic T and aff: aff_dim S = aff_dim T
  shows (rel_interior S) homeomorphic (rel_interior T)
  using assms [unfolded homeomorphic_minimal]
  unfolding homeomorphic_def
proof (clarify elim!: ex_forward)
  fix f g
  assume S: ∀ x∈S. f x ∈ T ∧ g (f x) = x and T: ∀ y∈T. g y ∈ S ∧ f (g y) = y
     and contf: continuous_on S f and contg: continuous_on T g
  then have fST: f ‘ S = T and gTS: g ‘ T = S and inj_on f S inj_on g T
    by (auto simp: inj_on_def intro: rev_image_eqI) metis+
  have fim: f ‘ rel_interior S ⊆ rel_interior T
    by (metis ‹inj_on f S› aff contf continuous_image_subset_rel_interior fST
order_refl)
  have gim: g ‘ rel_interior T ⊆ rel_interior S
    by (metis ‹inj_on g T› aff contg continuous_image_subset_rel_interior gTS
order_refl)
  show homeomorphism (rel_interior S) (rel_interior T) f g
    unfolding homeomorphism_def
  proof (intro conjI ballI)
    show gf: ⋀x. x ∈ rel_interior S ⟹ g (f x) = x
      using S rel_interior_subset by blast
    show fg: ⋀y. y ∈ rel_interior T ⟹ f (g y) = y
      using T mem_rel_interior_ball by blast
    have rel_interior T ⊆ f ‘ rel_interior S
    proof
      fix x assume x ∈ rel_interior T
      then have g x ∈ rel_interior S
        using gim by blast
      then show x ∈ f ‘ rel_interior S
        by (metis fg ‹x ∈ rel_interior T› imageI)
    qed
    moreover have f ‘ rel_interior S ⊆ rel_interior T
      by (metis ‹inj_on f S› aff contf continuous_image_subset_rel_interior fST
order_refl)
    ultimately show f ‘ rel_interior S = rel_interior T
      by blast
    show continuous_on (rel_interior S) f
      using contf continuous_on_subset rel_interior_subset by blast
    have rel_interior S ⊆ g ‘ rel_interior T
    proof
      fix x assume x ∈ rel_interior S
      then have f x ∈ rel_interior T
```

```
      using fim by blast
    then show x ∈ g ` rel_interior T
      by (metis gf ‹x ∈ rel_interior S› imageI)
  qed
  then show g ` rel_interior T = rel_interior S
    using gim by blast
  show continuous_on (rel_interior T) g
    using contg continuous_on_subset rel_interior_subset by blast
 qed
qed


lemma homeomorphic_aff_dim_le:
 fixes S :: 'a::euclidean_space set
 assumes S homeomorphic T rel_interior S ≠ {}
  shows aff_dim (affine hull S) ≤ aff_dim (affine hull T)
proof −
 obtain f g
  where S: ∀ x∈S. f x ∈ T ∧ g (f x) = x and T: ∀ y∈T. g y ∈ S ∧ f (g y) = y
    and contf: continuous_on S f and contg: continuous_on T g
  using assms [unfolded homeomorphic_minimal] by auto
 show ?thesis
 proof (rule invariance_of_dimension_affine_sets)
  show continuous_on (rel_interior S) f
    using contf continuous_on_subset rel_interior_subset by blast
  show f ` rel_interior S ⊆ affine hull T
    by (meson S hull_subset image_subsetI rel_interior_subset rev_subsetD)
  show inj_on f (rel_interior S)
    by (metis S inj_on_inverseI inj_on_subset rel_interior_subset)
 qed (simp_all add: openin_rel_interior assms)
qed

lemma homeomorphic_rel_interiors:
 fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
 assumes S homeomorphic T rel_interior S = {} ⟷ rel_interior T = {}
  shows (rel_interior S) homeomorphic (rel_interior T)
proof (cases rel_interior T = {})
 case True
 with assms show ?thesis by auto
next
 case False
 have aff_dim (affine hull S) ≤ aff_dim (affine hull T)
  using False assms homeomorphic_aff_dim_le by blast
 moreover have aff_dim (affine hull T) ≤ aff_dim (affine hull S)
  using False assms(1) homeomorphic_aff_dim_le homeomorphic_sym by auto
 ultimately have aff_dim S = aff_dim T by force
 then show ?thesis
  by (rule homeomorphic_rel_interiors_same_dimension [OF ‹S homeomorphic
T›])
```

qed

lemma *homeomorphic_rel_boundaries_same_dimension*:
fixes $S :: 'a::euclidean\_space\ set$ **and** $T :: 'b::euclidean\_space\ set$
assumes S *homeomorphic* T **and** *aff*: $aff\_dim\ S = aff\_dim\ T$
shows $(S - rel\_interior\ S)$ *homeomorphic* $(T - rel\_interior\ T)$
using *assms* [*unfolded homeomorphic_minimal*]
unfolding *homeomorphic_def*
proof (*clarify elim*!: *ex_forward*)
fix $f\ g$
assume S: $\forall x{\in}S.\ f\ x \in T \wedge g\ (f\ x) = x$ **and** T: $\forall y{\in}T.\ g\ y \in S \wedge f\ (g\ y) = y$
and *contf*: *continuous_on* $S\ f$ **and** *contg*: *continuous_on* $T\ g$
then have *fST*: $f\ `\ S = T$ **and** *gTS*: $g\ `\ T = S$ **and** *inj_on* $f\ S$ *inj_on* $g\ T$
by (*auto simp*: *inj_on_def intro*: *rev_image_eqI*) *metis*+
have *fim*: $f\ `\ rel\_interior\ S \subseteq rel\_interior\ T$
by (*metis* ‹*inj_on f S*› *aff contf continuous_image_subset_rel_interior fST order_refl*)
have *gim*: $g\ `\ rel\_interior\ T \subseteq rel\_interior\ S$
by (*metis* ‹*inj_on g T*› *aff contg continuous_image_subset_rel_interior gTS order_refl*)
show *homeomorphism* $(S - rel\_interior\ S)\ (T - rel\_interior\ T)\ f\ g$
unfolding *homeomorphism_def*
proof (*intro conjI ballI*)
show *gf*: $\bigwedge x.\ x \in S - rel\_interior\ S \Longrightarrow g\ (f\ x) = x$
using *S rel_interior_subset* **by** *blast*
show *fg*: $\bigwedge y.\ y \in T - rel\_interior\ T \Longrightarrow f\ (g\ y) = y$
using *T mem_rel_interior_ball* **by** *blast*
show $f\ `\ (S - rel\_interior\ S) = T - rel\_interior\ T$
using *S fST fim gim* **by** *auto*
show *continuous_on* $(S - rel\_interior\ S)\ f$
using *contf continuous_on_subset rel_interior_subset* **by** *blast*
show $g\ `\ (T - rel\_interior\ T) = S - rel\_interior\ S$
using *T gTS gim fim* **by** *auto*
show *continuous_on* $(T - rel\_interior\ T)\ g$
using *contg continuous_on_subset rel_interior_subset* **by** *blast*
qed
qed

lemma *homeomorphic_rel_boundaries*:
fixes $S :: 'a::euclidean\_space\ set$ **and** $T :: 'b::euclidean\_space\ set$
assumes S *homeomorphic* T $rel\_interior\ S = \{\} \longleftrightarrow rel\_interior\ T = \{\}$
shows $(S - rel\_interior\ S)$ *homeomorphic* $(T - rel\_interior\ T)$
proof (*cases* $rel\_interior\ T = \{\}$)
case *True*
with *assms* **show** *?thesis* **by** *auto*
next
case *False*
obtain $f\ g$

where S: $\forall x{\in}S.\ f\ x \in T \wedge g\ (f\ x) = x$ **and** T: $\forall y{\in}T.\ g\ y \in S \wedge f\ (g\ y) = y$
and *contf*: *continuous_on S f* **and** *contg*: *continuous_on T g*
using *assms* **by** (*auto simp*: *homeomorphic_minimal*)
have *aff_dim* (*affine hull S*) ≤ *aff_dim* (*affine hull T*)
using *False assms homeomorphic_aff_dim_le* **by** *blast*
moreover have *aff_dim* (*affine hull T*) ≤ *aff_dim* (*affine hull S*)
by (*meson False assms*(*1*) *homeomorphic_aff_dim_le homeomorphic_sym*)
ultimately have *aff_dim S* = *aff_dim T* **by** *force*
then show *?thesis*
by (*rule homeomorphic_rel_boundaries_same_dimension* [*OF* ‹*S homeomorphic T*›])
qed

proposition *uniformly_continuous_homeomorphism_UNIV_trivial*:
fixes f :: $'a$::*euclidean_space* ⇒ $'a$
assumes *contf*: *uniformly_continuous_on S f* **and** *hom*: *homeomorphism S UNIV f g*
shows *S* = *UNIV*
proof (*cases S* = {})
case *True*
then show *?thesis*
by (*metis UNIV_I hom empty_iff homeomorphism_def image_eqI*)
next
case *False*
have *inj g*
by (*metis UNIV_I hom homeomorphism_apply2 injI*)
then have *open* (*g* ‘ *UNIV*)
by (*blast intro*: *invariance_of_domain hom homeomorphism_cont2*)
then have *open S*
using *hom homeomorphism_image2* **by** *blast*
moreover have *complete S*
unfolding *complete_def*
proof *clarify*
fix σ
assume σ: $\forall n.\ \sigma\ n \in S$ **and** *Cauchy* σ
have *Cauchy* (*f* ∘ σ)
using *uniformly_continuous_imp_Cauchy_continuous* ‹*Cauchy* σ› σ *contf*
unfolding *Cauchy_continuous_on_def* **by** *blast*
then obtain *l* **where** (*f* ∘ σ) ⟶ *l*
by (*auto simp*: *convergent_eq_Cauchy* [*symmetric*])
show ∃ *l*∈*S*. σ ⟶ *l*
proof
show *g l* ∈ *S*
using *hom homeomorphism_image2* **by** *blast*
have (*g* ∘ (*f* ∘ σ)) ⟶ *g l*
by (*meson UNIV_I* ‹(*f* ∘ σ) ⟶ *l*› *continuous_on_sequentially hom homeomorphism_cont2*)
then show σ ⟶ *g l*
proof −

```
      have ∀ n. σ n = (g ∘ (f ∘ σ)) n
       by (metis (no_types) σ comp_eq_dest_lhs hom homeomorphism_apply1)
      then show ?thesis
       by (metis (no_types) LIMSEQ_iff ‹(g ∘ (f ∘ σ)) ⟶ g l›)
    qed
   qed
  qed
  then have closed S
   by (simp add: complete_eq_closed)
  ultimately show ?thesis
   using clopen [of S] False  by simp
qed
```

6.36.6 Formulation of loop homotopy in terms of maps out of type complex

```
lemma homotopic_circlemaps_imp_homotopic_loops:
  assumes homotopic_with_canon (λh. True) (sphere 0 1) S f g
   shows homotopic_loops S (f ∘ exp ∘ (λt. 2 * of_real pi * of_real t * i))
                           (g ∘ exp ∘ (λt. 2 * of_real pi * of_real t * i))
proof −
  have homotopic_with_canon (λf. True) {z. cmod z = 1} S f g
   using assms by (auto simp: sphere_def)
  moreover have continuous_on {0..1} (exp ∘ (λt. 2 * of_real pi * of_real t *
i))
    by (intro continuous_intros)
  moreover have (exp ∘ (λt. 2 * of_real pi * of_real t * i)) ‘ {0..1} ⊆ {z. cmod
z = 1}
   by (auto simp: norm_mult)
  ultimately
  show ?thesis
   apply (simp add: homotopic_loops_def comp_assoc)
   apply (rule homotopic_with_compose_continuous_right)
    apply (auto simp: pathstart_def pathfinish_def)
   done
qed

lemma homotopic_loops_imp_homotopic_circlemaps:
  assumes homotopic_loops S p q
   shows homotopic_with_canon (λh. True) (sphere 0 1) S
                              (p ∘ (λz. (Arg2pi z / (2 * pi))))
                              (q ∘ (λz. (Arg2pi z / (2 * pi))))
proof −
  obtain h where conth: continuous_on ({0..1::real} × {0..1}) h
           and him: h ‘ ({0..1} × {0..1}) ⊆ S
           and h0: (∀ x. h (0, x) = p x)
           and h1: (∀ x. h (1, x) = q x)
           and h01: (∀ t∈{0..1}. h (t, 1) = h (t, 0))
    using assms
```

```
    by (auto simp: homotopic_loops_def sphere_def homotopic_with_def path-
start_def pathfinish_def)
  define j where j ≡ λz. if 0 ≤ Im (snd z)
                         then h (fst z, Arg2pi (snd z) / (2 * pi))
                         else h (fst z, 1 − Arg2pi (cnj (snd z)) / (2 * pi))
  have Arg2pi_eq: 1 − Arg2pi (cnj y) / (2 * pi) = Arg2pi y / (2 * pi) ∨ Arg2pi
y = 0 ∧ Arg2pi (cnj y) = 0 if cmod y = 1 for y
   using that Arg2pi_eq_0_pi Arg2pi_eq_pi by (force simp: Arg2pi_cnj field_split_simps)
  show ?thesis
  proof (simp add: homotopic_with; intro conjI ballI exI)
    show continuous_on ({0..1} × sphere 0 1) (λw. h (fst w, Arg2pi (snd w) / (2
* pi)))
     proof (rule continuous_on_eq)
      show j: j x = h (fst x, Arg2pi (snd x) / (2 * pi)) if x ∈ {0..1} × sphere 0 1
for x
        using Arg2pi_eq that h01 by (force simp: j_def)
       have eq:  S = S ∩ (UNIV × {z. 0 ≤ Im z}) ∪ S ∩ (UNIV × {z. Im z ≤
0}) for S :: (real*complex)set
        by auto
       have §: Arg2pi z ≤ 2 * pi for z
        by (simp add: Arg2pi order_le_less)
       have c1: continuous_on ({0..1} × sphere 0 1 ∩ UNIV × {z. 0 ≤ Im z})
(λx. h (fst x, Arg2pi (snd x) / (2 * pi)))
        apply (intro continuous_intros continuous_on_compose2 [OF conth] con-
tinuous_on_compose2 [OF continuous_on_upperhalf_Arg2pi])
        by (auto simp: Arg2pi §)
       have c2: continuous_on ({0..1} × sphere 0 1 ∩ UNIV × {z. Im z ≤ 0})
(λx. h (fst x, 1 − Arg2pi (cnj (snd x)) / (2 * pi)))
        apply (intro continuous_intros continuous_on_compose2 [OF conth] con-
tinuous_on_compose2 [OF continuous_on_upperhalf_Arg2pi])
        by (auto simp: Arg2pi §)
      show continuous_on ({0..1} × sphere 0 1) j
        apply (simp add: j_def)
        apply (subst eq)
        apply (rule continuous_on_cases_local)
        using Arg2pi_eq h01
          by (force simp add: eq [symmetric] closedin_closed_Int closed_Times
closed_halfspace_Im_le closed_halfspace_Im_ge c1 c2)+
    qed
    have (λw. h (fst w, Arg2pi (snd w) / (2 * pi))) ‘ ({0..1} × sphere 0 1) ⊆ h ‘
({0..1} × {0..1})
      by (auto simp: Arg2pi_ge_0 Arg2pi_lt_2pi less_imp_le)
    also have ... ⊆ S
      using him by blast
    finally show (λw. h (fst w, Arg2pi (snd w) / (2 * pi))) ‘ ({0..1} × sphere 0
1) ⊆ S .
  qed (auto simp: h0 h1)
qed
```

lemma *simply_connected_homotopic_loops*:
simply_connected S ⟷
(∀ *p q*. *homotopic_loops S p p* ∧ *homotopic_loops S q q* ⟶ *homotopic_loops S p q*)
unfolding *simply_connected_def* **using** *homotopic_loops_refl* **by** *metis*

lemma *simply_connected_eq_homotopic_circlemaps1*:
fixes *f* :: *complex* ⇒ *'a*::*topological_space* **and** *g* :: *complex* ⇒ *'a*
assumes *S*: *simply_connected S*
and *contf*: *continuous_on* (*sphere 0 1*) *f* **and** *fim*: *f* ‘ (*sphere 0 1*) ⊆ *S*
and *contg*: *continuous_on* (*sphere 0 1*) *g* **and** *gim*: *g* ‘ (*sphere 0 1*) ⊆ *S*
shows *homotopic_with_canon* (λ*h*. *True*) (*sphere 0 1*) *S f g*
proof −
let *?h* = (λ*t*. *complex_of_real* (*2* ∗ *pi* ∗ *t*) ∗ i)
have *homotopic_loops S* (*f* ∘ *exp* ∘ *?h*) (*f* ∘ *exp* ∘ *?h*) ∧ *homotopic_loops S* (*g* ∘ *exp* ∘ *?h*) (*g* ∘ *exp* ∘ *?h*)
by (*simp add*: *homotopic_circlemaps_imp_homotopic_loops contf fim contg gim*)
then have *homotopic_loops S* (*f* ∘ *exp* ∘ *?h*) (*g* ∘ *exp* ∘ *?h*)
using *S simply_connected_homotopic_loops* **by** *blast*
then show *?thesis*
apply (*rule homotopic_with_eq* [*OF homotopic_loops_imp_homotopic_circlemaps*])
apply (*auto simp*: *o_def complex_norm_eq_1_exp mult.commute*)
done
qed

lemma *simply_connected_eq_homotopic_circlemaps2a*:
fixes *h* :: *complex* ⇒ *'a*::*topological_space*
assumes *conth*: *continuous_on* (*sphere 0 1*) *h* **and** *him*: *h* ‘ (*sphere 0 1*) ⊆ *S*
and *hom*: ⋀*f g*::*complex* ⇒ *'a*.
⟦*continuous_on* (*sphere 0 1*) *f*; *f* ‘ (*sphere 0 1*) ⊆ *S*;
continuous_on (*sphere 0 1*) *g*; *g* ‘ (*sphere 0 1*) ⊆ *S*⟧
⟹ *homotopic_with_canon* (λ*h*. *True*) (*sphere 0 1*) *S f g*
shows ∃ *a*. *homotopic_with_canon* (λ*h*. *True*) (*sphere 0 1*) *S h* (λ*x*. *a*)
by (*metis conth continuous_on_const him hom image_subset_iff*)

lemma *simply_connected_eq_homotopic_circlemaps2b*:
fixes *S* :: *'a*::*real_normed_vector set*
assumes ⋀*f g*::*complex* ⇒ *'a*.
⟦*continuous_on* (*sphere 0 1*) *f*; *f* ‘ (*sphere 0 1*) ⊆ *S*;
continuous_on (*sphere 0 1*) *g*; *g* ‘ (*sphere 0 1*) ⊆ *S*⟧
⟹ *homotopic_with_canon* (λ*h*. *True*) (*sphere 0 1*) *S f g*
shows *path_connected S*
proof (*clarsimp simp add*: *path_connected_eq_homotopic_points*)
fix *a b*
assume *a* ∈ *S b* ∈ *S*
then show *homotopic_loops S* (*linepath a a*) (*linepath b b*)
using *homotopic_circlemaps_imp_homotopic_loops* [*OF assms* [*of* λ*x*. *a* λ*x*.

b]]
by (*auto simp*: *o_def linepath_def*)
qed

lemma *simply_connected_eq_homotopic_circlemaps3*:
fixes *h* :: *complex* ⇒ *'a*::*real_normed_vector*
assumes *path_connected S*
and *hom*: ⋀*f*::*complex* ⇒ *'a*.
⟦*continuous_on* (*sphere 0 1*) *f*; *f* ‘(*sphere 0 1*) ⊆ *S*⟧
⟹ ∃ *a*. *homotopic_with_canon* (λ*h*. *True*) (*sphere 0 1*) *S f* (λ*x*. *a*)
shows *simply_connected S*
proof (*clarsimp simp add*: *simply_connected_eq_contractible_loop_some assms*)
fix *p*
assume *p*: *path p path_image p* ⊆ *S pathfinish p* = *pathstart p*
then have *homotopic_loops S p p*
by (*simp add*: *homotopic_loops_refl*)
then obtain *a* **where** *homp*: *homotopic_with_canon* (λ*h*. *True*) (*sphere 0 1*) *S*
(*p* ∘ (λ*z*. *Arg2pi z* / (*2* ∗ *pi*))) (λ*x*. *a*)
by (*metis homotopic_with_imp_subset2 homotopic_loops_imp_homotopic_circlemaps*
homotopic_with_imp_continuous hom)
show ∃ *a*. *a* ∈ *S* ∧ *homotopic_loops S p* (*linepath a a*)
proof (*intro exI conjI*)
show *a* ∈ *S*
using *homotopic_with_imp_subset2* [*OF homp*]
by (*metis dist_0_norm image_subset_iff mem_sphere norm_one*)
have *teq*: ⋀*t*. ⟦*0* ≤ *t*; *t* ≤ *1*⟧
⟹ *t* = *Arg2pi* (*exp* (*2* ∗ *of_real pi* ∗ *of_real t* ∗ i)) / (*2* ∗ *pi*) ∨ *t=1*
∧ *Arg2pi* (*exp* (*2* ∗ *of_real pi* ∗ *of_real t* ∗ i)) = *0*
using *Arg2pi_of_real* [*of 1*] **by** (*force simp*: *Arg2pi_exp*)
have *homotopic_loops S p* (*p* ∘ (λ*z*. *Arg2pi z* / (*2* ∗ *pi*)) ∘ *exp* ∘ (λ*t*. *2* ∗
complex_of_real pi ∗ *complex_of_real t* ∗ i))
using *p teq* **by** (*fastforce simp*: *pathfinish_def pathstart_def intro*: *homo-*
topic_loops_eq [*OF p*])
then show *homotopic_loops S p* (*linepath a a*)
by (*simp add*: *linepath_refl homotopic_loops_trans* [*OF _ homotopic_circlemaps_imp_homotopic_loops*
[*OF homp*, *simplified K_record_comp*]])
qed
qed

proposition *simply_connected_eq_homotopic_circlemaps*:
fixes *S* :: *'a*::*real_normed_vector set*
shows *simply_connected S* ⟷
(∀ *f g*::*complex* ⇒ *'a*.
continuous_on (*sphere 0 1*) *f* ∧ *f* ‘ (*sphere 0 1*) ⊆ *S* ∧
continuous_on (*sphere 0 1*) *g* ∧ *g* ‘ (*sphere 0 1*) ⊆ *S*
⟶ *homotopic_with_canon* (λ*h*. *True*) (*sphere 0 1*) *S f g*)
by (*metis simply_connected_eq_homotopic_circlemaps1 simply_connected_eq_homotopic_circlemaps2a*

simply_connected_eq_homotopic_circlemaps2b simply_connected_eq_homotopic_circlemaps3)

proposition *simply_connected_eq_contractible_circlemap*:
fixes $S :: 'a::real\_normed\_vector\ set$
shows *simply_connected S* $\longleftrightarrow$
path_connected S $\wedge$
$(\forall f::complex \Rightarrow 'a.$
continuous_on (*sphere 0 1*) *f* $\wedge$ *f* ‘(*sphere 0 1*) $\subseteq S$
$\longrightarrow$ ($\exists a.$ *homotopic_with_canon* ($\lambda h.$ *True*) (*sphere 0 1*) *S f* ($\lambda x.$ *a*)))
by (*metis simply_connected_eq_homotopic_circlemaps simply_connected_eq_homotopic_circlemaps2*

simply_connected_eq_homotopic_circlemaps3 simply_connected_imp_path_connected)

corollary *homotopy_eqv_simple_connectedness*:
fixes $S :: 'a::real\_normed\_vector\ set$ **and** $T :: 'b::real\_normed\_vector\ set$
shows *S homotopy_eqv T* $\Longrightarrow$ *simply_connected S* $\longleftrightarrow$ *simply_connected T*
by (*simp add*: *simply_connected_eq_homotopic_circlemaps homotopy_eqv_homotopic_triviality image_subset_iff_funcset*)

6.36.7 Homeomorphism of simple closed curves to circles

proposition *homeomorphic_simple_path_image_circle*:
fixes *a* :: *complex* **and** γ :: *real* $\Rightarrow$ $'a::t2\_space$
assumes *simple_path* γ **and** *loop*: *pathfinish* γ = *pathstart* γ **and** *0* < *r*
shows (*path_image* γ) *homeomorphic sphere a r*
proof −
have *homotopic_loops* (*path_image* γ) γ γ
by (*simp add*: *assms homotopic_loops_refl simple_path_imp_path*)
then have *hom*: *homotopic_with_canon* ($\lambda h.$ *True*) (*sphere 0 1*) (*path_image* γ)
($\gamma \circ$ ($\lambda z.$ *Arg2pi z* / (*2***pi*))) ($\gamma \circ$ ($\lambda z.$ *Arg2pi z* / (*2***pi*)))
by (*rule homotopic_loops_imp_homotopic_circlemaps*)
have $\exists g.$ *homeomorphism* (*sphere 0 1*) (*path_image* γ) ($\gamma \circ$ ($\lambda z.$ *Arg2pi z* / (*2***pi*))) *g*
proof (*rule homeomorphism_compact*)
show *continuous_on* (*sphere 0 1*) ($\gamma \circ$ ($\lambda z.$ *Arg2pi z* / (*2***pi*)))
using *hom homotopic_with_imp_continuous* **by** *blast*
show *inj_on* ($\gamma \circ$ ($\lambda z.$ *Arg2pi z* / (*2***pi*))) (*sphere 0 1*)
proof
fix *x y*
assume *xy*: *x* ∈ *sphere 0 1 y* ∈ *sphere 0 1*
and *eq*: ($\gamma \circ$ ($\lambda z.$ *Arg2pi z* / (*2***pi*))) *x* = ($\gamma \circ$ ($\lambda z.$ *Arg2pi z* / (*2***pi*))) *y*
then have (*Arg2pi x* / (*2***pi*)) = (*Arg2pi y* / (*2***pi*))
proof −
have (*Arg2pi x* / (*2***pi*)) ∈ {*0..1*} (*Arg2pi y* / (*2***pi*)) ∈ {*0..1*}
using *Arg2pi_ge_0 Arg2pi_lt_2pi dual_order.strict_iff_order* **by** *fastforce+*
with *eq* **show** *?thesis*
using ‹*simple_path* γ› *Arg2pi_lt_2pi*

```
        unfolding simple_path_def loop_free_def o_def
        by (metis eq_divide_eq_1 not_less_iff_gr_or_eq)
    qed
    with xy show x = y
      by (metis is_Arg_def Arg2pi Arg2pi_0 dist_0_norm divide_cancel_right
dual_order.strict_iff_order mem_sphere)
  qed
  have ⋀z. cmod z = 1 ⟹ ∃x∈{0..1}. γ (Arg2pi z / (2*pi)) = γ x
    by (metis Arg2pi_ge_0 Arg2pi_lt_2pi atLeastAtMost_iff divide_less_eq_1
less_eq_real_def zero_less_mult_iff pi_gt_zero zero_le_divide_iff zero_less_numeral)
   moreover have ∃z∈sphere 0 1. γ x = γ (Arg2pi z / (2*pi)) if 0 ≤ x x ≤ 1
for x
   proof (cases x=1)
     case True
     with Arg2pi_of_real [of 1] loop show ?thesis
       by (rule_tac x=1 in bexI) (auto simp: pathfinish_def pathstart_def ‹0 ≤
x›)
   next
     case False
     then have *: (Arg2pi (exp (i*(2* of_real pi* of_real x))) / (2*pi)) = x
       using that by (auto simp: Arg2pi_exp field_split_simps)
     show ?thesis
       by (rule_tac x=exp(i * of_real(2*pi*x)) in bexI) (auto simp: *)
  qed
  ultimately show (γ ∘ (λz. Arg2pi z / (2*pi))) ` sphere 0 1 = path_image γ
    by (auto simp: path_image_def image_iff)
  qed auto
  then have path_image γ homeomorphic sphere (0::complex) 1
    using homeomorphic_def homeomorphic_sym by blast
 also have ... homeomorphic sphere a r
   by (simp add: assms homeomorphic_spheres)
 finally show ?thesis .
qed

lemma homeomorphic_simple_path_images:
 fixes γ1 :: real ⇒ 'a::t2_space and γ2 :: real ⇒ 'b::t2_space
 assumes simple_path γ1 and loop: pathfinish γ1 = pathstart γ1
 assumes simple_path γ2 and loop: pathfinish γ2 = pathstart γ2
 shows (path_image γ1) homeomorphic (path_image γ2)
 by (meson assms homeomorphic_simple_path_image_circle homeomorphic_sym
homeomorphic_trans loop pi_gt_zero)
```

6.36.8 Dimension-based conditions for various homeomorphisms

```
lemma homeomorphic_subspaces_eq:
 fixes S :: 'a::euclidean_space set and T :: 'b::euclidean_space set
 assumes subspace S subspace T
 shows S homeomorphic T ⟷ dim S = dim T
```

proof
assume *S homeomorphic T*
then obtain *f g* **where** *hom*: *homeomorphism S T f g*
using *homeomorphic_def* **by** *blast*
show *dim S = dim T*
proof (*rule order_antisym*)
show *dim S ≤ dim T*
by (*metis assms dual_order.refl inj_onI homeomorphism_cont1* [*OF hom*] *homeomorphism_apply1* [*OF hom*] *homeomorphism_image1* [*OF hom*] *continuous_injective_image_subspace_dim_le*)
show *dim T ≤ dim S*
by (*metis assms dual_order.refl inj_onI homeomorphism_cont2* [*OF hom*] *homeomorphism_apply2* [*OF hom*] *homeomorphism_image2* [*OF hom*] *continuous_injective_image_subspace_dim_le*)
qed
next
assume *dim S = dim T*
then show *S homeomorphic T*
by (*simp add*: *assms homeomorphic_subspaces*)
qed

lemma *homeomorphic_affine_sets_eq*:
fixes *S* :: *'a*::*euclidean_space set* **and** *T* :: *'b*::*euclidean_space set*
assumes *affine S affine T*
shows *S homeomorphic T* ⟷ *aff_dim S = aff_dim T*
proof (*cases S = {} ∨ T = {}*)
case *True*
then show *?thesis*
using *assms homeomorphic_affine_sets* **by** *force*
next
case *False*
then obtain *a b* **where** *a ∈ S b ∈ T*
by *blast*
then have *subspace* ((+) (− *a*) ‘ *S*) *subspace* ((+) (− *b*) ‘ *T*)
using *affine_diffs_subspace assms* **by** *blast+*
then show *?thesis*
by (*metis affine_imp_convex assms homeomorphic_affine_sets homeomorphic_convex_sets*)
qed

lemma *homeomorphic_hyperplanes_eq*:
fixes *a* :: *'a*::*euclidean_space* **and** *c* :: *'b*::*euclidean_space*
assumes *a ≠ 0 c ≠ 0*
shows ({*x*. *a* ⋅ *x* = *b*} *homeomorphic* {*x*. *c* ⋅ *x* = *d*} ⟷ *DIM*(*'a*) = *DIM*(*'b*))
proof −
have *DIM*(*'a*) − *Suc 0* = *DIM*(*'b*) − *Suc 0* ⟹ *DIM*(*'a*) = *DIM*(*'b*)
by (*metis DIM_positive Suc_pred*)
then show *?thesis*
by (*auto simp*: *homeomorphic_affine_sets_eq affine_hyperplane assms*)

qed

lemma *homeomorphic_UNIV_UNIV*:
shows (*UNIV*::'*a set*) *homeomorphic* (*UNIV*::'*b set*) ⟷
DIM('*a*::*euclidean_space*) = *DIM*('*b*::*euclidean_space*)
by (*simp add*: *homeomorphic_subspaces_eq*)

lemma *simply_connected_sphere_gen*:
assumes *convex S bounded S* **and** *3*: *3* ≤ *aff_dim S*
shows *simply_connected*(*rel_frontier S*)
proof −
have *pa*: *path_connected* (*rel_frontier S*)
using *assms* **by** (*simp add*: *path_connected_sphere_gen*)
show *?thesis*
proof (*clarsimp simp add*: *simply_connected_eq_contractible_circlemap pa*)
fix *f*
assume *f*: *continuous_on* (*sphere* (*0*::*complex*) *1*) *f f* ‘ *sphere 0 1* ⊆ *rel_frontier S*
have *eq*: *sphere* (*0*::*complex*) *1* = *rel_frontier*(*cball 0 1*)
by *simp*
have *convex* (*cball* (*0*::*complex*) *1*)
by (*rule convex_cball*)
then obtain *c* **where** *homotopic_with_canon* (λ*z*. *True*) (*sphere* (*0*::*complex*) *1*) (*rel_frontier S*) *f* (λ*x*. *c*)
apply (*rule inessential_spheremap_lowdim_gen* [*OF* _ *bounded_cball* ‹*convex S*› ‹*bounded S*›, **where** *f*=*f*])
using *f 3* **by** (*auto simp*: *aff_dim_cball*)
then show ∃ *a*. *homotopic_with_canon* (λ*h*. *True*) (*sphere 0 1*) (*rel_frontier S*) *f* (λ*x*. *a*)
by *blast*
qed
qed

6.36.9 more invariance of domain

proposition *invariance_of_domain_sphere_affine_set_gen*:
fixes *f* :: '*a*::*euclidean_space* ⇒ '*b*::*euclidean_space*
assumes *contf*: *continuous_on S f* **and** *injf*: *inj_on f S* **and** *fim*: *f* ‘ *S* ⊆ *T*
and *U*: *bounded U convex U*
and *affine T* **and** *affTU*: *aff_dim T* < *aff_dim U*
and *ope*: *openin* (*top_of_set* (*rel_frontier U*)) *S*
shows *openin* (*top_of_set T*) (*f* ‘ *S*)
proof (*cases rel_frontier U* = {})
case *True*
then show *?thesis*
using *ope openin_subset* **by** *force*
next
case *False*
obtain *b c* **where** *b*: *b* ∈ *rel_frontier U* **and** *c*: *c* ∈ *rel_frontier U* **and** *b* ≠ *c*

```
    using ‹bounded U› rel_frontier_not_sing [of U] subset_singletonD False  by
fastforce
  obtain V :: 'a set where affine V and affV: aff_dim V = aff_dim U − 1
  proof (rule choose_affine_subset [OF affine_UNIV])
    show − 1 ≤ aff_dim U − 1
      by (metis aff_dim_empty aff_dim_geq aff_dim_negative_iff affTU diff_0
diff_right_mono not_le)
    show aff_dim U − 1 ≤ aff_dim (UNIV::'a set)
      by (metis aff_dim_UNIV aff_dim_le_DIM le_cases not_le zle_diff1_eq)
  qed auto
  have SU: S ⊆ rel_frontier U
    using ope openin_imp_subset by auto
  have homb: rel_frontier U − {b} homeomorphic V
   and homc: rel_frontier U − {c} homeomorphic V
    using homeomorphic_punctured_sphere_affine_gen [of U _ V]
    by (simp_all add: ‹affine V› affV U b c)
  then obtain g h j k
          where gh: homeomorphism (rel_frontier U − {b}) V g h
            and jk: homeomorphism (rel_frontier U − {c}) V j k
    by (auto simp: homeomorphic_def)
  with SU have hgsub: (h ‘ g ‘ (S − {b})) ⊆ S and kjsub: (k ‘ j ‘ (S − {c})) ⊆ S
    by (simp_all add: homeomorphism_def subset_eq)
  have [simp]: aff_dim T ≤ aff_dim V
    by (simp add: affTU affV)
  have openin (top_of_set T) ((f ∘ h) ‘ g ‘ (S − {b}))
  proof (rule invariance_of_domain_affine_sets [OF _ ‹affine V›])
    have openin (top_of_set (rel_frontier U − {b})) (S − {b})
     by (meson Diff_mono Diff_subset SU ope openin_delete openin_subset_trans
order_refl)
    then show openin (top_of_set V) (g ‘ (S − {b}))
      by (rule homeomorphism_imp_open_map [OF gh])
    show continuous_on (g ‘ (S − {b})) (f ∘ h)
    proof (rule continuous_on_compose)
      show continuous_on (g ‘ (S − {b})) h
       by (meson Diff_mono SU homeomorphism_def homeomorphism_of_subsets
gh set_eq_subset)
    qed (use contf continuous_on_subset hgsub in blast)
    show inj_on (f ∘ h) (g ‘ (S − {b}))
     by (smt (verit, del_insts) SU homeomorphism_def inj_on_def injf gh Diff_iff
comp_apply imageE subset_iff)
    show (f ∘ h) ‘ g ‘ (S − {b}) ⊆ T
      by (metis fim image_comp image_mono hgsub subset_trans)
  qed (auto simp: assms)
  moreover
  have openin (top_of_set T) ((f ∘ k) ‘ j ‘ (S − {c}))
  proof (rule invariance_of_domain_affine_sets [OF _ ‹affine V›])
    show openin (top_of_set V) (j ‘ (S − {c}))
     by (meson Diff_mono Diff_subset SU ope openin_delete openin_subset_trans
order_refl homeomorphism_imp_open_map [OF jk])
```

```
    show continuous_on (j ‘ (S − {c})) (f ∘ k)
    proof (rule continuous_on_compose)
      show continuous_on (j ‘ (S − {c})) k
       by (meson Diff_mono SU homeomorphism_def homeomorphism_of_subsets
jk set_eq_subset)
    qed (use contf continuous_on_subset kjsub in blast)
    show inj_on (f ∘ k) (j ‘ (S − {c}))
     by (smt (verit, del_insts) SU homeomorphism_def inj_on_def injf jk Diff_iff
comp_apply imageE subset_iff)
    show (f ∘ k) ‘ j ‘ (S − {c}) ⊆ T
      by (metis fim image_comp image_mono kjsub subset_trans)
  qed (auto simp: assms)
  ultimately have openin (top_of_set T) ((f ∘ h) ‘ g ‘ (S − {b}) ∪ ((f ∘ k) ‘ j
‘ (S − {c})))
    by (rule openin_Un)
  moreover have (f ∘ h) ‘ g ‘ (S − {b}) = f ‘ (S − {b})
  proof −
    have h ‘ g ‘ (S − {b}) = (S − {b})
    proof
      show h ‘ g ‘ (S − {b}) ⊆ S − {b}
       using homeomorphism_apply1 [OF gh] SU by (fastforce simp add: image_iff
image_subset_iff)
      show S − {b} ⊆ h ‘ g ‘ (S − {b})
      by (metis Diff_mono SU gh homeomorphism_image2 homeomorphism_of_subsets
set_eq_subset)
    qed
    then show ?thesis
      by (metis image_comp)
  qed
  moreover have (f ∘ k) ‘ j ‘ (S − {c}) = f ‘ (S − {c})
  proof −
    have k ‘ j ‘ (S − {c}) = (S − {c})
      using homeomorphism_apply1 [OF jk] SU
       by (meson Diff_mono homeomorphism_def homeomorphism_of_subsets jk
subset_refl)
    then show ?thesis
      by (metis image_comp)
  qed
  moreover have f ‘ (S − {b}) ∪ f ‘ (S − {c}) = f ‘ (S)
    using ‹b ≠ c› by blast
  ultimately show ?thesis
    by simp
qed


lemma invariance_of_domain_sphere_affine_set:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes contf: continuous_on S f and injf: inj_on f S and fim: f ‘ S ⊆ T
      and r ≠ 0 affine T and affTU: aff_dim T < DIM('a)
```

```
    and ope: openin (top_of_set (sphere a r)) S
  shows openin (top_of_set T) (f ‘ S)
proof (cases sphere a r = {})
  case True
  then show ?thesis
    using ope openin_subset by force
next
  case False
  show ?thesis
   proof (rule invariance_of_domain_sphere_affine_set_gen [OF contf injf fim
bounded_cball convex_cball ‹affine T›])
    show aff_dim T < aff_dim (cball a r)
     by (metis False affTU aff_dim_cball assms(4) linorder_cases sphere_empty)
    show openin (top_of_set (rel_frontier (cball a r))) S
     by (simp add: ‹r ≠ 0› ope)
  qed
qed

lemma no_embedding_sphere_lowdim:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes contf: continuous_on (sphere a r) f and injf: inj_on f (sphere a r)
and r > 0
  shows DIM('a) ≤ DIM('b)
proof −
  have False if DIM('a) > DIM('b)
  proof −
    have compact (f ‘ sphere a r)
      using compact_continuous_image
      by (simp add: compact_continuous_image contf)
    then have ¬ open (f ‘ sphere a r)
      using compact_open
     by (metis assms(3) image_is_empty not_less_iff_gr_or_eq sphere_eq_empty)
    then show False
     using invariance_of_domain_sphere_affine_set [OF contf injf subset_UNIV]
‹r > 0›
     by (metis aff_dim_UNIV affine_UNIV less_irrefl of_nat_less_iff open_openin
openin_subtopology_self subtopology_UNIV that)
  qed
  then show ?thesis
    using not_less by blast
qed

lemma simply_connected_sphere:
  fixes a :: 'a::euclidean_space
  assumes 3 ≤ DIM('a)
    shows simply_connected(sphere a r)
proof (cases rule: linorder_cases [of r 0])
  case less
  then show ?thesis by simp
```

next
case *equal*
then show *?thesis* **by** (*auto simp*: *convex_imp_simply_connected*)
next
case *greater*
then show *?thesis*
using *simply_connected_sphere_gen* [*of cball a r*] *assms*
by (*simp add*: *aff_dim_cball*)
qed

lemma *simply_connected_sphere_eq*:
fixes *a* :: *'a*::*euclidean_space*
shows *simply_connected*(*sphere a r*) ⟷ *3* ≤ *DIM*(*'a*) ∨ *r* ≤ *0* (**is** *?lhs* = *?rhs*)
proof (*cases r* ≤ *0*)
case *True*
have *simply_connected* (*sphere a r*)
using *True less_eq_real_def* **by** (*auto intro*: *convex_imp_simply_connected*)
with *True* **show** *?thesis* **by** *auto*
next
case *False*
show *?thesis*
proof
assume *L*: *?lhs*
have *False* **if** *DIM*(*'a*) = *1* ∨ *DIM*(*'a*) = *2*
using *that*
proof
assume *DIM*(*'a*) = *1*
with *L* **show** *False*
using *connected_sphere_eq simply_connected_imp_connected*
by (*metis False Suc_1 not_less_eq_eq order_refl*)
next
assume *DIM*(*'a*) = *2*
then have *sphere a r homeomorphic sphere* (*0*::*complex*) *1*
by (*metis DIM_complex False homeomorphic_spheres_gen not_less zero_less_one*)
then have *simply_connected*(*sphere* (*0*::*complex*) *1*)
using *L homeomorphic_simply_connected_eq* **by** *blast*
then obtain *a*::*complex* **where** *homotopic_with_canon* (λ*h*. *True*) (*sphere 0 1*) (*sphere 0 1*) *id* (λ*x*. *a*)
by (*metis continuous_on_id' id_apply image_id subset_refl simply_connected_eq_contractible_circlemap*)
then show *False*
using *contractible_sphere contractible_def not_one_le_zero* **by** *blast*
qed
with *False* **show** *?rhs*
by (*metis DIM_ge_Suc0 One_nat_def Suc_1 not_less_eq_eq numeral_3_eq_3 order_antisym_conv*)
next
assume *?rhs*
with *False* **show** *?lhs* **by** (*simp add*: *simply_connected_sphere*)

qed
qed

lemma *simply_connected_punctured_universe_eq*:
fixes *a* :: *'a*::*euclidean_space*
shows *simply_connected*(− {*a*}) ⟷ *3* ≤ *DIM*(*'a*)
proof −
have [*simp*]: *a* ∈ *rel_interior* (*cball a 1*)
by (*simp add*: *rel_interior_nonempty_interior*)
have [*simp*]: *affine hull cball a 1* − {*a*} = −{*a*}
by (*metis Compl_eq_Diff_UNIV aff_dim_cball aff_dim_lt_full not_less_iff_gr_or_eq zero_less_one*)
have *sphere a 1 homotopy_eqv* − {*a*}
using *homotopy_eqv_rel_frontier_punctured_affine_hull* [*of cball a 1 a*] **by** *auto*
then have *simply_connected*(− {*a*}) ⟷ *simply_connected*(*sphere a 1*)
using *homotopy_eqv_simple_connectedness* **by** *blast*
also have ... ⟷ *3* ≤ *DIM*(*'a*)
by (*simp add*: *simply_connected_sphere_eq*)
finally show *?thesis* **.**
qed

lemma *not_simply_connected_circle*:
fixes *a* :: *complex*
shows *0* < *r* ⟹ ¬ *simply_connected*(*sphere a r*)
by (*simp add*: *simply_connected_sphere_eq*)

proposition *simply_connected_punctured_convex*:
fixes *a* :: *'a*::*euclidean_space*
assumes *convex S* **and** *3*: *3* ≤ *aff_dim S*
shows *simply_connected*(*S* − {*a*})
proof (*cases a* ∈ *rel_interior S*)
case *True*
then obtain *e* **where** *a* ∈ *S 0* < *e* **and** *e*: *cball a e* ∩ *affine hull S* ⊆ *S*
by (*auto simp*: *rel_interior_cball*)
have *con*: *convex* (*cball a e* ∩ *affine hull S*)
by (*simp add*: *convex_Int*)
have *bo*: *bounded* (*cball a e* ∩ *affine hull S*)
by (*simp add*: *bounded_Int*)
have *affine hull S* ∩ *interior* (*cball a e*) ≠ {}
using ‹*0* < *e*› ‹*a* ∈ *S*› *hull_subset* **by** *fastforce*
then have *3* ≤ *aff_dim* (*affine hull S* ∩ *cball a e*)
by (*simp add*: *3 aff_dim_convex_Int_nonempty_interior* [*OF convex_affine_hull*])
also have ... = *aff_dim* (*cball a e* ∩ *affine hull S*)
by (*simp add*: *Int_commute*)
finally have *3* ≤ *aff_dim* (*cball a e* ∩ *affine hull S*) **.**
moreover have *rel_frontier* (*cball a e* ∩ *affine hull S*) *homotopy_eqv S* − {*a*}

proof (*rule homotopy_eqv_rel_frontier_punctured_convex*)
show $a \in$ *rel_interior* (*cball a e* $\cap$ *affine hull S*)
by (*meson IntI Int_mono* ‹$a \in S$› ‹$0 < e$› *e* ‹*cball a e* $\cap$ *affine hull S* $\subseteq S$› *ball_subset_cball centre_in_cball dual_order.strict_implies_order hull_inc hull_mono mem_rel_interior_ball*)
have *closed* (*cball a e* $\cap$ *affine hull S*)
by *blast*
then show *rel_frontier* (*cball a e* $\cap$ *affine hull S*) $\subseteq S$
by (*metis Diff_subset closure_closed dual_order.trans e rel_frontier_def*)
show $S \subseteq$ *affine hull* (*cball a e* $\cap$ *affine hull S*)
by (*metis* (*no_types*, *lifting*) *IntI* ‹$a \in S$› ‹$0 < e$› *affine_hull_convex_Int_nonempty_interior centre_in_ball convex_affine_hull empty_iff hull_subset inf_commute interior_cball subsetCE subsetI*)
qed (*auto simp*: *assms con bo*)
ultimately show *?thesis*
using *homotopy_eqv_simple_connectedness simply_connected_sphere_gen* [*OF con bo*]
by *blast*
next
case *False*
then have *rel_interior S* $\subseteq S - \{a\}$
by (*simp add*: *False rel_interior_subset subset_Diff_insert*)
moreover have $S - \{a\} \subseteq$ *closure S*
by (*meson Diff_subset closure_subset subset_trans*)
ultimately show *?thesis*
by (*metis contractible_imp_simply_connected contractible_convex_tweak_boundary_points* [*OF* ‹*convex S*›])
qed

corollary *simply_connected_punctured_universe*:
fixes *a* :: *'a*::*euclidean_space*
assumes $3 \leq$ *DIM*(*'a*)
shows *simply_connected*($- \{a\}$)
proof −
have [*simp*]: *affine hull cball a 1* = *UNIV*
by (*simp add*: *aff_dim_cball affine_hull_UNIV*)
have $a \in$ *rel_interior* (*cball a 1*)
by (*simp add*: *rel_interior_interior*)
then
have *simply_connected* (*rel_frontier* (*cball a 1*)) = *simply_connected* (*affine hull cball a 1* $- \{a\}$)
using *homotopy_eqv_rel_frontier_punctured_affine_hull homotopy_eqv_simple_connectedness*
by *blast*
then show *?thesis*
using *simply_connected_sphere* [*of a 1*, *OF assms*] **by** (*auto simp*: *Compl_eq_Diff_UNIV*)
qed

6.36.10 The power, squaring and exponential functions as covering maps

proposition *covering_space_power_punctured_plane*:
assumes *0 < n*
shows *covering_space (− {0}) (λz::complex. z^n) (− {0})*
proof −
consider *n = 1 | 2 ≤ n* **using** *assms* **by** *linarith*
then obtain *e* **where** *0 < e*
and *e*: *⋀w z. cmod(w − z) < e ∗ cmod z ⟹ (w^n = z^n ⟷ w = z)*
proof *cases*
assume *n = 1* **then show** *?thesis*
by (*rule_tac e=1* **in** *that*) *auto*
next
assume *2 ≤ n*
have *eq_if_pow_eq*:
w = z **if** *lt*: *cmod (w − z) < 2 ∗ sin (pi / real n) ∗ cmod z*
and *eq*: *w^n = z^n* **for** *w z*
proof (*cases z = 0*)
case *True* **with** *eq assms* **show** *?thesis* **by** (*auto simp*: *power_0_left*)
next
case *False*
then have *z ≠ 0* **by** *auto*
have *(w/z)^n = 1*
by (*metis False divide_self_if eq power_divide power_one*)
then obtain *j*::*nat* **where** *j*: *w / z = exp (2 ∗ of_real pi ∗ i ∗ j / n)* **and** *j < n*
using *Suc_leI assms ‹2 ≤ n› complex_roots_unity* [*THEN eqset_imp_iff*, *of n w/z*]
by *force*
have *cmod (w/z − 1) < 2 ∗ sin (pi / real n)*
using *lt assms ‹z ≠ 0›* **by** (*simp add*: *field_split_simps norm_divide*)
then have *cmod (exp (i ∗ of_real (2 ∗ pi ∗ j / n)) − 1) < 2 ∗ sin (pi / real n)*
by (*simp add*: *j field_simps*)
then have *2 ∗ |sin((2 ∗ pi ∗ j / n) / 2)| < 2 ∗ sin (pi / real n)*
by (*simp only*: *dist_exp_i_1*)
then have *sin_less*: *sin((pi ∗ j / n)) < sin (pi / real n)*
by (*simp add*: *field_simps*)
then have *w / z = 1*
proof (*cases j = 0*)
case *True* **then show** *?thesis* **by** (*auto simp*: *j*)
next
case *False*
then have *sin (pi / real n) ≤ sin((pi ∗ j / n))*
proof (*cases j / n ≤ 1/2*)
case *True*
show *?thesis*
using *‹j ≠ 0 › ‹j < n› True*

```
      by (intro sin_monotone_2pi_le) (auto simp: field_simps intro: order_trans
[of _ 0])
      next
        case False
        then have seq: sin(pi * j / n) = sin(pi * (n - j) / n)
              using ‹j < n› by (simp add: algebra_simps diff_divide_distrib
of_nat_diff)
        show ?thesis
          unfolding seq
          using ‹j < n› False
       by (intro sin_monotone_2pi_le) (auto simp: field_simps intro: order_trans
[of _ 0])
      qed
      with sin_less show ?thesis by force
    qed
    then show ?thesis by simp
  qed
  show ?thesis
  proof
    show 0 < 2 * sin (pi / real n)
      by (force simp: ‹2 ≤ n› sin_pi_divide_n_gt_0)
  qed (meson eq_if_pow_eq)
 qed
 have zn1: continuous_on (- {0}) (λz::complex. z^n)
   by (rule continuous_intros)+
 have zn2: (λz::complex. z^n) ‘ (- {0}) = - {0}
    using assms by (auto simp: image_def elim: exists_complex_root_nonzero
[where n = n])
 have zn3: ∃ T. z^n ∈ T ∧ open T ∧ 0 ∉ T ∧
            (∃ v. ⋃ v = -{0} ∩ (λz. z ^ n) -‘ T ∧
                (∀ u∈v. open u ∧ 0 ∉ u) ∧
                pairwise disjnt v ∧
                (∀ u∈v. Ex (homeomorphism u T (λz. z^n))))
        if z ≠ 0 for z::complex
 proof -
   define d where d ≡ min (1/2) (e/4) * norm z
   have 0 < d
     by (simp add: d_def ‹0 < e› ‹z ≠ 0›)
   have iff_x_eq_y: x^n = y^n ⟷ x = y
       if eq: w^n = z^n and x: x ∈ ball w d and y: y ∈ ball w d for w x y
   proof -
     have [simp]: norm z = norm w using that
       by (simp add: assms power_eq_imp_eq_norm)
     show ?thesis
     proof (cases w = 0)
       case True with ‹z ≠ 0› assms eq
       show ?thesis by (auto simp: power_0_left)
     next
       case False
```

```
      have cmod (x − y) < 2∗d
        using x y
          by (simp add: dist_norm [symmetric]) (metis dist_commute mult_2
dist_triangle_less_add)
      also have ... ≤ 2 ∗ e / 4 ∗ norm w
        using ‹e > 0› by (simp add: d_def min_mult_distrib_right)
      also have ... = e ∗ (cmod w / 2)
        by simp
      also have ... ≤ e ∗ cmod y
      proof (rule mult_left_mono)
        have cmod (w − y) < cmod w / 2 ⟹ cmod w / 2 ≤ cmod y
          by (metis (no_types) dist_0_norm dist_norm norm_triangle_half_l
not_le order_less_irrefl)
        then show cmod w / 2 ≤ cmod y
          using y by (simp add: dist_norm d_def min_mult_distrib_right)
      qed (use ‹e > 0› in auto)
      finally have cmod (x − y) < e ∗ cmod y .
      then show ?thesis by (rule e)
    qed
  qed
  then have inj: inj_on (λw. w^n) (ball z d)
    by (simp add: inj_on_def)
  have cont: continuous_on (ball z d) (λw. w ^ n)
    by (intro continuous_intros)
  have noncon: ¬ (λw::complex. w^n) constant_on UNIV
   by (metis UNIV_I assms constant_on_def power_one zero_neq_one zero_power)
  have im_eq: (λw. w^n) ‘ ball z′ d = (λw. w^n) ‘ ball z d
            if z′: z′^n = z^n for z′
  proof −
    have nz′: norm z′ = norm z using that assms power_eq_imp_eq_norm by
blast
    have (w ∈ (λw. w^n) ‘ ball z′ d) = (w ∈ (λw. w^n) ‘ ball z d) for w
    proof (cases w=0)
      case True with assms show ?thesis
        by (simp add: image_def ball_def nz′)
    next
      case False
      have z′ ≠ 0 using ‹z ≠ 0› nz′ by force
      have 1: (z∗x / z′)^n = x^n if x ≠ 0 for x
        using z′ that by (simp add: field_simps ‹z ≠ 0›)
      have 2: cmod (z − z ∗ x / z′) = cmod (z′ − x) if x ≠ 0 for x
      proof −
        have cmod (z − z ∗ x / z′) = cmod z ∗ cmod (1 − x / z′)
       by (metis (no_types) ab_semigroup_mult_class.mult_ac(1) divide_complex_def
mult.right_neutral norm_mult right_diff_distrib′)
        also have ... = cmod z′ ∗ cmod (1 − x / z′)
          by (simp add: nz′)
        also have ... = cmod (z′ − x)
          by (simp add: ‹z′ ≠ 0› diff_divide_eq_iff norm_divide)
```

```
        finally show ?thesis .
      qed
      have 3: (z'*x / z) ^ n = x ^ n if x ≠ 0 for x
        using z' that by (simp add: field_simps ‹z ≠ 0›)
      have 4: cmod (z' - z' * x / z) = cmod (z - x) if x ≠ 0 for x
      proof -
        have cmod (z * (1 - x * inverse z)) = cmod (z - x)
          by (metis ‹z ≠ 0› diff_divide_distrib divide_complex_def divide_self_if
nonzero_eq_divide_eq semiring_normalization_rules(7))
        then show ?thesis
          by (metis (no_types) mult.assoc divide_complex_def mult.right_neutral
norm_mult nz' right_diff_distrib')
      qed
      show ?thesis
        by (simp add: set_eq_iff image_def ball_def) (metis 1 2 3 4 diff_zero
dist_norm nz')
    qed
    then show ?thesis by blast
  qed

  have ex_ball: ∃B. (∃z'. B = ball z' d ∧ z' ^ n = z ^ n) ∧ x ∈ B
            if x ≠ 0 and eq: x ^ n = w ^ n and dzw: dist z w < d for x w
  proof -
    have w ≠ 0 by (metis assms power_eq_0_iff that(1) that(2))
    have [simp]: cmod x = cmod w
      using assms power_eq_imp_eq_norm eq by blast
    have [simp]: cmod (x * z / w - x) = cmod (z - w)
    proof -
      have cmod (x * z / w - x) = cmod x * cmod (z / w - 1)
        by (metis (no_types) mult.right_neutral norm_mult right_diff_distrib'
times_divide_eq_right)
      also have ... = cmod w * cmod (z / w - 1)
        by simp
      also have ... = cmod (z - w)
        by (simp add: ‹w ≠ 0› divide_diff_eq_iff nonzero_norm_divide)
      finally show ?thesis .
    qed
    show ?thesis
    proof (intro exI conjI)
      show (z / w * x) ^ n = z ^ n
        by (metis ‹w ≠ 0› eq nonzero_eq_divide_eq power_mult_distrib)
      show x ∈ ball (z / w * x) d
        using ‹d > 0› that
        by (simp add: ball_eq_ball_iff ‹z ≠ 0› ‹w ≠ 0› field_simps) (simp add:
dist_norm)
    qed auto
  qed

  show ?thesis
```

```
    proof (rule exI, intro conjI)
      show z ^ n ∈ (λw. w ^ n) ‘ ball z d
        using ‹d > 0› by simp
      show open ((λw. w ^ n) ‘ ball z d)
        by (rule invariance_of_domain [OF cont open_ball inj])
      show 0 ∉ (λw. w ^ n) ‘ ball z d
        using ‹z ≠ 0› assms by (force simp: d_def)
      show ∃v. ⋃v = - {0} ∩ (λz. z ^ n) -‘ (λw. w ^ n) ‘ ball z d ∧
                (∀u∈v. open u ∧ 0 ∉ u) ∧
                disjoint v ∧
                (∀u∈v. Ex (homeomorphism u ((λw. w ^ n) ‘ ball z d) (λz. z ^ n)))
      proof (rule exI, intro ballI conjI)
        show ⋃{ball z' d |z'. z'^n = z^n} = - {0} ∩ (λz. z ^ n) -‘ (λw. w ^ n) ‘
ball z d (is ?l = ?r)
        proof
          have ⋀z'. cmod z' < d ⟹ z' ^ n ≠ z ^ n
            by (auto simp add: assms d_def power_eq_imp_eq_norm that)
          then show ?l ⊆ ?r
            by auto (metis im_eq image_eqI mem_ball)
          show ?r ⊆ ?l
            by auto (meson ex_ball)
        qed
        show ⋀u. u ∈ {ball z' d |z'. z' ^ n = z ^ n} ⟹ 0 ∉ u
          by (force simp add: assms d_def power_eq_imp_eq_norm that)

        show disjoint {ball z' d |z'. z' ^ n = z ^ n}
        proof (clarsimp simp add: pairwise_def disjnt_iff)
          fix ξ ζ x
          assume ξ^n = z^n ζ^n = z^n ball ξ d ≠ ball ζ d
            and dist ξ x < d dist ζ x < d
          then have dist ξ ζ < d+d
            using dist_triangle_less_add by blast
          then have cmod (ξ - ζ) < 2*d
            by (simp add: dist_norm)
          also have ... ≤ e * cmod z
            using mult_right_mono ‹0 < e› that by (auto simp: d_def)
          finally have cmod (ξ - ζ) < e * cmod z .
          with e have ξ = ζ
            by (metis ‹ξ^n = z^n› ‹ζ^n = z^n› assms power_eq_imp_eq_norm)
          then show False
            using ‹ball ξ d ≠ ball ζ d› by blast
        qed
        show Ex (homeomorphism u ((λw. w ^ n) ‘ ball z d) (λz. z ^ n))
          if u ∈ {ball z' d |z'. z' ^ n = z ^ n} for u
        proof (rule invariance_of_domain_homeomorphism [of u λz. z^n])
          show open u
            using that by auto
          show continuous_on u (λz. z ^ n)
            by (intro continuous_intros)
```

```
        show inj_on (λz. z ^ n) u
          using that by (auto simp: iff_x_eq_y inj_on_def)
          show ⋀g. homeomorphism u ((λz. z ^ n) ' u) (λz. z ^ n) g ⟹ Ex
(homeomorphism u ((λw. w ^ n) ' ball z d) (λz. z ^ n))
          using im_eq that by clarify metis
       qed auto
     qed auto
   qed
  qed
  show ?thesis
   using assms
   apply (simp add: covering_space_def zn1 zn2)
   apply (subst zn2 [symmetric])
   apply (simp add: openin_open_eq open_Compl zn3)
   done
qed

corollary covering_space_square_punctured_plane:
  covering_space (− {0}) (λz::complex. z^2) (− {0})
  by (simp add: covering_space_power_punctured_plane)


proposition covering_space_exp_punctured_plane:
  covering_space UNIV (λz::complex. exp z) (− {0})
proof (simp add: covering_space_def, intro conjI ballI)
  show continuous_on UNIV (λz::complex. exp z)
    by (rule continuous_on_exp [OF continuous_on_id])
  show range exp = − {0::complex}
    by auto (metis exp_Ln range_eqI)
  show ∃ T. z ∈ T ∧ openin (top_of_set (− {0})) T ∧
           (∃ v. ⋃ v = exp −' T ∧ (∀ u∈v. open u) ∧ disjoint v ∧
                (∀ u∈v. ∃ q. homeomorphism u T exp q))
       if z ∈ − {0::complex} for z
  proof −
    have z ≠ 0
      using that by auto
    have ball (Ln z) 1 ⊆ ball (Ln z) pi
      using pi_ge_two by (simp add: ball_subset_ball_iff)
    then have inj_exp: inj_on exp (ball (Ln z) 1)
      using inj_on_exp_pi inj_on_subset by blast
    define twopi where twopi ≡ λn. of_real (2 * of_int n * pi) * i
    define V where V ≡ range (λn. (λx. x + twopi n) ' (ball(Ln z) 1))
    have exp_eq': exp w = exp z ⟷ (∃ n::int. w = z + twopi n) for z w
      by (simp add: exp_eq twopi_def)
    show ?thesis
    proof (intro exI conjI)
      show z ∈ exp ' (ball(Ln z) 1)
        by (metis ‹z ≠ 0› centre_in_ball exp_Ln rev_image_eqI zero_less_one)
      have open (− {0::complex})
```

by *blast*
with *inj_exp* **show** *openin* (*top_of_set* (− {*0*})) (*exp* ‘ *ball* (*Ln z*) *1*)
by (*auto simp*: *openin_open_eq invariance_of_domain continuous_on_exp* [*OF continuous_on_id*])
show *UV*: ⋃ *V* = *exp* −‘ *exp* ‘ *ball* (*Ln z*) *1*
by (*force simp*: *V_def twopi_def Complex_Transcendental.exp_eq image_iff*)
show ∀ *V*∈*V*. *open V*
by (*auto simp*: *V_def inj_on_def continuous_intros invariance_of_domain*)
have *2* ≤ *cmod* (*twopi m* − *twopi n*) **if** *m* ≠ *n* **for** *m n*
proof −
have *1* ≤ *abs* (*m* − *n*)
using *that* **by** *linarith*
then have *1* ≤ *cmod* (*of_int m* − *of_int n*) ∗ *1*
by (*metis mult.right_neutral norm_of_int of_int_1_le_iff of_int_abs of_int_diff*)
also have ... ≤ *cmod* (*of_int m* − *of_int n*) ∗ *of_real pi*
using *pi_ge_two*
by (*intro mult_left_mono*) *auto*
also have ... ≤ *cmod* ((*of_int m* − *of_int n*) ∗ *of_real pi* ∗ i)
by (*simp add*: *norm_mult*)
also have ... ≤ *cmod* (*of_int m* ∗ *of_real pi* ∗ i − *of_int n* ∗ *of_real pi* ∗ i)
by (*simp add*: *algebra_simps*)
finally have *1* ≤ *cmod* (*of_int m* ∗ *of_real pi* ∗ i − *of_int n* ∗ *of_real pi* ∗ i) **.**
then have *2* ∗ *1* ≤ *cmod* (*2* ∗ (*of_int m* ∗ *of_real pi* ∗ i − *of_int n* ∗ *of_real pi* ∗ i))
by (*metis mult_le_cancel_left_pos norm_mult_numeral1 zero_less_numeral*)
then show *?thesis*
by (*simp add*: *algebra_simps twopi_def*)
qed
then show *disjoint V*
unfolding *V_def pairwise_def disjnt_iff*
by (*smt* (*verit*, *best*) *add.commute add_diff_cancel_left′ add_diff_eq dist_commute dist_complex_def dist_triangle imageE mem_ball*)
show ∀ *u*∈*V*. ∃ *q*. *homeomorphism u* (*exp* ‘ *ball* (*Ln z*) *1*) *exp q*
proof
fix *u*
assume *u* ∈ *V*
then obtain *n* **where** *n*: *u* = (λ*x*. *x* + *twopi n*) ‘ (*ball*(*Ln z*) *1*)
by (*auto simp*: *V_def*)
have *compact* (*cball* (*Ln z*) *1*)
by *simp*
moreover have *continuous_on* (*cball* (*Ln z*) *1*) *exp*
by (*rule continuous_on_exp* [*OF continuous_on_id*])
moreover have *inj_on exp* (*cball* (*Ln z*) *1*)
using *pi_ge_two inj_on_subset* [*OF inj_on_exp_pi* [*of Ln z*]]
by (*simp add*: *subset_iff*)
ultimately obtain γ **where** *hom*: *homeomorphism* (*cball* (*Ln z*) *1*) (*exp* ‘ *cball* (*Ln z*) *1*) *exp* γ

```
      using homeomorphism_compact by blast
    have eq1: exp ‘ u = exp ‘ ball (Ln z) 1
      by (smt (verit) n exp_eq′ image_cong image_image)
    have γexp: γ (exp x) + twopi n = x if x ∈ u for x
    proof −
      have exp x = exp (x − twopi n)
        using exp_eq′ by auto
      then have γ (exp x) = γ (exp (x − twopi n))
        by simp
      also have ... = x − twopi n
      using ‹x ∈ u› by (auto simp: n intro: homeomorphism_apply1 [OF hom])
      finally show ?thesis
        by simp
    qed
    have exp2n: exp (γ (exp x) + twopi n) = exp x if dist (Ln z) x < 1 for x
      by (metis γexp exp_eq′ imageI mem_ball n that)
    have continuous_on (exp ‘ ball (Ln z) 1) γ
        by (meson ball_subset_cball continuous_on_subset hom homeomor-
phism_cont2 image_mono)
    then have cont: continuous_on (exp ‘ ball (Ln z) 1) (λx. γ x + twopi n)
      by (intro continuous_intros)
    have homeomorphism u (exp ‘ ball (Ln z) 1) exp ((λx. x + twopi n) ∘ γ)
      unfolding homeomorphism_def
      apply (intro conjI ballI eq1 continuous_on_exp [OF continuous_on_id])
       apply (auto simp: γexp exp2n cont n)
      apply (force simp: image_iff homeomorphism_apply1 [OF hom])+
      done
    then show ∃ q. homeomorphism u (exp ‘ ball (Ln z) 1) exp q by metis
  qed
 qed
 qed
qed
```

6.36.11 Hence the Borsukian results about mappings into circles

```
lemma inessential_eq_continuous_logarithm:
  fixes f :: 'a::real_normed_vector ⇒ complex
  shows (∃ a. homotopic_with_canon (λh. True) S (−{0}) f (λt. a)) ⟷
        (∃ g. continuous_on S g ∧ (∀ x ∈ S. f x = exp(g x)))
  (is ?lhs ⟷ ?rhs)
proof
  assume ?lhs thus ?rhs
   by (metis covering_space_lift_inessential_function covering_space_exp_punctured_plane)
next
  assume ?rhs
  then obtain g where contg: continuous_on S g and f: ⋀x. x ∈ S ⟹ f x =
exp(g x)
    by metis
```

```
  obtain a where homotopic_with_canon (λh. True) S (- {of_real 0}) (exp ∘
g) (λx. a)
 proof (rule nullhomotopic_through_contractible [OF contg _ _ _ contractible_UNIV])
   show continuous_on (UNIV::complex set) exp
     by (intro continuous_intros)
   show exp ∈ UNIV → -{of_real 0}
     by auto
 qed force
 then have homotopic_with_canon (λh. True) S (- {0}) f (λt. a)
   using f homotopic_with_eq by fastforce
 then show ?lhs ..
qed

corollary inessential_imp_continuous_logarithm_circle:
 fixes f :: 'a::real_normed_vector ⇒ complex
 assumes homotopic_with_canon (λh. True) S (sphere 0 1) f (λt. a)
 obtains g where continuous_on S g and ⋀x. x ∈ S ⟹ f x = exp(g x)
proof -
 have homotopic_with_canon (λh. True) S (- {0}) f (λt. a)
   using assms homotopic_with_subset_right by fastforce
 then show ?thesis
   by (metis inessential_eq_continuous_logarithm that)
qed


lemma inessential_eq_continuous_logarithm_circle:
 fixes f :: 'a::real_normed_vector ⇒ complex
 shows (∃a. homotopic_with_canon (λh. True) S (sphere 0 1) f (λt. a)) ⟷
       (∃g. continuous_on S g ∧ (∀x ∈ S. f x = exp(i * of_real(g x))))
 (is ?lhs ⟷ ?rhs)
proof
 assume L: ?lhs
 then obtain g where contg: continuous_on S g and g: ⋀x. x ∈ S ⟹ f x =
exp(g x)
   using inessential_imp_continuous_logarithm_circle by blast
 have f ` S ⊆ sphere 0 1
   by (metis L homotopic_with_imp_subset1)
 then have ⋀x. x ∈ S ⟹ Re (g x) = 0
   using g by auto
 then show ?rhs
    by (rule_tac x=Im ∘ g in exI) (auto simp: Euler g intro: contg continu-
ous_intros)
next
 assume ?rhs
 then obtain g where contg: continuous_on S g and g: ⋀x. x ∈ S ⟹ f x =
exp(i* of_real(g x))
   by metis
 obtain a where homotopic_with_canon (λh. True) S (sphere 0 1) ((exp ∘ (λz.
i*z)) ∘ (of_real ∘ g)) (λx. a)
```

```
  proof (rule nullhomotopic_through_contractible)
    show continuous_on S (complex_of_real ∘ g)
      by (intro conjI contg continuous_intros)
    show (complex_of_real ∘ g) ∈ S → ℝ
      by auto
    show continuous_on ℝ (exp ∘ (∗)i)
      by (intro continuous_intros)
    show (exp ∘ (∗)i) ∈ ℝ → sphere 0 1
      by (auto simp: complex_is_Real_iff)
  qed (auto simp: convex_Reals convex_imp_contractible)
  moreover have ⋀x. x ∈ S ⟹ (exp ∘ (∗)i ∘ (complex_of_real ∘ g)) x = f x
    by (simp add: g)
  ultimately have homotopic_with_canon (λh. True) S (sphere 0 1) f (λt. a)
    using homotopic_with_eq by force
  then show ?lhs ..
qed

proposition homotopic_with_sphere_times:
  fixes f :: 'a::real_normed_vector ⇒ complex
  assumes hom: homotopic_with_canon (λx. True) S (sphere 0 1) f g and conth:
continuous_on S h
      and hin: ⋀x. x ∈ S ⟹ h x ∈ sphere 0 1
    shows homotopic_with_canon (λx. True) S (sphere 0 1) (λx. f x ∗ h x) (λx.
g x ∗ h x)
proof −
  obtain k where contk: continuous_on ({0..1::real} × S) k
            and kim: k ‘ ({0..1} × S) ⊆ sphere 0 1
            and k0: ⋀x. k(0, x) = f x
            and k1: ⋀x. k(1, x) = g x
    using hom by (auto simp: homotopic_with_def)
  show ?thesis
    apply (simp add: homotopic_with)
    apply (rule_tac x=λz. k z ∗ (h ∘ snd)z in exI)
    using kim hin by (fastforce simp: conth norm_mult k0 k1 intro!: contk contin-
uous_intros)+
qed

proposition homotopic_circlemaps_divide:
  fixes f :: 'a::real_normed_vector ⇒ complex
    shows homotopic_with_canon (λx. True) S (sphere 0 1) f g ⟷
          continuous_on S f ∧ f ‘ S ⊆ sphere 0 1 ∧
          continuous_on S g ∧ g ‘ S ⊆ sphere 0 1 ∧
           (∃c. homotopic_with_canon (λx. True) S (sphere 0 1) (λx. f x / g x)
(λx. c))
proof −
  have homotopic_with_canon (λx. True) S (sphere 0 1) (λx. f x / g x) (λx. 1)
       if homotopic_with_canon (λx. True) S (sphere 0 1) (λx. f x / g x) (λx. c)
for c
  proof −
```

```
    have S = {} ∨ path_component (sphere 0 1) 1 c
      using homotopic_with_imp_subset2 [OF that] path_connected_sphere [of
0::complex 1]
      by (auto simp: path_connected_component)
    with subtopology_empty_iff_trivial
    have homotopic_with_canon (λx. True) S (sphere 0 1) (λx. 1) (λx. c)
      by (force simp add: homotopic_constant_maps)
    then show ?thesis
      using homotopic_with_symD homotopic_with_trans that by blast
  qed
  then have ∗: (∃ c. homotopic_with_canon (λx. True) S (sphere 0 1) (λx. f x /
g x) (λx. c)) ⟷
              homotopic_with_canon (λx. True) S (sphere 0 1) (λx. f x / g x) (λx.
1)
    by auto
  have homotopic_with_canon (λx. True) S (sphere 0 1) f g ⟷
          continuous_on S f ∧ f ‘ S ⊆ sphere 0 1 ∧
          continuous_on S g ∧ g ‘ S ⊆ sphere 0 1 ∧
          homotopic_with_canon (λx. True) S (sphere 0 1) (λx. f x / g x) (λx. 1)
       (is ?lhs ⟷ ?rhs)
  proof
    assume L: ?lhs
    have geq1 [simp]: ⋀x. x ∈ S ⟹ cmod (g x) = 1
      using homotopic_with_imp_subset2 [OF L]
      by (simp add: image_subset_iff)
    have cont: continuous_on S (inverse ∘ g)
    proof (rule continuous_intros)
      show continuous_on S g
        using homotopic_with_imp_continuous [OF L] by blast
      show continuous_on (g ‘ S) inverse
       by (rule continuous_on_subset [of sphere 0 1, OF continuous_on_inverse])
auto
    qed
    have [simp]: ⋀x. x ∈ S ⟹ g x ≠ 0
      using geq1 by fastforce
    have homotopic_with_canon (λx. True) S (sphere 0 1) (λx. f x / g x) (λx. 1)
      using homotopic_with_eq [OF homotopic_with_sphere_times [OF L cont]]
      by (auto simp: divide_inverse norm_inverse)
    with L show ?rhs
     by (simp add: homotopic_with_imp_continuous homotopic_with_imp_subset1
homotopic_with_imp_subset2)
  next
    assume ?rhs then show ?lhs
       by (elim conjE homotopic_with_eq [OF homotopic_with_sphere_times];
force)
  qed
  then show ?thesis
    by (simp add: ∗)
qed
```

6.36.12 Upper and lower hemicontinuous functions

And relation in the case of preimage map to open and closed maps, and fact that upper and lower hemicontinuity together imply continuity in the sense of the Hausdorff metric (at points where the function gives a bounded and nonempty set).

Many similar proofs below.

lemma *upper_hemicontinuous*:
  assumes $\bigwedge x.\ x \in S \Longrightarrow f\ x \subseteq T$
    shows $((\forall\, U.\ \mathit{openin}\ (\mathit{top\_of\_set}\ T)\ U$
                $\longrightarrow \mathit{openin}\ (\mathit{top\_of\_set}\ S)\ \{x \in S.\ f\ x \subseteq U\}) \longleftrightarrow$
          $(\forall\, U.\ \mathit{closedin}\ (\mathit{top\_of\_set}\ T)\ U$
                $\longrightarrow \mathit{closedin}\ (\mathit{top\_of\_set}\ S)\ \{x \in S.\ f\ x \cap U \neq \{\}\}))$
        (**is** *?lhs* = *?rhs*)
proof (*intro iffI allI impI*)
  fix *U*
  assume ∗ [*rule_format*]: *?lhs* **and** *closedin* (*top_of_set T*) *U*
  then have *openin* (*top_of_set T*) (*T* − *U*)
    by (*simp add*: *openin_diff*)
  then have *openin* (*top_of_set S*) {*x* ∈ *S*. *f x* ⊆ *T* − *U*}
    using ∗ [*of T−U*] **by** *blast*
  moreover have *S* − {*x* ∈ *S*. *f x* ⊆ *T* − *U*} = {*x* ∈ *S*. *f x* ∩ *U* ≠ {}}
    using *assms* **by** *blast*
  ultimately show *closedin* (*top_of_set S*) {*x* ∈ *S*. *f x* ∩ *U* ≠ {}}
    by (*simp add*: *openin_closedin_eq*)
next
  fix *U*
  assume ∗ [*rule_format*]: *?rhs* **and** *openin* (*top_of_set T*) *U*
  then have *closedin* (*top_of_set T*) (*T* − *U*)
    by (*simp add*: *closedin_diff*)
  then have *closedin* (*top_of_set S*) {*x* ∈ *S*. *f x* ∩ (*T* − *U*) ≠ {}}
    using ∗ [*of T−U*] **by** *blast*
  moreover have {*x* ∈ *S*. *f x* ∩ (*T* − *U*) ≠ {}} = *S* − {*x* ∈ *S*. *f x* ⊆ *U*}
    using *assms* **by** *auto*
  ultimately show *openin* (*top_of_set S*) {*x* ∈ *S*. *f x* ⊆ *U*}
    by (*simp add*: *openin_closedin_eq*)
qed

lemma *lower_hemicontinuous*:
  assumes $\bigwedge x.\ x \in S \Longrightarrow f\ x \subseteq T$
    shows $((\forall\, U.\ \mathit{closedin}\ (\mathit{top\_of\_set}\ T)\ U$
                $\longrightarrow \mathit{closedin}\ (\mathit{top\_of\_set}\ S)\ \{x \in S.\ f\ x \subseteq U\}) \longleftrightarrow$
          $(\forall\, U.\ \mathit{openin}\ (\mathit{top\_of\_set}\ T)\ U$
                $\longrightarrow \mathit{openin}\ (\mathit{top\_of\_set}\ S)\ \{x \in S.\ f\ x \cap U \neq \{\}\}))$
        (**is** *?lhs* = *?rhs*)
proof (*intro iffI allI impI*)
  fix *U*
  assume ∗ [*rule_format*]: *?lhs* **and** *openin* (*top_of_set T*) *U*

```
  then have closedin (top_of_set T) (T − U)
    by (simp add: closedin_diff)
  then have closedin (top_of_set S) {x ∈ S. f x ⊆ T−U}
    using ∗ [of T−U] by blast
  moreover have {x ∈ S. f x ⊆ T−U} = S − {x ∈ S. f x ∩ U ≠ {}}
    using assms by auto
  ultimately show openin (top_of_set S) {x ∈ S. f x ∩ U ≠ {}}
    by (simp add: openin_closedin_eq)
next
  fix U
  assume ∗ [rule_format]: ?rhs and closedin (top_of_set T) U
  then have openin (top_of_set T) (T − U)
    by (simp add: openin_diff)
  then have openin (top_of_set S) {x ∈ S. f x ∩ (T − U) ≠ {}}
    using ∗ [of T−U] by blast
  moreover have S − {x ∈ S. f x ∩ (T − U) ≠ {}} = {x ∈ S. f x ⊆ U}
    using assms by blast
  ultimately show closedin (top_of_set S) {x ∈ S. f x ⊆ U}
    by (simp add: openin_closedin_eq)
qed

lemma open_map_iff_lower_hemicontinuous_preimage:
  assumes f ‘ S ⊆ T
    shows ((∀ U. openin (top_of_set S) U
                ⟶ openin (top_of_set T) (f ‘ U)) ⟷
           (∀ U. closedin (top_of_set S) U
                ⟶ closedin (top_of_set T) {y ∈ T. {x. x ∈ S ∧ f x = y} ⊆ U}))
        (is ?lhs = ?rhs)
proof (intro iffI allI impI)
  fix U
  assume ∗ [rule_format]: ?lhs and closedin (top_of_set S) U
  then have openin (top_of_set S) (S − U)
    by (simp add: openin_diff)
  then have openin (top_of_set T) (f ‘ (S − U))
    using ∗ [of S−U] by blast
  moreover have T − (f ‘ (S − U)) = {y ∈ T. {x ∈ S. f x = y} ⊆ U}
    using assms by blast
  ultimately show closedin (top_of_set T) {y ∈ T. {x ∈ S. f x = y} ⊆ U}
    by (simp add: openin_closedin_eq)
next
  fix U
  assume ∗ [rule_format]: ?rhs and opeSU: openin (top_of_set S) U
  then have closedin (top_of_set S) (S − U)
    by (simp add: closedin_diff)
  then have closedin (top_of_set T) {y ∈ T. {x ∈ S. f x = y} ⊆ S − U}
    using ∗ [of S−U] by blast
  moreover have {y ∈ T. {x ∈ S. f x = y} ⊆ S − U} = T − (f ‘ U)
    using assms openin_imp_subset [OF opeSU] by auto
  ultimately show openin (top_of_set T) (f ‘ U)
```

using *assms openin_imp_subset* [*OF opeSU*] **by** (*force simp*: *openin_closedin_eq*)
qed

lemma *closed_map_iff_upper_hemicontinuous_preimage*:
assumes f ‘ $S \subseteq T$
shows $((\forall U.\ closedin\ (top\_of\_set\ S)\ U \longrightarrow closedin\ (top\_of\_set\ T)\ (f\ `\ U)) \longleftrightarrow (\forall U.\ openin\ (top\_of\_set\ S)\ U \longrightarrow openin\ (top\_of\_set\ T)\ \{y \in T.\ \{x.\ x \in S \wedge f\ x = y\} \subseteq U\}))$
(**is** *?lhs* = *?rhs*)
proof (*intro iffI allI impI*)
fix U
assume $*$ [*rule_format*]: *?lhs* **and** *opeSU*: *openin* (*top_of_set S*) U
then have *closedin* (*top_of_set S*) $(S - U)$
by (*simp add*: *closedin_diff*)
then have *closedin* (*top_of_set T*) $(f\ `\ (S - U))$
using $*$ [*of* $S - U$] **by** *blast*
moreover have $f\ `\ (S - U) = T - \{y \in T.\ \{x.\ x \in S \wedge f\ x = y\} \subseteq U\}$
using *assms openin_imp_subset* [*OF opeSU*] **by** *auto*
ultimately show *openin* (*top_of_set T*) $\{y \in T.\ \{x.\ x \in S \wedge f\ x = y\} \subseteq U\}$
using *assms openin_imp_subset* [*OF opeSU*] **by** (*force simp*: *openin_closedin_eq*)
next
fix U
assume $*$ [*rule_format*]: *?rhs* **and** *cloSU*: *closedin* (*top_of_set S*) U
then have *openin* (*top_of_set S*) $(S - U)$
by (*simp add*: *openin_diff*)
then have *openin* (*top_of_set T*) $\{y \in T.\ \{x \in S.\ f\ x = y\} \subseteq S - U\}$
using $*$ [*of* $S - U$] **by** *blast*
moreover have $(f\ `\ U) = T - \{y \in T.\ \{x \in S.\ f\ x = y\} \subseteq S - U\}$
using *assms closedin_imp_subset* [*OF cloSU*] **by** *auto*
ultimately show *closedin* (*top_of_set T*) $(f\ `\ U)$
by (*simp add*: *openin_closedin_eq*)
qed

proposition *upper_lower_hemicontinuous_explicit*:
fixes T :: ($'b$::{*real_normed_vector*,*heine_borel*}) *set*
assumes *fST*: $\bigwedge x.\ x \in S \Longrightarrow f\ x \subseteq T$
and *ope*: $\bigwedge U.\ openin\ (top\_of\_set\ T)\ U \Longrightarrow openin\ (top\_of\_set\ S)\ \{x \in S.\ f\ x \subseteq U\}$
and *clo*: $\bigwedge U.\ closedin\ (top\_of\_set\ T)\ U \Longrightarrow closedin\ (top\_of\_set\ S)\ \{x \in S.\ f\ x \subseteq U\}$
and $x \in S$ $0 < e$ **and** *bofx*: *bounded*$(f\ x)$ **and** *fx_ne*: $f\ x \neq \{\}$
obtains d **where** $0 < d$
$\bigwedge x'.\ [\![x' \in S;\ dist\ x\ x' < d]\!] \Longrightarrow (\forall y \in f\ x.\ \exists y'.\ y' \in f\ x' \wedge dist\ y\ y' < e) \wedge (\forall y' \in f\ x'.\ \exists y.\ y \in f\ x \wedge dist\ y'\ y < e)$
proof −
have *openin* (*top_of_set T*) $(T \cap (\bigcup a \in f\ x.\ \bigcup b \in ball\ 0\ e.\ \{a + b\}))$
by (*auto simp*: *open_sums openin_open_Int*)

with *ope* **have** *openin* (*top_of_set S*)
{*u* ∈ *S*. *f u* ⊆ *T* ∩ (⋃ *a*∈*f x*. ⋃ *b*∈*ball 0 e*. {*a* + *b*})} **by** *blast*
with ‹*0* < *e*› ‹*x* ∈ *S*› **obtain** *d1* **where** *d1* > *0* **and**
d1: ⋀*x'*. ⟦*x'* ∈ *S*; *dist x' x* < *d1*⟧ ⟹ *f x'* ⊆ *T* ∧ *f x'* ⊆ (⋃ *a* ∈ *f x*. ⋃ *b* ∈ *ball 0 e*. {*a* + *b*})
by (*force simp*: *openin_euclidean_subtopology_iff dest*: *fST*)
have *oo*: ⋀*U*. *openin* (*top_of_set T*) *U* ⟹
openin (*top_of_set S*) {*x* ∈ *S*. *f x* ∩ *U* ≠ {}}
using *lower_hemicontinuous fST clo* **by** *blast*
have *compact* (*closure*(*f x*))
by (*simp add*: *bofx*)
moreover have *closure*(*f x*) ⊆ (⋃ *a* ∈ *f x*. *ball a* (*e*/*2*))
using ‹*0* < *e*› **by** (*force simp*: *closure_approachable simp del*: *divide_const_simps*)
ultimately obtain *C* **where** *C* ⊆ *f x finite C closure*(*f x*) ⊆ (⋃ *a* ∈ *C*. *ball a* (*e*/*2*))
by (*meson compactE finite_subset_image Elementary_Metric_Spaces.open_ball compactE_image*)
then have *fx_cover*: *f x* ⊆ (⋃ *a* ∈ *C*. *ball a* (*e*/*2*))
by (*meson closure_subset order_trans*)
with *fx_ne* **have** *C* ≠ {}
by *blast*
have *xin*: *x* ∈ (⋂ *a* ∈ *C*. {*x* ∈ *S*. *f x* ∩ *T* ∩ *ball a* (*e*/*2*) ≠ {}})
using ‹*x* ∈ *S*› ‹*0* < *e*› *fST* ‹*C* ⊆ *f x*› **by** *force*
have *openin* (*top_of_set S*) {*x* ∈ *S*. *f x* ∩ (*T* ∩ *ball a* (*e*/*2*)) ≠ {}} **for** *a*
by (*simp add*: *openin_open_Int oo*)
then have *openin* (*top_of_set S*) (⋂ *a* ∈ *C*. {*x* ∈ *S*. *f x* ∩ *T* ∩ *ball a* (*e*/*2*) ≠ {}})
by (*simp add*: *Int_assoc openin_INT2* [*OF* ‹*finite C*› ‹*C* ≠ {}›])
with *xin* **obtain** *d2* **where** *d2*>*0*
and *d2*: ⋀*u v*. ⟦*u* ∈ *S*; *dist u x* < *d2*; *v* ∈ *C*⟧ ⟹ *f u* ∩ *T* ∩ *ball v* (*e*/*2*) ≠ {}
unfolding *openin_euclidean_subtopology_iff* **using** *xin* **by** *fastforce*
show *?thesis*
proof (*intro that conjI ballI*)
show *0* < *min d1 d2*
using ‹*0* < *d1*› ‹*0* < *d2*› **by** *linarith*
next
fix *x' y*
assume *x'* ∈ *S dist x x'* < *min d1 d2 y* ∈ *f x*
then have *dd2*: *dist x' x* < *d2*
by (*auto simp*: *dist_commute*)
obtain *a* **where** *a* ∈ *C y* ∈ *ball a* (*e*/*2*)
using *fx_cover* ‹*y* ∈ *f x*› **by** *auto*
then show ∃ *y'*. *y'* ∈ *f x'* ∧ *dist y y'* < *e*
using *d2* [*OF* ‹*x'* ∈ *S*› *dd2*] *dist_triangle_half_r* **by** *fastforce*
next
fix *x' y'*
assume *x'* ∈ *S dist x x'* < *min d1 d2 y'* ∈ *f x'*
then have *dist x' x* < *d1*

by (*auto simp*: *dist_commute*)
then have $y' \in (\bigcup a{\in}f\ x.\ \bigcup b{\in}ball\ 0\ e.\ \{a + b\})$
using *d1* [*OF* ‹$x' \in S$›] ‹$y' \in f\ x'$› **by** *force*
then show $\exists y.\ y \in f\ x \wedge dist\ y'\ y < e$
by *clarsimp* (*metis add_diff_cancel_left' dist_norm*)
qed
qed

6.36.13 Complex logs exist on various "well-behaved" sets

lemma *continuous_logarithm_on_contractible*:
fixes $f :: 'a{::}real\_normed\_vector \Rightarrow complex$
assumes *continuous_on S f contractible S* $\bigwedge z.\ z \in S \Longrightarrow f\ z \neq 0$
obtains g **where** *continuous_on S g* $\bigwedge x.\ x \in S \Longrightarrow f\ x = exp(g\ x)$
proof −
obtain c **where** *hom*: *homotopic_with_canon* $(\lambda h.\ True)\ S\ (-\{0\})\ f\ (\lambda x.\ c)$
using *nullhomotopic_from_contractible assms*
by (*metis imageE subset_Compl_singleton image_subset_iff_funcset*)
then show *?thesis*
by (*metis inessential_eq_continuous_logarithm that*)
qed

lemma *continuous_logarithm_on_simply_connected*:
fixes $f :: 'a{::}real\_normed\_vector \Rightarrow complex$
assumes *contf*: *continuous_on S f* **and** *S*: *simply_connected S locally path_connected S*
and *f*: $\bigwedge z.\ z \in S \Longrightarrow f\ z \neq 0$
obtains g **where** *continuous_on S g* $\bigwedge x.\ x \in S \Longrightarrow f\ x = exp(g\ x)$
using *covering_space_lift* [*OF covering_space_exp_punctured_plane S contf*]
by (*metis* (*full_types*) *f imageE subset_Compl_singleton image_subset_iff_funcset*)

lemma *continuous_logarithm_on_cball*:
fixes $f :: 'a{::}real\_normed\_vector \Rightarrow complex$
assumes *continuous_on* (*cball a r*) *f* **and** $\bigwedge z.\ z \in cball\ a\ r \Longrightarrow f\ z \neq 0$
obtains h **where** *continuous_on* (*cball a r*) *h* $\bigwedge z.\ z \in cball\ a\ r \Longrightarrow f\ z = exp(h\ z)$
using *assms continuous_logarithm_on_contractible convex_imp_contractible* **by** *blast*

lemma *continuous_logarithm_on_ball*:
fixes $f :: 'a{::}real\_normed\_vector \Rightarrow complex$
assumes *continuous_on* (*ball a r*) *f* **and** $\bigwedge z.\ z \in ball\ a\ r \Longrightarrow f\ z \neq 0$
obtains h **where** *continuous_on* (*ball a r*) *h* $\bigwedge z.\ z \in ball\ a\ r \Longrightarrow f\ z = exp(h\ z)$
using *assms continuous_logarithm_on_contractible convex_imp_contractible* **by** *blast*

lemma *continuous_sqrt_on_contractible*:
fixes $f :: 'a{::}real\_normed\_vector \Rightarrow complex$

assumes *continuous_on S f contractible S*
and $\bigwedge z.\ z \in S \Longrightarrow f\ z \neq 0$
obtains *g* **where** *continuous_on S g* $\bigwedge x.\ x \in S \Longrightarrow f\ x = (g\ x)\ \hat{}\ 2$
proof −
obtain *g* **where** *contg*: *continuous_on S g* **and** *feq*: $\bigwedge x.\ x \in S \Longrightarrow f\ x = exp(g\ x)$
using *continuous_logarithm_on_contractible* [*OF assms*] **by** *blast*
show *?thesis*
proof
show *continuous_on S* ($\lambda z.\ exp\ (g\ z\ /\ 2)$)
by (*rule continuous_on_compose2* [*of UNIV exp*]; *intro continuous_intros contg subset_UNIV*) *auto*
show $\bigwedge x.\ x \in S \Longrightarrow f\ x = (exp\ (g\ x\ /\ 2))^2$
by (*metis exp_double feq nonzero_mult_div_cancel_left times_divide_eq_right zero_neq_numeral*)
qed
qed

lemma *continuous_sqrt_on_simply_connected*:
fixes *f* :: *'a*::*real_normed_vector* ⇒ *complex*
assumes *contf*: *continuous_on S f* **and** *S*: *simply_connected S locally path_connected S*
and *f*: $\bigwedge z.\ z \in S \Longrightarrow f\ z \neq 0$
obtains *g* **where** *continuous_on S g* $\bigwedge x.\ x \in S \Longrightarrow f\ x = (g\ x)\ \hat{}\ 2$
proof −
obtain *g* **where** *contg*: *continuous_on S g* **and** *feq*: $\bigwedge x.\ x \in S \Longrightarrow f\ x = exp(g\ x)$
using *continuous_logarithm_on_simply_connected* [*OF assms*] **by** *blast*
show *?thesis*
proof
show *continuous_on S* ($\lambda z.\ exp\ (g\ z\ /\ 2)$)
by (*rule continuous_on_compose2* [*of UNIV exp*]; *intro continuous_intros contg subset_UNIV*) *auto*
show $\bigwedge x.\ x \in S \Longrightarrow f\ x = (exp\ (g\ x\ /\ 2))^2$
by (*metis exp_double feq nonzero_mult_div_cancel_left times_divide_eq_right zero_neq_numeral*)
qed
qed

6.36.14 Another simple case where sphere maps are nullhomotopic

lemma *inessential_spheremap_2_aux*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *complex*
assumes *2*: *2* < *DIM*(*'a*) **and** *contf*: *continuous_on* (*sphere a r*) *f*
and *fim*: *f* ∈ (*sphere a r*) → (*sphere 0 1*)
obtains *c* **where** *homotopic_with_canon* ($\lambda z.\ True$) (*sphere a r*) (*sphere 0 1*) *f* ($\lambda x.\ c$)
proof −

```
  obtain g where contg: continuous_on (sphere a r) g
            and feq: ⋀x. x ∈ sphere a r ⟹ f x = exp(g x)
  proof (rule continuous_logarithm_on_simply_connected [OF contf])
    show simply_connected (sphere a r)
      using 2 by (simp add: simply_connected_sphere_eq)
    show locally path_connected (sphere a r)
      by (simp add: locally_path_connected_sphere)
    show ⋀z.  z ∈ sphere a r ⟹ f z ≠ 0
      using fim by force
  qed auto
  have ∃g. continuous_on (sphere a r) g ∧ (∀x∈sphere a r. f x = exp (i * com-
plex_of_real (g x)))
  proof (intro exI conjI)
    show continuous_on (sphere a r) (Im ∘ g)
      by (intro contg continuous_intros continuous_on_compose)
    show ∀x∈sphere a r. f x = exp (i * complex_of_real ((Im ∘ g) x))
      using exp_eq_polar feq fim norm_exp_eq_Re
      by (auto simp flip: image_subset_iff_funcset)
  qed
  with inessential_eq_continuous_logarithm_circle that show ?thesis
    by metis
qed

lemma inessential_spheremap_2:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes a2: 2 < DIM('a) and b2: DIM('b) = 2
      and contf: continuous_on (sphere a r) f and fim: f ∈ (sphere a r) → (sphere
b s)
  obtains c where homotopic_with_canon (λz. True) (sphere a r) (sphere b s) f
(λx. c)
proof (cases s ≤ 0)
  case True
  then show ?thesis
    using contf contractible_sphere fim nullhomotopic_into_contractible that by
blast
next
  case False
  then have sphere b s homeomorphic sphere (0::complex) 1
    using assms by (simp add: homeomorphic_spheres_gen)
  then obtain h k where hk: homeomorphism (sphere b s) (sphere (0::complex)
1) h k
    by (auto simp: homeomorphic_def)
  then have conth: continuous_on (sphere b s) h
       and  contk: continuous_on (sphere 0 1) k
       and  him: h ∈ sphere b s → sphere 0 1
       and  kim: k ∈ sphere 0 1 → sphere b s
    by (force simp: homeomorphism_def)+
  obtain c where homotopic_with_canon (λz. True) (sphere a r) (sphere 0 1) (h
∘ f) (λx. c)
```

proof (*rule inessential_spheremap_2_aux* [*OF a2*])
show *continuous_on* (*sphere a r*) (*h* ∘ *f*)
by (*meson contf conth continuous_on_compose continuous_on_subset fim image_subset_iff_funcset*)
show (*h* ∘ *f*) ∈ *sphere a r* → *sphere 0 1*
using *fim him* **by** *force*
qed *auto*
then have *homotopic_with_canon* (λ*f*. *True*) (*sphere a r*) (*sphere b s*) (*k* ∘ (*h* ∘ *f*)) (*k* ∘ (λ*x*. *c*))
by (*rule homotopic_with_compose_continuous_left* [*OF _ contk kim*])
moreover have ⋀*x*. *r* = *dist a x* ⟹ *f x* = *k* (*h* (*f x*))
by (*metis fim hk homeomorphism_def image_subset_iff mem_sphere image_subset_iff_funcset*)
ultimately have *homotopic_with_canon* (λ*z*. *True*) (*sphere a r*) (*sphere b s*) *f* (λ*x*. *k c*)
by (*auto intro*: *homotopic_with_eq*)
then show *?thesis*
by (*metis that*)
qed

6.36.15 Holomorphic logarithms and square roots

lemma *g_imp_holomorphic_log*:
assumes *holf*: *f holomorphic_on S*
and *contg*: *continuous_on S g* **and** *feq*: ⋀*x*. *x* ∈ *S* ⟹ *f x* = *exp* (*g x*)
and *fnz*: ⋀*z*. *z* ∈ *S* ⟹ *f z* ≠ *0*
obtains *g* **where** *g holomorphic_on S* ⋀*z*. *z* ∈ *S* ⟹ *f z* = *exp*(*g z*)
proof −
have *contf*: *continuous_on S f*
by (*simp add*: *holf holomorphic_on_imp_continuous_on*)
have *g field_differentiable at z within S* **if** *f field_differentiable at z within S z* ∈ *S* **for** *z*
proof −
obtain *f′* **where** *f′*: ((λ*y*. (*f y* − *f z*) / (*y* − *z*)) ⟶ *f′*) (*at z within S*)
using ‹*f field_differentiable at z within S*› **by** (*auto simp*: *field_differentiable_def has_field_derivative_iff*)
then have *ee*: ((λ*x*. (*exp*(*g x*) − *exp*(*g z*)) / (*x* − *z*)) ⟶ *f′*) (*at z within S*)
by (*simp add*: *feq* ‹*z* ∈ *S*› *Lim_transform_within* [*OF _ zero_less_one*])
have (((λ*y*. *if y* = *g z then exp* (*g z*) *else* (*exp y* − *exp* (*g z*)) / (*y* − *g z*)) ∘ *g*) ⟶ *exp* (*g z*))
(*at z within S*)
proof (*rule tendsto_compose_at*)
show (*g* ⟶ *g z*) (*at z within S*)
using *contg continuous_on* ‹*z* ∈ *S*› **by** *blast*
show (λ*y*. *if y* = *g z then exp* (*g z*) *else* (*exp y* − *exp* (*g z*)) / (*y* − *g z*)) −*g z*→ *exp* (*g z*)
by (*simp add*: *LIM_offset_zero_iff DERIV_D cong*: *if_cong Lim_cong_within*)
qed *auto*
then have *dd*: ((λ*x*. *if g x* = *g z then exp*(*g z*) *else* (*exp*(*g x*) − *exp*(*g z*)) / (*g*

```
x − g z)) ⟶ exp(g z)) (at z within S)
      by (simp add: o_def)
    have continuous (at z within S) g
      using contg continuous_on_eq_continuous_within ‹z ∈ S› by blast
    then have (∀F x in at z within S. dist (g x) (g z) < 2∗pi)
      by (simp add: continuous_within tendsto_iff)
    then have ∀F x in at z within S. exp (g x) = exp (g z) ⟶ g x ≠ g z ⟶ x
= z
      by (rule eventually_mono) (auto simp: exp_eq dist_norm norm_mult)
    then have ((λy. (g y − g z) / (y − z)) ⟶ f' / exp (g z)) (at z within S)
      by (auto intro!: Lim_transform_eventually [OF tendsto_divide [OF ee dd]])
    then show ?thesis
      by (auto simp: field_differentiable_def has_field_derivative_iff)
  qed
  then have g holomorphic_on S
    using holf holomorphic_on_def by auto
  then show ?thesis
    using feq that by auto
qed

lemma contractible_imp_holomorphic_log:
  assumes holf: f holomorphic_on S
      and S: contractible S
      and fnz: ⋀z. z ∈ S ⟹ f z ≠ 0
  obtains g where g holomorphic_on S ⋀z. z ∈ S ⟹ f z = exp(g z)
proof −
  have contf: continuous_on S f
    by (simp add: holf holomorphic_on_imp_continuous_on)
  obtain g where contg: continuous_on S g and feq: ⋀x. x ∈ S ⟹ f x = exp (g
x)
    by (metis continuous_logarithm_on_contractible [OF contf S fnz])
  then show thesis
    using fnz g_imp_holomorphic_log holf that by blast
qed

lemma simply_connected_imp_holomorphic_log:
  assumes holf: f holomorphic_on S
      and S: simply_connected S locally path_connected S
      and fnz: ⋀z. z ∈ S ⟹ f z ≠ 0
  obtains g where g holomorphic_on S ⋀z. z ∈ S ⟹ f z = exp(g z)
proof −
  have contf: continuous_on S f
    by (simp add: holf holomorphic_on_imp_continuous_on)
  obtain g where contg: continuous_on S g and feq: ⋀x. x ∈ S ⟹ f x = exp (g
x)
    by (metis continuous_logarithm_on_simply_connected [OF contf S fnz])
  then show thesis
    using fnz g_imp_holomorphic_log holf that by blast
qed
```

lemma *contractible_imp_holomorphic_sqrt*:
assumes *holf*: *f holomorphic_on S*
and *S*: *contractible S*
and *fnz*: $\bigwedge z.\ z \in S \Longrightarrow f\ z \neq 0$
obtains *g* **where** *g holomorphic_on S* $\bigwedge z.\ z \in S \Longrightarrow f\ z = g\ z\ \hat{}\ 2$
proof −
obtain *g* **where** *holg*: *g holomorphic_on S* **and** *feq*: $\bigwedge z.\ z \in S \Longrightarrow f\ z = exp(g\ z)$
using *contractible_imp_holomorphic_log* [*OF assms*] **by** *blast*
show *?thesis*
proof
show *exp* ∘ (λ*z*. *z* / 2) ∘ *g holomorphic_on S*
by (*intro holomorphic_on_compose holg holomorphic_intros*) *auto*
show $\bigwedge z.\ z \in S \Longrightarrow f\ z = ((exp \circ (\lambda z.\ z\ /\ 2) \circ g)\ z)^2$
by (*simp add*: *feq flip*: *exp_double*)
qed
qed

lemma *simply_connected_imp_holomorphic_sqrt*:
assumes *holf*: *f holomorphic_on S*
and *S*: *simply_connected S locally path_connected S*
and *fnz*: $\bigwedge z.\ z \in S \Longrightarrow f\ z \neq 0$
obtains *g* **where** *g holomorphic_on S* $\bigwedge z.\ z \in S \Longrightarrow f\ z = g\ z\ \hat{}\ 2$
proof −
obtain *g* **where** *holg*: *g holomorphic_on S* **and** *feq*: $\bigwedge z.\ z \in S \Longrightarrow f\ z = exp(g\ z)$
using *simply_connected_imp_holomorphic_log* [*OF assms*] **by** *blast*
show *?thesis*
proof
show *exp* ∘ (λ*z*. *z* / 2) ∘ *g holomorphic_on S*
by (*intro holomorphic_on_compose holg holomorphic_intros*) *auto*
show $\bigwedge z.\ z \in S \Longrightarrow f\ z = ((exp \circ (\lambda z.\ z\ /\ 2) \circ g)\ z)^2$
by (*simp add*: *feq flip*: *exp_double*)
qed
qed

Related theorems about holomorphic inverse cosines.

lemma *contractible_imp_holomorphic_arccos*:
assumes *holf*: *f holomorphic_on S* **and** *S*: *contractible S*
and *non1*: $\bigwedge z.\ z \in S \Longrightarrow f\ z \neq 1 \wedge f\ z \neq -1$
obtains *g* **where** *g holomorphic_on S* $\bigwedge z.\ z \in S \Longrightarrow f\ z = cos(g\ z)$
proof −
have *hol1f*: (λ*z*. 1 − *f z* ^ 2) *holomorphic_on S*
by (*intro holomorphic_intros holf*)
obtain *g* **where** *holg*: *g holomorphic_on S* **and** *eq*: $\bigwedge z.\ z \in S \Longrightarrow 1 - (f\ z)^2 = (g\ z)^2$
using *contractible_imp_holomorphic_sqrt* [*OF hol1f S*]
by (*metis eq_iff_diff_eq_0 non1 power2_eq_1_iff*)

```
  have holfg: (λz. f z + i∗g z) holomorphic_on S
    by (intro holf holg holomorphic_intros)
  have ⋀z. z ∈ S ⟹ f z + i∗g z ≠ 0
   by (metis Arccos_body_lemma eq add.commute add.inverse_unique complex_i_mult_minus
power2_csqrt power2_eq_iff)
  then obtain h where holh: h holomorphic_on S and fgeq: ⋀z. z ∈ S ⟹ f z
+ i∗g z = exp (h z)
    using contractible_imp_holomorphic_log [OF holfg S] by metis
  show ?thesis
  proof
    show (λz. −i∗h z) holomorphic_on S
      by (intro holh holomorphic_intros)
    show f z = cos (− i∗h z) if z ∈ S for z
    proof −
      have (f z + i∗g z)∗(f z − i∗g z) = 1
        using that eq by (auto simp: algebra_simps power2_eq_square)
      then have f z − i∗g z = inverse (f z + i∗g z)
        using inverse_unique by force
      also have ... = exp (− h z)
        by (simp add: exp_minus fgeq that)
      finally have f z = exp (− h z) + i∗g z
        by (simp add: diff_eq_eq)
      with that show ?thesis
        by (simp add: cos_exp_eq flip: fgeq)
    qed
  qed
qed


lemma contractible_imp_holomorphic_arccos_bounded:
  assumes holf: f holomorphic_on S and S: contractible S and a ∈ S
      and non1: ⋀z. z ∈ S ⟹ f z ≠ 1 ∧ f z ≠ −1
  obtains g where g holomorphic_on S norm(g a) ≤ pi + norm(f a) ⋀z. z ∈ S
⟹ f z = cos(g z)
proof −
  obtain g where holg: g holomorphic_on S and feq: ⋀z. z ∈ S ⟹ f z = cos (g
z)
    using contractible_imp_holomorphic_arccos [OF holf S non1] by blast
  obtain b where cos b = f a norm b ≤ pi + norm (f a)
    using cos_Arccos norm_Arccos_bounded by blast
  then have cos b = cos (g a)
    by (simp add: ‹a ∈ S› feq)
  then consider n where n ∈ ℤ b = g a + of_real(2∗n∗pi) | n where n ∈ ℤ b
= −g a + of_real(2∗n∗pi)
    by (auto simp: complex_cos_eq)
  then show ?thesis
  proof cases
    case 1
    show ?thesis
```

```
    proof
      show (λz. g z + of_real(2*n*pi)) holomorphic_on S
        by (intro holomorphic_intros holg)
      show cmod (g a + of_real(2*n*pi)) ≤ pi + cmod (f a)
        using 1 ‹cmod b ≤ pi + cmod (f a)› by blast
      show ⋀z. z ∈ S ⟹ f z = cos (g z + complex_of_real (2*n*pi))
        by (metis ‹n ∈ ℤ› complex_cos_eq feq)
    qed
  next
    case 2
    show ?thesis
    proof
      show (λz. −g z + of_real(2*n*pi)) holomorphic_on S
        by (intro holomorphic_intros holg)
      show cmod (−g a + of_real(2*n*pi)) ≤ pi + cmod (f a)
        using 2 ‹cmod b ≤ pi + cmod (f a)› by blast
      show ⋀z. z ∈ S ⟹ f z = cos (−g z + complex_of_real (2*n*pi))
        by (metis ‹n ∈ ℤ› complex_cos_eq feq)
    qed
  qed
qed
```

6.36.16 The "Borsukian" property of sets

This doesn't have a standard name. Kuratowski uses "contractible with respect to $[S^1]$" while Whyburn uses "property b". It's closely related to unicoherence.

```
definition Borsukian where
  Borsukian S ≡
    ∀f. continuous_on S f ∧ f ∈ S → (− {0::complex})
      ⟶ (∃a. homotopic_with_canon (λh. True) S (− {0}) f (λx. a))

lemma Borsukian_retraction_gen:
  assumes Borsukian S continuous_on S h h ‘ S = T
      continuous_on T k  k ∈ T → S  ⋀y. y ∈ T ⟹ h(k y) = y
    shows Borsukian T
proof −
  interpret R: Retracts S h T k
    using assms by (simp add: image_subset_iff_funcset Retracts.intro)
  show ?thesis
    using assms
    apply (clarsimp simp add: Borsukian_def)
    apply (rule R.cohomotopically_trivial_retraction_null_gen [OF TrueI TrueI
refl, of −{0}], auto)
    done
qed

lemma retract_of_Borsukian: ⟦Borsukian T; S retract_of T⟧ ⟹ Borsukian S
```

by (*smt* (*verit*) *Borsukian_retraction_gen retract_of_def retraction retraction_def retraction_subset image_subset_iff_funcset*)

lemma *homeomorphic_Borsukian*: ⟦*Borsukian S*; *S homeomorphic T*⟧ ⟹ *Borsukian T*
using *Borsukian_retraction_gen order_refl*
by (*fastforce simp add*: *homeomorphism_def homeomorphic_def*)

lemma *homeomorphic_Borsukian_eq*:
S homeomorphic T ⟹ *Borsukian S* ⟷ *Borsukian T*
by (*meson homeomorphic_Borsukian homeomorphic_sym*)

lemma *Borsukian_translation*:
fixes *S* :: *'a*::*real_normed_vector set*
shows *Borsukian* (*image* (λ*x*. *a* + *x*) *S*) ⟷ *Borsukian S*
using *homeomorphic_Borsukian_eq homeomorphic_translation* **by** *blast*

lemma *Borsukian_injective_linear_image*:
fixes *f* :: *'a*::*euclidean_space* ⇒ *'b*::*euclidean_space*
assumes *linear f inj f*
shows *Borsukian*(*f* ‘ *S*) ⟷ *Borsukian S*
using *assms homeomorphic_Borsukian_eq linear_homeomorphic_image* **by** *blast*

lemma *homotopy_eqv_Borsukianness*:
fixes *S* :: *'a*::*real_normed_vector set*
and *T* :: *'b*::*real_normed_vector set*
assumes *S homotopy_eqv T*
shows (*Borsukian S* ⟷ *Borsukian T*)
by (*meson Borsukian_def assms homotopy_eqv_cohomotopic_triviality_null image_subset_iff_funcset*)

lemma *Borsukian_alt*:
fixes *S* :: *'a*::*real_normed_vector set*
shows
Borsukian S ⟷
(∀ *f g*. *continuous_on S f* ∧ *f* ∈ *S* → −{*0*} ∧
continuous_on S g ∧ *g* ∈ *S* → −{*0*}
⟶ *homotopic_with_canon* (λ*h*. *True*) *S* (− {*0*::*complex*}) *f g*)
unfolding *Borsukian_def homotopic_triviality*
by (*force simp add*: *path_connected_punctured_universe*)

lemma *Borsukian_continuous_logarithm*:
fixes *S* :: *'a*::*real_normed_vector set*
shows *Borsukian S* ⟷
(∀ *f*. *continuous_on S f* ∧ *f* ∈ *S* → (− {*0*::*complex*})
⟶ (∃ *g*. *continuous_on S g* ∧ (∀ *x* ∈ *S*. *f x* = *exp*(*g x*))))
by (*simp add*: *Borsukian_def inessential_eq_continuous_logarithm*)

lemma *Borsukian_continuous_logarithm_circle*:

```
  fixes S :: 'a::real_normed_vector set
  shows Borsukian S ⟷
          (∀f. continuous_on S f ∧ f ∈ S → sphere (0::complex) 1
              ⟶ (∃g. continuous_on S g ∧ (∀x ∈ S. f x = exp(g x))))
   (is ?lhs = ?rhs)
proof
  assume ?lhs then show ?rhs
    by (force simp: Borsukian_continuous_logarithm)
next
  assume RHS [rule_format]: ?rhs
  show ?lhs
  proof (clarsimp simp: Borsukian_continuous_logarithm Pi_iff)
    fix f :: 'a ⇒ complex
    assume contf: continuous_on S f and 0: ∀i∈S. f i ≠ 0
    then have continuous_on S (λx. f x / complex_of_real (cmod (f x)))
      by (intro continuous_intros) auto
    moreover have (λx. f x / complex_of_real (cmod (f x))) ∈ S → sphere 0 1
      using 0 by (auto simp: norm_divide)
    ultimately obtain g where contg: continuous_on S g
                  and fg: ∀x ∈ S. f x / complex_of_real (cmod (f x)) = exp(g x)
      using RHS [of λx. f x / of_real(norm(f x))] by auto
    show ∃g. continuous_on S g ∧ (∀x∈S. f x = exp (g x))
    proof (intro exI ballI conjI)
      show continuous_on S (λx. (Ln ∘ of_real ∘ norm ∘ f)x + g x)
        by (intro continuous_intros contf contg conjI) (use 0 in auto)
      show f x = exp ((Ln ∘ complex_of_real ∘ cmod ∘ f) x + g x) if x ∈ S for x
        using 0 that
        apply (simp add: exp_add)
      by (metis div_by_0 exp_Ln exp_not_eq_zero fg mult.commute nonzero_eq_divide_eq)
    qed
  qed
qed


lemma Borsukian_continuous_logarithm_circle_real:
  fixes S :: 'a::real_normed_vector set
  shows Borsukian S ⟷
        (∀f. continuous_on S f ∧ f ∈ S → sphere (0::complex) 1
            ⟶ (∃g. continuous_on S (complex_of_real ∘ g) ∧ (∀x ∈ S. f x =
exp(i * of_real(g x)))))
   (is ?lhs = ?rhs)
proof
  assume LHS: ?lhs
  show ?rhs
  proof (clarify)
    fix f :: 'a ⇒ complex
    assume continuous_on S f and f01: f ∈ S → sphere 0 1
     then obtain g where contg: continuous_on S g and ⋀x. x ∈ S ⟹ f x =
exp(g x)
```

```
    using LHS by (auto simp: Borsukian_continuous_logarithm_circle)
   then have ∀x∈S. f x = exp (i * complex_of_real ((Im ∘ g) x))
    using f01 exp_eq_polar norm_exp_eq_Re by (fastforce simp: Pi_iff)
   then show ∃g. continuous_on S (complex_of_real ∘ g) ∧ (∀x∈S. f x = exp
(i * complex_of_real (g x)))
    by (rule_tac x=Im ∘ g in exI) (force intro: continuous_intros contg)
  qed
next
  assume RHS [rule_format]: ?rhs
  show ?lhs
  proof (clarsimp simp: Borsukian_continuous_logarithm_circle)
   fix f :: 'a ⇒ complex
   assume continuous_on S f and f01: f ∈ S → sphere 0 1
  then obtain g where contg: continuous_on S (complex_of_real ∘ g) and ⋀x.
x ∈ S ⟹ f x =  exp(i * of_real(g x))
    by (metis RHS)
   then show ∃g. continuous_on S g ∧ (∀x∈S. f x = exp (g x))
     by (rule_tac x=λx. i* of_real(g x) in exI) (auto simp: continuous_intros
contg)
  qed
qed

lemma Borsukian_circle:
  fixes S :: 'a::real_normed_vector set
  shows Borsukian S ⟷
        (∀f. continuous_on S f ∧ f ∈ S → sphere (0::complex) 1
            ⟶ (∃a. homotopic_with_canon (λh. True) S (sphere (0::complex) 1)
f (λx. a)))
by (simp add: inessential_eq_continuous_logarithm_circle Borsukian_continuous_logarithm_circle_real)

lemma contractible_imp_Borsukian: contractible S ⟹ Borsukian S
 by (meson Borsukian_def nullhomotopic_from_contractible image_subset_iff_funcset)

lemma simply_connected_imp_Borsukian:
  fixes S :: 'a::real_normed_vector set
  shows ⟦simply_connected S; locally path_connected S⟧ ⟹ Borsukian S
 by (smt (verit, del_insts) continuous_logarithm_on_simply_connected Borsukian_continuous_logarithm_circle
     PiE mem_sphere_0 norm_eq_zero zero_neq_one)

lemma starlike_imp_Borsukian:
  fixes S :: 'a::real_normed_vector set
  shows starlike S ⟹ Borsukian S
  by (simp add: contractible_imp_Borsukian starlike_imp_contractible)

lemma Borsukian_empty: Borsukian {}
  by (auto simp: contractible_imp_Borsukian)

lemma Borsukian_UNIV: Borsukian (UNIV :: 'a::real_normed_vector set)
  by (auto simp: contractible_imp_Borsukian)
```

lemma *convex_imp_Borsukian*:
fixes $S :: 'a::real\_normed\_vector\ set$
shows $convex\ S \Longrightarrow Borsukian\ S$
by (*meson Borsukian_def convex_imp_contractible nullhomotopic_from_contractible*)

proposition *Borsukian_sphere*:
fixes $a :: 'a::euclidean\_space$
shows $3 \le DIM('a) \Longrightarrow Borsukian\ (sphere\ a\ r)$
using *ENR_sphere*
by (*blast intro*: *simply_connected_imp_Borsukian ENR_imp_locally_path_connected simply_connected_sphere*)

lemma *Borsukian_Un_lemma*:
fixes $S :: 'a::real\_normed\_vector\ set$
assumes *BS*: *Borsukian S* **and** *BT*: *Borsukian T* **and** *ST*: *connected*$(S \cap T)$
and $*$: $\bigwedge f\ g::'a \Rightarrow complex.$
$[\![continuous\_on\ S\ f;\ continuous\_on\ T\ g;\ \bigwedge x.\ x \in S \wedge x \in T \Longrightarrow f\ x = g\ x]\!]$
$\Longrightarrow continuous\_on\ (S \cup T)\ (\lambda x.\ if\ x \in S\ then\ f\ x\ else\ g\ x)$
shows *Borsukian*$(S \cup T)$
proof (*clarsimp simp add*: *Borsukian_continuous_logarithm Pi_iff*)
fix $f :: 'a \Rightarrow complex$
assume *contf*: $continuous\_on\ (S \cup T)\ f$ **and** *0*: $\forall i \in S \cup T.\ f\ i \neq 0$
then have *contfS*: $continuous\_on\ S\ f$ **and** *contfT*: $continuous\_on\ T\ f$
using *continuous_on_subset* **by** *auto*
have $[\![continuous\_on\ S\ f;\ f\ `\ S \subseteq -\{0\}]\!] \Longrightarrow \exists g.\ continuous\_on\ S\ g \wedge (\forall x \in S.\ f\ x = exp(g\ x))$
using *BS* **by** (*auto simp*: *Borsukian_continuous_logarithm*)
then obtain *g* **where** *contg*: $continuous\_on\ S\ g$ **and** *fg*: $\bigwedge x.\ x \in S \Longrightarrow f\ x = exp(g\ x)$
using *0 contfS* **by** *force*
have $[\![continuous\_on\ T\ f;\ f\ `\ T \subseteq -\{0\}]\!] \Longrightarrow \exists g.\ continuous\_on\ T\ g \wedge (\forall x \in T.\ f\ x = exp(g\ x))$
using *BT* **by** (*auto simp*: *Borsukian_continuous_logarithm*)
then obtain *h* **where** *conth*: $continuous\_on\ T\ h$ **and** *fh*: $\bigwedge x.\ x \in T \Longrightarrow f\ x = exp(h\ x)$
using *0 contfT* **by** *force*
show $\exists g.\ continuous\_on\ (S \cup T)\ g \wedge (\forall x \in S \cup T.\ f\ x = exp\ (g\ x))$
proof (*cases* $S \cap T = \{\}$)
case *True*
show *?thesis*
proof (*intro exI conjI*)
show $continuous\_on\ (S \cup T)\ (\lambda x.\ if\ x \in S\ then\ g\ x\ else\ h\ x)$
using *True* $*$ [*OF contg conth*]
by (*meson disjoint_iff*)
show $\forall x \in S \cup T.\ f\ x = exp\ (if\ x \in S\ then\ g\ x\ else\ h\ x)$
using *fg fh* **by** *auto*
qed

```
  next
    case False
    have (λx. g x − h x) constant_on S ∩ T
    proof (rule continuous_discrete_range_constant [OF ST])
      show continuous_on (S ∩ T) (λx. g x − h x)
        by (metis contg conth continuous_on_diff continuous_on_subset inf_le1
inf_le2)
      show ∃e>0. ∀y. y ∈ S ∩ T ∧ g y − h y ≠ g x − h x ⟶ e ≤ cmod (g y −
h y − (g x − h x))
          if x ∈ S ∩ T for x
      proof −
        have g y − g x = h y − h x
            if y ∈ S y ∈ T cmod (g y − g x − (h y − h x)) < 2 ∗ pi for y
        proof (rule exp_complex_eqI)
          have |Im (g y − g x) − Im (h y − h x)| ≤ cmod (g y − g x − (h y − h
x))
            by (metis abs_Im_le_cmod minus_complex.simps(2))
          then show |Im (g y − g x) − Im (h y − h x)| < 2 ∗ pi
            using that by linarith
          have exp (g x) = exp (h x) exp (g y) = exp (h y)
            using fg fh that ‹x ∈ S ∩ T› by fastforce+
          then show exp (g y − g x) = exp (h y − h x)
            by (simp add: exp_diff)
        qed
        then show ?thesis
          by (rule_tac x=2∗pi in exI) (fastforce simp add: algebra_simps)
      qed
    qed
    then obtain a where a: ⋀x. x ∈ S ∩ T ⟹ g x − h x = a
      by (auto simp: constant_on_def)
    with False have exp a = 1
    by (metis IntI disjoint_iff_not_equal divide_self_if exp_diff exp_not_eq_zero
fg fh)
    with a show ?thesis
      apply (rule_tac x=λx. if x ∈ S then g x else a + h x in exI)
      apply (intro ∗ contg conth continuous_intros conjI)
       apply (auto simp: algebra_simps fg fh exp_add)
      done
  qed
qed

proposition Borsukian_open_Un:
  fixes S :: 'a::real_normed_vector set
  assumes opeS: openin (top_of_set (S ∪ T)) S
      and opeT: openin (top_of_set (S ∪ T)) T
      and BS: Borsukian S and BT: Borsukian T and ST: connected(S ∩ T)
    shows Borsukian(S ∪ T)
  by (force intro: Borsukian_Un_lemma [OF BS BT ST] continuous_on_cases_local_open
[OF opeS opeT])
```

lemma *Borsukian_closed_Un*:
fixes S :: *'a::real_normed_vector set*
assumes *cloS*: *closedin* (*top_of_set* ($S \cup T$)) S
and *cloT*: *closedin* (*top_of_set* ($S \cup T$)) T
and *BS*: *Borsukian* S **and** *BT*: *Borsukian* T **and** *ST*: *connected*($S \cap T$)
shows *Borsukian*($S \cup T$)
by (*force intro*: *Borsukian_Un_lemma* [*OF BS BT ST*] *continuous_on_cases_local* [*OF cloS cloT*])

lemma *Borsukian_separation_compact*:
fixes S :: *complex set*
assumes *compact* S
shows *Borsukian* $S \longleftrightarrow$ *connected*($-$ S)
by (*simp add*: *Borsuk_separation_theorem Borsukian_circle assms*)

lemma *Borsukian_monotone_image_compact*:
fixes f :: *'a::euclidean_space* $\Rightarrow$ *'b::euclidean_space*
assumes *Borsukian* S **and** *contf*: *continuous_on* S f **and** *fim*: f ' $S = T$
and *compact* S **and** *conn*: $\bigwedge y.\ y \in T \Longrightarrow$ *connected* $\{x.\ x \in S \wedge f\ x = y\}$
shows *Borsukian* T
proof (*clarsimp simp*: *Borsukian_continuous_logarithm Pi_iff*)
fix g :: *'b* $\Rightarrow$ *complex*
assume *contg*: *continuous_on* T g **and** *0*: $\forall\, i \in T.\ g\ i \neq 0$
have *continuous_on* S ($g \circ f$)
using *contf contg continuous_on_compose fim* **by** *blast*
moreover have ($g \circ f$) ' $S \subseteq -\{0\}$
using *fim 0* **by** *auto*
ultimately obtain h **where** *conth*: *continuous_on* S h **and** *gfh*: $\bigwedge x.\ x \in S \Longrightarrow$ ($g \circ f$) x = *exp*($h\ x$)
using ‹*Borsukian S*› **by** (*auto simp*: *Borsukian_continuous_logarithm*)
have $\bigwedge y.\ \exists\, x.\ y \in T \longrightarrow x \in S \wedge f\ x = y$
using *fim* **by** *auto*
then obtain f' **where** f': $\bigwedge y.\ y \in T \longrightarrow f'\ y \in S \wedge f\ (f'\ y) = y$
by *metis*
have $*$: ($\lambda x.\ h\ x - h(f'\ y)$) *constant_on* $\{x.\ x \in S \wedge f\ x = y\}$ **if** $y \in T$ **for** y
proof (*rule continuous_discrete_range_constant* [*OF conn* [*OF that*], *of* $\lambda x.\ h\ x - h\ (f'\ y)$], *simp_all add*: *algebra_simps*)
show *continuous_on* $\{x \in S.\ f\ x = y\}$ ($\lambda x.\ h\ x - h\ (f'\ y)$)
by (*intro continuous_intros continuous_on_subset* [*OF conth*]) *auto*
show $\exists\, e{>}0.\ \forall\, u.\ u \in S \wedge f\ u = y \wedge h\ u \neq h\ x \longrightarrow e \leq$ *cmod* ($h\ u - h\ x$)
if x: $x \in S \wedge f\ x = y$ **for** x
proof −
have $h\ u = h\ x$ **if** $u \in S$ $f\ u = y$ *cmod* ($h\ u - h\ x$) $<$ *2* $*$ *pi* **for** u
proof (*rule exp_complex_eqI*)
have |*Im* ($h\ u$) − *Im* ($h\ x$)| $\leq$ *cmod* ($h\ u - h\ x$)
by (*metis abs_Im_le_cmod minus_complex.simps*(*2*))
then show |*Im* ($h\ u$) − *Im* ($h\ x$)| $<$ *2* $*$ *pi*
using *that* **by** *linarith*

```
      show exp (h u) = exp (h x)
        by (simp add: gfh [symmetric] x that)
    qed
    then show ?thesis
      by (rule_tac x=2*pi in exI) (fastforce simp add: algebra_simps)
  qed
 qed
 show ∃h. continuous_on T h ∧ (∀x∈T. g x = exp (h x))
 proof (intro exI conjI)
  show continuous_on T (h ∘ f')
  proof (rule continuous_from_closed_graph [of h ` S])
   show compact (h ` S)
    by (simp add: ‹compact S› compact_continuous_image conth)
   show (h ∘ f') ∈ T → h ` S
    by (auto simp: f')
   have h x = h (f' (f x)) if x ∈ S for x
    using * [of f x] fim that unfolding constant_on_def by clarsimp (metis f'
imageI right_minus_eq)
   moreover have ⋀x. x ∈ T ⟹ ∃u. u ∈ S ∧ x = f u ∧ h (f' x) = h u
    using f' by fastforce
   ultimately
   have eq: ((λx. (x, (h ∘ f') x)) ` T) =
            {p. ∃x. x ∈ S ∧ (x, p) ∈ (S × UNIV) ∩ ((λz. snd z − ((f ∘ fst) z,
(h ∘ fst) z)) −` {0})}
    using fim by (auto simp: image_iff)
   moreover have closed ...
   apply (intro closed_compact_projection [OF ‹compact S›] continuous_closed_preimage
                continuous_intros continuous_on_subset [OF contf] continu-
ous_on_subset [OF conth])
    by (auto simp: ‹compact S› closed_Times compact_imp_closed)
   ultimately show closed ((λx. (x, (h ∘ f') x)) ` T)
    by simp
  qed
 qed (use f' gfh in fastforce)
qed


lemma Borsukian_open_map_image_compact:
 fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
 assumes Borsukian S and contf: continuous_on S f and fim: f ` S = T and
compact S
   and ope: ⋀U. openin (top_of_set S) U
            ⟹ openin (top_of_set T) (f ` U)
  shows Borsukian T
proof (clarsimp simp add: Borsukian_continuous_logarithm_circle_real)
 fix g :: 'b ⇒ complex
 assume contg: continuous_on T g and gim: g ∈ T → sphere 0 1
 have continuous_on S (g ∘ f)
  using contf contg continuous_on_compose fim by blast
```

moreover have $(g \circ f) \in S \to \mathit{sphere}\ 0\ 1$
using *fim gim* **by** *auto*
ultimately obtain h **where** *cont_cxh*: *continuous_on* S $(\mathit{complex\_of\_real} \circ h)$
and *gfh*: $\bigwedge x.\ x \in S \Longrightarrow (g \circ f)\ x = \mathit{exp}(\mathrm{i} * \mathit{of\_real}(h\ x))$
using ‹*Borsukian S*› *Borsukian_continuous_logarithm_circle_real* **by** *metis*
then have *conth*: *continuous_on S h*
by *simp*
have $\exists x.\ x \in S \wedge f\ x = y \wedge (\forall x' \in S.\ f\ x' = y \longrightarrow h\ x \le h\ x')$ **if** $y \in T$ **for** y
proof −
have *1*: *compact* $(h\ `\ \{x \in S.\ f\ x = y\})$
proof (*rule compact_continuous_image*)
show *continuous_on* $\{x \in S.\ f\ x = y\}$ h
by (*rule continuous_on_subset* [*OF conth*]) *auto*
have *compact* $(S \cap f\ -`\ \{y\})$
using *that proper_map_from_compact* [*OF contf* _ ‹*compact S*›] *fim*
by *force*
then show *compact* $\{x \in S.\ f\ x = y\}$
by (*auto simp*: *vimage_def Int_def*)
qed
have *2*: $h\ `\ \{x \in S.\ f\ x = y\} \neq \{\}$
using *fim that* **by** *auto*
have $\exists s \in h\ `\ \{x \in S.\ f\ x = y\}.\ \forall t \in h\ `\ \{x \in S.\ f\ x = y\}.\ s \le t$
using *compact_attains_inf* [*OF 1 2*] **by** *blast*
then show *?thesis* **by** *auto*
qed
then obtain k **where** *kTS*: $\bigwedge y.\ y \in T \Longrightarrow k\ y \in S$
and *fk*: $\bigwedge y.\ y \in T \Longrightarrow f\ (k\ y) = y$
and *hle*: $\bigwedge x'\ y.\ [\![y \in T;\ x' \in S;\ f\ x' = y]\!] \Longrightarrow h\ (k\ y) \le h\ x'$
by *metis*
have *continuous_on T* $(h \circ k)$
proof (*clarsimp simp add*: *continuous_on_iff*)
fix y **and** e::*real*
assume $y \in T\ 0 < e$
moreover have *uniformly_continuous_on S* $(\mathit{complex\_of\_real} \circ h)$
using ‹*compact S*› *cont_cxh compact_uniformly_continuous* **by** *blast*
ultimately obtain d **where** $0 < d$
and d: $\bigwedge x\ x'.\ [\![x \in S;\ x' \in S;\ \mathit{dist}\ x'\ x < d]\!] \Longrightarrow \mathit{dist}\ (h\ x')\ (h\ x) < e$
by (*force simp*: *uniformly_continuous_on_def*)
obtain δ **where** $0 < \delta$ **and** δ:
$\bigwedge x'.\ [\![x' \in T;\ \mathit{dist}\ y\ x' < \delta]\!]$
$\Longrightarrow (\forall v \in \{z \in S.\ f\ z = y\}.\ \exists v'.\ v' \in \{z \in S.\ f\ z = x'\} \wedge \mathit{dist}\ v\ v' < d) \wedge$
$(\forall v' \in \{z \in S.\ f\ z = x'\}.\ \exists v.\ v \in \{z \in S.\ f\ z = y\} \wedge \mathit{dist}\ v'\ v < d)$
proof (*rule upper_lower_hemicontinuous_explicit* [*of T* $\lambda y.\ \{z \in S.\ f\ z = y\}$ *S*])
show $\bigwedge U.\ \mathit{openin}\ (\mathit{top\_of\_set}\ S)\ U$
$\Longrightarrow \mathit{openin}\ (\mathit{top\_of\_set}\ T)\ \{x \in T.\ \{z \in S.\ f\ z = x\} \subseteq U\}$
using *closed_map_iff_upper_hemicontinuous_preimage* [*OF fim* [*THEN equalityD1*]]

```
      by (simp add: Abstract_Topology_2.continuous_imp_closed_map ‹compact
S› contf fim)
     show ⋀U. closedin (top_of_set S) U ⟹
               closedin (top_of_set T) {x ∈ T. {z ∈ S. f z = x} ⊆ U}
      using ope open_map_iff_lower_hemicontinuous_preimage [OF fim [THEN
equalityD1]]
       by meson
     show bounded {z ∈ S. f z = y}
        by (metis (no_types, lifting) compact_imp_bounded [OF ‹compact S›]
bounded_subset mem_Collect_eq subsetI)
   qed (use ‹y ∈ T› ‹0 < d› fk kTS in ‹force+›)
   have dist (h (k y')) (h (k y)) < e if y' ∈ T dist y y' < δ for y'
   proof −
     have k1: k y ∈ S f (k y) = y and k2: k y' ∈ S f (k y') = y'
       by (auto simp: ‹y ∈ T› ‹y' ∈ T› kTS fk)
     have 1: ⋀v. ⟦v ∈ S; f v = y⟧ ⟹ ∃v'. v' ∈ {z ∈ S. f z = y'} ∧ dist v v' < d
      and 2: ⋀v'. ⟦v' ∈ S; f v' = y'⟧ ⟹ ∃v. v ∈ {z ∈ S. f z = y} ∧ dist v' v < d
       using δ [OF that] by auto
     then obtain w' w where w' ∈ S f w' = y' dist (k y) w' < d
       and w ∈ S f w = y dist (k y') w < d
       using 1 [OF k1] 2 [OF k2] by auto
     then show ?thesis
       using d [of w k y'] d [of w' k y] k1 k2 ‹y' ∈ T› ‹y ∈ T› hle
       by (fastforce simp: dist_norm abs_diff_less_iff algebra_simps)
   qed
   then show ∃d>0. ∀x'∈T. dist x' y < d ⟶ dist (h (k x')) (h (k y)) < e
     using ‹0 < δ› by (auto simp: dist_commute)
 qed
 then show ∃h. continuous_on T h ∧ (∀x∈T. g x = exp (i ∗ complex_of_real
(h x)))
   using fk gfh kTS by force
qed
```

If two points are separated by a closed set, there's a minimal one.

```
proposition closed_irreducible_separator:
 fixes a :: 'a::real_normed_vector
 assumes closed S and ab: ¬ connected_component (− S) a b
 obtains T where T ⊆ S closed T T ≠ {} ¬ connected_component (− T) a b
               ⋀U. U ⊂ T ⟹ connected_component (− U) a b
proof (cases a ∈ S ∨ b ∈ S)
 case True
 then show ?thesis
 proof
   assume ∗: a ∈ S
   show ?thesis
   proof
     show {a} ⊆ S
       using ∗ by blast
     show ¬ connected_component (− {a}) a b
```

```
      using connected_component_in by auto
    show ⋀U. U ⊂ {a} ⟹ connected_component (− U) a b
    by (metis connected_component_UNIV UNIV_I compl_bot_eq connected_component_eq_eq
less_le_not_le subset_singletonD)
   qed auto
 next
   assume ∗: b ∈ S
   show ?thesis
   proof
     show {b} ⊆ S
       using ∗ by blast
     show ¬ connected_component (− {b}) a b
       using connected_component_in by auto
     show ⋀U. U ⊂ {b} ⟹ connected_component (− U) a b
     by (metis connected_component_UNIV UNIV_I compl_bot_eq connected_component_eq_eq
less_le_not_le subset_singletonD)
   qed auto
 qed
next
 case False
 define A where A ≡ connected_component_set (− S) a
 define B where B ≡ connected_component_set (− (closure A)) b
 have a ∈ A
   using False A_def by auto
 have b ∈ B
   unfolding A_def B_def closure_Un_frontier
  using ab False ‹closed S› frontier_complement frontier_of_connected_component_subset
frontier_subset_closed by force
 have frontier B ⊆ frontier (connected_component_set (− closure A) b)
   using B_def by blast
 also have frsub: ... ⊆ frontier A
 proof −
   have ⋀A. closure (− closure (− A)) ⊆ closure A
    by (metis (no_types) closure_mono closure_subset compl_le_compl_iff dou-
ble_compl)
   then show ?thesis
     by (metis (no_types) closure_closure double_compl frontier_closures fron-
tier_of_connected_component_subset le_inf_iff subset_trans)
 qed
 finally have frBA: frontier B ⊆ frontier A .
 show ?thesis
 proof
   show frontier B ⊆ S
   proof −
     have frontier S ⊆ S
       by (simp add: ‹closed S› frontier_subset_closed)
     then show ?thesis
      using frsub frontier_complement frontier_of_connected_component_subset
       unfolding A_def B_def by blast
```

qed
show *closed* (*frontier B*)
by *simp*
show ¬ *connected_component* (− *frontier B*) *a b*
unfolding *connected_component_def*
proof *clarify*
fix *T*
assume *connected T* **and** *TB*: *T* ⊆ − *frontier B* **and** *a* ∈ *T* **and** *b* ∈ *T*
have *a* ∉ *B*
by (*metis A_def B_def ComplD* ‹*a* ∈ *A*› *assms*(*1*) *closed_open connected_component_subset in_closure_connected_component subsetD*)
have *T* ∩ *B* ≠ {}
using ‹*b* ∈ *B*› ‹*b* ∈ *T*› **by** *blast*
moreover have *T* − *B* ≠ {}
using ‹*a* ∉ *B*› ‹*a* ∈ *T*› **by** *blast*
ultimately show *False*
using *connected_Int_frontier* [*of T B*] *TB* ‹*connected T*› **by** *blast*
qed
moreover have *connected_component* (− *frontier B*) *a b* **if** *frontier B* = {}
using *connected_component_eq_UNIV that* **by** *auto*
ultimately show *frontier B* ≠ {}
by *blast*
show *connected_component* (− *U*) *a b* **if** *U* ⊂ *frontier B* **for** *U*
proof −
obtain *p* **where** *Usub*: *U* ⊆ *frontier B* **and** *p*: *p* ∈ *frontier B p* ∉ *U*
using ‹*U* ⊂ *frontier B*› **by** *blast*
show *?thesis*
unfolding *connected_component_def*
proof (*intro exI conjI*)
have *connected* ((*insert p A*) ∪ (*insert p B*))
proof (*rule connected_Un*)
show *connected* (*insert p A*)
by (*metis A_def IntD1 frBA* ‹*p* ∈ *frontier B*› *closure_insert closure_subset connected_connected_component connected_intermediate_closure frontier_closures insert_absorb subsetCE subset_insertI*)
show *connected* (*insert p B*)
by (*metis B_def IntD1* ‹*p* ∈ *frontier B*› *closure_insert closure_subset connected_connected_component connected_intermediate_closure frontier_closures insert_absorb subset_insertI*)
qed *blast*
then show *connected* (*insert p* (*B* ∪ *A*))
by (*simp add*: *sup.commute*)
have *A* ⊆ − *U*
using *A_def Usub* ‹*frontier B* ⊆ *S*› *connected_component_subset* **by** *fastforce*
moreover have *B* ⊆ − *U*
using *B_def Usub connected_component_subset frBA frontier_closures* **by** *fastforce*
ultimately show *insert p* (*B* ∪ *A*) ⊆ − *U*

```
        using p by auto
      qed (auto simp: ‹a ∈ A› ‹b ∈ B›)
    qed
  qed
qed

lemma frontier_minimal_separating_closed_pointwise:
  fixes S :: 'a::real_normed_vector set
  assumes S: closed S a ∉ S and nconn: ¬ connected_component (− S) a b
      and conn: ⋀T. ⟦closed T; T ⊂ S⟧ ⟹ connected_component (− T) a b
    shows frontier(connected_component_set (− S) a) = S (is ?F = S)
proof −
  have ?F ⊆ S
    by (simp add: S componentsI frontier_of_components_closed_complement)
  moreover have False if ?F ⊂ S
  proof −
    have connected_component (− ?F) a b
      by (simp add: conn that)
    then obtain T where connected T T ⊆ − ?F a ∈ T b ∈ T
      by (auto simp: connected_component_def)
    moreover have T ∩ ?F ≠ {}
    proof (rule connected_Int_frontier [OF ‹connected T›])
      show T ∩ connected_component_set (− S) a ≠ {}
        using ‹a ∉ S› ‹a ∈ T› by fastforce
      show T − connected_component_set (− S) a ≠ {}
        using ‹b ∈ T› nconn by blast
    qed
    ultimately show ?thesis
      by blast
  qed
  ultimately show ?thesis
    by blast
qed
```

6.36.17 Unicoherence (closed)

```
definition unicoherent where
  unicoherent U ≡
  ∀ S T. connected S ∧ connected T ∧ S ∪ T = U ∧
        closedin (top_of_set U) S ∧ closedin (top_of_set U) T
        ⟶ connected (S ∩ T)

lemma unicoherentI [intro?]:
  assumes ⋀S T. ⟦connected S; connected T; U = S ∪ T; closedin (top_of_set
U) S; closedin (top_of_set U) T⟧
          ⟹ connected (S ∩ T)
  shows unicoherent U
  using assms unfolding unicoherent_def by blast
```

lemma *unicoherentD*:
assumes *unicoherent U connected S connected T U = S ∪ T closedin (top_of_set U) S closedin (top_of_set U) T*
shows *connected (S ∩ T)*
using *assms* **unfolding** *unicoherent_def* **by** *blast*

proposition *homeomorphic_unicoherent*:
assumes *ST*: *S homeomorphic T* **and** *S*: *unicoherent S*
shows *unicoherent T*
proof −
obtain *f g* **where** *gf*: *⋀x. x ∈ S ⟹ g (f x) = x* **and** *fim*: *T = f ‘ S* **and** *gfim*: *g ‘ f ‘ S = S*
and *contf*: *continuous_on S f* **and** *contg*: *continuous_on (f ‘ S) g*
using *ST* **by** (*auto simp*: *homeomorphic_def homeomorphism_def*)
show *?thesis*
proof
fix *U V*
assume *connected U connected V* **and** *T*: *T = U ∪ V*
and *cloU*: *closedin (top_of_set T) U*
and *cloV*: *closedin (top_of_set T) V*
have *f ‘ (g ‘ U ∩ g ‘ V) ⊆ U f ‘ (g ‘ U ∩ g ‘ V) ⊆ V*
using *gf fim T* **by** *auto* (*metis UnCI image_iff*)+
moreover have *U ∩ V ⊆ f ‘ (g ‘ U ∩ g ‘ V)*
using *gf fim* **by** (*force simp*: *image_iff T*)
ultimately have *U ∩ V = f ‘ (g ‘ U ∩ g ‘ V)* **by** *blast*
moreover have *connected (f ‘ (g ‘ U ∩ g ‘ V))*
proof (*rule connected_continuous_image*)
show *continuous_on (g ‘ U ∩ g ‘ V) f*
using *T fim gfim* **by** (*metis Un_upper1 contf continuous_on_subset image_mono inf_le1*)
show *connected (g ‘ U ∩ g ‘ V)*
proof (*intro conjI unicoherentD* [*OF S*])
show *connected (g ‘ U) connected (g ‘ V)*
using ‹*connected U*› *cloU* ‹*connected V*› *cloV*
by (*metis Topological_Spaces.connected_continuous_image closedin_imp_subset contg continuous_on_subset fim*)+
show *S = g ‘ U ∪ g ‘ V*
using *T fim gfim* **by** *auto*
have *hom*: *homeomorphism T S g f*
by (*simp add*: *contf contg fim gf gfim homeomorphism_def*)
have *closedin (top_of_set T) U closedin (top_of_set T) V*
by (*simp_all add*: *cloU cloV*)
then show *closedin (top_of_set S) (g ‘ U)*
closedin (top_of_set S) (g ‘ V)
by (*blast intro*: *homeomorphism_imp_closed_map* [*OF hom*])+
qed
qed
ultimately show *connected (U ∩ V)* **by** *metis*
qed

qed

lemma *homeomorphic_unicoherent_eq*:
  S homeomorphic T $\Longrightarrow$ (*unicoherent S* $\longleftrightarrow$ *unicoherent T*)
 by (*meson homeomorphic_sym homeomorphic_unicoherent*)

lemma *unicoherent_translation*:
 fixes *S* :: *'a*::*real_normed_vector set*
 shows
  unicoherent (*image* ($\lambda x.$ *a* + *x*) *S*) $\longleftrightarrow$ *unicoherent S*
 using *homeomorphic_translation homeomorphic_unicoherent_eq* **by** *blast*

lemma *unicoherent_injective_linear_image*:
 fixes *f* :: *'a*::*euclidean_space* $\Rightarrow$ *'b*::*euclidean_space*
 assumes *linear f inj f*
 shows (*unicoherent*(*f* ' *S*) $\longleftrightarrow$ *unicoherent S*)
  using *assms homeomorphic_unicoherent_eq linear_homeomorphic_image* **by** *blast*

lemma *Borsukian_imp_unicoherent*:
 fixes *U* :: *'a*::*euclidean_space set*
 assumes *Borsukian U* **shows** *unicoherent U*
 unfolding *unicoherent_def*
proof *clarify*
 fix *S T*
 assume *connected S connected T U* = *S* $\cup$ *T*
   and *cloS*: *closedin* (*top_of_set* (*S* $\cup$ *T*)) *S*
   and *cloT*: *closedin* (*top_of_set* (*S* $\cup$ *T*)) *T*
 show *connected* (*S* $\cap$ *T*)
  unfolding *connected_closedin_eq*
 proof *clarify*
  fix *V W*
  assume *closedin* (*top_of_set* (*S* $\cap$ *T*)) *V*
    and *closedin* (*top_of_set* (*S* $\cap$ *T*)) *W*
    and *VW*: *V* $\cup$ *W* = *S* $\cap$ *T V* $\cap$ *W* = {} **and** *V* $\neq$ {} *W* $\neq$ {}
  then have *cloV*: *closedin* (*top_of_set U*) *V* **and** *cloW*: *closedin* (*top_of_set U*) *W*
   using ‹*U* = *S* $\cup$ *T*› *cloS cloT closedin_trans* **by** *blast*+
  obtain *q* **where** *contq*: *continuous_on U q*
      and *q01*: $\bigwedge x.$ *x* $\in$ *U* $\Longrightarrow$ *q x* $\in$ {*0..1*::*real*}
      and *qV*: $\bigwedge x.$ *x* $\in$ *V* $\Longrightarrow$ *q x* = *0* **and** *qW*: $\bigwedge x.$ *x* $\in$ *W* $\Longrightarrow$ *q x* = *1*
   by (*rule Urysohn_local* [*OF cloV cloW* ‹*V* $\cap$ *W* = {}›, *of 0 1*])
      (*fastforce simp*: *closed_segment_eq_real_ivl*)
  let *?h* = $\lambda x.$ *if x* $\in$ *S then exp*(*pi* $*$ i $*$ *q x*) *else 1 / exp*(*pi* $*$ i $*$ *q x*)
  have *eqST*: *exp*(*pi* $*$ i $*$ *q x*) = *1 / exp*(*pi* $*$ i $*$ *q x*) **if** *x* $\in$ *S* $\cap$ *T* **for** *x*
  proof −
   have *x* $\in$ *V* $\cup$ *W*

```
      using that ‹V ∪ W = S ∩ T› by blast
    with qV qW show ?thesis by force
  qed
  obtain g where contg: continuous_on U g
    and circle: g ∈ U → sphere 0 1
    and S: ⋀x. x ∈ S ⟹ g x = exp(pi * i * q x)
    and T: ⋀x. x ∈ T ⟹ g x = 1 / exp(pi * i * q x)
  proof
    show continuous_on U ?h
      unfolding ‹U = S ∪ T›
    proof (rule continuous_on_cases_local [OF cloS cloT])
      show continuous_on S (λx. exp (pi * i * q x))
      proof (intro continuous_intros)
        show continuous_on S q
          using ‹U = S ∪ T› continuous_on_subset contq by blast
      qed
      show continuous_on T (λx. 1 / exp (pi * i * q x))
      proof (intro continuous_intros)
        show continuous_on T q
          using ‹U = S ∪ T› continuous_on_subset contq by auto
      qed auto
    qed (use eqST in auto)
  qed (use eqST in ‹auto simp: norm_divide›)
  then obtain h where conth: continuous_on U h and heq: ⋀x. x ∈ U ⟹ g
x = exp (h x)
    by (metis Borsukian_continuous_logarithm_circle assms)
  obtain v w where v ∈ V w ∈ W
    using ‹V ≠ {}› ‹W ≠ {}› by blast
  then have vw: v ∈ S ∩ T w ∈ S ∩ T
    using VW by auto
   have iff: 2 * pi ≤ cmod (2 * of_int m * of_real pi * i − 2 * of_int n *
of_real pi * i)
        ⟷ 1 ≤ abs (m − n) for m n
  proof −
    have 2 * pi ≤ cmod (2 * of_int m * of_real pi * i − 2 * of_int n * of_real
pi * i)
          ⟷ 2 * pi ≤ cmod ((2 * pi * i) * (of_int m − of_int n))
      by (simp add: algebra_simps)
    also have ... ⟷ 2 * pi ≤ 2 * pi * cmod (of_int m − of_int n)
      by (simp add: norm_mult)
    also have ... ⟷ 1 ≤ abs (m − n)
      by simp (metis norm_of_int of_int_1_le_iff of_int_abs of_int_diff)
    finally show ?thesis .
  qed
  have *: ∃n::int. h x − (pi * i * q x) = (of_int(2*n) * pi) * i if x ∈ S for x
    using that S ‹U = S ∪ T› heq exp_eq [symmetric] by (simp add: alge-
bra_simps)
  moreover have (λx. h x − (pi * i * q x)) constant_on S
  proof (rule continuous_discrete_range_constant [OF ‹connected S›])
```

```
    have continuous_on S h continuous_on S q
      using ‹U = S ∪ T› continuous_on_subset conth contq by blast+
    then show continuous_on S (λx. h x − (pi ∗ i ∗ q x))
      by (intro continuous_intros)
    have 2∗pi ≤ cmod (h y − (pi ∗ i ∗ q y) − (h x − (pi ∗ i ∗ q x)))
      if x ∈ S y ∈ S and ne: h y − (pi ∗ i ∗ q y) ≠ h x − (pi ∗ i ∗ q x) for x y
      using ∗ [OF ‹x ∈ S›] ∗ [OF ‹y ∈ S›] ne by (auto simp: iff)
    then show ⋀x. x ∈ S ⟹
      ∃e>0. ∀y. y ∈ S ∧ h y − (pi ∗ i ∗ q y) ≠ h x − (pi ∗ i ∗ q x) ⟶
            e ≤ cmod (h y − (pi ∗ i ∗ q y) − (h x − (pi ∗ i ∗ q x)))
      by (rule_tac x=2∗pi in exI) auto
  qed
  ultimately
  obtain m where m: ⋀x. x ∈ S ⟹ h x − (pi ∗ i ∗ q x) = (of_int(2∗m) ∗ pi)
∗ i
    using vw by (force simp: constant_on_def)
  have ∗: ∃n::int. h x = − (pi ∗ i ∗ q x) + (of_int(2∗n) ∗ pi) ∗ i if x ∈ T for x
    unfolding exp_eq [symmetric]
      using that T ‹U = S ∪ T› by (simp add: exp_minus field_simps  heq
[symmetric])
  moreover have (λx. h x + (pi ∗ i ∗ q x)) constant_on T
  proof (rule continuous_discrete_range_constant [OF ‹connected T›])
    have continuous_on T h continuous_on T q
      using ‹U = S ∪ T› continuous_on_subset conth contq by blast+
    then show continuous_on T (λx. h x + (pi ∗ i ∗ q x))
      by (intro continuous_intros)
    have 2∗pi ≤ cmod (h y + (pi ∗ i ∗ q y) − (h x + (pi ∗ i ∗ q x)))
      if x ∈ T y ∈ T and ne: h y + (pi ∗ i ∗ q y) ≠ h x + (pi ∗ i ∗ q x) for x y
      using ∗ [OF ‹x ∈ T›] ∗ [OF ‹y ∈ T›] ne by (auto simp: iff)
    then show ⋀x. x ∈ T ⟹
      ∃e>0. ∀y. y ∈ T ∧ h y + (pi ∗ i ∗ q y) ≠ h x + (pi ∗ i ∗ q x) ⟶
            e ≤ cmod (h y + (pi ∗ i ∗ q y) − (h x + (pi ∗ i ∗ q x)))
      by (rule_tac x=2∗pi in exI) auto
  qed
  ultimately
  obtain n where n: ⋀x. x ∈ T ⟹ h x + (pi ∗ i ∗ q x) = (of_int(2∗n) ∗ pi)
∗ i
    using vw by (force simp: constant_on_def)
  show False
    using m [of v] m [of w] n [of v] n [of w] vw
    by (auto simp: algebra_simps ‹v ∈ V› ‹w ∈ W› qV qW)
 qed
qed


corollary contractible_imp_unicoherent:
 fixes U :: 'a::euclidean_space set
 assumes contractible U  shows unicoherent U
 by (simp add: Borsukian_imp_unicoherent assms contractible_imp_Borsukian)
```

corollary *convex_imp_unicoherent*:
fixes $U :: 'a::euclidean\_space\ set$
assumes *convex U* **shows** *unicoherent U*
by (*simp add*: *Borsukian_imp_unicoherent assms convex_imp_Borsukian*)

If the type class constraint can be relaxed, I don't know how!

corollary *unicoherent_UNIV*: *unicoherent* (*UNIV* :: *'a* :: *euclidean_space set*)
by (*simp add*: *convex_imp_unicoherent*)

lemma *unicoherent_monotone_image_compact*:
fixes $T :: 'b :: t2\_space\ set$
assumes *S*: *unicoherent S compact S* **and** *contf*: *continuous_on S f* **and** *fim*: *f ‘ S = T*
and *conn*: $\bigwedge y.\ y \in T \Longrightarrow connected\ (S \cap f -` \{y\})$
shows *unicoherent T*
proof
fix *U V*
assume *UV*: *connected U connected V T = U ∪ V*
and *cloU*: *closedin* (*top_of_set T*) *U*
and *cloV*: *closedin* (*top_of_set T*) *V*
moreover have *compact T*
using ‹*compact S*› *compact_continuous_image contf fim* **by** *blast*
ultimately have *closed U closed V*
by (*auto simp*: *closedin_closed_eq compact_imp_closed*)
let *?SUV* = $(S \cap f -` U) \cap (S \cap f -` V)$
have *UV_eq*: *f ‘ ?SUV = U ∩ V*
using ‹*T = U ∪ V*› *fim* **by** *force+*
have *connected* (*f ‘ ?SUV*)
proof (*rule connected_continuous_image*)
show *continuous_on ?SUV f*
by (*meson contf continuous_on_subset inf_le1*)
show *connected ?SUV*
proof (*rule unicoherentD* [*OF* ‹*unicoherent S*›, *of S ∩ f -‘ U S ∩ f -‘ V*])
have $\bigwedge C.\ closedin\ (top\_of\_set\ S)\ C \Longrightarrow closedin\ (top\_of\_set\ T)\ (f ` C)$
by (*metis* ‹*compact S*› *closed_subset closedin_compact closedin_imp_subset compact_continuous_image compact_imp_closed contf continuous_on_subset fim image_mono*)
then show *connected* $(S \cap f -` U)$ *connected* $(S \cap f -` V)$
using *UV* **by** (*auto simp*: *conn intro*: *connected_closed_monotone_preimage* [*OF contf fim*])
show $S = (S \cap f -` U) \cup (S \cap f -` V)$
using *UV fim* **by** *blast*
show *closedin* (*top_of_set S*) $(S \cap f -` U)$
closedin (*top_of_set S*) $(S \cap f -` V)$
by (*auto simp*: *continuous_on_imp_closedin cloU cloV contf fim*)
qed
qed

with *UV_eq* **show** *connected* $(U \cap V)$
by *simp*
qed

6.36.18 Several common variants of unicoherence

lemma *connected_frontier_simple*:
fixes $S :: 'a :: euclidean\_space\ set$
assumes *connected S connected*$(-S)$ **shows** *connected*(*frontier S*)
unfolding *frontier_closures*
by (*rule unicoherentD* [*OF unicoherent_UNIV*]; *simp add*: *assms connected_imp_connected_closure flip*: *closure_Un*)

lemma *connected_frontier_component_complement*:
fixes $S :: 'a :: euclidean\_space\ set$
assumes *connected S C* $\in$ *components*$(-S)$ **shows** *connected*(*frontier C*)
by (*meson assms component_complement_connected connected_frontier_simple in_components_connected*)

lemma *connected_frontier_disjoint*:
fixes $S :: 'a :: euclidean\_space\ set$
assumes *connected S connected T disjnt S T* **and** *ST*: *frontier S* $\subseteq$ *frontier T*
shows *connected*(*frontier S*)
proof (*cases S* $=$ *UNIV*)
case *True* **then show** *?thesis*
by *simp*
next
case *False*
then have $-S \neq \{\}$
by *blast*
then obtain *C* **where** *C*: *C* $\in$ *components*$(-S)$ **and** $T \subseteq C$
by (*metis ComplI disjnt_iff subsetI exists_component_superset* ‹*disjnt S T*› ‹*connected T*›)
moreover have *frontier S* $=$ *frontier C*
proof $-$
have *frontier C* $\subseteq$ *frontier S*
using *C frontier_complement frontier_of_components_subset* **by** *blast*
moreover have $x \in$ *frontier C* **if** $x \in$ *frontier S* **for** x
proof $-$
have $x \in$ *closure C*
using *that* **unfolding** *frontier_def*
by (*metis* (*no_types*) *Diff_eq ST* ‹$T \subseteq C$› *closure_mono contra_subsetD frontier_def le_inf_iff that*)
moreover have $x \notin$ *interior C*
using *that* **unfolding** *frontier_def*
by (*metis C Compl_eq_Diff_UNIV Diff_iff subsetD in_components_subset interior_diff interior_mono*)
ultimately show *?thesis*
by (*auto simp*: *frontier_def*)

qed
ultimately show *?thesis*
by *blast*
qed
ultimately show *?thesis*
using ‹*connected S*› *connected_frontier_component_complement* **by** *auto*
qed

6.36.19 Some separation results

lemma *separation_by_component_closed_pointwise*:
fixes *S* :: *'a* :: *euclidean_space set*
assumes *closed S* ¬ *connected_component* (− *S*) *a b*
obtains *C* **where** *C* ∈ *components S* ¬ *connected_component*(− *C*) *a b*
proof (*cases a* ∈ *S* ∨ *b* ∈ *S*)
case *True*
then show *?thesis*
using *connected_component_in componentsI that* **by** *fastforce*
next
case *False*
obtain *T* **where** *T* ⊆ *S closed T T* ≠ {}
and *nab*: ¬ *connected_component* (− *T*) *a b*
and *conn*: ⋀*U*. *U* ⊂ *T* ⟹ *connected_component* (− *U*) *a b*
using *closed_irreducible_separator* [*OF assms*] **by** *metis*
moreover have *connected T*
proof −
have *ab*: *frontier*(*connected_component_set* (− *T*) *a*) = *T frontier*(*connected_component_set* (− *T*) *b*) = *T*
using *frontier_minimal_separating_closed_pointwise*
by (*metis False* ‹*T* ⊆ *S*› ‹*closed T*› *connected_component_sym conn connected_component_eq_empty connected_component_intermediate_subset empty_subsetI nab*)+
have *connected* (*frontier* (*connected_component_set* (− *T*) *a*))
proof (*rule connected_frontier_disjoint*)
show *disjnt* (*connected_component_set* (− *T*) *a*) (*connected_component_set* (− *T*) *b*)
unfolding *disjnt_iff*
by (*metis connected_component_eq connected_component_eq_empty connected_component_idemp mem_Collect_eq nab*)
show *frontier* (*connected_component_set* (− *T*) *a*) ⊆ *frontier* (*connected_component_set* (− *T*) *b*)
by (*simp add*: *ab*)
qed *auto*
with *ab* ‹*closed T*› **show** *?thesis*
by *simp*
qed
ultimately obtain *C* **where** *C* ∈ *components S T* ⊆ *C*
using *exists_component_superset* [*of T S*] **by** *blast*
then show *?thesis*

by (*meson Compl_anti_mono connected_component_of_subset nab that*)
qed

lemma *separation_by_component_closed*:
fixes $S :: 'a :: euclidean\_space\ set$
assumes *closed* S $\neg$ *connected*$(-\ S)$
obtains C **where** $C \in components\ S$ $\neg$ *connected*$(-\ C)$
proof $-$
obtain $x\ y$ **where** *closed* S $x \notin S$ $y \notin S$ **and** $\neg$ *connected_component* $(-\ S)$ $x\ y$
using *assms* **by** (*auto simp*: *connected_iff_connected_component*)
then obtain C **where** $C \in components\ S$ $\neg$ *connected_component*$(-\ C)$ $x\ y$
using *separation_by_component_closed_pointwise* **by** *metis*
then show *thesis*
by (*metis Compl_iff* ‹$x \notin S$› ‹$y \notin S$› *connected_component_eq_self in_components_subset mem_Collect_eq subsetD that*)
qed

lemma *separation_by_Un_closed_pointwise*:
fixes $S :: 'a :: euclidean\_space\ set$
assumes *ST*: *closed* S *closed* T $S \cap T = \{\}$
and *conS*: *connected_component* $(-\ S)$ a b **and** *conT*: *connected_component* $(-\ T)$ a b
shows *connected_component* $(-\ (S \cup T))$ a b
proof (*rule ccontr*)
have $a \notin S$ $b \notin S$ $a \notin T$ $b \notin T$
using *conS conT connected_component_in* **by** *auto*
assume $\neg$ *connected_component* $(-\ (S \cup T))$ a b
then obtain C **where** $C \in components\ (S \cup T)$ **and** C: $\neg$ *connected_component*$(-\ C)$ a b
using *separation_by_component_closed_pointwise assms* **by** *blast*
then have $C \subseteq S \vee C \subseteq T$
proof $-$
have *connected* C $C \subseteq S \cup T$
using ‹$C \in components\ (S \cup T)$› *in_components_subset* **by** (*blast elim*: *componentsE*)+
moreover then have $C \cap T = \{\} \vee C \cap S = \{\}$
by (*metis Int_empty_right ST inf.commute connected_closed*)
ultimately show *?thesis*
by *blast*
qed
then show *False*
by (*meson Compl_anti_mono C conS conT connected_component_of_subset*)
qed

lemma *separation_by_Un_closed*:
fixes $S :: 'a :: euclidean\_space\ set$
assumes *ST*: *closed* S *closed* T $S \cap T = \{\}$ **and** *conS*: *connected*$(-\ S)$ **and** *conT*: *connected*$(-\ T)$

shows *connected*(− (*S* ∪ *T*))
using *assms separation_by_Un_closed_pointwise*
by (*fastforce simp add*: *connected_iff_connected_component*)

lemma *open_unicoherent_UNIV*:
fixes *S* :: *'a* :: *euclidean_space set*
assumes *open S open T connected S connected T S* ∪ *T* = *UNIV*
shows *connected*(*S* ∩ *T*)
proof −
have *connected*(− (−*S* ∪ −*T*))
by (*metis closed_Compl compl_sup compl_top_eq double_compl separation_by_Un_closed assms*)
then show *?thesis*
by *simp*
qed

lemma *separation_by_component_open_aux*:
fixes *S* :: *'a* :: *euclidean_space set*
assumes *ST*: *closed S closed T S* ∩ *T* = {}
and *S* ≠ {} *T* ≠ {}
obtains *C* **where** *C* ∈ *components*(−(*S* ∪ *T*)) *C* ≠ {} *frontier C* ∩ *S* ≠ {} *frontier C* ∩ *T* ≠ {}
proof (*rule ccontr*)
let *?S* = *S* ∪ ⋃{*C* ∈ *components*(− (*S* ∪ *T*)). *frontier C* ⊆ *S*}
let *?T* = *T* ∪ ⋃{*C* ∈ *components*(− (*S* ∪ *T*)). *frontier C* ⊆ *T*}
assume ¬ *thesis*
with *that* **have** ∗: *frontier C* ∩ *S* = {} ∨ *frontier C* ∩ *T* = {}
if *C*: *C* ∈ *components* (− (*S* ∪ *T*)) *C* ≠ {} **for** *C*
using *C* **by** *blast*
have ∃ *A B*::*'a set*. *closed A* ∧ *closed B* ∧ *UNIV* ⊆ *A* ∪ *B* ∧ *A* ∩ *B* = {} ∧ *A* ≠ {} ∧ *B* ≠ {}
proof (*intro exI conjI*)
have *frontier* (⋃{*C* ∈ *components* (− *S* ∩ − *T*). *frontier C* ⊆ *S*}) ⊆ *S*
using *subset_trans* [*OF frontier_Union_subset_closure*]
by (*metis* (*no_types*, *lifting*) *SUP_least* ‹*closed S*› *closure_minimal mem_Collect_eq*)
then have *frontier ?S* ⊆ *S*
by (*simp add*: *frontier_subset_eq assms subset_trans* [*OF frontier_Un_subset*])
then show *closed ?S*
using *frontier_subset_eq* **by** *fastforce*
have *frontier* (⋃{*C* ∈ *components* (− *S* ∩ − *T*). *frontier C* ⊆ *T*}) ⊆ *T*
using *subset_trans* [*OF frontier_Union_subset_closure*]
by (*metis* (*no_types*, *lifting*) *SUP_least* ‹*closed T*› *closure_minimal mem_Collect_eq*)
then have *frontier ?T* ⊆ *T*
by (*simp add*: *frontier_subset_eq assms subset_trans* [*OF frontier_Un_subset*])
then show *closed ?T*
using *frontier_subset_eq* **by** *fastforce*
have *UNIV* ⊆ (*S* ∪ *T*) ∪ ⋃(*components*(− (*S* ∪ *T*)))
using *Union_components* **by** *blast*
also have ... ⊆ *?S* ∪ *?T*

```
    proof −
      have C ∈ components (−(S ∪ T)) ∧ frontier C ⊆ S ∨
            C ∈ components (−(S ∪ T)) ∧ frontier C ⊆ T
        if C ∈ components (− (S ∪ T)) C ≠ {} for C
        using ∗ [OF that] that
        by clarify (metis (no_types, lifting) UnE ‹closed S› ‹closed T› closed_Un
disjoint_iff_not_equal frontier_of_components_closed_complement subsetCE)
      then show ?thesis
        by blast
    qed
    finally show UNIV ⊆ ?S ∪ ?T .
    have ⋃{C ∈ components (− (S ∪ T)). frontier C ⊆ S} ∪
         ⋃{C ∈ components (− (S ∪ T)). frontier C ⊆ T} ⊆ − (S ∪ T)
      using in_components_subset by fastforce
    moreover have ⋃{C ∈ components (− (S ∪ T)). frontier C ⊆ S} ∩
                  ⋃{C ∈ components (− (S ∪ T)). frontier C ⊆ T} = {}
    proof −
      have C ∩ C′ = {} if C ∈ components (− (S ∪ T)) frontier C ⊆ S
                          C′ ∈ components (− (S ∪ T)) frontier C′ ⊆ T for C C′
      proof −
        have NUN: − S ∩ − T ≠ UNIV
          using ‹T ≠ {}› by blast
        have C ≠ C′
        proof
          assume C = C′
          with that have frontier C′ ⊆ S ∩ T
            by simp
          also have ... = {}
            using ‹S ∩ T = {}› by blast
          finally have C′ = {} ∨ C′ = UNIV
            using frontier_eq_empty by auto
          then show False
         using ‹C = C′› NUN that by (force simp: dest: in_components_nonempty
in_components_subset)
        qed
        with that show ?thesis
          by (simp add: components_nonoverlap [of _ −(S ∪ T)])
      qed
      then show ?thesis
        by blast
    qed
    ultimately show ?S ∩ ?T = {}
      using ST by blast
    show ?S ≠ {} ?T ≠ {}
      using ‹S ≠ {}› ‹T ≠ {}› by blast+
  qed
    then show False
      by (metis Compl_disjoint connected_UNIV compl_bot_eq compl_unique
connected_closedD inf_sup_absorb sup_compl_top_left1 top.extremum_uniqueI)
```

qed

proposition *separation_by_component_open*:
fixes $S :: 'a :: euclidean\_space\ set$
assumes *open* S **and** *non*: $\neg\ connected(-\ S)$
obtains C **where** $C \in components\ S\ \neg\ connected(-\ C)$
proof −
obtain $T\ U$
where *closed* T *closed* U **and** TU: $T \cup U = -\ S\ T \cap U = \{\}\ T \neq \{\}\ U \neq \{\}$
using *assms* **by** (*auto simp*: *connected_closed_set closed_def*)
then obtain C **where** C: $C \in components(-(T \cup U))\ C \neq \{\}$
and *frontier* $C \cap T \neq \{\}$ *frontier* $C \cap U \neq \{\}$
using *separation_by_component_open_aux* [*OF* ‹*closed* T› ‹*closed* U› ‹$T \cap U = \{\}$›] **by** *force*
show *thesis*
proof
show $C \in components\ S$
using $C(1)$ $TU(1)$ **by** *auto*
show $\neg$ *connected* $(-\ C)$
proof
assume *connected* $(-\ C)$
then have *connected* (*frontier* C)
using *connected_frontier_simple* [*of* C] ‹$C \in components\ S$› *in_components_connected* **by** *blast*
then show *False*
unfolding *connected_closed*
by (*metis* $C(1)$ $TU(2)$ ‹*closed* T› ‹*closed* U› ‹*frontier* $C \cap T \neq \{\}$› ‹*frontier* $C \cap U \neq \{\}$› *closed_Un frontier_of_components_closed_complement inf_bot_right inf_commute*)
qed
qed
qed

lemma *separation_by_Un_open*:
fixes $S :: 'a :: euclidean\_space\ set$
assumes *open* S *open* T $S \cap T = \{\}$ **and** cS: *connected*$(-S)$ **and** cT: *connected*$(-T)$
shows *connected*$(-\ (S \cup T))$
using *assms unicoherent_UNIV* **unfolding** *unicoherent_def* **by** *force*

lemma *nonseparation_by_component_eq*:
fixes $S :: 'a :: euclidean\_space\ set$
assumes *open* $S \vee$ *closed* S
shows $((\forall\ C \in components\ S.\ connected(-C)) \longleftrightarrow connected(-\ S))$ (**is** *?lhs* = *?rhs*)
proof

```
  assume ?lhs with assms show ?rhs
   by (meson separation_by_component_closed separation_by_component_open)
next
  assume ?rhs with assms show ?lhs
    using component_complement_connected by force
qed
```

Another interesting equivalent of an inessential mapping into C-0

```
proposition inessential_eq_extensible:
  fixes f :: 'a::euclidean_space ⇒ complex
  assumes closed S
  shows (∃a. homotopic_with_canon (λh. True) S (−{0}) f (λt. a)) ⟷
         (∃g. continuous_on UNIV g ∧ (∀x ∈ S. g x = f x) ∧ (∀x. g x ≠ 0))
    (is ?lhs = ?rhs)
proof
  assume ?lhs
  then obtain a where a: homotopic_with_canon (λh. True) S (−{0}) f (λt. a)
..
  show ?rhs
  proof (cases S = {})
    case True
    with a show ?thesis by force
  next
    case False
    have anr: ANR (−{0::complex})
      by (simp add: ANR_delete open_Compl open_imp_ANR)
    obtain g where contg: continuous_on UNIV g and gim: g ‘ UNIV ⊆ −{0}
                   and gf: ⋀x. x ∈ S ⟹ g x = f x
   proof (rule Borsuk_homotopy_extension_homotopic [OF _ _ continuous_on_const
_ homotopic_with_symD [OF a]])
      show closedin (top_of_set UNIV) S
        using assms by auto
      show (λt. a) ∈ UNIV → − {0}
        using a homotopic_with_imp_subset2 False by blast
    qed (use anr that in ‹force+›)
    then show ?thesis
      by force
  qed
next
  assume ?rhs
  then obtain g where contg: continuous_on UNIV g
         and gf: ⋀x. x ∈ S ⟹ g x = f x and non0: ⋀x. g x ≠ 0
    by metis
  obtain h k::'a⇒'a where hk: homeomorphism (ball 0 1) UNIV h k
    using homeomorphic_ball01_UNIV homeomorphic_def by blast
  then have continuous_on (ball 0 1) (g ∘ h)
    by (meson contg continuous_on_compose continuous_on_subset homeomor-
phism_cont1 top_greatest)
  then obtain j where contj: continuous_on (ball 0 1) j
```

```
                and j: ⋀z. z ∈ ball 0 1 ⟹ exp(j z) = (g ∘ h) z
   by (metis (mono_tags, opaque_lifting) continuous_logarithm_on_ball comp_apply
non0)
  have [simp]: ⋀x. x ∈ S ⟹ h (k x) = x
    using hk homeomorphism_apply2 by blast
  have ∃ζ. continuous_on S ζ∧ (∀ x∈S. f x = exp (ζ x))
  proof (intro exI conjI ballI)
    show continuous_on S (j ∘ k)
    proof (rule continuous_on_compose)
      show continuous_on S k
        by (meson continuous_on_subset hk homeomorphism_cont2 top_greatest)
      show continuous_on (k ‘ S) j
          by (auto intro: continuous_on_subset [OF contj] simp flip: homeomor-
phism_image2 [OF hk])
    qed
    show f x = exp ((j ∘ k) x) if x ∈ S for x
    proof −
      have f x = (g ∘ h) (k x)
        by (simp add: gf that)
      also have ... = exp (j (k x))
        by (metis rangeI homeomorphism_image2 [OF hk] j)
      finally show ?thesis by simp
    qed
  qed
  then show ?lhs
    by (simp add: inessential_eq_continuous_logarithm)
qed

lemma inessential_on_clopen_Union:
  fixes ℱ :: 'a::euclidean_space set set
  assumes T: path_connected T
      and ⋀S. S ∈ ℱ ⟹ closedin (top_of_set (⋃ℱ)) S
      and ⋀S. S ∈ ℱ ⟹ openin (top_of_set (⋃ℱ)) S
      and hom: ⋀S. S ∈ ℱ ⟹ ∃ a. homotopic_with_canon (λx. True) S T f (λx.
a)
  obtains a where homotopic_with_canon (λx. True) (⋃ℱ) T f (λx. a)
proof (cases ⋃ℱ = {})
  case True
  with that show ?thesis
    using homotopic_with_canon_on_empty by fastforce
next
  case False
  then obtain C where C ∈ ℱ C ≠ {}
    by blast
  then obtain a where clo: closedin (top_of_set (⋃ℱ)) C
    and ope: openin (top_of_set (⋃ℱ)) C
    and homotopic_with_canon (λx. True) C T f (λx. a)
    using assms by blast
  with ‹C ≠ {}› have f ‘ C ⊆ T a ∈ T
```

using *homotopic_with_imp_subset1 homotopic_with_imp_subset2* **by** *blast+*
have *homotopic_with_canon* (λ*x*. *True*) (⋃ℱ) *T f* (λ*x*. *a*)
proof (*rule homotopic_on_clopen_Union*)
show ⋀*S*. *S* ∈ ℱ ⟹ *closedin* (*top_of_set* (⋃ℱ)) *S*
⋀*S*. *S* ∈ ℱ ⟹ *openin* (*top_of_set* (⋃ℱ)) *S*
by (*simp_all add*: *assms*)
show *homotopic_with_canon* (λ*x*. *True*) *S T f* (λ*x*. *a*) **if** *S* ∈ ℱ **for** *S*
proof (*cases S* = {})
case *False*
then obtain *b* **where** *b* ∈ *S*
by *blast*
obtain *c* **where** *c*: *homotopic_with_canon* (λ*x*. *True*) *S T f* (λ*x*. *c*)
using ‹*S* ∈ ℱ› *hom* **by** *blast*
then have *c* ∈ *T*
using ‹*b* ∈ *S*› *homotopic_with_imp_subset2* **by** *blast*
then have *homotopic_with_canon* (λ*x*. *True*) *S T* (λ*x*. *a*) (λ*x*. *c*)
using *T* ‹*a* ∈ *T*› **by** (*simp add*: *homotopic_constant_maps path_connected_component*)
then show *?thesis*
using *c homotopic_with_symD homotopic_with_trans* **by** *blast*
qed (*simp add*: *homotopic_on_empty*)
qed
then show *?thesis* **..**
qed

proposition *Janiszewski_dual*:
fixes *S* :: *complex set*
assumes
compact S compact T connected S connected T connected(− (*S* ∪ *T*))
shows *connected*(*S* ∩ *T*)
proof −
have *ST*: *compact* (*S* ∪ *T*)
by (*simp add*: *assms compact_Un*)
with *Borsukian_imp_unicoherent* [*of S* ∪ *T*] *ST assms*
show *?thesis*
by (*simp add*: *Borsukian_separation_compact closed_subset compact_imp_closed unicoherentD*)
qed

end

6.37 The Jordan Curve Theorem and Applications

theory *Jordan_Curve*
imports *Arcwise_Connected Further_Topology*
begin

6.37.1 Janiszewski's theorem

lemma *Janiszewski_weak*:

```
  fixes a b::complex
  assumes compact S compact T and conST: connected(S ∩ T)
      and ccS: connected_component (− S) a b and ccT: connected_component
(− T) a b
    shows connected_component (− (S ∪ T)) a b
proof −
  have [simp]: a ∉ S a ∉ T b ∉ S b ∉ T
    by (meson ComplD ccS ccT connected_component_in)+
  have clo: closedin (top_of_set (S ∪ T)) S closedin (top_of_set (S ∪ T)) T
    by (simp_all add: assms closed_subset compact_imp_closed)
  obtain g where contg: continuous_on S g
           and g: ⋀x. x ∈ S ⟹ exp (i* of_real (g x)) = (x − a) /R cmod (x −
a) / ((x − b) /R cmod (x − b))
    using ccS ‹compact S›
   apply (simp add: Borsuk_maps_homotopic_in_connected_component_eq [symmetric])
    apply (subst (asm) homotopic_circlemaps_divide)
    apply (auto simp: inessential_eq_continuous_logarithm_circle)
    done
  obtain h where conth: continuous_on T h
           and h: ⋀x. x ∈ T ⟹ exp (i* of_real (h x)) = (x − a) /R cmod (x −
a) / ((x − b) /R cmod (x − b))
    using ccT ‹compact T›
   apply (simp add: Borsuk_maps_homotopic_in_connected_component_eq [symmetric])
    apply (subst (asm) homotopic_circlemaps_divide)
    apply (auto simp: inessential_eq_continuous_logarithm_circle)
    done
  have continuous_on (S ∪ T) (λx. (x − a) /R cmod (x − a)) continuous_on (S
∪ T) (λx. (x − b) /R cmod (x − b))
    by (intro continuous_intros; force)+
  moreover have (λx. (x − a) /R cmod (x − a)) ‘ (S ∪ T) ⊆ sphere 0 1 (λx. (x
− b) /R cmod (x − b)) ‘ (S ∪ T) ⊆ sphere 0 1
    by (auto simp: divide_simps)
  moreover have ∃g. continuous_on (S ∪ T) g ∧
                  (∀x∈S ∪ T. (x − a) /R cmod (x − a) / ((x − b) /R cmod (x −
b)) = exp (i*complex_of_real (g x)))
  proof (cases S ∩ T = {})
    case True
    then have continuous_on (S ∪ T) (λx. if x ∈ S then g x else h x)
      using continuous_on_cases_local [OF clo contg conth]
      by (meson disjoint_iff)
    then show ?thesis
      by (rule_tac x=(λx. if x ∈ S then g x else h x) in exI) (auto simp: g h)
  next
    case False
    have diffpi: ∃n. g x = h x + 2* of_int n*pi if x ∈ S ∩ T for x
    proof −
      have exp (i* of_real (g x)) = exp (i* of_real (h x))
        using that by (simp add: g h)
      then obtain n where complex_of_real (g x) = complex_of_real (h x) + 2*
```

```
of_int n*complex_of_real pi
      apply (simp add: exp_eq)
     by (metis complex_i_not_zero distrib_left mult.commute mult_cancel_left)
    then show ?thesis
     using of_real_eq_iff by (fastforce intro!: exI [where x=n])
  qed
  have contgh: continuous_on (S ∩ T) (λx. g x − h x)
      by (intro continuous_intros continuous_on_subset [OF contg] continuous_on_subset [OF conth]) auto
  moreover have disc:
      ∃e>0. ∀y. y ∈ S ∩ T ∧ g y − h y ≠ g x − h x ⟶ e ≤ norm ((g y − h y) − (g x − h x))
       if x ∈ S ∩ T for x
  proof −
    obtain nx where nx: g x = h x + 2* of_int nx*pi
     using ‹x ∈ S ∩ T› diffpi by blast
    have 2*pi ≤ norm (g y − h y − (g x − h x)) if y: y ∈ S ∩ T and neq: g y − h y ≠ g x − h x for y
    proof −
     obtain ny where ny: g y = h y + 2* of_int ny*pi
      using ‹y ∈ S ∩ T› diffpi by blast
     { assume nx ≠ ny
      then have 1 ≤ |real_of_int ny − real_of_int nx|
       by linarith
      then have (2*pi)*1 ≤ (2*pi)*|real_of_int ny − real_of_int nx|
       by simp
      also have ... = |2*real_of_int ny*pi − 2*real_of_int nx*pi|
       by (simp add: algebra_simps abs_if)
       finally have 2*pi ≤ |2*real_of_int ny*pi − 2*real_of_int nx*pi| by simp
     }
     with neq show ?thesis
      by (simp add: nx ny)
    qed
    then show ?thesis
     by (rule_tac x=2*pi in exI) auto
  qed
  ultimately have (λx. g x − h x) constant_on S ∩ T
   using continuous_discrete_range_constant [OF conST contgh] by blast
  then obtain z where z: ⋀x. x ∈ S ∩ T ⟹ g x − h x = z
   by (auto simp: constant_on_def)
  obtain w where exp(i * of_real(h w)) = exp (i * of_real(z + h w))
   using disc z False
   by auto (metis diff_add_cancel g h of_real_add)
  then have [simp]: exp (i* of_real z) = 1
   by (metis cis_conv_exp cis_mult exp_not_eq_zero mult_cancel_right1)
  show ?thesis
  proof (intro exI conjI)
   show continuous_on (S ∪ T) (λx. if x ∈ S then g x else z + h x)
```

```
        by (intro continuous_intros continuous_on_cases_local [OF clo contg]
conth) (use z in force)
    qed (auto simp: g h algebra_simps exp_add)
  qed
  ultimately have homotopic_with_canon (λx. True) (S ∪ T) (sphere 0 1)
                  (λx. (x − a) /R cmod (x − a)) (λx. (x − b) /R cmod (x − b))
   by (subst homotopic_circlemaps_divide) (auto simp: inessential_eq_continuous_logarithm_circle)
  moreover have compact (S ∪ T)
    using assms by blast
  ultimately show ?thesis
    using assms Borsuk_maps_homotopic_in_connected_component_eq by fast-
force
qed


theorem Janiszewski:
  fixes a b :: complex
  assumes compact S closed T and conST: connected (S ∩ T)
      and ccS: connected_component (− S) a b and ccT: connected_component
(− T) a b
    shows connected_component (− (S ∪ T)) a b
proof −
  have path_component(− T) a b
    by (simp add: ‹closed T› ccT open_Compl open_path_connected_component)
  then obtain g where g: path g path_image g ⊆ − T pathstart g = a pathfinish
g = b
    by (auto simp: path_component_def)
  then obtain C where C: compact C connected C a ∈ C b ∈ C C ∩ T = {}
    by fastforce
  obtain r where 0 < r and r: C ∪ S ⊆ ball 0 r
   by (metis ‹compact C› ‹compact S› bounded_Un compact_imp_bounded bounded_subset_ballD)
  have connected_component (− (S ∪ (T ∩ cball 0 r ∪ sphere 0 r))) a b
  proof (rule Janiszewski_weak [OF ‹compact S›])
    show comT': compact ((T ∩ cball 0 r) ∪ sphere 0 r)
      by (simp add: ‹closed T› closed_Int_compact compact_Un)
    have S ∩ (T ∩ cball 0 r ∪ sphere 0 r) = S ∩ T
      using r by auto
    with conST show connected (S ∩ (T ∩ cball 0 r ∪ sphere 0 r))
      by simp
    show connected_component (− (T ∩ cball 0 r ∪ sphere 0 r)) a b
      using conST C r
      apply (simp add: connected_component_def)
      apply (rule_tac x=C in exI)
      by auto
  qed (simp add: ccS)
  then obtain U where U: connected U U ⊆ − S U ⊆ − T ∪ − cball 0 r U ⊆
− sphere 0 r a ∈ U b ∈ U
    by (auto simp: connected_component_def)
  show ?thesis
```

unfolding *connected_component_def*
proof (*intro exI conjI*)
show $U \subseteq - (S \cup T)$
using *U r* ‹*0 < r*› ‹*a ∈ C*› *connected_Int_frontier* [*of U cball 0 r*]
apply *simp*
by (*metis ball_subset_cball compl_inf disjoint_eq_subset_Compl disjoint_iff_not_equal inf.orderE inf_sup_aci*(*3*) *subsetCE*)
qed (*auto simp*: *U*)
qed

lemma *Janiszewski_connected*:
fixes *S* :: *complex set*
assumes *ST*: *compact S closed T connected*($S \cap T$)
and *notST*: *connected* ($- S$) *connected* ($- T$)
shows *connected*($- (S \cup T)$)
using *Janiszewski* [*OF ST*] **by** (*metis IntD1 IntD2 notST compl_sup connected_iff_connected_compone*

6.37.2 The Jordan Curve theorem

lemma *exists_double_arc*:
fixes *g* :: *real* ⇒ *'a*::*real_normed_vector*
assumes *simple_path g pathfinish g* = *pathstart g a* ∈ *path_image g b* ∈ *path_image g a* ≠ *b*
obtains *u d* **where** *arc u arc d pathstart u* = *a pathfinish u* = *b*
pathstart d = *b pathfinish d* = *a*
(*path_image u*) ∩ (*path_image d*) = {*a*,*b*}
(*path_image u*) ∪ (*path_image d*) = *path_image g*
proof −
obtain *u* **where** *u*: *0* ≤ *u u* ≤ *1 g u* = *a*
using *assms* **by** (*auto simp*: *path_image_def*)
define *h* **where** *h* ≡ *shiftpath u g*
have *simple_path h*
using ‹*simple_path g*› *simple_path_shiftpath* ‹*0* ≤ *u*› ‹*u* ≤ *1*› *assms*(*2*) *h_def* **by** *blast*
have *pathstart h* = *g u*
by (*simp add*: ‹*u* ≤ *1*› *h_def pathstart_shiftpath*)
have *pathfinish h* = *g u*
by (*simp add*: ‹*0* ≤ *u*› *assms h_def pathfinish_shiftpath*)
have *pihg*: *path_image h* = *path_image g*
by (*simp add*: ‹*0* ≤ *u*› ‹*u* ≤ *1*› *assms h_def path_image_shiftpath*)
then obtain *v* **where** *v*: *0* ≤ *v v* ≤ *1 h v* = *b*
using *assms* **by** (*metis* (*mono_tags*, *lifting*) *atLeastAtMost_iff imageE path_image_def*)
show *?thesis*
proof
show *arc* (*subpath 0 v h*)
by (*metis* (*no_types*) ‹*pathstart h* = *g u*› ‹*simple_path h*› *arc_simple_path_subpath* ‹*a ≠ b*› *atLeastAtMost_iff zero_le_one order_refl pathstart_def u*(*3*) *v*)
show *arc* (*subpath v 1 h*)
by (*metis* (*no_types*) ‹*pathfinish h* = *g u*› ‹*simple_path h*› *arc_simple_path_subpath*

```
‹a ≠ b› atLeastAtMost_iff zero_le_one order_refl pathfinish_def u(3) v)
    show pathstart (subpath 0 v h) = a
      by (metis ‹pathstart h = g u› pathstart_def pathstart_subpath u(3))
    show pathfinish (subpath 0 v h) = b  pathstart (subpath v 1 h) = b
      by (simp_all add: v(3))
    show pathfinish (subpath v 1 h) = a
      by (metis ‹pathfinish h = g u› pathfinish_def pathfinish_subpath u(3))
    show path_image (subpath 0 v h) ∩ path_image (subpath v 1 h) = {a, b}
    proof
      have loop_free h
        using ‹simple_path h› simple_path_def by blast
      then show path_image (subpath 0 v h) ∩ path_image (subpath v 1 h) ⊆ {a,
b}
        using v ‹pathfinish (subpath v 1 h) = a›
        apply (clarsimp simp add: loop_free_def path_image_subpath Ball_def)
        by (smt (verit))
      show {a, b} ⊆ path_image (subpath 0 v h) ∩ path_image (subpath v 1 h)
        using v ‹pathstart (subpath 0 v h) = a› ‹pathfinish (subpath v 1 h) = a›
        by (auto simp: path_image_subpath image_iff Bex_def)
    qed
    show path_image (subpath 0 v h) ∪ path_image (subpath v 1 h) = path_image
g
      using v path_image_subpath pihg path_image_def
      by (metis (full_types) image_Un ivl_disj_un_two_touch(4))
  qed
qed


theorem Jordan_curve:
  fixes c :: real ⇒ complex
  assumes simple_path c and loop: pathfinish c = pathstart c
  obtains inner outer where
              inner ≠ {} open inner connected inner
              outer ≠ {} open outer connected outer
              bounded inner ¬ bounded outer inner ∩ outer = {}
              inner ∪ outer = − path_image c
              frontier inner = path_image c
              frontier outer = path_image c
proof −
  have path c
    by (simp add: assms simple_path_imp_path)
  have hom: (path_image c) homeomorphic (sphere(0::complex) 1)
    by (simp add: assms homeomorphic_simple_path_image_circle)
  with Jordan_Brouwer_separation have ¬ connected (− (path_image c))
    by fastforce
  then obtain inner where inner: inner ∈ components (− path_image c) and
bounded inner
    using cobounded_has_bounded_component [of − (path_image c)]
   using ‹¬ connected (− path_image c)› ‹simple_path c› bounded_simple_path_image
```

by *force*
obtain *outer* **where** *outer*: *outer* ∈ *components* (− *path_image c*) **and** ¬ *bounded outer*
using *cobounded_unbounded_components* [*of* − (*path_image c*)]
using ‹*path c*› *bounded_path_image* **by** *auto*
show *?thesis*
proof
show *inner* ≠ {}
using *inner in_components_nonempty* **by** *auto*
show *open inner*
by (*meson* ‹*simple_path c*› *compact_imp_closed compact_simple_path_image inner open_Compl open_components*)
show *connected inner*
using *in_components_connected inner* **by** *blast*
show *outer* ≠ {}
using *outer in_components_nonempty* **by** *auto*
show *open outer*
by (*meson* ‹*simple_path c*› *compact_imp_closed compact_simple_path_image outer open_Compl open_components*)
show *connected outer*
using *in_components_connected outer* **by** *blast*
show *inner_outer*: *inner* ∩ *outer* = {}
by (*meson* ‹¬ *bounded outer*› ‹*bounded inner*› ‹*connected outer*› *bounded_subset components_maximal in_components_subset inner outer*)
show *fro_inner*: *frontier inner* = *path_image c*
by (*simp add*: *Jordan_Brouwer_frontier* [*OF hom inner*])
show *fro_outer*: *frontier outer* = *path_image c*
by (*simp add*: *Jordan_Brouwer_frontier* [*OF hom outer*])
have *False* **if** *m*: *middle* ∈ *components* (− *path_image c*) **and** *middle* ≠ *inner middle* ≠ *outer* **for** *middle*
proof −
have *frontier middle* = *path_image c*
by (*simp add*: *Jordan_Brouwer_frontier* [*OF hom*] *that*)
obtain *middle*: *open middle connected middle middle* ≠ {}
by (*metis fro_inner frontier_closed in_components_maximal m open_Compl open_components*)
obtain *a0 b0* **where** *a0* ∈ *path_image c b0* ∈ *path_image c a0* ≠ *b0*
using *simple_path_image_uncountable* [*OF* ‹*simple_path c*›]
by (*metis Diff_cancel countable_Diff_eq countable_empty insert_iff subsetI subset_singleton_iff*)
obtain *a b g* **where** *ab*: *a* ∈ *path_image c b* ∈ *path_image c a* ≠ *b*
and *g*: *arc g pathstart g* = *a pathfinish g* = *b*
and *pag_sub*: *path_image g* − {*a*,*b*} ⊆ *middle*
proof (*rule dense_accessible_frontier_point_pairs* [*OF* ‹*open middle*› ‹*connected middle*›, *of path_image c* ∩ *ball a0* (*dist a0 b0*) *path_image c* ∩ *ball b0* (*dist a0 b0*)])
show *openin* (*top_of_set* (*frontier middle*)) (*path_image c* ∩ *ball a0* (*dist a0 b0*))
openin (*top_of_set* (*frontier middle*)) (*path_image c* ∩ *ball b0* (*dist a0*

```
b0))
        by (simp_all add: ‹frontier middle = path_image c› openin_open_Int)
      show path_image c ∩ ball a0 (dist a0 b0) ≠ path_image c ∩ ball b0 (dist
a0 b0)
        using ‹a0 ≠ b0› ‹b0 ∈ path_image c› by auto
      show path_image c ∩ ball a0 (dist a0 b0) ≠ {}
        using ‹a0 ∈ path_image c› ‹a0 ≠ b0› by auto
      show path_image c ∩ ball b0 (dist a0 b0) ≠ {}
        using ‹b0 ∈ path_image c› ‹a0 ≠ b0› by auto
    qed (use arc_distinct_ends arc_imp_simple_path simple_path_endless that
in fastforce)
    obtain u d where arc u arc d
                 and pathstart u = a pathfinish u = b pathstart d = b pathfinish d
= a
                 and ud_ab: (path_image u) ∩ (path_image d) = {a,b}
                 and ud_Un: (path_image u) ∪ (path_image d) = path_image c
      using exists_double_arc [OF assms ab] by blast
    obtain x y where x ∈ inner y ∈ outer
      using ‹inner ≠ {}› ‹outer ≠ {}› by auto
    have inner ∩ middle = {} middle ∩ outer = {}
      using components_nonoverlap inner outer m that by blast+
   have connected_component (− (path_image u ∪ path_image g ∪ (path_image
d ∪ path_image g))) x y
    proof (rule Janiszewski)
      show compact (path_image u ∪ path_image g)
        by (simp add: ‹arc g› ‹arc u› compact_Un compact_arc_image)
      show closed (path_image d ∪ path_image g)
        by (simp add: ‹arc d› ‹arc g› closed_Un closed_arc_image)
       show connected ((path_image u ∪ path_image g) ∩ (path_image d ∪
path_image g))
        using ud_ab
        by (metis Un_insert_left g connected_arc_image insert_absorb pathfin-
ish_in_path_image pathstart_in_path_image sup_bot_left sup_commute sup_inf_distrib1)
      show connected_component (− (path_image u ∪ path_image g)) x y
        unfolding connected_component_def
      proof (intro exI conjI)
        have connected ((inner ∪ (path_image c − path_image u)) ∪ (outer ∪
(path_image c − path_image u)))
        proof (rule connected_Un)
          show connected (inner ∪ (path_image c − path_image u))
            using connected_intermediate_closure [OF ‹connected inner›]
            by (metis Diff_subset closure_Un_frontier dual_order.refl fro_inner
sup.mono sup_ge1)
          show connected (outer ∪ (path_image c − path_image u))
            using connected_intermediate_closure [OF ‹connected outer›]
          by (simp add: Diff_eq closure_Un_frontier fro_outer sup_inf_distrib1)
          have (inner ∩ outer) ∪ (path_image c − path_image u) ≠ {}
          using ‹arc d› ‹pathfinish d = a› ‹pathstart d = b› arc_imp_simple_path
nonempty_simple_path_endless ud_Un ud_ab by fastforce
```

```
            then show (inner ∪ (path_image c − path_image u)) ∩ (outer ∪
(path_image c − path_image u)) ≠ {}
              by auto
          qed
          then show connected (inner ∪ outer ∪ (path_image c − path_image u))
            by (metis sup.right_idem sup_assoc sup_commute)
          have inner ⊆ − path_image u outer ⊆ − path_image u
            using in_components_subset inner outer ud_Un by auto
          moreover have inner ⊆ − path_image g outer ⊆ − path_image g
            using ‹inner ∩ middle = {}› ‹inner ⊆ − path_image u›
           using ‹middle ∩ outer = {}› ‹outer ⊆ − path_image u› pag_sub ud_ab
by fastforce+
          moreover have path_image c − path_image u ⊆ − path_image g
            using in_components_subset m pag_sub ud_ab by fastforce
          ultimately show inner ∪ outer ∪ (path_image c − path_image u) ⊆ −
(path_image u ∪ path_image g)
            by force
          show x ∈ inner ∪ outer ∪ (path_image c − path_image u)
            by (auto simp: ‹x ∈ inner›)
          show y ∈ inner ∪ outer ∪ (path_image c − path_image u)
            by (auto simp: ‹y ∈ outer›)
        qed
        show connected_component (− (path_image d ∪ path_image g)) x y
          unfolding connected_component_def
        proof (intro exI conjI)
          have connected ((inner ∪ (path_image c − path_image d)) ∪ (outer ∪
(path_image c − path_image d)))
          proof (rule connected_Un)
            show connected (inner ∪ (path_image c − path_image d))
            using connected_intermediate_closure [OF ‹connected inner›] fro_inner
              by (simp add: closure_Un_frontier sup.coboundedI2)
            show connected (outer ∪ (path_image c − path_image d))
              using connected_intermediate_closure [OF ‹connected outer›]
              by (simp add: closure_Un_frontier fro_outer sup.coboundedI2)
            have (inner ∩ outer) ∪ (path_image c − path_image d) ≠ {}
            using ‹arc u› ‹pathfinish u = b› ‹pathstart u = a› arc_imp_simple_path
nonempty_simple_path_endless ud_Un ud_ab by fastforce
            then show (inner ∪ (path_image c − path_image d)) ∩ (outer ∪
(path_image c − path_image d)) ≠ {}
              by auto
          qed
          then show connected (inner ∪ outer ∪ (path_image c − path_image d))
            by (metis sup.right_idem sup_assoc sup_commute)
          have inner ⊆ − path_image d outer ⊆ − path_image d
            using in_components_subset inner outer ud_Un by auto
          moreover have inner ⊆ − path_image g outer ⊆ − path_image g
            using ‹inner ∩ middle = {}› ‹inner ⊆ − path_image d›
           using ‹middle ∩ outer = {}› ‹outer ⊆ − path_image d› pag_sub ud_ab
by fastforce+
```

```
        moreover have path_image c − path_image d ⊆ − path_image g
          using in_components_subset m pag_sub ud_ab by fastforce
        ultimately show inner ∪ outer ∪ (path_image c − path_image d) ⊆ −
(path_image d ∪ path_image g)
          by force
        show x ∈ inner ∪ outer ∪ (path_image c − path_image d)
          by (auto simp: ‹x ∈ inner›)
        show y ∈ inner ∪ outer ∪ (path_image c − path_image d)
          by (auto simp: ‹y ∈ outer›)
      qed
    qed
      then have connected_component (− (path_image u ∪ path_image d ∪
path_image g)) x y
      by (simp add: Un_ac)
    moreover have ¬(connected_component (− (path_image c)) x y)
     by (metis (no_types, lifting) ‹¬ bounded outer› ‹bounded inner› ‹x ∈ inner›
‹y ∈ outer› componentsE connected_component_eq inner mem_Collect_eq outer)
    ultimately show False
      by (auto simp: ud_Un [symmetric] connected_component_def)
  qed
  then have components (− path_image c) = {inner,outer}
    using inner outer by blast
  then have Union (components (− path_image c)) = inner ∪ outer
    by simp
  then show inner ∪ outer = − path_image c
    by auto
 qed (auto simp: ‹bounded inner› ‹¬ bounded outer›)
qed


corollary Jordan_disconnected:
 fixes c :: real ⇒ complex
 assumes simple_path c pathfinish c = pathstart c
 shows ¬ connected(− path_image c)
 using Jordan_curve [OF assms]
 by (metis Jordan_Brouwer_separation assms homeomorphic_simple_path_image_circle
zero_less_one)


corollary Jordan_inside_outside:
 fixes c :: real ⇒ complex
 assumes simple_path c pathfinish c = pathstart c
   shows inside(path_image c) ≠ {} ∧
         open(inside(path_image c)) ∧
         connected(inside(path_image c)) ∧
         outside(path_image c) ≠ {} ∧
         open(outside(path_image c)) ∧
         connected(outside(path_image c)) ∧
         bounded(inside(path_image c)) ∧
```

```
        ¬ bounded(outside(path_image c)) ∧
        inside(path_image c) ∩ outside(path_image c) = {} ∧
        inside(path_image c) ∪ outside(path_image c) =
        − path_image c ∧
        frontier(inside(path_image c)) = path_image c ∧
        frontier(outside(path_image c)) = path_image c
proof −
  obtain inner outer
    where ∗: inner ≠ {} open inner connected inner
          outer ≠ {} open outer connected outer
          bounded inner ¬ bounded outer inner ∩ outer = {}
          inner ∪ outer = − path_image c
          frontier inner = path_image c
          frontier outer = path_image c
    using Jordan_curve [OF assms] by blast
  then have inner: inside(path_image c) = inner
   by (metis dual_order.antisym inside_subset interior_eq interior_inside_frontier)
  have outer: outside(path_image c) = outer
    using ‹inner ∪ outer = − path_image c› ‹inside (path_image c) = inner›
        outside_inside ‹inner ∩ outer = {}› by auto
  show ?thesis
    using ∗ by (auto simp: inner outer)
qed
```

Triple-curve or "theta-curve" theorem

Proof that there is no fourth component taken from Kuratowski's Topology vol 2, para 61, II.

```
theorem split_inside_simple_closed_curve:
  fixes c :: real ⇒ complex
  assumes simple_path c1 and c1: pathstart c1 = a pathfinish c1 = b
      and simple_path c2 and c2: pathstart c2 = a pathfinish c2 = b
      and simple_path c  and c: pathstart c = a pathfinish c = b
      and a ≠ b
      and c1c2: path_image c1 ∩ path_image c2 = {a,b}
      and c1c: path_image c1 ∩ path_image c = {a,b}
      and c2c: path_image c2 ∩ path_image c = {a,b}
      and ne_12: path_image c ∩ inside(path_image c1 ∪ path_image c2) ≠ {}
   obtains inside(path_image c1 ∪ path_image c) ∩ inside(path_image c2 ∪
path_image c) = {}
             inside(path_image c1 ∪ path_image c) ∪ inside(path_image c2 ∪
path_image c) ∪
          (path_image c − {a,b}) = inside(path_image c1 ∪ path_image c2)
proof −
  let ?Θ = path_image c  let ?Θ1 = path_image c1  let ?Θ2 = path_image c2
   have sp: simple_path (c1 +++ reversepath c2) simple_path (c1 +++ re-
versepath c) simple_path (c2 +++ reversepath c)
    using assms by (auto simp: simple_path_join_loop_eq arc_simple_path sim-
ple_path_reversepath)
```

then have *op_in12*: *open* (*inside* (*?Θ1* ∪ *?Θ2*))
and *op_out12*: *open* (*outside* (*?Θ1* ∪ *?Θ2*))
and *op_in1c*: *open* (*inside* (*?Θ1* ∪ *?Θ*))
and *op_in2c*: *open* (*inside* (*?Θ2* ∪ *?Θ*))
and *op_out1c*: *open* (*outside* (*?Θ1* ∪ *?Θ*))
and *op_out2c*: *open* (*outside* (*?Θ2* ∪ *?Θ*))
and *co_in1c*: *connected* (*inside* (*?Θ1* ∪ *?Θ*))
and *co_in2c*: *connected* (*inside* (*?Θ2* ∪ *?Θ*))
and *co_out12c*: *connected* (*outside* (*?Θ1* ∪ *?Θ2*))
and *co_out1c*: *connected* (*outside* (*?Θ1* ∪ *?Θ*))
and *co_out2c*: *connected* (*outside* (*?Θ2* ∪ *?Θ*))
and *pa_c*: *?Θ* − {*pathstart c*, *pathfinish c*} ⊆ − *?Θ1*
?Θ − {*pathstart c*, *pathfinish c*} ⊆ − *?Θ2*
and *pa_c1*: *?Θ1* − {*pathstart c1*, *pathfinish c1*} ⊆ − *?Θ2*
?Θ1 − {*pathstart c1*, *pathfinish c1*} ⊆ − *?Θ*
and *pa_c2*: *?Θ2* − {*pathstart c2*, *pathfinish c2*} ⊆ − *?Θ1*
?Θ2 − {*pathstart c2*, *pathfinish c2*} ⊆ − *?Θ*
and *co_c*: *connected*(*?Θ* − {*pathstart c*,*pathfinish c*})
and *co_c1*: *connected*(*?Θ1* − {*pathstart c1*,*pathfinish c1*})
and *co_c2*: *connected*(*?Θ2* − {*pathstart c2*,*pathfinish c2*})
and *fr_in*: *frontier*(*inside*(*?Θ1* ∪ *?Θ2*)) = *?Θ1* ∪ *?Θ2*
frontier(*inside*(*?Θ2* ∪ *?Θ*)) = *?Θ2* ∪ *?Θ*
frontier(*inside*(*?Θ1* ∪ *?Θ*)) = *?Θ1* ∪ *?Θ*
and *fr_out*: *frontier*(*outside*(*?Θ1* ∪ *?Θ2*)) = *?Θ1* ∪ *?Θ2*
frontier(*outside*(*?Θ2* ∪ *?Θ*)) = *?Θ2* ∪ *?Θ*
frontier(*outside*(*?Θ1* ∪ *?Θ*)) = *?Θ1* ∪ *?Θ*
using *Jordan_inside_outside* [*of c1 +++ reversepath c2*]
using *Jordan_inside_outside* [*of c1 +++ reversepath c*]
using *Jordan_inside_outside* [*of c2 +++ reversepath c*] *assms*
apply (*simp_all add*: *path_image_join closed_Un closed_simple_path_image open_inside open_outside*)
apply (*blast* | *metis connected_simple_path_endless*)+
done
have *inout_12*: *inside* (*?Θ1* ∪ *?Θ2*) ∩ (*?Θ* − {*pathstart c*, *pathfinish c*}) ≠ {}
by (*metis* (*no_types*, *lifting*) *c c1c ne_12 Diff_Int_distrib Diff_empty Int_empty_right Int_left_commute inf_sup_absorb inf_sup_aci*(1) *inside_no_overlap*)
have *pi_disjoint*: *?Θ* ∩ *outside*(*?Θ1* ∪ *?Θ2*) = {}
proof (*rule ccontr*)
assume *?Θ* ∩ *outside* (*?Θ1* ∪ *?Θ2*) ≠ {}
then show *False*
using *connectedD* [*OF co_c*, *of inside*(*?Θ1* ∪ *?Θ2*) *outside*(*?Θ1* ∪ *?Θ2*)]
using *c c1c2 pa_c op_in12 op_out12 inout_12*
apply *clarsimp*
by (*smt* (*verit*, *ccfv_threshold*) *Diff_Int_distrib Diff_cancel Diff_empty Int_assoc inf_sup_absorb inf_sup_aci*(1) *outside_no_overlap*)
qed
have *out_sub12*: *outside*(*?Θ1* ∪ *?Θ2*) ⊆ *outside*(*?Θ1* ∪ *?Θ*) *outside*(*?Θ1* ∪ *?Θ2*) ⊆ *outside*(*?Θ2* ∪ *?Θ*)
by (*metis Un_commute pi_disjoint outside_Un_outside_Un*)+

have *pa1_disj_in2*: *?Θ1* ∩ *inside* (*?Θ2* ∪ *?Θ*) = {}
proof (*rule ccontr*)
assume *ne*: *?Θ1* ∩ *inside* (*?Θ2* ∪ *?Θ*) ≠ {}
have *1*: *inside* (*?Θ* ∪ *?Θ2*) ∩ *?Θ* = {}
by (*metis* (*no_types*) *Diff_Int_distrib Diff_cancel inf_sup_absorb inf_sup_aci*(*3*) *inside_no_overlap*)
have *2*: *outside* (*?Θ* ∪ *?Θ2*) ∩ *?Θ* = {}
by (*metis* (*no_types*) *Int_empty_right Int_left_commute inf_sup_absorb outside_no_overlap*)
have *path_image c1* ∩ *outside* (*path_image c2* ∪ *path_image c*) = {}
using *connectedD* [*OF co_c1*, *of inside*(*?Θ2* ∪ *?Θ*) *outside*(*?Θ2* ∪ *?Θ*)]
pa_c1 op_in2c op_out2c ne c1 c2c 1 2 **by** (*auto simp*: *inf_sup_aci*)
then have *outside* (*?Θ2* ∪ *?Θ*) ⊆ *outside* (*?Θ1* ∪ *?Θ2*)
by (*metis outside_Un_outside_Un sup_commute*)
with *out_sub12*
have *outside*(*?Θ1* ∪ *?Θ2*) = *outside*(*?Θ2* ∪ *?Θ*) **by** *blast*
then have *frontier*(*outside*(*?Θ1* ∪ *?Θ2*)) = *frontier*(*outside*(*?Θ2* ∪ *?Θ*))
by *simp*
then show *False*
using *inout_12 pi_disjoint c c1c c2c fr_out* **by** *auto*
qed
have *pa2_disj_in1*: *?Θ2* ∩ *inside*(*?Θ1* ∪ *?Θ*) = {}
proof (*rule ccontr*)
assume *ne*: *?Θ2* ∩ *inside* (*?Θ1* ∪ *?Θ*) ≠ {}
have *1*: *inside* (*?Θ* ∪ *?Θ1*) ∩ *?Θ* = {}
by (*metis* (*no_types*) *Diff_Int_distrib Diff_cancel inf_sup_absorb inf_sup_aci*(*3*) *inside_no_overlap*)
have *2*: *outside* (*?Θ* ∪ *?Θ1*) ∩ *?Θ* = {}
by (*metis* (*no_types*) *Int_empty_right Int_left_commute inf_sup_absorb outside_no_overlap*)
have *outside* (*?Θ1* ∪ *?Θ*) ⊆ *outside* (*?Θ1* ∪ *?Θ2*)
apply (*rule outside_Un_outside_Un*)
using *connectedD* [*OF co_c2*, *of inside*(*?Θ1* ∪ *?Θ*) *outside*(*?Θ1* ∪ *?Θ*)]
pa_c2 op_in1c op_out1c ne c2 c1c 1 2 **by** (*auto simp*: *inf_sup_aci*)
with *out_sub12*
have *outside*(*?Θ1* ∪ *?Θ2*) = *outside*(*?Θ1* ∪ *?Θ*)
by *blast*
then have *frontier*(*outside*(*?Θ1* ∪ *?Θ2*)) = *frontier*(*outside*(*?Θ1* ∪ *?Θ*))
by *simp*
then show *False*
using *inout_12 pi_disjoint c c1c c2c fr_out* **by** *auto*
qed
have *in_sub_in1*: *inside*(*?Θ1* ∪ *?Θ*) ⊆ *inside*(*?Θ1* ∪ *?Θ2*)
using *pa2_disj_in1 out_sub12* **by** (*auto simp*: *inside_outside*)
have *in_sub_in2*: *inside*(*?Θ2* ∪ *?Θ*) ⊆ *inside*(*?Θ1* ∪ *?Θ2*)
using *pa1_disj_in2 out_sub12* **by** (*auto simp*: *inside_outside*)
have *in_sub_out12*: *inside*(*?Θ1* ∪ *?Θ*) ⊆ *outside*(*?Θ2* ∪ *?Θ*)
proof
fix *x*

```
    assume x: x ∈ inside (?Θ1 ∪ ?Θ)
    then have xnot: x ∉ ?Θ
      by (simp add: inside_def)
    obtain z where zim: z ∈ ?Θ1 and zout: z ∈ outside(?Θ2 ∪ ?Θ)
      unfolding outside_inside
       using nonempty_simple_path_endless [OF ‹simple_path c1›] c1 c1c c1c2
pa1_disj_in2 by auto
    obtain e where e > 0 and e: ball z e ⊆ outside(?Θ2 ∪ ?Θ)
      using zout op_out2c open_contains_ball_eq by blast
    have z ∈ frontier (inside (?Θ1 ∪ ?Θ))
      using zim by (auto simp: fr_in)
    then obtain w where w1: w ∈ inside (?Θ1 ∪ ?Θ) and dwz: dist w z < e
      using zim ‹e > 0› by (auto simp: frontier_def closure_approachable)
    then have w2: w ∈ outside (?Θ2 ∪ ?Θ)
      by (metis e dist_commute mem_ball subsetCE)
    then have connected_component (− ?Θ2 ∩ − ?Θ) z w
      unfolding connected_component_def
      by (metis co_out2c compl_sup inside_Un_outside sup_ge2 zout)
    moreover have connected_component (− ?Θ2 ∩ − ?Θ) w x
      unfolding connected_component_def
      using pa2_disj_in1 co_in1c x w1 union_with_outside by fastforce
    ultimately have eq: connected_component_set (− ?Θ2 ∩ − ?Θ) x =
                        connected_component_set (− ?Θ2 ∩ − ?Θ) z
      by (metis (mono_tags, lifting) connected_component_eq mem_Collect_eq)
    show x ∈ outside (?Θ2 ∪ ?Θ)
      using zout x pa2_disj_in1 by (auto simp: outside_def eq xnot)
  qed
  have in_sub_out21: inside(?Θ2 ∪ ?Θ) ⊆ outside(?Θ1 ∪ ?Θ)
  proof
    fix x
    assume x: x ∈ inside (?Θ2 ∪ ?Θ)
    then have xnot: x ∉ ?Θ
      by (simp add: inside_def)
    obtain z where zim: z ∈ ?Θ2 and zout: z ∈ outside(?Θ1 ∪ ?Θ)
      unfolding outside_inside
       using nonempty_simple_path_endless [OF ‹simple_path c2›] c1c2 c2 c2c
pa2_disj_in1 by auto
    obtain e where e > 0 and e: ball z e ⊆ outside(?Θ1 ∪ ?Θ)
      using zout op_out1c open_contains_ball_eq by blast
    have z ∈ frontier (inside (?Θ2 ∪ ?Θ))
      using zim by (auto simp: fr_in)
    then obtain w where w2: w ∈ inside (?Θ2 ∪ ?Θ) and dwz: dist w z < e
      using zim ‹e > 0› by (auto simp: frontier_def closure_approachable)
    then have w1: w ∈ outside (?Θ1 ∪ ?Θ)
      by (metis e dist_commute mem_ball subsetCE)
    then have connected_component (− ?Θ1 ∩ − ?Θ) z w
      unfolding connected_component_def
      by (metis Compl_Un co_out1c inside_Un_outside sup_ge2 zout)
    moreover have connected_component (− ?Θ1 ∩ − ?Θ) w x
```

```
    unfolding connected_component_def
    using pa1_disj_in2 co_in2c x w2 union_with_outside by fastforce
  ultimately have eq: connected_component_set (− ?Θ1 ∩ − ?Θ) x =
                      connected_component_set (− ?Θ1 ∩ − ?Θ) z
    by (metis (no_types, lifting) connected_component_eq mem_Collect_eq)
  show x ∈ outside (?Θ1 ∪ ?Θ)
    using zout x pa1_disj_in2 by (auto simp: outside_def eq xnot)
qed
show ?thesis
proof
  show inside (?Θ1 ∪ ?Θ) ∩ inside (?Θ2 ∪ ?Θ) = {}
  by (metis Int_Un_distrib in_sub_out12 bot_eq_sup_iff disjoint_eq_subset_Compl
outside_inside)
  have ∗: outside (?Θ1 ∪ ?Θ) ∩ outside (?Θ2 ∪ ?Θ) ⊆ outside (?Θ1 ∪ ?Θ2)
  proof (rule components_maximal)
    show out_in: outside (?Θ1 ∪ ?Θ2) ∈ components (− (?Θ1 ∪ ?Θ2))
      unfolding outside_in_components co_out12c
      using co_out12c fr_out(1) by force
    have conn_U: connected (− (closure (inside (?Θ1 ∪ ?Θ)) ∪ closure (inside
(?Θ2 ∪ ?Θ))))
    proof (rule Janiszewski_connected, simp_all)
      show bounded (inside (?Θ1 ∪ ?Θ))
     by (simp add: ‹simple_path c1› ‹simple_path c› bounded_inside bounded_simple_path_image)
      have if1: − (inside (?Θ1 ∪ ?Θ) ∪ frontier (inside (?Θ1 ∪ ?Θ))) = − ?Θ1
∩ − ?Θ ∩ − inside (?Θ1 ∪ ?Θ)
        by (metis (no_types, lifting) Int_commute Jordan_inside_outside c
c1 compl_sup path_image_join path_image_reversepath pathfinish_join pathfin-
ish_reversepath pathstart_join pathstart_reversepath sp(2) closure_Un_frontier
fr_out(3))
      then show connected (− closure (inside (?Θ1 ∪ ?Θ)))
        by (metis Compl_Un outside_inside co_out1c closure_Un_frontier)
      have if2: − (inside (?Θ2 ∪ ?Θ) ∪ frontier (inside (?Θ2 ∪ ?Θ))) = − ?Θ2
∩ − ?Θ ∩ − inside (?Θ2 ∪ ?Θ)
        by (metis (no_types, lifting) Int_commute Jordan_inside_outside c
c2 compl_sup path_image_join path_image_reversepath pathfinish_join pathfin-
ish_reversepath pathstart_join pathstart_reversepath sp(3) closure_Un_frontier
fr_out(2))
      then show connected (− closure (inside (?Θ2 ∪ ?Θ)))
        by (metis Compl_Un outside_inside co_out2c closure_Un_frontier)
      have connected(?Θ)
        by (metis ‹simple_path c› connected_simple_path_image)
      moreover
      have closure (inside (?Θ1 ∪ ?Θ)) ∩ closure (inside (?Θ2 ∪ ?Θ)) = ?Θ
        (is ?lhs = ?rhs)
      proof
        show ?lhs ⊆ ?rhs
        proof clarify
          fix x
          assume x: x ∈ closure (inside (?Θ1 ∪ ?Θ)) x ∈ closure (inside (?Θ2 ∪
```

```
?Θ))
        then have x ∉ inside (?Θ1 ∪ ?Θ)
       by (meson closure_iff_nhds_not_empty in_sub_out12 inside_Int_outside
op_in1c)
        with fr_in x show x ∈ ?Θ
          by (metis c1c c1c2 closure_Un_frontier pa1_disj_in2 Int_iff Un_iff
insert_disjoint(2) insert_subset subsetI subset_antisym)
       qed
       show ?rhs ⊆ ?lhs
        using if1 if2 closure_Un_frontier by fastforce
      qed
      ultimately
       show connected (closure (inside (?Θ1 ∪ ?Θ)) ∩ closure (inside (?Θ2 ∪
?Θ)))
       by auto
    qed
    show connected (outside (?Θ1 ∪ ?Θ) ∩ outside (?Θ2 ∪ ?Θ))
      using fr_in conn_U  by (simp add: closure_Un_frontier outside_inside
Un_commute)
    show outside (?Θ1 ∪ ?Θ) ∩ outside (?Θ2 ∪ ?Θ) ⊆ − (?Θ1 ∪ ?Θ2)
    by clarify (metis Diff_Compl Diff_iff Un_iff inf_sup_absorb outside_inside)
    show outside (?Θ1 ∪ ?Θ2) ∩
         (outside (?Θ1 ∪ ?Θ) ∩ outside (?Θ2 ∪ ?Θ)) ≠ {}
      by (metis Int_assoc out_in inf.orderE out_sub12(1) out_sub12(2) out-
side_in_components)
   qed
   show inside (?Θ1 ∪ ?Θ) ∪ inside (?Θ2 ∪ ?Θ) ∪ (?Θ − {a, b}) = inside (?Θ1
∪ ?Θ2)
    (is ?lhs = ?rhs)
   proof
    have  path_image c − {a, b} ⊆ inside (path_image c1 ∪ path_image c2)
      using c1c c2c inside_outside pi_disjoint by fastforce
    then show ?lhs ⊆ ?rhs
      by (simp add: in_sub_in1 in_sub_in2)
   have inside (?Θ1 ∪ ?Θ2) ⊆ inside (?Θ1 ∪ ?Θ) ∪ inside (?Θ2 ∪ ?Θ) ∪ (?Θ)
      using Compl_anti_mono [OF ∗] by (force simp: inside_outside)
    moreover have inside (?Θ1 ∪ ?Θ2) ⊆ −{a,b}
      using c1 union_with_outside by fastforce
    ultimately show ?rhs ⊆ ?lhs by auto
   qed
  qed
qed

end
```

6.38 Polynomial Functions: Extremal Behaviour and Root Counts

theory *Poly_Roots*
imports *Complex_Main*
begin

6.38.1 Basics about polynomial functions: extremal behaviour and root counts

lemma *sub_polyfun*:
fixes $x :: 'a::\{comm\_ring,monoid\_mult\}$
shows $(\sum i{\le}n.\ a\ i * x\hat{}i) - (\sum i{\le}n.\ a\ i * y\hat{}i) =$
$(x - y) * (\sum j{<}n.\ \sum k{=}\ Suc\ j..n.\ a\ k * y\hat{}(k - Suc\ j) * x\hat{}j)$
proof −
have $(\sum i{\le}n.\ a\ i * x\hat{}i) - (\sum i{\le}n.\ a\ i * y\hat{}i) =$
$(\sum i{\le}n.\ a\ i * (x\hat{}i - y\hat{}i))$
by (*simp add*: *algebra_simps sum_subtractf* [*symmetric*])
also have ... $= (\sum i{\le}n.\ a\ i * (x - y) * (\sum j{<}i.\ y\hat{}(i - Suc\ j) * x\hat{}j))$
by (*simp add*: *power_diff_sumr2 ac_simps*)
also have ... $= (x - y) * (\sum i{\le}n.\ (\sum j{<}i.\ a\ i * y\hat{}(i - Suc\ j) * x\hat{}j))$
by (*simp add*: *sum_distrib_left ac_simps*)
also have ... $= (x - y) * (\sum j{<}n.\ (\sum i{=}Suc\ j..n.\ a\ i * y\hat{}(i - Suc\ j) * x\hat{}j))$
by (*simp add*: *sum.nested_swap'*)
finally show *?thesis* **.**
qed

lemma *sub_polyfun_alt*:
fixes $x :: 'a::\{comm\_ring,monoid\_mult\}$
shows $(\sum i{\le}n.\ a\ i * x\hat{}i) - (\sum i{\le}n.\ a\ i * y\hat{}i) =$
$(x - y) * (\sum j{<}n.\ \sum k{<}n{-}j.\ a\ (j{+}k{+}1) * y\hat{}k * x\hat{}j)$
proof −
{ fix j
have $(\sum k = Suc\ j..n.\ a\ k * y\hat{}(k - Suc\ j) * x\hat{}j) =$
$(\sum k < n - j.\ a\ (Suc\ (j + k)) * y\hat{}k * x\hat{}j)$
by (*rule sum.reindex_bij_witness*[**where** $i{=}\lambda i.\ i + Suc\ j$ **and** $j{=}\lambda i.\ i - Suc\ j$]) *auto* **}**
then show *?thesis*
by (*simp add*: *sub_polyfun*)
qed

lemma *polyfun_linear_factor*:
fixes $a :: 'a::\{comm\_ring,monoid\_mult\}$
shows $\exists b.\ \forall z.\ (\sum i{\le}n.\ c\ i * z\hat{}i) =$
$(z{-}a) * (\sum i{<}n.\ b\ i * z\hat{}i) + (\sum i{\le}n.\ c\ i * a\hat{}i)$
proof −
{ fix z
have $(\sum i{\le}n.\ c\ i * z\hat{}i) - (\sum i{\le}n.\ c\ i * a\hat{}i) =$
$(z - a) * (\sum j{<}n.\ (\sum k = Suc\ j..n.\ c\ k * a\hat{}(k - Suc\ j)) * z\hat{}j)$

```
    by (simp add: sub_polyfun sum_distrib_right)
  then have (∑ i≤n. c i * z^i) =
        (z − a) * (∑ j<n. (∑ k = Suc j..n. c k * a^(k − Suc j)) * z^j)
        + (∑ i≤n. c i * a^i)
    by (simp add: algebra_simps) }
 then show ?thesis
  by (intro exI allI)
qed

lemma polyfun_linear_factor_root:
 fixes a :: 'a::{comm_ring,monoid_mult}
 assumes (∑ i≤n. c i * a^i) = 0
 shows ∃ b. ∀ z. (∑ i≤n. c i * z^i) = (z−a) * (∑ i<n. b i * z^i)
 using polyfun_linear_factor [of c n a] assms
 by simp

lemma adhoc_norm_triangle: a + norm(y) ≤ b ==> norm(x) ≤ a ==> norm(x
+ y) ≤ b
 by (metis norm_triangle_mono order.trans order_refl)

proposition polyfun_extremal_lemma:
 fixes c :: nat ⇒ 'a::real_normed_div_algebra
 assumes e > 0
   shows ∃ M. ∀ z. M ≤ norm z ⟶ norm(∑ i≤n. c i * z^i) ≤ e * norm(z) ^
Suc n
proof (induction n)
 case 0
 show ?case
  by (rule exI [where x=norm (c 0) / e]) (auto simp: mult.commute pos_divide_le_eq
assms)
next
 case (Suc n)
 then obtain M where M: ∀ z. M ≤ norm z ⟶ norm (∑ i≤n. c i * z^i) ≤ e
* norm z ^ Suc n ..
 show ?case
 proof (rule exI [where x=max 1 (max M ((e + norm(c(Suc n))) / e))], clarify)
   fix z::'a
   assume max 1 (max M ((e + norm (c (Suc n))) / e)) ≤ norm z
   then have norm1: 0 < norm z M ≤ norm z (e + norm (c (Suc n))) / e ≤
norm z
     by auto
   then have norm2: (e + norm (c (Suc n))) ≤ e * norm z  (norm z * norm z
^ n) > 0
     apply (metis assms less_divide_eq mult.commute not_le)
     using norm1 apply (metis mult_pos_pos zero_less_power)
     done
   have e * (norm z * norm z ^ n) + norm (c (Suc n) * (z * z ^ n)) =
       (e + norm (c (Suc n))) * (norm z * norm z ^ n)
     by (simp add: norm_mult norm_power algebra_simps)
```

```
    also have ... ≤ (e * norm z) * (norm z * norm z ^ n)
      using norm2
      using assms mult_mono by fastforce
    also have ... = e * (norm z * (norm z * norm z ^ n))
      by (simp add: algebra_simps)
    finally have e * (norm z * norm z ^ n) + norm (c (Suc n) * (z * z ^ n))
                 ≤ e * (norm z * (norm z * norm z ^ n)) .
    then show norm (∑ i≤Suc n. c i * z^i) ≤ e * norm z ^ Suc (Suc n) using
M norm1
      by (drule_tac x=z in spec) (auto simp: intro!: adhoc_norm_triangle)
    qed
qed

lemma norm_lemma_xy: assumes |b| + 1 ≤ norm(y) − a norm(x) ≤ a shows
b ≤ norm(x + y)
proof −
  have b ≤ norm y − norm x
    using assms by linarith
  then show ?thesis
    by (metis (no_types) add.commute norm_diff_ineq order_trans)
qed

proposition polyfun_extremal:
  fixes c :: nat ⇒ 'a::real_normed_div_algebra
  assumes ∃k. k ≠ 0 ∧ k ≤ n ∧ c k ≠ 0
    shows eventually (λz. norm(∑ i≤n. c i * z^i) ≥ B) at_infinity
using assms
proof (induction n)
  case 0 then show ?case
    by simp
next
  case (Suc n)
  show ?case
  proof (cases c (Suc n) = 0)
    case True
    with Suc show ?thesis
      by auto (metis diff_is_0_eq diffs0_imp_equal less_Suc_eq_le not_less_eq)
  next
    case False
    with polyfun_extremal_lemma [of norm(c (Suc n)) / 2 c n]
    obtain M where M: ⋀z. M ≤ norm z ⟹
              norm (∑ i≤n. c i * z^i) ≤ norm (c (Suc n)) / 2 * norm z ^ Suc n
      by auto
    show ?thesis
    unfolding eventually_at_infinity
    proof (rule exI [where x=max M (max 1 ((|B| + 1) / (norm (c (Suc n)) /
2)))], clarsimp)
      fix z::'a
      assume les: M ≤ norm z  1 ≤ norm z  (|B| * 2 + 2) / norm (c (Suc n)) ≤
```

```
norm z
      then have |B| * 2 + 2 ≤ norm z * norm (c (Suc n))
        by (metis False pos_divide_le_eq zero_less_norm_iff)
      then have |B| * 2 + 2 ≤ norm z ^ (Suc n) * norm (c (Suc n))
          by (metis ‹1 ≤ norm z› order.trans mult_right_mono norm_ge_zero
self_le_power zero_less_Suc)
      then show B ≤ norm ((∑ i≤n. c i * z^i) + c (Suc n) * (z * z ^ n)) using
M les
       apply auto
        apply (rule norm_lemma_xy [where a = norm (c (Suc n)) * norm z ^
(Suc n) / 2])
       apply (simp_all add: norm_mult norm_power)
       done
    qed
  qed
qed

proposition polyfun_rootbound:
 fixes c :: nat ⇒ 'a::{comm_ring,real_normed_div_algebra}
 assumes ∃k. k ≤ n ∧ c k ≠ 0
   shows finite {z. (∑ i≤n. c i * z^i) = 0} ∧ card {z. (∑ i≤n. c i * z^i) = 0}
≤ n
using assms
proof (induction n arbitrary: c)
 case (Suc n) show ?case
 proof (cases {z. (∑ i≤Suc n. c i * z^i) = 0} = {})
   case False
   then obtain a where a: (∑ i≤Suc n. c i * a^i) = 0
     by auto
   from polyfun_linear_factor_root [OF this]
   obtain b where ⋀z. (∑ i≤Suc n. c i * z^i) = (z − a) * (∑ i< Suc n. b i *
z^i)
     by auto
   then have b: ⋀z. (∑ i≤Suc n. c i * z^i) = (z − a) * (∑ i≤n. b i * z^i)
     by (metis lessThan_Suc_atMost)
   then have ins_ab: {z. (∑ i≤Suc n. c i * z^i) = 0} = insert a {z. (∑ i≤n. b i
* z^i) = 0}
     by auto
   have c0: c 0 = − (a * b 0) using  b [of 0]
     by simp
   then have extr_prem: ¬ (∃k≤n. b k ≠ 0) ⟹ ∃k. k ≠ 0 ∧ k ≤ Suc n ∧ c k
≠ 0
     by (metis Suc.prems le0 minus_zero mult_zero_right)
   have ∃k≤n. b k ≠ 0
     apply (rule ccontr)
     using polyfun_extremal [OF extr_prem, of 1]
     apply (auto simp: eventually_at_infinity b simp del: sum.atMost_Suc)
     apply (drule_tac x=of_real ba in spec, simp)
     done
```

```
  then show ?thesis using Suc.IH [of b] ins_ab
    by (auto simp: card_insert_if)
  qed simp
qed simp

corollary
  fixes c :: nat ⇒ 'a::{comm_ring,real_normed_div_algebra}
  assumes ∃k. k ≤ n ∧ c k ≠ 0
    shows polyfun_rootbound_finite: finite {z. (∑i≤n. c i * z^i) = 0}
      and polyfun_rootbound_card:   card {z. (∑i≤n. c i * z^i) = 0} ≤ n
using polyfun_rootbound [OF assms] by auto

proposition polyfun_finite_roots:
  fixes c :: nat ⇒ 'a::{comm_ring,real_normed_div_algebra}
    shows  finite {z. (∑i≤n. c i * z^i) = 0} ⟷ (∃k. k ≤ n ∧ c k ≠ 0)
proof (cases  ∃k≤n. c k ≠ 0)
  case True then show ?thesis
    by (blast intro: polyfun_rootbound_finite)
next
  case False then show ?thesis
    by (auto simp: infinite_UNIV_char_0)
qed

lemma polyfun_eq_0:
  fixes c :: nat ⇒ 'a::{comm_ring,real_normed_div_algebra}
    shows  (∀z. (∑i≤n. c i * z^i) = 0) ⟷ (∀k. k ≤ n ⟶ c k = 0)
proof (cases (∀z. (∑i≤n. c i * z^i) = 0))
  case True
  then have ¬ finite {z. (∑i≤n. c i * z^i) = 0}
    by (simp add: infinite_UNIV_char_0)
  with True show ?thesis
    by (metis (poly_guards_query) polyfun_rootbound_finite)
next
  case False
  then show ?thesis
    by auto
qed

theorem polyfun_eq_const:
  fixes c :: nat ⇒ 'a::{comm_ring,real_normed_div_algebra}
    shows  (∀z. (∑i≤n. c i * z^i) = k) ⟷ c 0 = k ∧ (∀k. k ≠ 0 ∧ k ≤ n ⟶
c k = 0)
proof −
  {fix z
    have (∑i≤n. c i * z^i) = (∑i≤n. (if i = 0 then c 0 − k else c i) * z^i) + k
      by (induct n) auto
  } then
  have (∀z. (∑i≤n. c i * z^i) = k) ⟷ (∀z. (∑i≤n. (if i = 0 then c 0 − k else
c i) * z^i) = 0)
```

```
    by auto
  also have ... ⟷ c 0 = k ∧ (∀k. k ≠ 0 ∧ k ≤ n ⟶ c k = 0)
    by (auto simp: polyfun_eq_0)
  finally show ?thesis .
qed

end
```

6.39 Generalised Binomial Theorem

The proof of the Generalised Binomial Theorem and related results. We prove the generalised binomial theorem for complex numbers, following the proof at: https://proofwiki.org/wiki/Binomial_Theorem/General_Binomial_Theorem

```
theory Generalised_Binomial_Theorem
imports
  Complex_Main
  Complex_Transcendental
  Summation_Tests
begin

lemma gbinomial_ratio_limit:
  fixes a :: 'a :: real_normed_field
  assumes a ∉ ℕ
  shows (λn. (a gchoose n) / (a gchoose Suc n)) ——⟶ −1
proof (rule Lim_transform_eventually)
  let ?f = λn. inverse (a / of_nat (Suc n) − of_nat n / of_nat (Suc n))
  from eventually_gt_at_top[of 0::nat]
    show eventually (λn. ?f n = (a gchoose n) /(a gchoose Suc n)) sequentially
  proof eventually_elim
    fix n :: nat assume n: n > 0
    then obtain q where q: n = Suc q by (cases n) blast
    let ?P = ∏ i=0..<n. a − of_nat i
    from n have (a gchoose n) / (a gchoose Suc n) = (of_nat (Suc n) :: 'a) *
                  (?P / (∏ i=0..n. a − of_nat i))
      by (simp add: gbinomial_prod_rev atLeastLessThanSuc_atLeastAtMost)
    also from q have (∏ i=0..n. a − of_nat i) = ?P * (a − of_nat n)
    by (simp add: prod.atLeast0_atMost_Suc atLeastLessThanSuc_atLeastAtMost)
   also have ?P / ... = (?P / ?P) / (a − of_nat n) by (rule divide_divide_eq_left[symmetric])
    also from assms have ?P / ?P = 1 by auto
    also have of_nat (Suc n) * (1 / (a − of_nat n)) =
                  inverse (inverse (of_nat (Suc n)) * (a − of_nat n)) by (simp add:
field_simps)
    also have inverse (of_nat (Suc n)) * (a − of_nat n) = a / of_nat (Suc n) −
of_nat n / of_nat (Suc n)
      by (simp add: field_simps del: of_nat_Suc)
    finally show ?f n = (a gchoose n) / (a gchoose Suc n) by simp
  qed
```

```
  have (λn. norm a / (of_nat (Suc n))) ⟶ 0
    unfolding divide_inverse
    by (intro tendsto_mult_right_zero LIMSEQ_inverse_real_of_nat)
  hence (λn. a / of_nat (Suc n)) ⟶ 0
     by (subst tendsto_norm_zero_iff[symmetric]) (simp add: norm_divide del:
of_nat_Suc)
  hence ?f ⟶ inverse (0 - 1)
    by (intro tendsto_inverse tendsto_diff LIMSEQ_n_over_Suc_n) simp_all
  thus ?f ⟶ -1 by simp
qed

lemma conv_radius_gchoose:
  fixes a :: 'a :: {real_normed_field,banach}
  shows conv_radius (λn. a gchoose n) = (if a ∈ ℕ then ∞ else 1)
proof (cases a ∈ ℕ)
  assume a: a ∈ ℕ
  have eventually (λn. (a gchoose n) = 0) sequentially
    using eventually_gt_at_top[of nat ⌊norm a⌋]
   by eventually_elim (insert a, auto elim!: Nats_cases simp: binomial_gbinomial[symmetric])
  from conv_radius_cong'[OF this] a show ?thesis by simp
next
  assume a: a ∉ ℕ
  from tendsto_norm[OF gbinomial_ratio_limit[OF this]]
    have conv_radius (λn. a gchoose n) = 1
   by (intro conv_radius_ratio_limit_nonzero[of _ 1]) (simp_all add: norm_divide)
  with a show ?thesis by simp
qed

theorem gen_binomial_complex:
  fixes z :: complex
  assumes norm z < 1
  shows   (λn. (a gchoose n) * z^n) sums (1 + z) powr a
proof -
  define K where K = 1 - (1 - norm z) / 2
  from assms have K: K > 0 K < 1 norm z < K
     unfolding K_def by (auto simp: field_simps intro!: add_pos_nonneg)
  let ?f = λn. a gchoose n and ?f' = diffs (λn. a gchoose n)
  have summable_strong: summable (λn. ?f n * z ^ n) if norm z < 1 for z using
that
    by (intro summable_in_conv_radius) (simp_all add: conv_radius_gchoose)
  with K have summable: summable (λn. ?f n * z ^ n) if norm z < K for z using
that by auto
  hence summable': summable (λn. ?f' n * z ^ n) if norm z < K for z using that
    by (intro termdiff_converges[of _ K]) simp_all

  define f f' where [abs_def]: f z = (∑ n. ?f n * z ^ n) f' z = (∑ n. ?f' n * z ^
n) for z
  {
```

```
    fix z :: complex assume z: norm z < K
    from summable_mult2[OF summable'[OF z], of z]
      have summable1: summable (λn. ?f' n * z ^ Suc n) by (simp add: mult_ac)
    hence summable2: summable (λn. of_nat n * ?f n * z^n)
      unfolding diffs_def by (subst (asm) summable_Suc_iff)

    have (1 + z) * f' z = (∑ n. ?f' n * z^n) + (∑ n. ?f' n * z^Suc n)
      unfolding f_f'_def using summable' z by (simp add: algebra_simps sum-
inf_mult)
    also have (∑ n. ?f' n * z^n) = (∑ n. of_nat (Suc n) * ?f (Suc n) * z^n)
      by (intro suminf_cong) (simp add: diffs_def)
    also have (∑ n. ?f' n * z^Suc n) = (∑ n. of_nat n * ?f n * z ^ n)
      using summable1 suminf_split_initial_segment[OF summable1] unfolding
diffs_def
      by (subst suminf_split_head, subst (asm) summable_Suc_iff) simp_all
    also have (∑ n. of_nat (Suc n) * ?f (Suc n) * z^n) + (∑ n. of_nat n * ?f n
* z^n) =
                (∑ n. a * ?f n * z^n)
      by (subst gbinomial_mult_1, subst suminf_add)
         (insert summable'[OF z] summable2,
          simp_all add: summable_powser_split_head algebra_simps diffs_def)
    also have ... = a * f z unfolding f_f'_def
     by (subst suminf_mult[symmetric]) (simp_all add: summable[OF z] mult_ac)
    finally have a * f z = (1 + z) * f' z by simp
  } note deriv = this

  have [derivative_intros]: (f has_field_derivative f' z) (at z) if norm z < of_real
K for z
    unfolding f_f'_def using K that
    by (intro termdiffs_strong[of ?f K z] summable_strong) simp_all
  have f 0 = (∑ n. if n = 0 then 1 else 0) unfolding f_f'_def by (intro sum-
inf_cong) simp
  also have ... = 1 using sums_single[of 0 λ_. 1::complex] unfolding sums_iff
by simp
  finally have [simp]: f 0 = 1 .

  have ∃ c. ∀ z∈ball 0 K. f z * (1 + z) powr (−a) = c
  proof (rule has_field_derivative_zero_constant)
    fix z :: complex assume z': z ∈ ball 0 K
    hence z: norm z < K by simp
    with K have nz: 1 + z ≠ 0 by (auto dest!: minus_unique)
    from z K have norm z < 1 by simp
   hence (1 + z) ∉ ℝ≤0 by (cases z) (auto simp: Complex_eq complex_nonpos_Reals_iff)
    hence ((λz. f z * (1 + z) powr (−a)) has_field_derivative
             f' z * (1 + z) powr (−a) − a * f z * (1 + z) powr (−a−1)) (at z)
using z
      by (auto intro!: derivative_eq_intros)
    also from z have a * f z = (1 + z) * f' z by (rule deriv)
     finally show ((λz. f z * (1 + z) powr (−a)) has_field_derivative 0) (at z
```

```
within ball 0 K)
      using nz by (simp add: field_simps powr_diff at_within_open[OF z'])
  qed simp_all
  then obtain c where c: ⋀z. z ∈ ball 0 K ⟹ f z * (1 + z) powr (−a) = c by
blast
  from c[of 0] and K have c = 1 by simp
  with c[of z] have f z = (1 + z) powr a using K
    by (simp add: powr_minus field_simps dist_complex_def)
  with summable K show ?thesis unfolding f_f'_def by (simp add: sums_iff)
qed

lemma gen_binomial_complex':
  fixes x y :: real and a :: complex
  assumes |x| < |y|
  shows   (λn. (a gchoose n) * of_real x^n * of_real y powr (a − of_nat n)) sums
              of_real (x + y) powr a (is ?P x y)
proof −
  {
    fix x y :: real assume xy: |x| < |y| y ≥ 0
    hence y > 0 by simp
    note xy = xy this
    from xy have (λn. (a gchoose n) * of_real (x / y) ^ n) sums (1 + of_real (x
/ y)) powr a
      by (intro gen_binomial_complex) (simp add: norm_divide)
    hence (λn. (a gchoose n) * of_real (x / y) ^ n * y powr a) sums
            ((1 + of_real (x / y)) powr a * y powr a)
      by (rule sums_mult2)
    also have (1 + complex_of_real (x / y)) = complex_of_real (1 + x/y) by
simp
    also from xy have ... powr a * of_real y powr a = (... * y) powr a
      by (subst powr_times_real[symmetric]) (simp_all add: field_simps)
   also from xy have complex_of_real (1 + x / y) * complex_of_real y = of_real
(x + y)
      by (simp add: field_simps)
    finally have ?P x y using xy by (simp add: field_simps powr_diff powr_nat)
  } note A = this

  show ?thesis
  proof (cases y < 0)
    assume y: y < 0
    with assms have xy: x + y < 0 by simp
    with assms have |−x| < |−y| −y ≥ 0 by simp_all
    note A[OF this]
    also have complex_of_real (−x + −y) = − complex_of_real (x + y) by simp
    also from xy assms have ... powr a = (−1) powr −a * of_real (x + y) powr a
      by (subst powr_neg_real_complex) (simp add: abs_real_def split: if_split_asm)
    also {
      fix n :: nat
      from y have (a gchoose n) * of_real (−x) ^ n * of_real (−y) powr (a −
```

```
of_nat n) =
                    (a gchoose n) * (-of_real x / -of_real y) ^ n * (- of_real y)
powr a
       by (subst power_divide) (simp add: powr_diff powr_nat)
     also from y have (- of_real y) powr a = (-1) powr -a * of_real y powr a
       by (subst powr_neg_real_complex) simp
     also have -complex_of_real x / -complex_of_real y = complex_of_real x
/ complex_of_real y
       by simp
    also have ... ^ n = of_real x ^ n / of_real y ^ n by (simp add: power_divide)
     also have (a gchoose n) * ... * ((-1) powr -a * of_real y powr a) =
                  (-1) powr -a * ((a gchoose n) * of_real x ^ n * of_real y powr
(a - n))
       by (simp add: algebra_simps powr_diff powr_nat)
     finally have (a gchoose n) * of_real (- x) ^ n * of_real (- y) powr (a -
of_nat n) =
                   (-1) powr -a * ((a gchoose n) * of_real x ^ n * of_real y powr
(a - of_nat n)) .
   }
   note sums_cong[OF this]
   finally show ?thesis by (simp add: sums_mult_iff)
 qed (insert A[of x y] assms, simp_all add: not_less)
qed

lemma gen_binomial_complex'':
 fixes x y :: real and a :: complex
 assumes |y| < |x|
 shows   (λn. (a gchoose n) * of_real x powr (a - of_nat n) * of_real y ^ n)
sums
             of_real (x + y) powr a
 using gen_binomial_complex'[OF assms] by (simp add: mult_ac add.commute)

lemma gen_binomial_real:
 fixes z :: real
 assumes |z| < 1
 shows   (λn. (a gchoose n) * z^n) sums (1 + z) powr a
proof -
 from assms have norm (of_real z :: complex) < 1 by simp
 from gen_binomial_complex[OF this]
   have (λn. (of_real a gchoose n :: complex) * of_real z ^ n) sums
          (of_real (1 + z)) powr (of_real a) by simp
 also have (of_real (1 + z) :: complex) powr (of_real a) = of_real ((1 + z)
powr a)
   using assms by (subst powr_of_real) simp_all
 also have (of_real a gchoose n :: complex) = of_real (a gchoose n) for n
   by (simp add: gbinomial_prod_rev)
 hence (λn. (of_real a gchoose n :: complex) * of_real z ^ n) =
        (λn. of_real ((a gchoose n) * z ^ n)) by (intro ext) simp
 finally show ?thesis by (simp only: sums_of_real_iff)
```

qed

lemma *gen_binomial_real'*:
fixes *x y a* :: *real*
assumes $|x| < y$
shows $(\lambda n.\ (a\ gchoose\ n) * x\hat{}n * y\ powr\ (a - of\_nat\ n))\ sums\ (x + y)\ powr\ a$
proof −
from *assms* **have** $y > 0$ **by** *simp*
note $xy = this\ assms$
from *assms* **have** $|x\ /\ y| < 1$ **by** *simp*
hence $(\lambda n.\ (a\ gchoose\ n) * (x\ /\ y)\ \hat{}\ n)\ sums\ (1 + x\ /\ y)\ powr\ a$
by (*rule gen_binomial_real*)
hence $(\lambda n.\ (a\ gchoose\ n) * (x\ /\ y)\ \hat{}\ n * y\ powr\ a)\ sums\ ((1 + x\ /\ y)\ powr\ a * y\ powr\ a)$
by (*rule sums_mult2*)
with *xy* **show** *?thesis*
by (*simp add*: *field_simps powr_divide powr_diff powr_realpow*)
qed

lemma *one_plus_neg_powr_powser*:
fixes *z s* :: *complex*
assumes $norm\ (z :: complex) < 1$
shows $(\lambda n.\ (-1)\hat{}n * ((s + n - 1)\ gchoose\ n) * z\hat{}n)\ sums\ (1 + z)\ powr\ (-s)$
using *gen_binomial_complex*[*OF assms*, *of* $-s$] **by** (*simp add*: *gbinomial_minus*)

lemma *gen_binomial_real''*:
fixes *x y a* :: *real*
assumes $|y| < x$
shows $(\lambda n.\ (a\ gchoose\ n) * x\ powr\ (a - of\_nat\ n) * y\hat{}n)\ sums\ (x + y)\ powr\ a$
using *gen_binomial_real'*[*OF assms*] **by** (*simp add*: *mult_ac add.commute*)

lemma *sqrt_series'*:
$|z| < a \Longrightarrow (\lambda n.\ ((1/2)\ gchoose\ n) * a\ powr\ (1/2 - real\_of\_nat\ n) * z\ \hat{}\ n)\ sums$
$sqrt\ (a + z :: real)$
using *gen_binomial_real''*[*of z a 1/2*] **by** (*simp add*: *powr_half_sqrt*)

lemma *sqrt_series*:
$|z| < 1 \Longrightarrow (\lambda n.\ ((1/2)\ gchoose\ n) * z\ \hat{}\ n)\ sums\ sqrt\ (1 + z)$
using *gen_binomial_real*[*of z 1/2*] **by** (*simp add*: *powr_half_sqrt*)

end

6.40 Vitali Covering Theorem and an Application to Negligibility

theory *Vitali_Covering_Theorem*
imports
HOL−Combinatorics.Permutations
Equivalence_Lebesgue_Henstock_Integration
begin

lemma *stretch_Galois*:
fixes $x :: real^{\wedge\prime}n$
shows $(\bigwedge k.\ m\ k \neq 0) \Longrightarrow ((y = (\chi\ k.\ m\ k * x\$k)) \longleftrightarrow (\chi\ k.\ y\$k\ /\ m\ k) = x)$
by *auto*

lemma *lambda_swap_Galois*:
$(x = (\chi\ i.\ y\ \$\ Transposition.transpose\ m\ n\ i) \longleftrightarrow (\chi\ i.\ x\ \$\ Transposition.transpose\ m\ n\ i) = y)$
by (*auto*; *simp add*: *pointfree_idE vec_eq_iff*)

lemma *lambda_add_Galois*:
fixes $x :: real^{\wedge\prime}n$
shows $m \neq n \Longrightarrow (x = (\chi\ i.\ if\ i = m\ then\ y\$m + y\$n\ else\ y\$i) \longleftrightarrow (\chi\ i.\ if\ i = m\ then\ x\$m - x\$n\ else\ x\$i) = y)$
by (*safe*; *simp add*: *vec_eq_iff*)

lemma *Vitali_covering_lemma_cballs_balls*:
fixes $a :: 'a \Rightarrow 'b{::}euclidean\_space$
assumes $\bigwedge i.\ i \in K \Longrightarrow 0 < r\ i \wedge r\ i \leq B$
obtains C **where** *countable* C $C \subseteq K$
pairwise $(\lambda i\ j.\ disjnt\ (cball\ (a\ i)\ (r\ i))\ (cball\ (a\ j)\ (r\ j)))\ C$
$\bigwedge i.\ i \in K \Longrightarrow \exists j.\ j \in C \wedge$
$\neg\ disjnt\ (cball\ (a\ i)\ (r\ i))\ (cball\ (a\ j)\ (r\ j)) \wedge$
$cball\ (a\ i)\ (r\ i) \subseteq ball\ (a\ j)\ (5 * r\ j)$
proof (*cases* $K = \{\}$)
case *True*
with *that* **show** *?thesis*
by *auto*
next
case *False*
then have $B > 0$
using *assms less_le_trans* **by** *auto*
have *rgt0*[*simp*]: $\bigwedge i.\ i \in K \Longrightarrow 0 < r\ i$
using *assms* **by** *auto*
let *?djnt* = $pairwise\ (\lambda i\ j.\ disjnt\ (cball\ (a\ i)\ (r\ i))\ (cball\ (a\ j)\ (r\ j)))$
have $\exists C.\ \forall n.\ (C\ n \subseteq K \wedge$
$(\forall i \in C\ n.\ B/2\ \hat{}\ n \leq r\ i) \wedge$ *?djnt* $(C\ n) \wedge$
$(\forall i \in K.\ B/2\ \hat{}\ n < r\ i$
$\longrightarrow (\exists j.\ j \in C\ n \wedge$

```
                    ¬ disjnt (cball (a i) (r i)) (cball (a j) (r j)) ∧
                    cball (a i) (r i) ⊆ ball (a j) (5 * r j)))) ∧ (C n ⊆ C(Suc n))
  proof (rule dependent_nat_choice, safe)
    fix C n
    define D where D ≡ {i ∈ K. B/2 ^ Suc n < r i ∧ (∀j∈C. disjnt (cball(a i)(r
i)) (cball (a j) (r j)))}
    let ?cover_ar = λi j. ¬ disjnt (cball (a i) (r i)) (cball (a j) (r j)) ∧
                          cball (a i) (r i) ⊆ ball (a j) (5 * r j)
    assume C ⊆ K
      and Ble: ∀i∈C. B/2 ^ n ≤ r i
      and djntC: ?djnt C
      and cov_n: ∀i∈K. B/2 ^ n < r i ⟶ (∃j. j ∈ C ∧ ?cover_ar i j)
    have *: ∀C∈chains {C. C ⊆ D ∧ ?djnt C}. ⋃C ∈ {C. C ⊆ D ∧ ?djnt C}
    proof (clarsimp simp: chains_def)
      fix C
      assume C: C ⊆ {C. C ⊆ D ∧ ?djnt C} and chain⊆ C
      show ⋃C ⊆ D ∧ ?djnt (⋃C)
        unfolding pairwise_def
      proof (intro ballI conjI impI)
        show ⋃C ⊆ D
          using C by blast
      next
        fix x y
        assume x ∈ ⋃C and y ∈ ⋃C and x ≠ y
        then obtain X Y where XY: x ∈ X X ∈ C y ∈ Y Y ∈ C
          by blast
        then consider X ⊆ Y | Y ⊆ X
          by (meson ‹chain⊆ C› chain_subset_def)
        then show disjnt (cball (a x) (r x)) (cball (a y) (r y))
        proof cases
          case 1
          with C XY ‹x ≠ y› show ?thesis
            unfolding pairwise_def by blast
        next
          case 2
          with C XY ‹x ≠ y› show ?thesis
            unfolding pairwise_def by blast
        qed
      qed
    qed
    obtain E where E ⊆ D and djntE: ?djnt E and maximalE: ⋀X. ⟦X ⊆ D;
?djnt X; E ⊆ X⟧ ⟹ X = E
      using Zorn_Lemma [OF *] by safe blast
    show ∃L. (L ⊆ K ∧
              (∀i∈L. B/2 ^ Suc n ≤ r i) ∧ ?djnt L ∧
             (∀i∈K. B/2 ^ Suc n < r i ⟶ (∃j. j ∈ L ∧ ?cover_ar i j))) ∧ C ⊆ L
    proof (intro exI conjI ballI)
      show C ∪ E ⊆ K
        using D_def ‹C ⊆ K› ‹E ⊆ D› by blast
```

```
    show B/2 ^ Suc n ≤ r i if i: i ∈ C ∪ E for i
      using i
    proof
      assume i ∈ C
      have B/2 ^ Suc n ≤ B/2 ^ n
        using ‹B > 0› by (simp add: field_split_simps)
      also have ... ≤ r i
        using Ble ‹i ∈ C› by blast
      finally show ?thesis .
    qed (use D_def ‹E ⊆ D› in auto)
    show ?djnt (C ∪ E)
      using D_def ‹C ⊆ K› ‹E ⊆ D› djntC djntE
      unfolding pairwise_def disjnt_def by blast
  next
    fix i
    assume i ∈ K
    show B/2 ^ Suc n < r i ⟶ (∃j. j ∈ C ∪ E ∧ ?cover_ar i j)
    proof (cases r i ≤ B/2^n)
      case False
      then show ?thesis
        using cov_n ‹i ∈ K› by auto
    next
      case True
      have cball (a i) (r i) ⊆ ball (a j) (5 * r j)
        if less: B/2 ^ Suc n < r i and j: j ∈ C ∪ E
          and nondis: ¬ disjnt (cball (a i) (r i)) (cball (a j) (r j)) for j
      proof −
        obtain x where x: dist (a i) x ≤ r i dist (a j) x ≤ r j
          using nondis by (force simp: disjnt_def)
        have dist (a i) (a j) ≤ dist (a i) x + dist x (a j)
          by (simp add: dist_triangle)
        also have ... ≤ r i + r j
          by (metis add_mono_thms_linordered_semiring(1) dist_commute x)
        finally have aij: dist (a i) (a j) + r i < 5 * r j if r i < 2 * r j
          using that by auto
        show ?thesis
          using j
        proof
          assume j ∈ C
          have B/2^n < 2 * r j
            using Ble True ‹j ∈ C› less by auto
          with aij True show cball (a i) (r i) ⊆ ball (a j) (5 * r j)
            by (simp add: cball_subset_ball_iff)
        next
          assume j ∈ E
          then have B/2 ^ n < 2 * r j
            using D_def ‹E ⊆ D› by auto
          with True have r i < 2 * r j
            by auto
```

```
          with aij show cball (a i) (r i) ⊆ ball (a j) (5 * r j)
            by (simp add: cball_subset_ball_iff)
        qed
      qed
    moreover have ∃j. j ∈ C ∪ E ∧ ¬ disjnt (cball (a i) (r i)) (cball (a j) (r j))
      if B/2 ^ Suc n < r i
    proof (rule classical)
      assume NON: ¬ ?thesis
      show ?thesis
      proof (cases i ∈ D)
        case True
        have insert i E = E
        proof (rule maximalE)
          show insert i E ⊆ D
            by (simp add: True ‹E ⊆ D›)
          show pairwise (λi j. disjnt (cball (a i) (r i)) (cball (a j) (r j))) (insert i
E)
            using False NON by (auto simp: pairwise_insert djntE disjnt_sym)
        qed auto
        then show ?thesis
          using ‹i ∈ K› assms by fastforce
      next
        case False
        with that show ?thesis
          by (auto simp: D_def disjnt_def ‹i ∈ K›)
      qed
    qed
    ultimately
    show B/2 ^ Suc n < r i ⟶
        (∃j. j ∈ C ∪ E ∧
            ¬ disjnt (cball (a i) (r i)) (cball (a j) (r j)) ∧
            cball (a i) (r i) ⊆ ball (a j) (5 * r j))
      by blast
    qed
  qed auto
qed (use assms in force)
then obtain F where FK: ⋀n. F n ⊆ K
          and Fle: ⋀n i. i ∈ F n ⟹ B/2 ^ n ≤ r i
          and Fdjnt: ⋀n. ?djnt (F n)
          and FF: ⋀n i. ⟦i ∈ K; B/2 ^ n < r i⟧
                ⟹ ∃j. j ∈ F n ∧ ¬ disjnt (cball (a i) (r i)) (cball (a j) (r j)) ∧
                      cball (a i) (r i) ⊆ ball (a j) (5 * r j)
          and inc: ⋀n. F n ⊆ F(Suc n)
  by (force simp: all_conj_distrib)
show thesis
proof
  have *: countable I
    if I ⊆ K and pw: pairwise (λi j. disjnt (cball (a i) (r i)) (cball (a j) (r j)))
I for I
```

```
    proof −
      show ?thesis
      proof (rule countable_image_inj_on [of λi. cball(a i)(r i)])
        show countable ((λi. cball (a i) (r i)) ‘ I)
        proof (rule countable_disjoint_nonempty_interior_subsets)
          show disjoint ((λi. cball (a i) (r i)) ‘ I)
            by (auto simp: dest: pairwiseD [OF pw] intro: pairwise_imageI)
          show ⋀S. ⟦S ∈ (λi. cball (a i) (r i)) ‘ I; interior S = {}⟧ ⟹ S = {}
            using ‹I ⊆ K›
            by (auto simp: not_less [symmetric])
        qed
      next
        have ⋀x y. ⟦x ∈ I; y ∈ I; a x = a y; r x = r y⟧ ⟹ x = y
          using pw ‹I ⊆ K› assms
          apply (clarsimp simp: pairwise_def disjnt_def)
       by (metis assms centre_in_cball subsetD empty_iff inf.idem less_eq_real_def)
        then show inj_on (λi. cball (a i) (r i)) I
          using ‹I ⊆ K› by (fastforce simp: inj_on_def cball_eq_cball_iff dest:
assms)
      qed
    qed
    show (Union(range F)) ⊆ K
      using FK by blast
    moreover show pairwise (λi j. disjnt (cball (a i) (r i)) (cball (a j) (r j)))
(Union(range F))
    proof (rule pairwise_chain_Union)
      show chain⊆ (range F)
      unfolding chain_subset_def by clarify (meson inc lift_Suc_mono_le linear
subsetCE)
    qed (use Fdjnt in blast)
    ultimately show countable (Union(range F))
      by (blast intro: ∗)
  next
    fix i assume i ∈ K
    then obtain n where (1/2) ^ n < r i / B
      using ‹B > 0› assms real_arch_pow_inv by fastforce
    then have B2: B/2 ^ n < r i
      using ‹B > 0› by (simp add: field_split_simps)
    have 0 < r i r i ≤ B
      by (auto simp: ‹i ∈ K› assms)
    show ∃j. j ∈ (Union(range F)) ∧
          ¬ disjnt (cball (a i) (r i)) (cball (a j) (r j)) ∧
          cball (a i) (r i) ⊆ ball (a j) (5 ∗ r j)
      using FF [OF ‹i ∈ K› B2] by auto
  qed
qed
```

6.40.1 Vitali covering theorem

lemma *Vitali_covering_lemma_cballs*:
fixes $a :: 'a \Rightarrow 'b::euclidean\_space$
assumes S: $S \subseteq (\bigcup i{\in}K.\ cball\ (a\ i)\ (r\ i))$
and r: $\bigwedge i.\ i \in K \Longrightarrow 0 < r\ i \wedge r\ i \leq B$
obtains C **where** $countable\ C\ C \subseteq K$
$pairwise\ (\lambda i\ j.\ disjnt\ (cball\ (a\ i)\ (r\ i))\ (cball\ (a\ j)\ (r\ j)))\ C$
$S \subseteq (\bigcup i{\in}C.\ cball\ (a\ i)\ (5 * r\ i))$
proof −
obtain C **where** C: $countable\ C\ C \subseteq K$
$pairwise\ (\lambda i\ j.\ disjnt\ (cball\ (a\ i)\ (r\ i))\ (cball\ (a\ j)\ (r\ j)))\ C$
and cov: $\bigwedge i.\ i \in K \Longrightarrow \exists j.\ j \in C \wedge \neg\ disjnt\ (cball\ (a\ i)\ (r\ i))\ (cball\ (a\ j)\ (r\ j)) \wedge$
$cball\ (a\ i)\ (r\ i) \subseteq ball\ (a\ j)\ (5 * r\ j)$
by (*rule Vitali_covering_lemma_cballs_balls* [*OF r*, **where** *a*=*a*]) (*blast intro*: *that*)+
show *?thesis*
proof
have $(\bigcup i{\in}K.\ cball\ (a\ i)\ (r\ i)) \subseteq (\bigcup i{\in}C.\ cball\ (a\ i)\ (5 * r\ i))$
using *cov subset_iff* **by** *fastforce*
with S **show** $S \subseteq (\bigcup i{\in}C.\ cball\ (a\ i)\ (5 * r\ i))$
by *blast*
qed (*use C* **in** *auto*)
qed

lemma *Vitali_covering_lemma_balls*:
fixes $a :: 'a \Rightarrow 'b::euclidean\_space$
assumes S: $S \subseteq (\bigcup i{\in}K.\ ball\ (a\ i)\ (r\ i))$
and r: $\bigwedge i.\ i \in K \Longrightarrow 0 < r\ i \wedge r\ i \leq B$
obtains C **where** $countable\ C\ C \subseteq K$
$pairwise\ (\lambda i\ j.\ disjnt\ (ball\ (a\ i)\ (r\ i))\ (ball\ (a\ j)\ (r\ j)))\ C$
$S \subseteq (\bigcup i{\in}C.\ ball\ (a\ i)\ (5 * r\ i))$
proof −
obtain C **where** C: $countable\ C\ C \subseteq K$
and pw: $pairwise\ (\lambda i\ j.\ disjnt\ (cball\ (a\ i)\ (r\ i))\ (cball\ (a\ j)\ (r\ j)))\ C$
and cov: $\bigwedge i.\ i \in K \Longrightarrow \exists j.\ j \in C \wedge \neg\ disjnt\ (cball\ (a\ i)\ (r\ i))\ (cball\ (a\ j)\ (r\ j)) \wedge$
$cball\ (a\ i)\ (r\ i) \subseteq ball\ (a\ j)\ (5 * r\ j)$
by (*rule Vitali_covering_lemma_cballs_balls* [*OF r*, **where** *a*=*a*]) (*blast intro*: *that*)+
show *?thesis*
proof
have $(\bigcup i{\in}K.\ ball\ (a\ i)\ (r\ i)) \subseteq (\bigcup i{\in}C.\ ball\ (a\ i)\ (5 * r\ i))$
using *cov subset_iff*
by *clarsimp* (*meson less_imp_le mem_ball mem_cball subset_eq*)
with S **show** $S \subseteq (\bigcup i{\in}C.\ ball\ (a\ i)\ (5 * r\ i))$
by *blast*
show $pairwise\ (\lambda i\ j.\ disjnt\ (ball\ (a\ i)\ (r\ i))\ (ball\ (a\ j)\ (r\ j)))\ C$
using *pw*

by (*clarsimp simp*: *pairwise_def*) (*meson ball_subset_cball disjnt_subset1 disjnt_subset2*)
qed (*use C* **in** *auto*)
qed

theorem *Vitali_covering_theorem_cballs*:
fixes *a* :: *'a* ⇒ *'n*::*euclidean_space*
assumes *r*: ⋀*i*. *i* ∈ *K* ⟹ 0 < *r i*
and *S*: ⋀*x d*. ⟦*x* ∈ *S*; 0 < *d*⟧
⟹ ∃ *i*. *i* ∈ *K* ∧ *x* ∈ *cball* (*a i*) (*r i*) ∧ *r i* < *d*
obtains *C* **where** *countable C C* ⊆ *K*
pairwise (λ*i j*. *disjnt* (*cball* (*a i*) (*r i*)) (*cball* (*a j*) (*r j*))) *C*
negligible(*S* − (⋃ *i* ∈ *C*. *cball* (*a i*) (*r i*)))
proof −
let *?μ* = *measure lebesgue*
have ∗: ∃ *C*. *countable C* ∧ *C* ⊆ *K* ∧
pairwise (λ*i j*. *disjnt* (*cball* (*a i*) (*r i*)) (*cball* (*a j*) (*r j*))) *C* ∧
negligible(*S* − (⋃ *i* ∈ *C*. *cball* (*a i*) (*r i*)))
if *r01*: ⋀*i*. *i* ∈ *K* ⟹ 0 < *r i* ∧ *r i* ≤ 1
and *Sd*: ⋀*x d*. ⟦*x* ∈ *S*; 0 < *d*⟧ ⟹ ∃ *i*. *i* ∈ *K* ∧ *x* ∈ *cball* (*a i*) (*r i*) ∧ *r i* < *d*
for *K r* **and** *a* :: *'a* ⇒ *'n*
proof −
obtain *C* **where** *C*: *countable C C* ⊆ *K*
and *pwC*: *pairwise* (λ*i j*. *disjnt* (*cball* (*a i*) (*r i*)) (*cball* (*a j*) (*r j*))) *C*
and *cov*: ⋀*i*. *i* ∈ *K* ⟹ ∃ *j*. *j* ∈ *C* ∧ ¬ *disjnt* (*cball* (*a i*) (*r i*)) (*cball* (*a j*) (*r j*)) ∧
cball (*a i*) (*r i*) ⊆ *ball* (*a j*) (5 ∗ *r j*)
by (*rule Vitali_covering_lemma_cballs_balls* [*of K r 1 a*]) (*auto simp*: *r01*)
have *ar_injective*: ⋀*x y*. ⟦*x* ∈ *C*; *y* ∈ *C*; *a x* = *a y*; *r x* = *r y*⟧ ⟹ *x* = *y*
using ‹*C* ⊆ *K*› *pwC cov*
by (*force simp*: *pairwise_def disjnt_def*)
show *?thesis*
proof (*intro exI conjI*)
show *negligible* (*S* − (⋃ *i*∈*C*. *cball* (*a i*) (*r i*)))
proof (*clarsimp simp*: *negligible_on_intervals* [*of S*−*T* **for** *T*])
fix *l u*
show *negligible* ((*S* − (⋃ *i*∈*C*. *cball* (*a i*) (*r i*))) ∩ *cbox l u*)
unfolding *negligible_outer_le*
proof (*intro allI impI*)
fix *e*::*real*
assume *e* > 0
define *D* **where** *D* ≡ {*i* ∈ *C*. ¬ *disjnt* (*ball*(*a i*) (5 ∗ *r i*)) (*cbox l u*)}
then have *D* ⊆ *C*
by *auto*
have *countable D*
unfolding *D_def* **using** ‹*countable C*› **by** *simp*
have *UD*: (⋃ *i*∈*D*. *cball* (*a i*) (*r i*)) ∈ *lmeasurable*

```
      proof (rule fmeasurableI2)
        show cbox (l - 6 *R One) (u + 6 *R One) ∈ lmeasurable
          by blast
        have y ∈ cbox (l - 6 *R One) (u + 6 *R One)
         if i ∈ C and x: x ∈ cbox l u and ai: dist (a i) y ≤ r i dist (a i) x < 5
* r i
          for i x y
        proof -
          have d6: dist y x < 6 * r i
            using dist_triangle3 [of y x a i] that by linarith
          show ?thesis
          proof (clarsimp simp: mem_box algebra_simps)
            fix j::'n
            assume j: j ∈ Basis
            then have xyj: |x · j - y · j| ≤ dist y x
            by (metis Basis_le_norm dist_commute dist_norm inner_diff_left)
            have l · j ≤ x · j
              using ‹j ∈ Basis› mem_box ‹x ∈ cbox l u› by blast
            also have ... ≤ y · j + 6 * r i
              using d6 xyj by (auto simp: algebra_simps)
            also have ... ≤ y · j + 6
              using r01 [of i] ‹C ⊆ K› ‹i ∈ C› by auto
            finally have l: l · j ≤ y · j + 6 .
            have y · j ≤ x · j + 6 * r i
              using d6 xyj by (auto simp: algebra_simps)
            also have ... ≤ u · j + 6 * r i
              using j  x by (auto simp: mem_box)
            also have ... ≤ u · j + 6
              using r01 [of i] ‹C ⊆ K› ‹i ∈ C› by auto
            finally have u: y · j ≤ u · j + 6 .
            show l · j ≤ y · j + 6 ∧ y · j ≤ u · j + 6
              using l u by blast
          qed
        qed
       then show (⋃i∈D. cball (a i) (r i)) ⊆ cbox (l - 6 *R One) (u + 6 *R
One)
          by (force simp: D_def disjnt_def)
        show (⋃i∈D. cball (a i) (r i)) ∈ sets lebesgue
          using ‹countable D› by auto
      qed
      obtain D1 where D1 ⊆ D finite D1
        and measD1: ?μ (⋃i∈D. cball (a i) (r i)) - e / 5 ^ DIM('n) < ?μ
(⋃i∈D1. cball (a i) (r i))
      proof (rule measure_countable_Union_approachable [where e = e / 5 ^
(DIM('n))])
        show countable ((λi. cball (a i) (r i)) ‘ D)
          using ‹countable D› by auto
        show ⋀d. d ∈ (λi. cball (a i) (r i)) ‘ D ⟹ d ∈ lmeasurable
          by auto
```

```
      show ⋀D'. ⟦D' ⊆ (λi. cball (a i) (r i)) ‘ D; finite D'⟧ ⟹ ?μ (⋃ D') ≤
?μ (⋃i∈D. cball (a i) (r i))
        by (fastforce simp add: intro!: measure_mono_fmeasurable UD)
      qed (use ‹e > 0› in ‹auto dest: finite_subset_image›)
      show ∃ T. (S − (⋃i∈C. cball (a i) (r i))) ∩
            cbox l u ⊆ T ∧ T ∈ lmeasurable ∧ ?μ T ≤ e
      proof (intro exI conjI)
        show (S − (⋃i∈C. cball (a i) (r i))) ∩ cbox l u ⊆ (⋃i∈D − D1. ball
(a i) (5 * r i))
        proof clarify
          fix x
          assume x: x ∈ cbox l u x ∈ S x ∉ (⋃i∈C. cball (a i) (r i))
          have closed (⋃i∈D1. cball (a i) (r i))
            using ‹finite D1› by blast
          moreover have x ∉ (⋃j∈D1. cball (a j) (r j))
            using x ‹D1 ⊆ D› unfolding D_def by blast
          ultimately obtain q where q > 0 and q: ball x q ⊆ − (⋃i∈D1. cball
(a i) (r i))
            by (metis (no_types, lifting) ComplI open_contains_ball closed_def)
          obtain i where i ∈ K and xi: x ∈ cball (a i) (r i) and ri: r i < q/2
            using Sd [OF ‹x ∈ S›] ‹q > 0› half_gt_zero by blast
          then obtain j where j ∈ C
                          and nondisj: ¬ disjnt (cball (a i) (r i)) (cball (a j) (r j))
                          and sub5j:  cball (a i) (r i) ⊆ ball (a j) (5 * r j)
            using cov [OF ‹i ∈ K›] by metis
          show x ∈ (⋃i∈D − D1. ball (a i) (5 * r i))
          proof
            show j ∈ D − D1
            proof
              show j ∈ D
                  using ‹j ∈ C› sub5j ‹x ∈ cbox l u› xi by (auto simp: D_def
disjnt_def)
              obtain y where yi: dist (a i) y ≤ r i and yj: dist (a j) y ≤ r j
                using disjnt_def nondisj by fastforce
              have dist x y ≤ r i + r i
                by (metis add_mono dist_commute dist_triangle_le mem_cball
xi yi)
              also have ... < q
                using ri by linarith
              finally have y ∈ ball x q
                by simp
              with yj q show j ∉ D1
                by (auto simp: disjoint_UN_iff)
            qed
            show x ∈ ball (a j) (5 * r j)
              using xi sub5j by blast
          qed
        qed
        have 3: ?μ (⋃i∈D2. ball (a i) (5 * r i)) ≤ e
```

```
          if D2: D2 ⊆ D − D1 and finite D2 for D2
        proof −
          have rgt0: 0 < r i if i ∈ D2 for i
            using ‹C ⊆ K› D_def ‹i ∈ D2› D2 r01
            by (simp add: subset_iff)
          then have inj: inj_on (λi. ball (a i) (5 ∗ r i)) D2
                using ‹C ⊆ K› D2 by (fastforce simp: inj_on_def D_def
ball_eq_ball_iff intro: ar_injective)
           have ?μ (⋃i∈D2. ball (a i) (5 ∗ r i)) ≤ sum (?μ) ((λi. ball (a i) (5
∗ r i)) ‘ D2)
            using that by (force intro: measure_Union_le)
          also have ... = (∑i∈D2. ?μ (ball (a i) (5 ∗ r i)))
            by (simp add: comm_monoid_add_class.sum.reindex [OF inj])
          also have ... = (∑i∈D2. 5 ^ DIM('n) ∗ ?μ (ball (a i) (r i)))
          proof (rule sum.cong [OF refl])
            fix i assume i ∈ D2
            thus ?μ (ball (a i) (5 ∗ r i)) = 5 ^ DIM('n) ∗ ?μ (ball (a i) (r i))
              using content_ball_conv_unit_ball[of 5 ∗ r i a i]
                    content_ball_conv_unit_ball[of r i a i] rgt0[of i] by auto
          qed
          also have ... = (∑i∈D2. ?μ (ball (a i) (r i))) ∗ 5 ^ DIM('n)
            by (simp add: sum_distrib_left mult.commute)
          finally have ?μ (⋃i∈D2. ball (a i) (5 ∗ r i)) ≤ (∑i∈D2. ?μ (ball (a
i) (r i))) ∗ 5 ^ DIM('n) .
          moreover have (∑i∈D2. ?μ (ball (a i) (r i))) ≤ e / 5 ^ DIM('n)
          proof −
            have D12_dis: ((⋃x∈D1. cball (a x) (r x)) ∩ (⋃x∈D2. cball (a x)
(r x))) ≤ {}
            proof clarify
              fix w d1 d2
              assume d1 ∈ D1 w d1 d2 ∈ cball (a d1) (r d1) d2 ∈ D2 w d1 d2
∈ cball (a d2) (r d2)
              then show w d1 d2 ∈ {}
                  by (metis DiffE disjnt_iff subsetCE D2 ‹D1 ⊆ D› ‹D ⊆ C›
pairwiseD [OF pwC, of d1 d2])
            qed
            have inj: inj_on (λi. cball (a i) (r i)) D2
           using rgt0 D2 ‹D ⊆ C› by (force simp: inj_on_def cball_eq_cball_iff
intro!: ar_injective)
            have ds: disjoint ((λi. cball (a i) (r i)) ‘ D2)
              using D2 ‹D ⊆ C› by (auto intro: pairwiseI pairwiseD [OF pwC])
             have (∑i∈D2. ?μ (ball (a i) (r i))) = (∑i∈D2. ?μ (cball (a i) (r
i)))
              by (simp add: content_cball_conv_ball)
            also have ... = sum ?μ ((λi. cball (a i) (r i)) ‘ D2)
              by (simp add: comm_monoid_add_class.sum.reindex [OF inj])
            also have ... = ?μ (⋃i∈D2. cball (a i) (r i))
               by (auto intro: measure_Union′ [symmetric] ds simp add: ‹finite
D2›)
```

```
            finally have ?μ (⋃i∈D1. cball (a i) (r i)) + (∑i∈D2. ?μ (ball (a
i) (r i))) =
                         ?μ (⋃i∈D1. cball (a i) (r i)) + ?μ (⋃i∈D2. cball (a i)
(r i))
              by simp
            also have ... = ?μ (⋃ i ∈ D1 ∪ D2. cball (a i) (r i))
             using D12_dis by (simp add: measure_Un3 ‹finite D1› ‹finite D2›
fmeasurable.finite_UN)
            also have ... ≤ ?μ (⋃i∈D. cball (a i) (r i))
           using D2 ‹D1 ⊆ D› by (fastforce intro!: measure_mono_fmeasurable
[OF _ _ UD] ‹finite D1› ‹finite D2›)
            finally have ?μ (⋃i∈D1. cball (a i) (r i)) + (∑i∈D2. ?μ (ball (a
i) (r i))) ≤ ?μ (⋃i∈D. cball (a i) (r i)) .
            with measD1 show ?thesis
              by simp
          qed
            ultimately show ?thesis
              by (simp add: field_split_simps)
        qed
        have co: countable (D − D1)
          by (simp add: ‹countable D›)
        show (⋃i∈D − D1. ball (a i) (5 ∗ r i)) ∈ lmeasurable
          using ‹e > 0› by (auto simp: fmeasurable_UN_bound [OF co _ 3])
        show ?μ (⋃i∈D − D1. ball (a i) (5 ∗ r i)) ≤ e
          using ‹e > 0› by (auto simp: measure_UN_bound [OF co _ 3])
      qed
    qed
  qed
 qed (use C pwC in auto)
qed
define K' where K' ≡ {i ∈ K. r i ≤ 1}
have 1: ⋀i. i ∈ K' ⟹ 0 < r i ∧ r i ≤ 1
 using K'_def r by auto
have 2: ∃ i. i ∈ K' ∧ x ∈ cball (a i) (r i) ∧ r i < d
 if x ∈ S ∧ 0 < d for x d
 using that by (auto simp: K'_def dest!: S [where d = min d 1])
have K' ⊆ K
 using K'_def by auto
then show thesis
 using ∗ [OF 1 2] that by fastforce
qed


theorem Vitali_covering_theorem_balls:
 fixes a :: 'a ⇒ 'b::euclidean_space
 assumes S: ⋀x d. ⟦x ∈ S; 0 < d⟧ ⟹ ∃ i. i ∈ K ∧ x ∈ ball (a i) (r i) ∧ r i < d
 obtains C where countable C C ⊆ K
   pairwise (λi j. disjnt (ball (a i) (r i)) (ball (a j) (r j))) C
   negligible(S − (⋃ i ∈ C. ball (a i) (r i)))
```

proof −
have *1*: $\exists i.\ i \in \{i \in K.\ 0 < r\ i\} \wedge x \in cball\ (a\ i)\ (r\ i) \wedge r\ i < d$
if *xd*: $x \in S\ d > 0$ **for** *x d*
by (*metis* (*mono_tags*, *lifting*) *assms ball_eq_empty less_eq_real_def mem_Collect_eq mem_ball mem_cball not_le xd*(*1*) *xd*(*2*))
obtain *C* **where** *C*: *countable C* $C \subseteq K$
and *pw*: *pairwise* $(\lambda i\ j.\ disjnt\ (cball\ (a\ i)\ (r\ i))\ (cball\ (a\ j)\ (r\ j)))\ C$
and *neg*: *negligible*$(S - (\bigcup i \in C.\ cball\ (a\ i)\ (r\ i)))$
by (*rule Vitali_covering_theorem_cballs* [*of* $\{i \in K.\ 0 < r\ i\}$ *r S a*, *OF _ 1*]) *auto*
show *thesis*
proof
show *pairwise* $(\lambda i\ j.\ disjnt\ (ball\ (a\ i)\ (r\ i))\ (ball\ (a\ j)\ (r\ j)))\ C$
apply (*rule pairwise_mono* [*OF pw*])
apply (*auto simp*: *disjnt_def*)
by (*meson disjoint_iff_not_equal less_imp_le mem_cball*)
have *negligible* $(\bigcup i \in C.\ sphere\ (a\ i)\ (r\ i))$
by (*auto intro*: *negligible_sphere* ‹*countable C*›)
then have *negligible* $(S - (\bigcup i \in C.\ cball(a\ i)(r\ i)) \cup (\bigcup i \in C.\ sphere\ (a\ i)\ (r\ i)))$
by (*rule negligible_Un* [*OF neg*])
then show *negligible* $(S - (\bigcup i \in C.\ ball\ (a\ i)\ (r\ i)))$
by (*rule negligible_subset*) *force*
qed (*use C* **in** *auto*)
qed

lemma *negligible_eq_zero_density_alt*:
negligible S $\longleftrightarrow$
$(\forall x \in S.\ \forall e > 0.$
$\exists d\ U.\ 0 < d \wedge d \le e \wedge S \cap ball\ x\ d \subseteq U \wedge$
$U \in lmeasurable \wedge measure\ lebesgue\ U < e * measure\ lebesgue\ (ball\ x\ d))$
(**is** _ = $(\forall x \in S.\ \forall e > 0.\ ?Q\ x\ e)$)
proof (*intro iffI ballI allI impI*)
fix *x* **and** *e* :: *real*
assume *negligible S* **and** $x \in S$ **and** $e > 0$
then
show $\exists d\ U.\ 0 < d \wedge d \le e \wedge S \cap ball\ x\ d \subseteq U \wedge U \in lmeasurable \wedge$
measure lebesgue U $< e *$ *measure lebesgue* (*ball x d*)
apply (*rule_tac x*=*e* **in** *exI*)
apply (*rule_tac x*=$S \cap ball\ x\ e$ **in** *exI*)
apply (*auto simp*: *negligible_imp_measurable negligible_Int negligible_imp_measure0 zero_less_measure_iff*
intro: *mult_pos_pos content_ball_pos*)
done
next
assume *R* [*rule_format*]: $\forall x \in S.\ \forall e > 0.\ ?Q\ x\ e$
let $?\mu$ = *measure lebesgue*

```
  have ∃U. openin (top_of_set S) U ∧ z ∈ U ∧ negligible U
    if z ∈ S for z
  proof (intro exI conjI)
    show openin (top_of_set S) (S ∩ ball z 1)
      by (simp add: openin_open_Int)
    show z ∈ S ∩ ball z 1
      using ‹z ∈ S› by auto
    show negligible (S ∩ ball z 1)
    proof (clarsimp simp: negligible_outer_le)
      fix e :: real
      assume e > 0
      let ?K = {(x,d). x ∈ S ∧ 0 < d ∧ ball x d ⊆ ball z 1 ∧
                       (∃U. S ∩ ball x d ⊆ U ∧ U ∈ lmeasurable ∧
                            ?μ U < e / ?μ (ball z 1) * ?μ (ball x d))}
      obtain C where countable C and Csub: C ⊆ ?K
        and pwC: pairwise (λi j. disjnt (ball (fst i) (snd i)) (ball (fst j) (snd j))) C
        and negC: negligible((S ∩ ball z 1) - (⋃i ∈ C. ball (fst i) (snd i)))
      proof (rule Vitali_covering_theorem_balls [of S ∩ ball z 1 ?K fst snd])
        fix x and d :: real
        assume x: x ∈ S ∩ ball z 1 and d > 0
        obtain k where k > 0 and k: ball x k ⊆ ball z 1
          by (meson Int_iff open_ball openE x)
        let ?ε = min (e / ?μ (ball z 1) / 2) (min (d / 2) k)
       obtain r U where r: r > 0 r ≤ ?ε and U: S ∩ ball x r ⊆ U U ∈ lmeasurable
          and mU: ?μ U < ?ε * ?μ (ball x r)
       using R [of x ?ε] ‹d > 0› ‹e > 0› ‹k > 0› x by (auto simp: content_ball_pos)
        show ∃i. i ∈ ?K ∧ x ∈ ball (fst i) (snd i) ∧ snd i < d
        proof (rule exI [of _ (x,r)], simp, intro conjI exI)
          have ball x r ⊆ ball x k
            using r by (simp add: ball_subset_ball_iff)
          also have ... ⊆ ball z 1
            using ball_subset_ball_iff k by auto
          finally show ball x r ⊆ ball z 1 .
          have ?ε * ?μ (ball x r) ≤ e * content (ball x r) / content (ball z 1)
            using r ‹e > 0› by (simp add: ord_class.min_def field_split_simps
content_ball_pos)
          with mU show ?μ U < e * content (ball x r) / content (ball z 1)
            by auto
        qed (use r U x in auto)
      qed
      have ∃U. case p of (x,d) ⇒ S ∩ ball x d ⊆ U ∧
                         U ∈ lmeasurable ∧ ?μ U < e / ?μ (ball z 1) * ?μ (ball x d)
        if p ∈ C for p
        using that Csub unfolding case_prod_unfold by blast
      then obtain U where U:
            ⋀p. p ∈ C ⟹
                case p of (x,d) ⇒ S ∩ ball x d ⊆ U p ∧
                U p ∈ lmeasurable ∧ ?μ (U p) < e / ?μ (ball z 1) * ?μ (ball x d)
        by (rule that [OF someI_ex])
```

```
    let ?T = ((S ∩ ball z 1) − (⋃(x,d)∈C. ball x d)) ∪ ⋃(U ‘ C)
    show ∃ T. S ∩ ball z 1 ⊆ T ∧ T ∈ lmeasurable ∧ ?μ T ≤ e
    proof (intro exI conjI)
      show S ∩ ball z 1 ⊆ ?T
        using U by fastforce
      { have Um: U i ∈ lmeasurable if i ∈ C for i
          using that U by blast
        have lee: ?μ (⋃i∈I. U i) ≤ e if I ⊆ C finite I for I
        proof −
          have ?μ (⋃(x,d)∈I. ball x d) ≤ ?μ (ball z 1)
            apply (rule measure_mono_fmeasurable)
                using ‹I ⊆ C› ‹finite I› Csub by (force simp: prod.case_eq_if
sets.finite_UN)+
          then have le1: (?μ (⋃(x,d)∈I. ball x d) / ?μ (ball z 1)) ≤ 1
            by (simp add: content_ball_pos)
          have ?μ (⋃i∈I. U i) ≤ (∑i∈I. ?μ (U i))
            using that U by (blast intro: measure_UNION_le)
          also have ... ≤ (∑(x,r)∈I. e / ?μ (ball z 1) ∗ ?μ (ball x r))
            by (rule sum_mono) (use ‹I ⊆ C› U in force)
          also have ... = (e / ?μ (ball z 1)) ∗ (∑(x,r)∈I. ?μ (ball x r))
            by (simp add: case_prod_app prod.case_distrib sum_distrib_left)
          also have ... = e ∗ (?μ (⋃(x,r)∈I. ball x r) / ?μ (ball z 1))
            apply (subst measure_UNION′)
          using that pwC by (auto simp: case_prod_unfold elim: pairwise_mono)
          also have ... ≤ e
             by (metis mult.commute mult.left_neutral mult_le_cancel_iff1 ‹e >
0› le1)
          finally show ?thesis .
        qed
        have ⋃(U ‘ C) ∈ lmeasurable ?μ (⋃(U ‘ C)) ≤ e
          using ‹e > 0› Um lee
              by(auto intro!: fmeasurable_UN_bound [OF ‹countable C›] mea-
sure_UN_bound [OF ‹countable C›])
      }
      moreover have ?μ ?T = ?μ (⋃(U ‘ C))
      proof (rule measure_negligible_symdiff [OF ‹⋃(U ‘ C) ∈ lmeasurable›])
        show negligible((⋃(U ‘ C) − ?T) ∪ (?T − ⋃(U ‘ C)))
          by (force intro!: negligible_subset [OF negC])
      qed
      ultimately show ?T ∈ lmeasurable  ?μ ?T ≤ e
            by (simp_all add: fmeasurable.Un negC negligible_imp_measurable
split_def)
    qed
  qed
 qed
 with locally_negligible_alt show negligible S
   by metis
qed
```

proposition *negligible_eq_zero_density*:
  negligible S $\longleftrightarrow$
  $(\forall x{\in}S.\ \forall r{>}0.\ \forall e{>}0.\ \exists d.\ 0 < d \wedge d \leq r \wedge$
    $(\exists U.\ S \cap ball\ x\ d \subseteq U \wedge U \in lmeasurable \wedge measure\ lebesgue\ U < e * measure\ lebesgue\ (ball\ x\ d)))$
proof −
  let *?Q* = $\lambda x\ d\ e.\ \exists U.\ S \cap ball\ x\ d \subseteq U \wedge U \in lmeasurable \wedge measure\ lebesgue\ U < e * content\ (ball\ x\ d)$
  have $(\forall e{>}0.\ \exists d{>}0.\ d \leq e \wedge$ *?Q x d e*$) = (\forall r{>}0.\ \forall e{>}0.\ \exists d{>}0.\ d \leq r \wedge$ *?Q x d e*$)$
    if $x \in S$ **for** x
  proof (*intro iffI allI impI*)
    fix *r* :: *real* **and** *e* :: *real*
    assume *L* [*rule_format*]: $\forall e{>}0.\ \exists d{>}0.\ d \leq e \wedge$ *?Q x d e* **and** $r > 0\ e > 0$
    show $\exists d{>}0.\ d \leq r \wedge$ *?Q x d e*
      using *L* [*of min r e*] **apply** (*rule ex_forward*)
     using ‹$r > 0$› ‹$e > 0$› **by** (*auto intro*: *less_le_trans elim*!: *ex_forward simp*: *content_ball_pos*)
  qed *auto*
  then show *?thesis*
    by (*force simp*: *negligible_eq_zero_density_alt*)
qed

end

6.41 Change of Variables Theorems

theory *Change_Of_Vars*
  imports *Vitali_Covering_Theorem Determinants*

begin

6.41.1 Measurable Shear and Stretch

proposition
  fixes *a* :: *real^'n*
  assumes $m \neq n$ **and** *ab_ne*: *cbox a b* $\neq \{\}$ **and** *an*: $0 \leq a\$n$
  shows *measurable_shear_interval*: $(\lambda x.\ \chi\ i.\ if\ i = m\ then\ x\$m + x\$n\ else\ x\$i)$ ‘ *(cbox a b)* $\in$ *lmeasurable*
      (**is** *?f* ‘ *_* $\in$ *_*)
   and *measure_shear_interval*: *measure lebesgue* $((\lambda x.\ \chi\ i.\ if\ i = m\ then\ x\$m + x\$n\ else\ x\$i)$ ‘ *cbox a b*)
            = *measure lebesgue* (*cbox a b*) (**is** *?Q*)
proof −
  have *lin*: *linear ?f*
    by (*rule linearI*) (*auto simp*: *plus_vec_def scaleR_vec_def algebra_simps*)
  show *fab*: *?f* ‘ *cbox a b* $\in$ *lmeasurable*
    by (*simp add*: *lin measurable_linear_image_interval*)
  let *?c* = $\chi\ i.\ if\ i = m\ then\ b\$m + b\$n\ else\ b\$i$

let *?mn = axis m 1 − axis n (1::real)*
have *eq1*: *measure lebesgue (cbox a ?c)*
= measure lebesgue (?f ‘ cbox a b)
+ measure lebesgue (cbox a ?c ∩ {x. ?mn · x ≤ a$m})
+ measure lebesgue (cbox a ?c ∩ {x. ?mn · x ≥ b$m})
proof (*rule measure_Un3_negligible*)
show *cbox a ?c ∩ {x. ?mn · x ≤ a$m} ∈ lmeasurable cbox a ?c ∩ {x. ?mn · x ≥ b$m} ∈ lmeasurable*
by (*auto simp*: *convex_Int convex_halfspace_le convex_halfspace_ge bounded_Int measurable_convex*)
have *negligible {x. ?mn · x = a$m}*
by (*metis ‹m ≠ n› axis_index_axis eq_iff_diff_eq_0 negligible_hyperplane*)
moreover have *?f ‘ cbox a b ∩ (cbox a ?c ∩ {x. ?mn · x ≤ a $ m}) ⊆ {x. ?mn · x = a$m}*
using *‹m ≠ n› antisym_conv* **by** (*fastforce simp*: *algebra_simps mem_box_cart inner_axis'*)
ultimately show *negligible ((?f ‘ cbox a b) ∩ (cbox a ?c ∩ {x. ?mn · x ≤ a $ m}))*
by (*rule negligible_subset*)
have *negligible {x. ?mn · x = b$m}*
by (*metis ‹m ≠ n› axis_index_axis eq_iff_diff_eq_0 negligible_hyperplane*)
moreover have *(?f ‘ cbox a b) ∩ (cbox a ?c ∩ {x. ?mn · x ≥ b$m}) ⊆ {x. ?mn · x = b$m}*
using *‹m ≠ n› antisym_conv* **by** (*fastforce simp*: *algebra_simps mem_box_cart inner_axis'*)
ultimately show *negligible (?f ‘ cbox a b ∩ (cbox a ?c ∩ {x. ?mn · x ≥ b$m}))*
by (*rule negligible_subset*)
have *negligible {x. ?mn · x = b$m}*
by (*metis ‹m ≠ n› axis_index_axis eq_iff_diff_eq_0 negligible_hyperplane*)
moreover have *(cbox a ?c ∩ {x. ?mn · x ≤ a $ m} ∩ (cbox a ?c ∩ {x. ?mn · x ≥ b$m})) ⊆ {x. ?mn · x = b$m}*
using *‹m ≠ n› ab_ne*
apply (*clarsimp simp*: *algebra_simps mem_box_cart inner_axis'*)
by (*smt* (*verit*, *ccfv_SIG*) *interval_ne_empty_cart*(*1*))
ultimately show *negligible (cbox a ?c ∩ {x. ?mn · x ≤ a $ m} ∩ (cbox a ?c ∩ {x. ?mn · x ≥ b$m}))*
by (*rule negligible_subset*)
show *?f ‘ cbox a b ∪ cbox a ?c ∩ {x. ?mn · x ≤ a $ m} ∪ cbox a ?c ∩ {x. ?mn · x ≥ b$m} = cbox a ?c* (**is** *?lhs = _*)
proof
show *?lhs ⊆ cbox a ?c*
by (*auto simp*: *mem_box_cart add_mono*) (*meson add_increasing2 an order_trans*)
show *cbox a ?c ⊆ ?lhs*
apply (*clarsimp simp*: *algebra_simps image_iff inner_axis' lambda_add_Galois* [*OF ‹m ≠ n›*])
by (*smt* (*verit*, *del_insts*) *mem_box_cart*(*2*) *vec_lambda_beta*)
qed
qed (*fact fab*)

let *?d* = χ *i. if i = m then a $ m − b $ m else 0*
have *eq2*: *measure lebesgue (cbox a ?c ∩ {x. ?mn · x ≤ a $ m}) + measure lebesgue (cbox a ?c ∩ {x. ?mn · x ≥ b$m})*
= measure lebesgue (cbox a (χ i. if i = m then a $ m + b $ n else b $ i))
proof (*rule measure_translate_add[of cbox a ?c ∩ {x. ?mn · x ≤ a$m} cbox a ?c ∩ {x. ?mn · x ≥ b$m}*
(χ i. if i = m then a$m − b$m else 0) cbox a (χ i. if i = m then a$m + b$n else b$i)])
show (*cbox a ?c ∩ {x. ?mn · x ≤ a$m}*) ∈ *lmeasurable*
cbox a ?c ∩ {x. ?mn · x ≥ b$m} ∈ *lmeasurable*
by (*auto simp*: *convex_Int convex_halfspace_le convex_halfspace_ge bounded_Int measurable_convex*)
have ⋀*x*. ⟦*x $ n + a $ m ≤ x $ m*⟧
⟹ *x* ∈ (+) (*χ i. if i = m then a $ m − b $ m else 0*) ‘ {*x. x $ n + b $ m ≤ x $ m*}
using ‹*m ≠ n*›
by (*rule_tac x=x − (χ i. if i = m then a$m − b$m else 0*) **in** *image_eqI*)
(*simp_all add*: *mem_box_cart*)
then have *imeq*: (+) *?d* ‘ {*x. b $ m ≤ ?mn · x*} = {*x. a $ m ≤ ?mn · x*}
using ‹*m ≠ n*› **by** (*auto simp*: *mem_box_cart inner_axis′ algebra_simps*)
have ⋀*x*. ⟦*0 ≤ a $ n*; *x $ n + a $ m ≤ x $ m*;
∀ *i*. *i ≠ m* ⟶ *a $ i ≤ x $ i ∧ x $ i ≤ b $ i*⟧
⟹ *a $ m ≤ x $ m*
using ‹*m ≠ n*› **by** *force*
then have (+) *?d* ‘ (*cbox a ?c ∩ {x. b $ m ≤ ?mn · x}*)
= *cbox a (χ i. if i = m then a $ m + b $ n else b $ i) ∩ {x. a $ m ≤ ?mn · x}*
using *an ab_ne*
apply (*simp add*: *cbox_translation* [*symmetric*] *translation_Int interval_ne_empty_cart imeq*)
apply (*auto simp*: *mem_box_cart inner_axis′ algebra_simps if_distrib all_if_distrib*)
by (*metis* (*full_types*) *add_mono mult_2_right*)
then show *cbox a ?c ∩ {x. ?mn · x ≤ a $ m} ∪*
(+) *?d* ‘ (*cbox a ?c ∩ {x. b $ m ≤ ?mn · x}*) =
cbox a (χ i. if i = m then a $ m + b $ n else b $ i) (**is** *?lhs = ?rhs*)
using *an* ‹*m ≠ n*›
apply (*auto simp*: *mem_box_cart inner_axis′ algebra_simps if_distrib all_if_distrib*, *force*)
apply (*drule_tac x=n* **in** *spec*)+
by (*meson ab_ne add_mono_thms_linordered_semiring*(*3*) *dual_order.trans interval_ne_empty_cart*(*1*))
have *negligible*{*x. ?mn · x = a$m*}
by (*metis* ‹*m ≠ n*› *axis_index_axis eq_iff_diff_eq_0 negligible_hyperplane*)
moreover have (*cbox a ?c ∩ {x. ?mn · x ≤ a $ m} ∩*
(+) *?d* ‘ (*cbox a ?c ∩ {x. b $ m ≤ ?mn · x}*)) ⊆ {*x. ?mn · x = a$m*}
using ‹*m ≠ n*› *antisym_conv* **by** (*fastforce simp*: *algebra_simps mem_box_cart inner_axis′*)

```
    ultimately show negligible (cbox a ?c ∩ {x. ?mn · x ≤ a $ m} ∩
                                (+) ?d ' (cbox a ?c ∩ {x. b $ m ≤ ?mn · x}))
      by (rule negligible_subset)
  qed
  have ac_ne: cbox a ?c ≠ {}
   by (smt (verit, del_insts) ab_ne an interval_ne_empty_cart(1) vec_lambda_beta)
  have ax_ne: cbox a (χ i. if i = m then a $ m + b $ n else b $ i) ≠ {}
    using ab_ne an
    by (smt (verit, ccfv_threshold) interval_ne_empty_cart(1) vec_lambda_beta)
  have eq3: measure lebesgue (cbox a ?c) = measure lebesgue (cbox a (χ i. if i =
m then a$m + b$n else b$i)) + measure lebesgue (cbox a b)
   by (simp add: content_cbox_if_cart ab_ne ac_ne ax_ne algebra_simps prod.delta_remove
          if_distrib [of λu. u − z for z] prod.remove)
  show ?Q
    using eq1 eq2 eq3 by (simp add: algebra_simps)
qed


proposition
  fixes S :: (real^'n) set
  assumes S ∈ lmeasurable
  shows measurable_stretch: ((λx. χ k. m k * x$k) ' S) ∈ lmeasurable (is  ?f ' S
∈ _)
    and measure_stretch: measure lebesgue ((λx. χ k. m k * x$k) ' S) = |prod m
UNIV| * measure lebesgue S
    (is ?MEQ)
proof −
  have (?f ' S) ∈ lmeasurable ∧ ?MEQ
  proof (cases ∀ k. m k ≠ 0)
    case True
    have m0: 0 < |prod m UNIV|
      using True by simp
    have (indicat_real (?f ' S) has_integral |prod m UNIV| * measure lebesgue S)
UNIV
    proof (clarsimp simp add: has_integral_alt [where i=UNIV])
      fix e :: real
      assume e > 0
      have (indicat_real S has_integral (measure lebesgue S)) UNIV
        using assms lmeasurable_iff_has_integral by blast
      then obtain B where B>0
        and B: ⋀a b. ball 0 B ⊆ cbox a b ⟹
                     ∃ z. (indicat_real S has_integral z) (cbox a b) ∧
                          |z − measure lebesgue S| < e / |prod m UNIV|
          by (simp add: has_integral_alt [where i=UNIV]) (metis (full_types)
divide_pos_pos m0  m0 ‹e > 0›)
      show ∃ B>0. ∀ a b. ball 0 B ⊆ cbox a b ⟶
                 (∃ z. (indicat_real (?f ' S) has_integral z) (cbox a b) ∧
                       |z − |prod m UNIV| * measure lebesgue S| < e)
      proof (intro exI conjI allI)
```

```
      let ?C = Max (range (λk. |m k|)) * B
      show ?C > 0
        using True ‹B > 0› by (simp add: Max_gr_iff)
      show ball 0 ?C ⊆ cbox u v ⟶
             (∃z. (indicat_real (?f ‘ S) has_integral z) (cbox u v) ∧
                  |z - |prod m UNIV| * measure lebesgue S| < e) for u v
      proof
        assume uv: ball 0 ?C ⊆ cbox u v
        with ‹?C > 0› have cbox_ne: cbox u v ≠ {}
          using centre_in_ball by blast
        let ?α = λk. u$k / m k
        let ?β = λk. v$k / m k
        have invm0: ⋀k. inverse (m k) ≠ 0
          using True by auto
        have ball 0 B ⊆ (λx. χ k. x $ k / m k) ‘ ball 0 ?C
        proof clarsimp
          fix x :: real^'n
          assume x: norm x < B
          have [simp]: |Max (range (λk. |m k|))| = Max (range (λk. |m k|))
            by (meson Max_ge abs_ge_zero abs_of_nonneg finite finite_imageI
order_trans rangeI)
          have norm (χ k. m k * x $ k) ≤ norm (Max (range (λk. |m k|)) *R x)
            by (rule norm_le_componentwise_cart) (auto simp: abs_mult intro:
mult_right_mono)
          also have ... < ?C
            using x ‹0 < (MAX k. |m k|) * B› ‹0 < B› zero_less_mult_pos2 by
fastforce
          finally have norm (χ k. m k * x $ k) < ?C .
          then show x ∈ (λx. χ k. x $ k / m k) ‘ ball 0 ?C
            using stretch_Galois [of inverse ∘ m] True by (auto simp: image_iff
field_simps)
        qed
        then have Bsub: ball 0 B ⊆ cbox (χ k. min (?α k) (?β k)) (χ k. max (?α
k) (?β k))
            using cbox_ne uv image_stretch_interval_cart [of inverse ∘ m u v,
symmetric]
          by (force simp: field_simps)
        obtain z where zint: (indicat_real S has_integral z) (cbox (χ k. min (?α
k) (?β k)) (χ k. max (?α k) (?β k)))
                  and zless: |z - measure lebesgue S| < e / |prod m UNIV|
          using B [OF Bsub] by blast
        have ind: indicat_real (?f ‘ S) = (λx. indicator S (χ k. x$k / m k))
          using True stretch_Galois [of m] by (force simp: indicator_def)
        show ∃z. (indicat_real (?f ‘ S) has_integral z) (cbox u v) ∧
                    |z - |prod m UNIV| * measure lebesgue S| < e
        proof (simp add: ind, intro conjI exI)
          have ((λx. indicat_real S (χ k. x $ k/ m k)) has_integral z *R |prod m
UNIV|)
                ((λx. χ k. x $ k * m k) ‘ cbox (χ k. min (?α k) (?β k)) (χ k. max
```

```
(?α k) (?β k)))
            using True has_integral_stretch_cart [OF zint, of inverse ∘ m]
            by (simp add: field_simps prod_dividef)
          moreover have ((λx. χ k. x $ k * m k) ' cbox (χ k. min (?α k) (?β k))
(χ k. max (?α k) (?β k))) = cbox u v
          using True image_stretch_interval_cart [of inverse ∘ m u v, symmetric]
              image_stretch_interval_cart [of λk. 1 u v, symmetric] ‹cbox u v ≠
{}›
            by (simp add: field_simps image_comp o_def)
           ultimately show ((λx. indicat_real S (χ k. x $ k/ m k)) has_integral
z *R |prod m UNIV|) (cbox u v)
            by simp
          have |z *R |prod m UNIV| − |prod m UNIV| * measure lebesgue S|
              = |prod m UNIV| * |z − measure lebesgue S|
          by (metis (no_types, opaque_lifting) abs_abs abs_scaleR mult.commute
real_scaleR_def right_diff_distrib')
          also have ... < e
            using zless True by (simp add: field_simps)
          finally show |z *R |prod m UNIV| − |prod m UNIV| * measure lebesgue
S| < e .
        qed
       qed
     qed
   qed
   then show ?thesis
    by (auto simp: has_integral_integrable integral_unique lmeasure_integral_UNIV
measurable_integrable)
 next
   case False
   then obtain k where m k = 0 and prm: prod m UNIV = 0
     by auto
   have nfS: negligible (?f ' S)
    by (rule negligible_subset [OF negligible_standard_hyperplane_cart]) (use ‹m
k = 0› in auto)
   then show ?thesis
     by (simp add: negligible_iff_measure prm)
 qed
 then show (?f ' S) ∈ lmeasurable ?MEQ
   by metis+
qed


proposition
 fixes f :: real^'n::{finite,wellorder} ⇒ real^'n::_
  assumes linear f S ∈ lmeasurable
  shows measurable_linear_image: (f ' S) ∈ lmeasurable
    and measure_linear_image: measure lebesgue (f ' S) = |det (matrix f)| *
measure lebesgue S (is ?Q f S)
proof −
```

```
  have ∀S ∈ lmeasurable. (f ` S) ∈ lmeasurable ∧ ?Q f S
  proof (rule induct_linear_elementary [OF ‹linear f›]; intro ballI)
    fix f g and S :: (real,'n) vec set
    assume linear f and linear g
      and f [rule_format]: ∀S ∈ lmeasurable. f ` S ∈ lmeasurable ∧ ?Q f S
      and g [rule_format]: ∀S ∈ lmeasurable. g ` S ∈ lmeasurable ∧ ?Q g S
      and S: S ∈ lmeasurable
    then have gS: g ` S ∈ lmeasurable
      by blast
    show (f ∘ g) ` S ∈ lmeasurable ∧ ?Q (f ∘ g) S
      using f [OF gS] g [OF S] matrix_compose [OF ‹linear g› ‹linear f›]
      by (simp add: o_def image_comp abs_mult det_mul)
  next
    fix f :: real^'n::_ ⇒ real^'n::_ and i and S :: (real^'n::_) set
    assume linear f and 0: ⋀x. f x $ i = 0 and S ∈ lmeasurable
    then have ¬ inj f
      by (metis (full_types) linear_injective_imp_surjective one_neq_zero surjE
vec_component)
    have detf: det (matrix f) = 0
      using ‹¬ inj f› det_nz_iff_inj[OF ‹linear f›] by blast
    show f ` S ∈ lmeasurable ∧ ?Q f S
    proof
      show f ` S ∈ lmeasurable
        using lmeasurable_iff_indicator_has_integral ‹linear f› ‹¬ inj f› negligi-
ble_UNIV negligible_linear_singular_image by blast
      have measure lebesgue (f ` S) = 0
      by (meson ‹¬ inj f› ‹linear f› negligible_imp_measure0 negligible_linear_singular_image)
      also have ... = |det (matrix f)| * measure lebesgue S
        by (simp add: detf)
      finally show ?Q f S .
    qed
  next
    fix c and S :: (real^'n::_) set
    assume S ∈ lmeasurable
    show (λa. χ i. c i * a $ i) ` S ∈ lmeasurable ∧ ?Q (λa. χ i. c i * a $ i) S
    proof
      show (λa. χ i. c i * a $ i) ` S ∈ lmeasurable
        by (simp add: ‹S ∈ lmeasurable› measurable_stretch)
      show ?Q (λa. χ i. c i * a $ i) S
          by (simp add: measure_stretch [OF ‹S ∈ lmeasurable›, of c] axis_def
matrix_def det_diagonal)
    qed
  next
    fix m :: 'n and n :: 'n and S :: (real, 'n) vec set
    assume m ≠ n and S ∈ lmeasurable
    let ?h = λv::(real, 'n) vec. χ i. v $ Transposition.transpose m n i
    have lin: linear ?h
      by (rule linearI) (simp_all add: plus_vec_def scaleR_vec_def)
   have meq: measure lebesgue ((λv::(real, 'n) vec. χ i. v $ Transposition.transpose
```

```
m n i) ‘ cbox a b)
            = measure lebesgue (cbox a b) for a b
    proof (cases cbox a b = {})
      case True then show ?thesis
        by simp
    next
      case False
      then have him: ?h ‘ (cbox a b) ≠ {}
        by blast
      have eq: ?h ‘ (cbox a b) = cbox (?h a) (?h b)
          by (auto simp: image_iff lambda_swap_Galois mem_box_cart) (metis
transpose_involutory)+
      show ?thesis
        using him prod.permute [OF permutes_swap_id, where S=UNIV and
g=λi. (b − a)$i, symmetric]
        by (simp add: eq content_cbox_cart False)
    qed
    have (χ i j. if Transposition.transpose m n i = j then 1 else 0) = (χ i j. if j =
Transposition.transpose m n i then 1 else (0::real))
      by (auto intro!: Cart_lambda_cong)
    then have matrix ?h = transpose(χ i j. mat 1 $ i $ Transposition.transpose m
n j)
      by (auto simp: matrix_eq transpose_def axis_def mat_def matrix_def)
    then have 1: |det (matrix ?h)| = 1
    by (simp add: det_permute_columns permutes_swap_id sign_swap_id abs_mult)
    show ?h ‘ S ∈ lmeasurable ∧ ?Q ?h S
      using measure_linear_sufficient [OF lin ‹S ∈ lmeasurable›] meq 1 by force
  next
    fix m n :: 'n and S :: (real, 'n) vec set
    assume m ≠ n and S ∈ lmeasurable
    let ?h = λv::(real, 'n) vec. χ i. if i = m then v $ m + v $ n else v $ i
    have lin: linear ?h
      by (rule linearI) (auto simp: algebra_simps plus_vec_def scaleR_vec_def
vec_eq_iff)
    consider m < n | n < m
      using ‹m ≠ n› less_linear by blast
    then have 1: det(matrix ?h) = 1
    proof cases
      assume m < n
      have ∗: matrix ?h $ i $ j = (0::real) if j < i for i j :: 'n
      proof −
        have axis j 1 = (χ n. if n = j then 1 else (0::real))
          using axis_def by blast
        then have (χ p q. if p = m then axis q 1 $ m + axis q 1 $ n else axis q 1
$ p) $ i $ j = (0::real)
          using ‹j < i› axis_def ‹m < n› by auto
        with ‹m < n› show ?thesis
          by (auto simp: matrix_def axis_def cong: if_cong)
      qed
```

```
      show ?thesis
       using ‹m ≠ n› by (subst det_upperdiagonal [OF *]) (auto simp: matrix_def
axis_def cong: if_cong)
    next
      assume n < m
      have *: matrix ?h $ i $ j = (0::real) if j > i for i j :: 'n
      proof −
        have axis j 1 = (χ n. if n = j then 1 else (0::real))
          using axis_def by blast
        then have (χ p q. if p = m then axis q 1 $ m + axis q 1 $ n else axis q 1
$ p) $ i $ j = (0::real)
          using ‹j > i› axis_def ‹m > n› by auto
        with ‹m > n› show ?thesis
          by (auto simp: matrix_def axis_def cong: if_cong)
      qed
      show ?thesis
        using ‹m ≠ n›
       by (subst det_lowerdiagonal [OF *]) (auto simp: matrix_def axis_def cong:
if_cong)
    qed
   have meq: measure lebesgue (?h ‘ (cbox a b)) = measure lebesgue (cbox a b) for
a b
    proof (cases cbox a b = {})
      case True then show ?thesis by simp
    next
      case False
      then have ne: (+) (χ i. if i = n then − a $ n else 0) ‘ cbox a b ≠ {}
        by auto
      let ?v = χ i. if i = n then − a $ n else 0
      have ?h ‘ cbox a b
          = (+) (χ i. if i = m ∨ i = n then a $ n else 0) ‘ ?h ‘ (+) ?v ‘ (cbox a b)
        using ‹m ≠ n› unfolding image_comp o_def by (force simp: vec_eq_iff)
      then have measure lebesgue (?h ‘ (cbox a b))
              = measure lebesgue ((λv. χ i. if i = m then v $ m + v $ n else v $ i) ‘
                                  (+) ?v ‘ cbox a b)
        by (rule ssubst) (rule measure_translation)
      also have ... = measure lebesgue ((λv. χ i. if i = m then v $ m + v $ n else
v $ i) ‘ cbox (?v +a) (?v + b))
        by (metis (no_types, lifting) cbox_translation)
      also have ... = measure lebesgue ((+) ?v ‘ cbox a b)
        apply (subst measure_shear_interval)
        using ‹m ≠ n› ne apply auto
        apply (simp add: cbox_translation)
        by (metis cbox_borel cbox_translation measure_completion sets_lborel)
      also have ... = measure lebesgue (cbox a b)
        by (rule measure_translation)
        finally show ?thesis .
      qed
    show ?h ‘ S ∈ lmeasurable ∧ ?Q ?h S
```

```
    using measure_linear_sufficient [OF lin ‹S ∈ lmeasurable›] meq 1 by force
  qed
  with assms show (f ‘ S) ∈ lmeasurable ?Q f S
    by metis+
qed


lemma
 fixes f :: real^'n::{finite,wellorder} ⇒ real^'n::_
  assumes f: orthogonal_transformation f and S: S ∈ lmeasurable
  shows measurable_orthogonal_image: f ‘ S ∈ lmeasurable
    and measure_orthogonal_image: measure lebesgue (f ‘ S) = measure lebesgue
S
proof −
  have linear f
    by (simp add: f orthogonal_transformation_linear)
  then show f ‘ S ∈ lmeasurable
    by (metis S measurable_linear_image)
  show measure lebesgue (f ‘ S) = measure lebesgue S
    by (simp add: measure_linear_image ‹linear f› S f)
qed

proposition measure_semicontinuous_with_hausdist_explicit:
  assumes bounded S and neg: negligible(frontier S) and e > 0
  obtains d where d > 0
                ⋀T. ⟦T ∈ lmeasurable; ⋀y. y ∈ T ⟹ ∃x. x ∈ S ∧ dist x y < d⟧
                    ⟹ measure lebesgue T < measure lebesgue S + e
proof (cases S = {})
  case True
  with that ‹e > 0› show ?thesis by force
next
  case False
  then have frS: frontier S ≠ {}
    using ‹bounded S› frontier_eq_empty not_bounded_UNIV by blast
  have S ∈ lmeasurable
    by (simp add: ‹bounded S› measurable_Jordan neg)
  have null: (frontier S) ∈ null_sets lebesgue
    by (metis neg negligible_iff_null_sets)
  have frontier S ∈ lmeasurable and mS0: measure lebesgue (frontier S) = 0
    using neg negligible_imp_measurable negligible_iff_measure by blast+
  with ‹e > 0› sets_lebesgue_outer_open
  obtain U where open U
    and U: frontier S ⊆ U U − frontier S ∈ lmeasurable emeasure lebesgue (U −
frontier S) < e
    by (metis fmeasurableD)
  with null have U ∈ lmeasurable
   by (metis borel_open measurable_Diff_null_set sets_completionI_sets sets_lborel)
  have measure lebesgue (U − frontier S) = measure lebesgue U
   using mS0 by (simp add: ‹U ∈ lmeasurable› fmeasurableD measure_Diff_null_set
```

null)
with *U* **have** *mU*: *measure lebesgue U* < *e*
by (*simp add*: *emeasure_eq_measure2 ennreal_less_iff*)
show *?thesis*
proof
have *U* ≠ *UNIV*
using ‹*U* ∈ *lmeasurable*› **by** *auto*
then have − *U* ≠ {}
by *blast*
with ‹*open U*› ‹*frontier S* ⊆ *U*› **show** *setdist* (*frontier S*) (− *U*) > *0*
by (*auto simp*: ‹*bounded S*› *open_closed compact_frontier_bounded setdist_gt_0_compact_closed frS*)
fix *T*
assume *T* ∈ *lmeasurable*
and *T*: ⋀*t*. *t* ∈ *T* ⟹ ∃ *y*. *y* ∈ *S* ∧ *dist y t* < *setdist* (*frontier S*) (− *U*)
then have *measure lebesgue T* − *measure lebesgue S* ≤ *measure lebesgue* (*T* − *S*)
by (*simp add*: ‹*S* ∈ *lmeasurable*› *measure_diff_le_measure_setdiff*)
also have ... ≤ *measure lebesgue U*
proof −
have *T* − *S* ⊆ *U*
proof *clarify*
fix *x*
assume *x* ∈ *T* **and** *x* ∉ *S*
then obtain *y* **where** *y* ∈ *S* **and** *y*: *dist y x* < *setdist* (*frontier S*) (− *U*)
using *T* **by** *blast*
have *closed_segment x y* ∩ *frontier S* ≠ {}
using *connected_Int_frontier* ‹*x* ∉ *S*› ‹*y* ∈ *S*› **by** *blast*
then obtain *z* **where** *z*: *z* ∈ *closed_segment x y z* ∈ *frontier S*
by *auto*
with *y* **have** *dist z x* < *setdist*(*frontier S*) (− *U*)
by (*auto simp*: *dist_commute dest*!: *dist_in_closed_segment*)
with *z* **have** *False* **if** *x* ∈ −*U*
using *setdist_le_dist* [*OF* ‹*z* ∈ *frontier S*› *that*] **by** *auto*
then show *x* ∈ *U*
by *blast*
qed
then show *?thesis*
by (*simp add*: ‹*S* ∈ *lmeasurable*› ‹*T* ∈ *lmeasurable*› ‹*U* ∈ *lmeasurable*› *fmeasurableD measure_mono_fmeasurable sets.Diff*)
qed
finally have *measure lebesgue T* − *measure lebesgue S* ≤ *measure lebesgue U* **.**
with *mU* **show** *measure lebesgue T* < *measure lebesgue S* + *e*
by *linarith*
qed
qed

proposition
fixes *f* :: *real*^'*n*::{*finite*,*wellorder*} ⇒ *real*^'*n*::_

```
  assumes S: S ∈ lmeasurable
  and deriv: ⋀x. x ∈ S ⟹ (f has_derivative f' x) (at x within S)
  and int: (λx. |det (matrix (f' x))|) integrable_on S
  and bounded: ⋀x. x ∈ S ⟹ |det (matrix (f' x))| ≤ B
  shows measurable_bounded_differentiable_image:
      f ‘ S ∈ lmeasurable
    and measure_bounded_differentiable_image:
      measure lebesgue (f ‘ S) ≤ B * measure lebesgue S (is ?M)
proof −
  have f ‘ S ∈ lmeasurable ∧ measure lebesgue (f ‘ S) ≤ B * measure lebesgue S
  proof (cases B < 0)
    case True
    then have S = {}
     by (meson abs_ge_zero bounded empty_iff equalityI less_le_trans linorder_not_less
subsetI)
    then show ?thesis
      by auto
  next
    case False
    then have B ≥ 0
      by arith
    let ?μ = measure lebesgue
    have f_diff: f differentiable_on S
      using deriv by (auto simp: differentiable_on_def differentiable_def)
    have eps: f ‘ S ∈ lmeasurable ?μ (f ‘ S) ≤ (B+e) * ?μ S (is ?ME)
            if e > 0 for e
    proof −
      have eps_d: f ‘ S ∈ lmeasurable  ?μ (f ‘ S) ≤ (B+e) * (?μ S + d) (is ?MD)
              if d > 0 for d
      proof −
        obtain T where T: open T S ⊆ T and TS: (T−S) ∈ lmeasurable and
emeasure lebesgue (T−S) < ennreal d
          using S ‹d > 0› sets_lebesgue_outer_open by blast
        then have ?μ (T−S) < d
          by (metis emeasure_eq_measure2 ennreal_leI not_less)
        with S T TS have T ∈ lmeasurable and Tless: ?μ T < ?μ S + d
          by (auto simp: measurable_measure_Diff dest!: fmeasurable_Diff_D)
        have ∃r. 0 < r ∧ r < d ∧ ball x r ⊆ T ∧ f ‘ (S ∩ ball x r) ∈ lmeasurable ∧
                  ?μ (f ‘ (S ∩ ball x r)) ≤ (B + e) * ?μ (ball x r)
          if x ∈ S d > 0 for x d
        proof −
          have lin: linear (f' x)
            and lim0: ((λy. (f y − (f x + f' x (y − x))) /R norm(y − x)) ⟶ 0)
(at x within S)
          using deriv ‹x ∈ S› by (auto simp: has_derivative_within bounded_linear.linear
field_simps)
          have bo: bounded (f' x ‘ ball 0 1)
            by (simp add: bounded_linear_image linear_linear lin)
          have neg: negligible (frontier (f' x ‘ ball 0 1))
```

```
      using deriv has_derivative_linear ‹x ∈ S›
      by (auto intro!: negligible_convex_frontier [OF convex_linear_image])
    let ?unit_vol = content (ball (0 :: real ^ 'n :: {finite, wellorder}) 1)
    have 0: 0 < e * ?unit_vol
      using ‹e > 0› by (simp add: content_ball_pos)
    obtain k where k > 0 and k:
        ⋀U. ⟦U ∈ lmeasurable; ⋀y. y ∈ U ⟹ ∃z. z ∈ f' x ‘ ball 0 1 ∧ dist
z y < k⟧
            ⟹ ?μ U < ?μ (f' x ‘ ball 0 1) + e * ?unit_vol
      using measure_semicontinuous_with_hausdist_explicit [OF bo neg 0]
by blast
    obtain l where l > 0 and l: ball x l ⊆ T
      using ‹x ∈ S› ‹open T› ‹S ⊆ T› openE by blast
    obtain ζ where 0 < ζ
      and ζ: ⋀y. ⟦y ∈ S; y ≠ x; dist y x < ζ⟧
            ⟹ norm (f y - (f x + f' x (y - x))) / norm (y - x) < k
        using lim0 ‹k > 0› by (simp add: Lim_within) (auto simp add:
field_simps)
    define r where r ≡ min (min l (ζ/2)) (min 1 (d/2))
    show ?thesis
    proof (intro exI conjI)
      show r > 0 r < d
        using ‹l > 0› ‹ζ > 0› ‹d > 0› by (auto simp: r_def)
      have r ≤ l
        by (auto simp: r_def)
      with l show ball x r ⊆ T
        by auto
      have ex_lessK: ∃x' ∈ ball 0 1. dist (f' x x') ((f y - f x) /R r) < k
        if y ∈ S and dist x y < r for y
      proof (cases y = x)
        case True
        with lin linear_0 ‹k > 0› that show ?thesis
          by (rule_tac x=0 in bexI) (auto simp: linear_0)
      next
        case False
        then show ?thesis
        proof (rule_tac x=(y - x) /R r in bexI)
          have f' x ((y - x) /R r) = f' x (y - x) /R r
            by (simp add: lin linear_scale)
          then have dist (f' x ((y - x) /R r)) ((f y - f x) /R r) = norm (f'
x (y - x) /R r - (f y - f x) /R r)
            by (simp add: dist_norm)
          also have ... = norm (f' x (y - x) - (f y - f x)) / r
            using ‹r > 0› by (simp add: divide_simps scale_right_diff_distrib
[symmetric])
          also have ... ≤ norm (f y - (f x + f' x (y - x))) / norm (y - x)
            using that ‹r > 0› False by (simp add: field_split_simps dist_norm
norm_minus_commute mult_right_mono)
          also have ... < k
```

```
                using that ‹0 < ζ› by (simp add: dist_commute r_def ζ [OF ‹y
∈ S› False])
              finally show dist (f' x ((y − x) /R r)) ((f y − f x) /R r) < k .
              show (y − x) /R r ∈ ball 0 1
                    using that ‹r > 0› by (simp add: dist_norm divide_simps
norm_minus_commute)
            qed
          qed
          let ?rfs = (λx. x /R r) ‘ (+) (− f x) ‘ f ‘ (S ∩ ball x r)
          have rfs_mble: ?rfs ∈ lmeasurable
          proof (rule bounded_set_imp_lmeasurable)
            have f differentiable_on S ∩ ball x r
              using f_diff by (auto simp: fmeasurableD differentiable_on_subset)
            with S show ?rfs ∈ sets lebesgue
                by (auto simp: sets.Int intro!: lebesgue_sets_translation differen-
tiable_image_in_sets_lebesgue)
            let ?B = (λ(x, y). x + y) ‘ (f' x ‘ ball 0 1 × ball 0 k)
            have bounded ?B
              by (simp add: bounded_plus [OF bo])
            moreover have ?rfs ⊆ ?B
              apply (auto simp: dist_norm image_iff dest!: ex_lessK)
               by (metis (no_types, opaque_lifting) add.commute diff_add_cancel
dist_0_norm dist_commute dist_norm mem_ball)
            ultimately show bounded (?rfs)
              by (rule bounded_subset)
          qed
          then have (λx. r *R x) ‘ ?rfs ∈ lmeasurable
            by (simp add: measurable_linear_image)
          with ‹r > 0› have (+) (− f x) ‘ f ‘ (S ∩ ball x r) ∈ lmeasurable
            by (simp add: image_comp o_def)
          then have (+) (f x) ‘ (+) (− f x) ‘ f ‘ (S ∩ ball x r) ∈ lmeasurable
            using  measurable_translation by blast
          then show fsb: f ‘ (S ∩ ball x r) ∈ lmeasurable
            by (simp add: image_comp o_def)
          have ?μ (f ‘ (S ∩ ball x r)) = ?μ (?rfs) * r ^ CARD('n)
            using ‹r > 0› fsb
              by (simp add: measure_linear_image measure_translation_subtract
measurable_translation_subtract field_simps cong: image_cong_simp)
        also have ... ≤ (|det (matrix (f' x))| * ?unit_vol + e * ?unit_vol) * r
^ CARD('n)
          proof −
            have ?μ (?rfs) < ?μ (f' x ‘ ball 0 1) + e * ?unit_vol
              using rfs_mble by (force intro: k dest!: ex_lessK)
          then have ?μ (?rfs) < |det (matrix (f' x))| * ?unit_vol + e * ?unit_vol
              by (simp add: lin measure_linear_image [of f' x])
            with ‹r > 0› show ?thesis
              by auto
          qed
          also have ... ≤ (B + e) * ?μ (ball x r)
```

```
          using bounded [OF ‹x ∈ S›] ‹r > 0›
            by (simp add: algebra_simps content_ball_conv_unit_ball[of r]
content_ball_pos)
        finally show ?μ (f ‘ (S ∩ ball x r)) ≤ (B + e) * ?μ (ball x r) .
      qed
    qed
    then obtain r where
      r0d: ⋀x d. ⟦x ∈ S; d > 0⟧ ⟹ 0 < r x d ∧ r x d < d
      and rT: ⋀x d. ⟦x ∈ S; d > 0⟧ ⟹ ball x (r x d) ⊆ T
      and r: ⋀x d. ⟦x ∈ S; d > 0⟧ ⟹
          (f ‘ (S ∩ ball x (r x d))) ∈ lmeasurable ∧
          ?μ (f ‘ (S ∩ ball x (r x d))) ≤ (B + e) * ?μ (ball x (r x d))
      by metis
    obtain C where countable C and Csub: C ⊆ {(x,r x t) |x t. x ∈ S ∧ 0 <
t}
      and pwC: pairwise (λi j. disjnt (ball (fst i) (snd i)) (ball (fst j) (snd j)))
C
      and negC: negligible(S − (⋃i ∈ C. ball (fst i) (snd i)))
      apply (rule Vitali_covering_theorem_balls [of S {(x,r x t) |x t. x ∈ S ∧
0 < t} fst snd])
       apply auto
      by (metis dist_eq_0_iff r0d)
    let ?UB = (⋃(x,s) ∈ C. ball x s)
    have eq: f ‘ (S ∩ ?UB) = (⋃(x,s) ∈ C. f ‘ (S ∩ ball x s))
      by auto
    have mle: ?μ (⋃(x,s) ∈ K. f ‘ (S ∩ ball x s)) ≤ (B + e) * (?μ S + d)  (is
?l ≤ ?r)
      if K ⊆ C and finite K for K
    proof −
      have gt0: b > 0 if (a, b) ∈ K for a b
        using Csub that ‹K ⊆ C› r0d by auto
      have inj: inj_on (λ(x, y). ball x y) K
        by (force simp: inj_on_def ball_eq_ball_iff dest: gt0)
      have disjnt: disjoint ((λ(x, y). ball x y) ‘ K)
        using pwC that pairwise_image pairwise_mono by fastforce
      have ?l ≤ (∑ i∈K. ?μ (case i of (x, s) ⇒ f ‘ (S ∩ ball x s)))
      proof (rule measure_UNION_le [OF ‹finite K›], clarify)
        fix x r
        assume (x,r) ∈ K
        then have x ∈ S
          using Csub ‹K ⊆ C› by auto
        show f ‘ (S ∩ ball x r) ∈ sets lebesgue
          by (meson Int_lower1 S differentiable_on_subset f_diff fmeasurableD
lmeasurable_ball order_refl sets.Int differentiable_image_in_sets_lebesgue)
      qed
      also have ... ≤ (∑(x,s) ∈ K. (B + e) * ?μ (ball x s))
        using Csub r ‹K ⊆ C›  by (intro sum_mono) auto
      also have ... = (B + e) * (∑(x,s) ∈ K. ?μ (ball x s))
        by (simp add: prod.case_distrib sum_distrib_left)
```

```
        also have ... = (B + e) * sum ?μ ((λ(x, y). ball x y) ' K)
         using ‹B ≥ 0› ‹e > 0› by (simp add: inj sum.reindex prod.case_distrib)
        also have ... = (B + e) * ?μ (⋃(x,s) ∈ K. ball x s)
          using ‹B ≥ 0› ‹e > 0› that
          by (subst measure_Union') (auto simp: disjnt measure_Union')
        also have ... ≤ (B + e) * ?μ T
          using ‹B ≥ 0› ‹e > 0› that apply simp
         using measure_mono_fmeasurable [OF _ _ ‹T ∈ lmeasurable›] Csub rT
              by (smt (verit) SUP_least measure_nonneg measure_notin_sets
mem_Collect_eq old.prod.case subset_iff)
        also have ... ≤ (B + e) * (?μ S + d)
          using ‹B ≥ 0› ‹e > 0› Tless by simp
        finally show ?thesis .
      qed
      have fSUB_mble: (f ' (S ∩ ?UB)) ∈ lmeasurable
        unfolding eq using Csub r False ‹e > 0› that
        by (auto simp: intro!: fmeasurable_UN_bound [OF ‹countable C› _ mle])
      have fSUB_meas: ?μ (f ' (S ∩ ?UB)) ≤ (B + e) * (?μ S + d)  (is ?MUB)
        unfolding eq using Csub r False ‹e > 0› that
        by (auto simp: intro!: measure_UN_bound [OF ‹countable C› _ mle])
      have neg: negligible ((f ' (S ∩ ?UB) - f ' S) ∪ (f ' S - f ' (S ∩ ?UB)))
      proof (rule negligible_subset [OF negligible_differentiable_image_negligible
[OF order_refl negC, where f=f]])
        show f differentiable_on S - (⋃i∈C. ball (fst i) (snd i))
          by (meson DiffE differentiable_on_subset subsetI f_diff)
      qed force
      show f ' S ∈ lmeasurable
        by (rule lmeasurable_negligible_symdiff [OF fSUB_mble neg])
      show ?MD
        using fSUB_meas measure_negligible_symdiff [OF fSUB_mble neg] by
simp
    qed
    show f ' S ∈ lmeasurable
      using eps_d [of 1] by simp
    show ?ME
    proof (rule field_le_epsilon)
      fix δ :: real
      assume 0 < δ
      then show ?μ (f ' S) ≤ (B + e) * ?μ S + δ
       using eps_d [of δ / (B+e)] ‹e > 0› ‹B ≥ 0› by (auto simp: divide_simps
mult_ac)
    qed
  qed
  show ?thesis
  proof (cases ?μ S = 0)
    case True
    with eps have ?μ (f ' S) = 0
      by (metis mult_zero_right not_le zero_less_measure_iff)
    then show ?thesis
```

```
      using eps [of 1] by (simp add: True)
    next
      case False
      have ?μ (f ‘ S) ≤ B * ?μ S
      proof (rule field_le_epsilon)
        fix e :: real
        assume e > 0
        then show ?μ (f ‘ S) ≤ B * ?μ S + e
       using eps [of e / ?μ S] False by (auto simp: algebra_simps zero_less_measure_iff)
      qed
      with eps [of 1] show ?thesis by auto
    qed
  qed
  then show f ‘ S ∈ lmeasurable ?M by blast+
qed

lemma m_diff_image_weak:
 fixes f :: real^'n::{finite,wellorder} ⇒ real^'n::_
  assumes S: S ∈ lmeasurable
    and deriv: ⋀x. x ∈ S ⟹ (f has_derivative f' x) (at x within S)
    and int: (λx. |det (matrix (f' x))|) integrable_on S
  shows f ‘ S ∈ lmeasurable ∧ measure lebesgue (f ‘ S) ≤ integral S (λx. |det
(matrix (f' x))|)
proof −
  let ?μ = measure lebesgue
  have aint_S: (λx. |det (matrix (f' x))|) absolutely_integrable_on S
    using int unfolding absolutely_integrable_on_def by auto
  define m where m ≡ integral S (λx. |det (matrix (f' x))|)
  have *: f ‘ S ∈ lmeasurable ?μ (f ‘ S) ≤ m + e * ?μ S
    if e > 0 for e
  proof −
    define T where T ≡ λn. {x ∈ S. n * e ≤ |det (matrix (f' x))| ∧
                                  |det (matrix (f' x))| < (Suc n) * e}
    have meas_t: T n ∈ lmeasurable for n
    proof −
      have *: (λx. |det (matrix (f' x))|) ∈ borel_measurable (lebesgue_on S)
        using aint_S by (simp add: S borel_measurable_restrict_space_iff fmea-
surableD set_integrable_def)
      have [intro]: x ∈ sets (lebesgue_on S) ⟹ x ∈ sets lebesgue for x
        using S sets_restrict_space_subset by blast
      have {x ∈ S. real n * e ≤ |det (matrix (f' x))|} ∈ sets lebesgue
      using * by (auto simp: borel_measurable_iff_halfspace_ge space_restrict_space)
      then have 1: {x ∈ S. real n * e ≤ |det (matrix (f' x))|} ∈ lmeasurable
        using S by (simp add: fmeasurableI2)
      have {x ∈ S. |det (matrix (f' x))| < (1 + real n) * e} ∈ sets lebesgue
      using * by (auto simp: borel_measurable_iff_halfspace_less space_restrict_space)
      then have 2: {x ∈ S. |det (matrix (f' x))| < (1 + real n) * e} ∈ lmeasurable
        using S by (simp add: fmeasurableI2)
      show ?thesis
```

```
        using fmeasurable.Int [OF 1 2] by (simp add: T_def Int_def cong:
conj_cong)
    qed
    have aint_T: ⋀k. (λx. |det (matrix (f' x))|) absolutely_integrable_on T k
      using set_integrable_subset [OF aint_S] meas_t T_def by blast
    have Seq: S = (⋃ n. T n)
      apply (auto simp: T_def)
      apply (rule_tac x = nat ⌊|det (matrix (f' x))| / e⌋ in exI)
       by (smt (verit, del_insts) divide_nonneg_nonneg floor_eq_iff of_nat_nat
pos_divide_less_eq that zero_le_floor)
    have meas_ft: f ‘ T n ∈ lmeasurable for n
    proof (rule measurable_bounded_differentiable_image)
      show T n ∈ lmeasurable
        by (simp add: meas_t)
    next
      fix x :: (real,'n) vec
      assume x ∈ T n
      show (f has_derivative f' x) (at x within T n)
          by (metis (no_types, lifting) ‹x ∈ T n› deriv has_derivative_subset
mem_Collect_eq subsetI T_def)
      show |det (matrix (f' x))| ≤ (Suc n) * e
        using ‹x ∈ T n› T_def by auto
    next
      show (λx. |det (matrix (f' x))|) integrable_on T n
        using aint_T absolutely_integrable_on_def by blast
    qed
    have disT: disjoint (range T)
      unfolding disjoint_def
    proof clarsimp
      show T m ∩ T n = {} if T m ≠ T n for m n
        using that
      proof (induction m n rule: linorder_less_wlog)
        case (less m n)
        with ‹e > 0› show ?case
          unfolding T_def
          proof (clarsimp simp add: Collect_conj_eq [symmetric])
            fix x
             assume e > 0  m < n  n * e ≤ |det (matrix (f' x))|  |det (matrix (f'
x))| < (1 + real m) * e
            then have n < 1 + real m
                by (metis (no_types, opaque_lifting) less_le_trans mult.commute
not_le mult_le_cancel_iff2)
            then show False
              using less.hyps by linarith
          qed
      qed auto
    qed
    have injT: inj_on T ({n. T n ≠ {}})
      unfolding inj_on_def
```

```
  proof clarsimp
    show m = n if T m = T n T n ≠ {} for m n
      using that
    proof (induction m n rule: linorder_less_wlog)
      case (less m n)
      have False if T n ⊆ T m x ∈ T n for x
        using ‹e > 0› ‹m < n› that
      apply (auto simp: T_def mult.commute intro: less_le_trans dest!: subsetD)
        by (smt (verit, best) mult_less_cancel_left_disj nat_less_real_le)
      then show ?case
        using less.prems by blast
    qed auto
  qed
  have sum_eq_Tim: (∑k≤n. f (T k)) = sum f (T ‘ {..n}) if f {} = 0 for f ::
_ ⇒ real and n
  proof (subst sum.reindex_nontrivial)
    fix i j assume i ∈ {..n} j ∈ {..n} i ≠ j T i = T j
    with that injT [unfolded inj_on_def] show f (T i) = 0
      by simp metis
  qed (use atMost_atLeast0 in auto)
  let ?B = m + e * ?μ S
  have (∑k≤n. ?μ (f ‘ T k)) ≤ ?B for n
  proof −
    have (∑k≤n. ?μ (f ‘ T k)) ≤ (∑k≤n. ((k+1) * e) * ?μ(T k))
    proof (rule sum_mono [OF measure_bounded_differentiable_image])
      show (f has_derivative f' x) (at x within T k) if x ∈ T k for k x
       using that unfolding T_def by (blast intro: deriv has_derivative_subset)
      show (λx. |det (matrix (f' x))|) integrable_on T k for k
        using absolutely_integrable_on_def aint_T by blast
      show |det (matrix (f' x))| ≤ real (k + 1) * e if x ∈ T k for k x
        using T_def that by auto
    qed (use meas_t in auto)
    also have ... ≤ (∑k≤n. (k * e) * ?μ(T k)) + (∑k≤n. e * ?μ(T k))
      by (simp add: algebra_simps sum.distrib)
    also have ... ≤ ?B
    proof (rule add_mono)
      have (∑k≤n. real k * e * ?μ (T k)) = (∑k≤n. integral (T k) (λx. k * e))
        by (simp add: lmeasure_integral [OF meas_t]
              flip: integral_mult_right integral_mult_left)
      also have ... ≤ (∑k≤n. integral (T k) (λx.  (abs (det (matrix (f' x))))))
      proof (rule sum_mono)
        fix k
        assume k ∈ {..n}
       show integral (T k) (λx. k * e) ≤ integral (T k) (λx. |det (matrix (f' x))|)
        proof (rule integral_le [OF integrable_on_const [OF meas_t]])
          show (λx. |det (matrix (f' x))|) integrable_on T k
            using absolutely_integrable_on_def aint_T by blast
        next
          fix x assume x ∈ T k
```

```
          show k * e ≤ |det (matrix (f' x))|
            using ‹x ∈ T k› T_def by blast
        qed
      qed
    also have ... = sum (λT. integral T (λx. |det (matrix (f' x))|)) (T ‘ {..n})
      by (auto intro: sum_eq_Tim)
    also have ... = integral (⋃k≤n. T k) (λx. |det (matrix (f' x))|)
    proof (rule integral_unique [OF has_integral_Union, symmetric])
      fix S assume S ∈ T ‘ {..n}
       then show ((λx. |det (matrix (f' x))|) has_integral integral S (λx. |det
(matrix (f' x))|)) S
        using absolutely_integrable_on_def aint_T by blast
      next
        show pairwise (λS S'. negligible (S ∩ S')) (T ‘ {..n})
           using disT unfolding disjnt_iff by (auto simp: pairwise_def intro!:
empty_imp_negligible)
      qed auto
    also have ... ≤ m
      unfolding m_def
    proof (rule integral_subset_le)
      have (λx. |det (matrix (f' x))|) absolutely_integrable_on (⋃k≤n. T k)
      proof (rule set_integrable_subset [OF aint_S])
        show ⋃ (T ‘ {..n}) ∈ sets lebesgue
          by (intro measurable meas_t fmeasurableD)
      qed (force simp: Seq)
      then show (λx. |det (matrix (f' x))|) integrable_on (⋃k≤n. T k)
        using absolutely_integrable_on_def by blast
    qed (use Seq int in auto)
    finally show (∑k≤n. real k * e * ?μ (T k)) ≤ m .
  next
    have (∑k≤n. ?μ (T k)) = sum ?μ (T ‘ {..n})
      by (auto intro: sum_eq_Tim)
    also have ... = ?μ (⋃k≤n. T k)
      using S disT by (auto simp: pairwise_def meas_t intro: measure_Union'
[symmetric])
    also have ... ≤ ?μ S
    using S by (auto simp: Seq intro: meas_t fmeasurableD measure_mono_fmeasurable)
    finally have (∑k≤n. ?μ (T k)) ≤ ?μ S .
    then show (∑k≤n. e * ?μ (T k)) ≤ e * ?μ S
    by (metis less_eq_real_def ordered_comm_semiring_class.comm_mult_left_mono
sum_distrib_left that)
  qed
  finally show (∑k≤n. ?μ (f ‘ T k)) ≤ ?B .
qed
  moreover have measure lebesgue (⋃k≤n. f ‘ T k) ≤ (∑k≤n. ?μ (f ‘ T k))
for n
  by (simp add: fmeasurableD meas_ft measure_UNION_le)
  ultimately have B_ge_m: ?μ (⋃k≤n. (f ‘ T k)) ≤ ?B for n
  by (meson order_trans)
```

```
    have (⋃ n. f ‘ T n) ∈ lmeasurable
      by (rule fmeasurable_countable_Union [OF meas_ft B_ge_m])
    moreover have ?μ (⋃ n. f ‘ T n) ≤ m + e * ?μ S
      by (rule measure_countable_Union_le [OF meas_ft B_ge_m])
    ultimately show f ‘ S ∈ lmeasurable ?μ (f ‘ S) ≤ m + e * ?μ S
      by (auto simp: Seq image_Union)
  qed
  show ?thesis
  proof
    show f ‘ S ∈ lmeasurable
      using * linordered_field_no_ub by blast
    let ?x = m − ?μ (f ‘ S)
    have False if ?μ (f ‘ S) > integral S (λx. |det (matrix (f′ x))|)
    proof −
      have ml: m < ?μ (f ‘ S)
        using m_def that by blast
      then have ?μ S ≠ 0
        using *(2) bgauge_existence_lemma by fastforce
      with ml have 0: 0 < − (m − ?μ (f ‘ S))/2 / ?μ S
        using that zero_less_measure_iff by force
      then show ?thesis
      using * [OF 0] that by (auto simp: field_split_simps m_def split: if_split_asm)
    qed
    then show ?μ (f ‘ S) ≤ integral S (λx. |det (matrix (f′ x))|)
      by fastforce
  qed
qed


theorem
 fixes f :: real^'n::{finite,wellorder} ⇒ real^'n::_
  assumes S: S ∈ sets lebesgue
    and deriv: ⋀x. x ∈ S ⟹ (f has_derivative f′ x) (at x within S)
    and int: (λx. |det (matrix (f′ x))|) integrable_on S
  shows measurable_differentiable_image: f ‘ S ∈ lmeasurable
    and measure_differentiable_image:
      measure lebesgue (f ‘ S) ≤ integral S (λx. |det (matrix (f′ x))|) (is ?M)
proof −
  let ?I = λn::nat. cbox (vec (−n)) (vec n) ∩ S
  let ?μ = measure lebesgue
  have x ∈ cbox (vec (− real (nat ⌈norm x⌉))) (vec (real (nat ⌈norm x⌉))) for x
:: real^'n::_
    apply (simp add: mem_box_cart)
   by (smt (verit, best) Finite_Cartesian_Product.norm_nth_le nat_ceiling_le_eq

       real_nat_ceiling_ge real_norm_def)
  then have Seq: S = (⋃ n. ?I n)
    by auto
  have fIn: f ‘ ?I n ∈ lmeasurable
```

```
    and mfIn: ?μ (f ‘ ?I n) ≤ integral S (λx. |det (matrix (f' x))|) (is ?MN) for
n
  proof −
    have In: ?I n ∈ lmeasurable
      by (simp add: S bounded_Int bounded_set_imp_lmeasurable sets.Int)
    moreover have ⋀x. x ∈ ?I n ⟹ (f has_derivative f' x) (at x within ?I n)
      by (meson Int_iff deriv has_derivative_subset subsetI)
    moreover have int_In: (λx. |det (matrix (f' x))|) integrable_on ?I n
    by (metis (mono_tags) Int_commute int integrable_altD(1) integrable_restrict_Int)
    ultimately have f ‘ ?I n ∈ lmeasurable ?μ (f ‘ ?I n) ≤ integral (?I n) (λx.
|det (matrix (f' x))|)
      using m_diff_image_weak by metis+
    moreover have integral (?I n) (λx. |det (matrix (f' x))|) ≤ integral S (λx.
|det (matrix (f' x))|)
      by (simp add: int_In int integral_subset_le)
    ultimately show f ‘ ?I n ∈ lmeasurable ?MN
      by auto
  qed
  have ?I k ⊆ ?I n if k ≤ n for k n
    by (rule Int_mono) (use that in ‹auto simp: subset_interval_imp_cart›)
  then have (⋃k≤n. f ‘ ?I k) = f ‘ ?I n for n
    by (fastforce simp add:)
  with mfIn have ?μ (⋃k≤n. f ‘ ?I k) ≤ integral S (λx. |det (matrix (f' x))|) for
n
    by simp
  then have (⋃n. f ‘ ?I n) ∈ lmeasurable ?μ (⋃n. f ‘ ?I n) ≤ integral S (λx. |det
(matrix (f' x))|)
    by (rule fmeasurable_countable_Union [OF fIn] measure_countable_Union_le
[OF fIn])+
  then show f ‘ S ∈ lmeasurable ?M
    by (metis Seq image_UN)+
qed


lemma borel_measurable_simple_function_limit_increasing:
  fixes f :: 'a::euclidean_space ⇒ real
  shows (f ∈ borel_measurable lebesgue ∧ (∀x. 0 ≤ f x)) ⟷
        (∃g. (∀n x. 0 ≤ g n x ∧ g n x ≤ f x) ∧ (∀n x. g n x ≤ (g(Suc n) x)) ∧
             (∀n. g n ∈ borel_measurable lebesgue) ∧ (∀n. finite(range (g n))) ∧
             (∀x. (λn. g n x) ⟶ f x))
        (is ?lhs = ?rhs)
proof
  assume f: ?lhs
  have leb_f: {x. a ≤ f x ∧ f x < b} ∈ sets lebesgue for a b
  proof −
    have {x. a ≤ f x ∧ f x < b} = {x. f x < b} − {x. f x < a}
      by auto
    also have ... ∈ sets lebesgue
      using borel_measurable_vimage_halfspace_component_lt [of f UNIV] f by
```

```
auto
    finally show ?thesis .
  qed
  have g n x ≤ f x
       if inc_g: ⋀n x. 0 ≤ g n x ∧ g n x ≤ g (Suc n) x
          and meas_g: ⋀n. g n ∈ borel_measurable lebesgue
          and fin: ⋀n. finite(range (g n)) and lim: ⋀x. (λn. g n x) ⟶ f x for
g n x
  proof −
    have ∃r>0. ∀N. ∃n≥N. dist (g n x) (f x) ≥ r if g n x > f x
    proof −
      have g: g n x ≤ g (N + n) x for N
        by (rule transitive_stepwise_le) (use inc_g in auto)
      have ∃m≥N. g n x − f x ≤ dist (g m x) (f x) for N
      proof
        show N ≤ N + n ∧ g n x − f x ≤ dist (g (N + n) x) (f x)
          using g [of N] by (auto simp: dist_norm)
      qed
      with that show ?thesis
        using diff_gt_0_iff_gt by blast
    qed
    with lim show ?thesis
      unfolding lim_sequentially
      by (meson less_le_not_le not_le_imp_less)
  qed
  moreover
  let ?Ω = λn k. indicator {y. k/2^n ≤ f y ∧ f y < (k+1)/2^n}
  let ?g = λn x. (∑k::real | k ∈ ℤ ∧ |k| ≤ 2 ^(2*n). k/2^n * ?Ω n k x)
  have ∃g. (∀n x. 0 ≤ g n x ∧ g n x ≤ (g(Suc n) x)) ∧
           (∀n. g n ∈ borel_measurable lebesgue) ∧ (∀n. finite(range (g n))) ∧(∀x.
(λn. g n x) ⟶ f x)
  proof (intro exI allI conjI)
    show 0 ≤ ?g n x for n x
    proof (clarify intro!: ordered_comm_monoid_add_class.sum_nonneg)
      fix k::real
      assume k ∈ ℤ and k: |k| ≤ 2 ^(2*n)
      show 0 ≤ k/2^n * ?Ω n k x
        using f ‹k ∈ ℤ› apply (clarsimp simp: indicator_def field_split_simps
Ints_def)
      by (smt (verit) int_less_real_le mult_nonneg_nonneg of_int_0 zero_le_power)
    qed
    show ?g n x ≤ ?g (Suc n) x for n x
    proof −
      have ?g n x =
           (∑k | k ∈ ℤ ∧ |k| ≤ 2 ^(2*n).
             k/2^n * (indicator {y. k/2^n ≤ f y ∧ f y < (k+1/2)/2^n} x +
             indicator {y. (k+1/2)/2^n ≤ f y ∧ f y < (k+1)/2^n} x))
        by (rule sum.cong [OF refl]) (simp add: indicator_def field_split_simps)
     also have ... = (∑k | k ∈ ℤ ∧ |k| ≤ 2 ^(2*n). k/2^n * indicator {y. k/2^n
```

```
≤ f y ∧ f y < (k+1/2)/2^n} x) +
                        (∑k | k ∈ ℤ ∧ |k| ≤ 2 ^ (2*n). k/2^n * indicator {y.
(k+1/2)/2^n ≤ f y ∧ f y < (k+1)/2^n} x)
      by (simp add: comm_monoid_add_class.sum.distrib algebra_simps)
    also have ... = (∑k | k ∈ ℤ ∧ |k| ≤ 2 ^(2*n). (2 * k)/2 ^ Suc n * indicator
{y. (2 * k)/2 ^ Suc n ≤ f y ∧ f y < (2 * k+1)/2 ^ Suc n} x) +
                        (∑k | k ∈ ℤ ∧ |k| ≤ 2 ^(2*n). (2 * k)/2 ^ Suc n * indicator
{y. (2 * k+1)/2 ^ Suc n ≤ f y ∧ f y < ((2 * k+1) + 1)/2 ^ Suc n} x)
      by (force simp: field_simps indicator_def intro: sum.cong)
      also have ... ≤ (∑k | k ∈ ℤ ∧ |k| ≤ 2 ^(2 * Suc n). k/2 ^ Suc n *
(indicator {y. k/2 ^ Suc n ≤ f y ∧ f y < (k+1)/2 ^ Suc n} x))
          (is ?a + _ ≤ ?b)
    proof −
      have *: ⟦sum f I ≤ sum h I; a + sum h I ≤ b⟧ ⟹ a + sum f I ≤ b for I
a b f and h :: real⇒real
        by linarith
      let ?h = λk. (2*k+1)/2 ^ Suc n *
                  (indicator {y. (2 * k+1)/2 ^ Suc n ≤ f y ∧ f y < ((2*k+1) +
1)/2 ^ Suc n} x)
      show ?thesis
      proof (rule *)
        show (∑k | k ∈ ℤ ∧ |k| ≤ 2 ^(2*n).
              2 * k/2 ^ Suc n * indicator {y. (2 * k+1)/2 ^ Suc n ≤ f y ∧ f y
< (2 * k+1 + 1)/2 ^ Suc n} x)
            ≤ sum ?h {k ∈ ℤ. |k| ≤ 2 ^(2*n)}
          by (rule sum_mono) (simp add: indicator_def field_split_simps)
      next
        have α: ?a = (∑k ∈ (*)2 ` {k ∈ ℤ. |k| ≤ 2 ^(2*n)}.
                  k/2 ^ Suc n * indicator {y. k/2 ^ Suc n ≤ f y ∧ f y < (k+1)/2
^ Suc n} x)
      by (auto simp: inj_on_def field_simps comm_monoid_add_class.sum.reindex)
        have β: sum ?h {k ∈ ℤ. |k| ≤ 2 ^(2*n)}
              = (∑k ∈ (λx. 2*x + 1) ` {k ∈ ℤ. |k| ≤ 2 ^(2*n)}.
                k/2 ^ Suc n * indicator {y. k/2 ^ Suc n ≤ f y ∧ f y < (k+1)/2
^ Suc n} x)
      by (auto simp: inj_on_def field_simps comm_monoid_add_class.sum.reindex)
        have 0: (*) 2 ` {k ∈ ℤ. P k} ∩ (λx. 2 * x + 1) ` {k ∈ ℤ. P k} = {} for
P :: real ⇒ bool
        proof −
          have 2 * i ≠ 2 * j + 1 for i j :: int by arith
          thus ?thesis
            unfolding Ints_def by auto (use of_int_eq_iff in fastforce)
        qed
        have ?a + sum ?h {k ∈ ℤ. |k| ≤ 2 ^(2*n)}
            = (∑k ∈ (*)2 ` {k ∈ ℤ. |k| ≤ 2 ^(2*n)} ∪ (λx. 2*x + 1) ` {k ∈ ℤ.
|k| ≤ 2 ^(2*n)}.
                k/2 ^ Suc n * indicator {y. k/2 ^ Suc n ≤ f y ∧ f y < (k+1)/2 ^
Suc n} x)
          unfolding α β
```

```
          using finite_abs_int_segment [of 2 ^ (2*n)]
          by (subst sum_Un) (auto simp: 0)
        also have ... ≤ ?b
        proof (rule sum_mono2)
          show finite {k::real. k ∈ ℤ ∧ |k| ≤ 2 ^ (2 * Suc n)}
            by (rule finite_abs_int_segment)
          show (*) 2 ` {k::real. k ∈ ℤ ∧ |k| ≤ 2^(2*n)} ∪ (λx. 2*x + 1) ` {k ∈
ℤ. |k| ≤ 2^(2*n)} ⊆ {k ∈ ℤ. |k| ≤ 2 ^ (2 * Suc n)}
            apply (clarsimp simp: image_subset_iff)
            using one_le_power [of 2::real 2*n] by linarith
          have *: ⟦x ∈ (S ∪ T) − U; ⋀x. x ∈ S ⟹ x ∈ U; ⋀x. x ∈ T ⟹ x ∈
U⟧ ⟹ P x for S T U P
            by blast
          have 0 ≤ b if b ∈ ℤ f x * (2 * 2^n) < b + 1 for b
        by (smt (verit, ccfv_SIG) Ints_cases f int_le_real_less mult_nonneg_nonneg
of_int_add one_le_power that)
          then show 0 ≤ b/2 ^ Suc n * indicator {y. b/2 ^ Suc n ≤ f y ∧ f y <
(b + 1)/2 ^ Suc n} x
              if b ∈ {k ∈ ℤ. |k| ≤ 2 ^ (2 * Suc n)} −
                   ((*) 2 ` {k ∈ ℤ. |k| ≤ 2 ^ (2*n)} ∪ (λx. 2*x + 1) ` {k ∈ ℤ.
|k| ≤ 2 ^ (2*n)}) for b
            using that by (simp add: indicator_def divide_simps)
        qed
        finally show ?a + sum ?h {k ∈ ℤ. |k| ≤ 2 ^ (2*n)} ≤ ?b .
      qed
    qed
    finally show ?thesis .
  qed
  show ?g n ∈ borel_measurable lebesgue for n
  apply (intro borel_measurable_indicator borel_measurable_times borel_measurable_sum)
    using leb_f sets_restrict_UNIV by auto
  show finite (range (?g n)) for n
  proof −
    have (∑k | k ∈ ℤ ∧ |k| ≤ 2 ^ (2*n). k/2^n * ?Ω n k x)
          ∈ (λk. k/2^n) ` {k ∈ ℤ. |k| ≤ 2 ^ (2*n)} for x
   proof (cases ∃k. k ∈ ℤ ∧ |k| ≤ 2 ^ (2*n) ∧ k/2^n ≤ f x ∧ f x < (k+1)/2^n)
      case True
      then show ?thesis
        by (blast intro: indicator_sum_eq)
    next
      case False
      then have (∑k | k ∈ ℤ ∧ |k| ≤ 2 ^ (2*n). k/2^n * ?Ω n k x) = 0
        by auto
      then show ?thesis by force
    qed
    then have range (?g n) ⊆ ((λk. (k/2^n)) ` {k. k ∈ ℤ ∧ |k| ≤ 2 ^ (2*n)})
      by auto
    moreover have finite ((λk::real. (k/2^n)) ` {k ∈ ℤ. |k| ≤ 2 ^ (2*n)})
      by (intro finite_imageI finite_abs_int_segment)
```

```
      ultimately show ?thesis
        by (rule finite_subset)
    qed
    show (λn. ?g n x) ⟶ f x for x
    proof (clarsimp simp add: lim_sequentially)
      fix e::real
      assume e > 0
      obtain N1 where N1: 2 ^ N1 > abs(f x)
        using real_arch_pow by fastforce
      obtain N2 where N2: (1/2) ^ N2 < e
        using real_arch_pow_inv ‹e > 0› by fastforce
      have dist (∑ k | k ∈ ℤ ∧ |k| ≤ 2 ^ (2*n). k/2^n * ?Ω n k x) (f x) < e if N1
+ N2 ≤ n for n
      proof −
        let ?m = real_of_int ⌊2^n * f x⌋
        have |?m| ≤ 2^n * 2^N1
          using N1 apply (simp add: f)
       by (meson floor_mono le_floor_iff less_le_not_le mult_le_cancel_left_pos
zero_less_numeral zero_less_power)
        also have ... ≤ 2 ^ (2*n)
          by (metis that add_leD1 add_le_cancel_left mult.commute mult_2_right
one_less_numeral_iff
                    power_add power_increasing_iff semiring_norm(76))
        finally have m_le: |?m| ≤ 2 ^ (2*n) .
        have ?m/2^n ≤ f x f x < (?m + 1)/2^n
          by (auto simp: mult.commute pos_divide_le_eq mult_imp_less_div_pos)
       then have eq: dist (∑ k | k ∈ ℤ ∧ |k| ≤ 2 ^ (2*n). k/2^n * ?Ω n k x) (f x)
                    = dist (?m/2^n) (f x)
          by (subst indicator_sum_eq [of ?m]) (auto simp: m_le)
        have |2^n| * |?m/2^n − f x| = |2^n * (?m/2^n − f x)|
          by (simp add: abs_mult)
        also have ... < 2 ^ N2 * e
          using N2 by (simp add: divide_simps mult.commute) linarith
        also have ... ≤ |2^n| * e
          using that ‹e > 0› by auto
        finally show ?thesis
          using eq by (simp add: dist_real_def)
      qed
      then show ∃no. ∀n≥no. dist (∑ k | k ∈ ℤ ∧ |k| ≤ 2 ^ (2*n). k * ?Ω n k
x/2^n) (f x) < e
        by force
    qed
  qed
  ultimately show ?rhs
    by metis
next
  assume RHS: ?rhs
  with borel_measurable_simple_function_limit [of f UNIV, unfolded lebesgue_on_UNIV_eq]
  show ?lhs
```

by (*blast intro*: *order_trans*)
qed

6.41.2 Borel measurable Jacobian determinant

lemma *lemma_partial_derivatives0*:
fixes $f :: 'a::euclidean\_space \Rightarrow 'b::euclidean\_space$
assumes *linear f* **and** *lim0*: $((\lambda x.\ f\ x\ /_R\ norm\ x) \longrightarrow 0)\ (at\ 0\ within\ S)$
and *lb*: $\bigwedge v.\ v \neq 0 \Longrightarrow (\exists k{>}0.\ \forall e{>}0.\ \exists x.\ x \in S - \{0\} \wedge norm\ x < e \wedge k * norm\ x \leq |v \cdot x|)$
shows $f\ x = 0$
proof −
interpret *linear f* **by** *fact*
have $dim\ \{x.\ f\ x = 0\} \leq DIM('a)$
by (*rule dim_subset_UNIV*)
moreover have *False* **if** *less*: $dim\ \{x.\ f\ x = 0\} < DIM('a)$
proof −
obtain d **where** $d \neq 0$ **and** d: $\bigwedge y.\ f\ y = 0 \Longrightarrow d \cdot y = 0$
using *orthogonal_to_subspace_exists* [*OF less*] *orthogonal_def*
by (*metis* (*mono_tags*, *lifting*) *mem_Collect_eq span_base*)
then obtain k **where** $k > 0$
and k: $\bigwedge e.\ e > 0 \Longrightarrow \exists y.\ y \in S - \{0\} \wedge norm\ y < e \wedge k * norm\ y \leq |d \cdot y|$
using *lb* **by** *blast*
have $\exists h.\ \forall n.\ ((h\ n \in S \wedge h\ n \neq 0 \wedge k * norm\ (h\ n) \leq |d \cdot h\ n|) \wedge norm\ (h\ n) < 1\ /\ real\ (Suc\ n)) \wedge$
$norm\ (h\ (Suc\ n)) < norm\ (h\ n)$
proof (*rule dependent_nat_choice*)
show $\exists y.\ (y \in S \wedge y \neq 0 \wedge k * norm\ y \leq |d \cdot y|) \wedge norm\ y < 1\ /\ real\ (Suc\ 0)$
by *simp* (*metis DiffE insertCI k not_less not_one_le_zero*)
qed (*use k* [*of min* (*norm x*) (*1/*(*Suc n* + *1*))) **for** *x n*] **in** *auto*)
then obtain α **where** α: $\bigwedge n.\ \alpha\ n \in S - \{0\}$ **and** *kd*: $\bigwedge n.\ k * norm(\alpha\ n) \leq |d \cdot \alpha\ n|$
and *norm_lt*: $\bigwedge n.\ norm(\alpha\ n) < 1/(Suc\ n)$
by *force*
let $?\beta = \lambda n.\ \alpha\ n\ /_R\ norm\ (\alpha\ n)$
have *com*: $\bigwedge g.\ (\forall n.\ g\ n \in sphere\ (0::'a)\ 1) \Longrightarrow \exists l \in sphere\ 0\ 1.\ \exists \varrho::nat\Rightarrow nat.\ strict\_mono\ \varrho \wedge (g \circ \varrho) \longrightarrow l$
using *compact_sphere compact_def* **by** *metis*
moreover have $\forall n.\ ?\beta\ n \in sphere\ 0\ 1$
using α **by** *auto*
ultimately obtain $l::'a$ **and** $\varrho::nat\Rightarrow nat$
where l: $l \in sphere\ 0\ 1$ **and** *strict_mono* ϱ **and** *to_l*: $(?\beta \circ \varrho) \longrightarrow l$
by *meson*
moreover have *continuous* $(at\ l)\ (\lambda x.\ (|d \cdot x| - k))$
by (*intro continuous_intros*)
ultimately have *lim_dl*: $((\lambda x.\ (|d \cdot x| - k)) \circ (?\beta \circ \varrho)) \longrightarrow (|d \cdot l| - k)$
by (*meson continuous_imp_tendsto*)

```
    have ∀⇩F i in sequentially. 0 ≤ ((λx. |d • x| − k) ∘ ((λn. α n /⇩R norm (α n))
∘ ϱ)) i
      using α kd by (auto simp: field_split_simps)
    then have k ≤ |d • l|
      using tendsto_lowerbound [OF lim_dl, of 0] by auto
    moreover have d • l = 0
    proof (rule d)
      show f l = 0
      proof (rule LIMSEQ_unique [of f ∘ ?β ∘ ϱ])
        have isCont f l
            using ‹linear f› linear_continuous_at linear_conv_bounded_linear by
blast
        then show (f ∘ (λn. α n /⇩R norm (α n)) ∘ ϱ) ⟶ f l
          unfolding comp_assoc
          using to_l continuous_imp_tendsto by blast
        have α ⟶ 0
          using norm_lt LIMSEQ_norm_0 by metis
        with ‹strict_mono ϱ› have (α ∘ ϱ) ⟶ 0
          by (metis LIMSEQ_subseq_LIMSEQ)
        with lim0 α have ((λx. f x /⇩R norm x) ∘ (α ∘ ϱ)) ⟶ 0
          by (force simp: tendsto_at_iff_sequentially)
        then show (f ∘ (λn. α n /⇩R norm (α n)) ∘ ϱ) ⟶ 0
          by (simp add: o_def scale)
      qed
    qed
    ultimately show False
      using ‹k > 0› by auto
  qed
  ultimately have dim: dim {x. f x = 0} = DIM('a)
    by force
  then show ?thesis
   by (metis (mono_tags, lifting) dim_eq_full UNIV_I eq_0_on_span mem_Collect_eq
span_raw_def)
qed

lemma lemma_partial_derivatives:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes linear f and lim: ((λx. f (x − a) /⇩R norm (x − a)) ⟶ 0) (at a
within S)
    and lb: ⋀v. v ≠ 0 ⟹ (∃k>0. ∀e>0. ∃x ∈ S − {a}. norm(a − x) < e ∧ k
* norm(a − x) ≤ |v • (x − a)|)
  shows f x = 0
proof −
  have ((λx. f x /⇩R norm x) ⟶ 0) (at 0 within (λx. x−a) ‘ S)
    using lim by (simp add: Lim_within dist_norm)
  then show ?thesis
  proof (rule lemma_partial_derivatives0 [OF ‹linear f›])
    fix v :: 'a
    assume v: v ≠ 0
```

show $\exists k{>}0.\ \forall e{>}0.\ \exists x.\ x \in (\lambda x.\ x - a)\ `\ S - \{0\} \wedge norm\ x < e \wedge k * norm\ x \leq |v \cdot x|$
using *lb* [*OF v*] **by** (*force simp*: *norm_minus_commute*)
qed
qed

proposition *borel_measurable_partial_derivatives*:
fixes $f :: real\hat{}'m::\{finite,wellorder\} \Rightarrow real\hat{}'n$
assumes *S*: $S \in sets\ lebesgue$
and *f*: $\bigwedge x.\ x \in S \Longrightarrow (f\ has\_derivative\ f'\ x)\ (at\ x\ within\ S)$
shows $(\lambda x.\ (matrix(f'\ x)\$m\$n)) \in borel\_measurable\ (lebesgue\_on\ S)$
proof −
have *contf*: *continuous_on S f*
using *continuous_on_eq_continuous_within f has_derivative_continuous* **by** *blast*
have $\{x \in S.\ (matrix\ (f'\ x)\$m\$n) \leq b\} \in sets\ lebesgue$ **for** *b*
proof (*rule sets_negligible_symdiff*)
let *?T* = $\{x \in S.\ \forall e{>}0.\ \exists d{>}0.\ \exists A.\ A\$m\$n < b \wedge (\forall i\ j.\ A\$i\$j \in \mathbb{Q}) \wedge$
$(\forall y \in S.\ norm(y - x) < d \longrightarrow norm(f\ y - f\ x - A *v (y - x)) \leq e * norm(y - x))\}$
let *?U* = $S \cap$
$(\bigcap e \in \{e \in \mathbb{Q}.\ e > 0\}.$
$\bigcup A \in \{A.\ A\$m\$n < b \wedge (\forall i\ j.\ A\$i\$j \in \mathbb{Q})\}.$
$\bigcup d \in \{d \in \mathbb{Q}.\ 0 < d\}.$
$S \cap (\bigcap y \in S.\ \{x \in S.\ norm(y - x) < d \longrightarrow norm(f\ y - f\ x - A *v (y - x)) \leq e * norm(y - x)\}))$
have *?T* = *?U*
proof (*intro set_eqI iffI*)
fix *x*
assume *xT*: $x \in ?T$
then show $x \in ?U$
proof (*clarsimp simp add*:)
fix *q* :: *real*
assume $q \in \mathbb{Q}\ q > 0$
then obtain *d A* **where** $d > 0$ **and** *A*: $A\ \$\ m\ \$\ n < b \bigwedge i\ j.\ A\ \$\ i\ \$\ j \in \mathbb{Q}$
$\bigwedge y.\ [\![y \in S;\ norm\ (y - x) < d]\!] \Longrightarrow norm\ (f\ y - f\ x - A *v (y - x)) \leq q * norm\ (y - x)$
using *xT* **by** *auto*
then obtain δ **where** $d > \delta\ \delta > 0\ \delta \in \mathbb{Q}$
using *Rats_dense_in_real* **by** *blast*
with *A* **show** $\exists A.\ A\ \$\ m\ \$\ n < b \wedge (\forall i\ j.\ A\ \$\ i\ \$\ j \in \mathbb{Q}) \wedge$
$(\exists s.\ s \in \mathbb{Q} \wedge 0 < s \wedge (\forall y{\in}S.\ norm\ (y - x) < s \longrightarrow norm\ (f\ y - f\ x - A *v (y - x)) \leq q * norm\ (y - x)))$
by *force*
qed
next
fix *x*
assume *xU*: $x \in ?U$

```
      then show x ∈ ?T
      proof clarsimp
        fix e :: real
        assume e > 0
        then obtain ε where ε: e > ε ε > 0 ε ∈ ℚ
          using Rats_dense_in_real by blast
        with xU obtain A r where x ∈ S and Ar: A $ m $ n < b ∀ i j. A $ i $ j
∈ ℚ r ∈ ℚ r > 0
          and ∀ y∈S. norm (y − x) < r ⟶ norm (f y − f x − A *v (y − x)) ≤ ε
* norm (y − x)
          by (auto simp: split: if_split_asm)
        then have ∀ y∈S. norm (y − x) < r ⟶ norm (f y − f x − A *v (y − x))
≤ e * norm (y − x)
          by (meson ‹e > ε› less_eq_real_def mult_right_mono norm_ge_zero
order_trans)
        then show ∃ d>0. ∃ A. A $ m $ n < b ∧ (∀ i j. A $ i $ j ∈ ℚ) ∧ (∀ y∈S.
norm (y − x) < d ⟶ norm (f y − f x − A *v (y − x)) ≤ e * norm (y − x))
          using ‹x ∈ S› Ar by blast
      qed
    qed
    moreover have ?U ∈ sets lebesgue
    proof −
      have coQ: countable {e ∈ ℚ. 0 < e}
        using countable_Collect countable_rat by blast
      have ne: {e ∈ ℚ. (0::real) < e} ≠ {}
        using zero_less_one Rats_1 by blast
      have coA: countable {A. A $ m $ n < b ∧ (∀ i j. A $ i $ j ∈ ℚ)}
      proof (rule countable_subset)
        show countable {A. ∀ i j. A $ i $ j ∈ ℚ}
          using countable_vector [OF countable_vector, of λi j. ℚ] by (simp add:
countable_rat)
      qed blast
      have *: ⟦U ≠ {} ⟹ closedin (top_of_set S) (S ∩ ⋂ U)⟧
            ⟹ closedin (top_of_set S) (S ∩ ⋂ U) for U
        by fastforce
      have eq: {x::(real,'m)vec. P x ∧ (Q x ⟶ R x)} = {x. P x ∧ ¬ Q x} ∪ {x.
P x ∧ R x} for P Q R
        by auto
      have sets: S ∩ (⋂ y∈S. {x ∈ S. norm (y − x) < d ⟶ norm (f y − f x − A
*v (y − x)) ≤ e * norm (y − x)})
                ∈ sets lebesgue for e A d
      proof −
        have clo: closedin (top_of_set S)
                  {x ∈ S. norm (y − x) < d ⟶ norm (f y − f x − A *v (y − x))
≤ e * norm (y − x)}
          for y
        proof −
          have cont1: continuous_on S (λx. norm (y − x))
           and  cont2: continuous_on S (λx. e * norm (y − x) − norm (f y − f x
```

```
− (A *v y − A *v x)))
          by (force intro: contf continuous_intros)+
        have clo1: closedin (top_of_set S) {x ∈ S. d ≤ norm (y − x)}
         using continuous_closedin_preimage [OF cont1, of {d..}] by (simp add:
vimage_def Int_def)
        have clo2: closedin (top_of_set S)
                    {x ∈ S. norm (f y − f x − (A *v y − A *v x)) ≤ e * norm (y
− x)}
         using continuous_closedin_preimage [OF cont2, of {0..}] by (simp add:
vimage_def Int_def)
        show ?thesis
          by (auto simp: eq not_less matrix_vector_mult_diff_distrib intro: clo1
clo2)
       qed
       show ?thesis
         by (rule lebesgue_closedin [of S]) (force intro: * S clo)+
     qed
     show ?thesis
      by (intro sets sets.Int S sets.countable_UN'' sets.countable_INT'' coQ coA)
auto
   qed
   ultimately show ?T ∈ sets lebesgue
     by simp
   let ?M = (?T − {x ∈ S. matrix (f' x) $ m $ n ≤ b} ∪ ({x ∈ S. matrix (f' x)
$ m $ n ≤ b} − ?T))
   let ?Θ = λx v. ∀ξ>0. ∃e>0. ∀y ∈ S−{x}. norm (x − y) < e ⟶ |v • (y −
x)| < ξ * norm (x − y)
   have nN: negligible {x ∈ S. ∃v≠0. ?Θ x v}
     unfolding negligible_eq_zero_density
   proof clarsimp
     fix x v and r e :: real
     assume x ∈ S v ≠ 0 r > 0 e > 0
     and Theta [rule_format]: ?Θ x v
     moreover have (norm v * e / 2) / CARD('m) ^ CARD('m) > 0
       by (simp add: ‹v ≠ 0› ‹e > 0›)
     ultimately obtain d where d > 0
       and dless: ⋀y. ⟦y ∈ S − {x}; norm (x − y) < d⟧ ⟹
                  |v • (y − x)| < ((norm v * e / 2) / CARD('m) ^ CARD('m))
* norm (x − y)
       by metis
     let ?W = ball x (min d r) ∩ {y. |v • (y − x)| < (norm v * e/2 * min d r) /
CARD('m) ^ CARD('m)}
     have open {x. |v • (x − a)| < b} for a b
       by (intro open_Collect_less continuous_intros)
     show ∃d>0. d ≤ r ∧
          (∃U. {x' ∈ S. ∃v≠0. ?Θ x' v} ∩ ball x d ⊆ U ∧
              U ∈ lmeasurable ∧ measure lebesgue U < e * content (ball x d))
     proof (intro exI conjI)
       show 0 < min d r min d r ≤ r
```

```
        using ‹r > 0› ‹d > 0› by auto
      show {x′ ∈ S. ∃ v. v ≠ 0 ∧ (∀ξ>0. ∃ e>0. ∀ z∈S − {x′}. norm (x′ − z)
< e ⟶ |v • (z − x′)| < ξ * norm (x′ − z))} ∩ ball x (min d r) ⊆ ?W
        proof (clarsimp simp: dist_norm norm_minus_commute)
          fix y w
          assume y ∈ S w ≠ 0
            and less [rule_format]:
                ∀ξ>0. ∃ e>0. ∀ z∈S − {y}. norm (y − z) < e ⟶ |w • (z − y)|
< ξ * norm (y − z)
            and d: norm (y − x) < d and r: norm (y − x) < r
           show |v • (y − x)| < norm v * e * min d r / (2 * real CARD('m) ^
CARD('m))
          proof (cases y = x)
            case True
            with ‹r > 0› ‹d > 0› ‹e > 0› ‹v ≠ 0› show ?thesis
              by simp
          next
            case False
           have |v • (y − x)| < norm v * e / 2 / real (CARD('m) ^ CARD('m))
* norm (x − y)
                  by (metis Diff_iff False ‹y ∈ S› d dless empty_iff insert_iff
norm_minus_commute)
             also have ... ≤ norm v * e * min d r / (2 * real CARD('m) ^
CARD('m))
              using d r ‹e > 0› by (simp add: field_simps norm_minus_commute
mult_left_mono)
            finally show ?thesis .
          qed
        qed
        show ?W ∈ lmeasurable
          by (simp add: fmeasurable_Int_fmeasurable borel_open)
        obtain k::'m where True
          by metis
       obtain T where T: orthogonal_transformation T and v: v = T(norm v
*R axis k (1::real))
          using rotation_rightward_line by metis
        define b where b ≡ norm v
        have b > 0
          using ‹v ≠ 0› by (auto simp: b_def)
         obtain eqb: inv T v = b *R axis k (1::real) and inj T bij T and invT:
orthogonal_transformation (inv T)
              by (metis UNIV_I b_def  T v bij_betw_inv_into_left orthogo-
nal_transformation_inj orthogonal_transformation_bij orthogonal_transformation_inv)
        let ?v = χ i. min d r / CARD('m)
         let ?v′ = χ i. if i = k then (e/2 * min d r) / CARD('m) ^ CARD('m)
else min d r
        let ?x′ = inv T x
         let ?W′ = (ball ?x′ (min d r) ∩ {y. |(y − ?x′)$k| < e * min d r / (2 *
CARD('m) ^ CARD('m))})
```

```
        have abs: x − e ≤ y ∧ y ≤ x + e ⟷ abs(y − x) ≤ e for x y e::real
          by auto
        have ?W = T ‘ ?W'
        proof −
          have 1: T ‘ (ball (inv T x) (min d r)) = ball x (min d r)
              by (simp add: T image_orthogonal_transformation_ball orthogonal_transformation_surj surj_f_inv_f)
           have 2: {y. |v • (y − x)| < b ∗ e ∗ min d r / (2 ∗ real CARD('m) ^ CARD('m))} =
                    T ‘ {y. |y $ k − ?x' $ k| < e ∗ min d r / (2 ∗ real CARD('m) ^ CARD('m))}
          proof −
           have ∗: |T (b ∗R axis k 1) • (y − x)| = b ∗ |inv T y $ k − ?x' $ k| for y
           proof −
            have |T (b ∗R axis k 1) • (y − x)| = |(b ∗R axis k 1) • inv T (y − x)|
                by (metis (no_types, opaque_lifting) b_def eqb invT orthogonal_transformation_def v)
             also have ... = b ∗ |(axis k 1) • inv T (y − x)|
               using ‹b > 0› by (simp add: abs_mult)
             also have ... = b ∗ |inv T y $ k − ?x' $ k|
               using orthogonal_transformation_linear [OF invT]
               by (simp add: inner_axis' linear_diff)
             finally show ?thesis
               by simp
            qed
            show ?thesis
              using v b_def [symmetric]
              using ‹b > 0› by (simp add: ∗ bij_image_Collect_eq [OF ‹bij T›] mult_less_cancel_left_pos times_divide_eq_right [symmetric] del: times_divide_eq_right)
          qed
          show ?thesis
           using ‹b > 0› by (simp add: image_Int ‹inj T› 1 2 b_def [symmetric])
        qed
        moreover have ?W' ∈ lmeasurable
          by (auto intro: fmeasurable_Int_fmeasurable)
        ultimately have measure lebesgue ?W = measure lebesgue ?W'
          by (metis measure_orthogonal_image T)
        also have ... ≤ measure lebesgue (cbox (?x' − ?v') (?x' + ?v'))
        proof (rule measure_mono_fmeasurable)
          show ?W' ⊆ cbox (?x' − ?v') (?x' + ?v')
        apply (clarsimp simp add: mem_box_cart abs dist_norm norm_minus_commute simp del: min_less_iff_conj min.bounded_iff)
            by (metis component_le_norm_cart less_eq_real_def le_less_trans vector_minus_component)
        qed auto
        also have ... ≤ e/2 ∗ measure lebesgue (cbox (?x' − ?v) (?x' + ?v))
        proof −
          have cbox (?x' − ?v) (?x' + ?v) ≠ {}
```

```
                using ‹r > 0› ‹d > 0› by (auto simp: interval_eq_empty_cart
divide_less_0_iff)
            with ‹r > 0› ‹d > 0› ‹e > 0› show ?thesis
              apply (simp add: content_cbox_if_cart mem_box_cart)
              apply (auto simp: prod_nonneg)
           apply (simp add: abs if_distrib prod.delta_remove field_simps power_diff
split: if_split_asm)
            done
          qed
          also have ... ≤ e/2 * measure lebesgue (cball ?x' (min d r))
          proof (rule mult_left_mono [OF measure_mono_fmeasurable])
            have *: norm (?x' − y) ≤ min d r
              if y: ⋀i. |?x' $ i − y $ i| ≤ min d r / real CARD('m) for y
            proof −
              have norm (?x' − y) ≤ (∑ i∈UNIV. |(?x' − y) $ i|)
                by (rule norm_le_l1_cart)
              also have ... ≤ real CARD('m) * (min d r / real CARD('m))
                by (rule sum_bounded_above) (use y in auto)
              finally show ?thesis
                by simp
            qed
            show cbox (?x' − ?v) (?x' + ?v) ⊆ cball ?x' (min d r)
                 apply (clarsimp simp only: mem_box_cart dist_norm mem_cball
intro!: *)
              by (simp add: abs_diff_le_iff abs_minus_commute)
          qed (use ‹e > 0› in auto)
          also have ... < e * content (cball ?x' (min d r))
            using ‹r > 0› ‹d > 0› ‹e > 0› by (auto intro: content_cball_pos)
          also have ... = e * content (ball x (min d r))
           using ‹r > 0› ‹d > 0› content_ball_conv_unit_ball[of min d r inv T x]
                 content_ball_conv_unit_ball[of min d r x]
            by (simp add: content_cball_conv_ball)
          finally show measure lebesgue ?W < e * content (ball x (min d r)) .
        qed
      qed
      have *: (⋀x. (x ∉ S) ⟹ (x ∈ T ⟷ x ∈ U)) ⟹ (T − U) ∪ (U − T) ⊆ S
for S T U :: (real,'m) vec set
        by blast
      have MN: ?M ⊆ {x ∈ S. ∃ v≠0. ?Θ x v}
      proof (rule *)
        fix x
        assume x: x ∉ {x ∈ S. ∃ v≠0. ?Θ x v}
        show (x ∈ ?T) ⟷ (x ∈ {x ∈ S. matrix (f' x) $ m $ n ≤ b})
        proof (cases x ∈ S)
          case True
          then have x: ¬ ?Θ x v if v ≠ 0 for v
            using x that by force
          show ?thesis
          proof (rule iffI; clarsimp)
```

```
      assume b: ∀e>0. ∃d>0. ∃A. A $ m $ n < b ∧ (∀i j. A $ i $ j ∈ ℚ) ∧
                          (∀y∈S. norm (y − x) < d ⟶ norm (f y − f x − A
*v (y − x)) ≤ e * norm (y − x))
                (is ∀e>0. ∃d>0. ∃A. ?Φ e d A)
      then have ∀k. ∃d>0. ∃A. ?Φ (1 / Suc k) d A
      by (metis (no_types, opaque_lifting) less_Suc_eq_0_disj of_nat_0_less_iff
zero_less_divide_1_iff)
      then obtain δ A where δ: ⋀k. δ k > 0
                        and Ab: ⋀k. A k $ m $ n < b
                        and A: ⋀k y. ⟦y ∈ S; norm (y − x) < δ k⟧ ⟹
                                  norm (f y − f x − A k *v (y − x)) ≤ 1/(Suc k)
* norm (y − x)
        by metis
      have ∀i j. ∃a. (λn. A n $ i $ j) ⟶⟶ a
      proof (intro allI)
        fix i j
        have vax: (A n *v axis j 1) $ i = A n $ i $ j for n
          by (metis cart_eq_inner_axis matrix_vector_mul_component)
        let ?CA = {x. Cauchy (λn. (A n) *v x)}
        have subspace ?CA
          unfolding subspace_def convergent_eq_Cauchy [symmetric]
            by (force simp: algebra_simps intro: tendsto_intros)
        then have CA_eq: ?CA = span ?CA
          by (metis span_eq_iff)
        also have ... = UNIV
        proof −
          have dim ?CA ≤ CARD('m)
            using dim_subset_UNIV[of ?CA] by auto
          moreover have False if less: dim ?CA < CARD('m)
          proof −
           obtain d where d ≠ 0 and d: ⋀y. y ∈ span ?CA ⟹ orthogonal d y
             using less by (force intro: orthogonal_to_subspace_exists [of ?CA])
            with x [OF ‹d ≠ 0›] obtain ξ where ξ > 0
             and ξ: ⋀e. e > 0 ⟹ ∃y ∈ S − {x}. norm (x − y) < e ∧ ξ * norm
(x − y) ≤ |d • (y − x)|
              by (fastforce simp: not_le Bex_def)
            obtain γ z where γSx: ⋀i. γ i ∈ S − {x}
                        and γle:  ⋀i. ξ * norm(γ i − x) ≤ |d • (γ i − x)|
                        and γx:   γ ⟶⟶ x
                        and z:    (λn. (γ n − x) /R norm (γ n − x)) ⟶⟶ z
            proof −
              have ∃γ. (∀i. (γ i ∈ S − {x} ∧
                                ξ * norm(γ i − x) ≤ |d • (γ i − x)| ∧ norm(γ i − x)
< 1/Suc i) ∧
                                norm(γ(Suc i) − x) < norm(γ i − x))
              proof (rule dependent_nat_choice)
                show ∃y. y ∈ S − {x} ∧ ξ * norm (y − x) ≤ |d • (y − x)| ∧
norm (y − x) < 1 / Suc 0
                 using ξ [of 1] by (auto simp: dist_norm norm_minus_commute)
```

```
                next
                  fix y i
                 assume y ∈ S − {x} ∧ ξ ∗ norm (y − x) ≤ |d • (y − x)| ∧ norm
(y − x) < 1/Suc i
                  then have min (norm(y − x)) (1/((Suc i) + 1)) > 0
                    by auto
                   then obtain y′ where y′ ∈ S − {x} and y′: norm (x − y′) <
min (norm (y − x)) (1/((Suc i) + 1))
                                        ξ ∗ norm (x − y′) ≤ |d • (y′ − x)|
                    using ξ by metis
                  with ξ show ∃ y′. (y′ ∈ S − {x} ∧ ξ ∗ norm (y′ − x) ≤ |d • (y′
− x)| ∧
                                norm (y′ − x) < 1/(Suc (Suc i))) ∧ norm (y′ − x) <
norm (y − x)
                    by (auto simp: dist_norm norm_minus_commute)
                qed
                then obtain γ where
                     γSx: ⋀i. γ i ∈ S − {x}
                     and γle: ⋀i. ξ ∗ norm(γ i − x) ≤ |d • (γ i − x)|
                     and γconv: ⋀i. norm(γ i − x) < 1/(Suc i)
                  by blast
                let ?f = λi. (γ i − x) /R norm (γ i − x)
                have ?f i ∈ sphere 0 1 for i
                  using γSx by auto
                then obtain l ϱ where l ∈ sphere 0 1 strict_mono ϱ and l: (?f ∘
ϱ) ⟶ l
              using compact_sphere [of 0::(real,'m) vec 1] unfolding compact_def
by meson
                show thesis
                proof
                  show (γ ∘ ϱ) i ∈ S − {x} ξ ∗ norm ((γ ∘ ϱ) i − x) ≤ |d • ((γ ∘
ϱ) i − x)| for i
                    using γSx γle by auto
                  have γ ⟶ x
                  proof (clarsimp simp add: LIMSEQ_def dist_norm)
                    fix r :: real
                    assume r > 0
                    with real_arch_invD obtain no where no ≠ 0 real no > 1/r
                            by (metis divide_less_0_1_iff not_less_iff_gr_or_eq
of_nat_0_eq_iff reals_Archimedean2)
                    with γconv show ∃ no. ∀ n≥no. norm (γ n − x) < r
                   by (metis ‹r > 0› add.commute divide_inverse inverse_inverse_eq
inverse_less_imp_less less_trans mult.left_neutral nat_le_real_less of_nat_Suc)
                  qed
                  with ‹strict_mono ϱ› show (γ ∘ ϱ) ⟶ x
                    by (metis LIMSEQ_subseq_LIMSEQ)
                  show (λn. ((γ ∘ ϱ) n − x) /R norm ((γ ∘ ϱ) n − x)) ⟶ l
                    using l by (auto simp: o_def)
                qed
```

```
            qed
            have isCont (λx. (|d • x| − ξ)) z
              by (intro continuous_intros)
            from isCont_tendsto_compose [OF this z]
            have lim: (λy. |d • ((γ y − x) /R norm (γ y − x))| − ξ) ⟶ |d •
z| − ξ
              by auto
             moreover have ∀F i in sequentially. 0 ≤ |d • ((γ i − x) /R norm
(γ i − x))| − ξ
            proof (rule eventuallyI)
              fix n
              show 0 ≤ |d • ((γ n − x) /R norm (γ n − x))| − ξ
                using γle [of n] γSx by (auto simp: abs_mult divide_simps)
            qed
            ultimately have ξ ≤ |d • z|
              using tendsto_lowerbound [where a=0] by fastforce
            have Cauchy (λn. (A n) ∗v z)
            proof (clarsimp simp add: Cauchy_def)
              fix ε :: real
              assume 0 < ε
              then obtain N::nat where N > 0 and N: ε/2 > 1/N
          by (metis half_gt_zero inverse_eq_divide neq0_conv real_arch_inverse)
              show ∃M. ∀m≥M. ∀n≥M. dist (A m ∗v z) (A n ∗v z) < ε
              proof (intro exI allI impI)
                fix i j
                assume ij: N ≤ i N ≤ j
                let ?V = λi k. A i ∗v ((γ k − x) /R norm (γ k − x))
                have ∀F k in sequentially. dist (γ k) x < min (δ i) (δ j)
                 using γx [unfolded tendsto_iff] by (meson min_less_iff_conj δ)
                then have even: ∀F k in sequentially. norm (?V i k − ?V j k) −
2 / N ≤ 0
                proof (rule eventually_mono, clarsimp)
                  fix p
                  assume p: dist (γ p) x < δ i dist (γ p) x < δ j
                  let ?C = λk. f (γ p) − f x − A k ∗v (γ p − x)
                  have norm ((A i − A j) ∗v (γ p − x)) = norm (?C j − ?C i)
                    by (simp add: algebra_simps)
                  also have ... ≤ norm (?C j) + norm (?C i)
                    using norm_triangle_ineq4 by blast
                    also have ... ≤ 1/(Suc j) ∗ norm (γ p − x) + 1/(Suc i) ∗
norm (γ p − x)
                    by (metis A Diff_iff γSx dist_norm p add_mono)
                  also have ... ≤ 1/N ∗ norm (γ p − x) + 1/N ∗ norm (γ p −
x)
                      using ij ‹N > 0› by (intro add_mono mult_right_mono)
(auto simp: field_simps)
                  also have ... = 2 / N ∗ norm (γ p − x)
                    by simp
                  finally have no_le: norm ((A i − A j) ∗v (γ p − x)) ≤ 2 / N
```

```
* norm (γ p − x) .
                  have norm (?V i p − ?V j p) =
                        norm ((A i − A j) *v ((γ p − x) /R norm (γ p − x)))
                    by (simp add: algebra_simps)
                  also have ... = norm ((A i − A j) *v (γ p − x)) / norm (γ p
− x)
                    by (simp add: divide_inverse matrix_vector_mult_scaleR)
                  also have ... ≤ 2 / N
                    using no_le by (auto simp: field_split_simps)
                  finally show norm (?V i p − ?V j p) ≤ 2 / N .
                qed
                have isCont (λw. (norm(A i *v w − A j *v w) − 2 / N)) z
                  by (intro continuous_intros)
                from isCont_tendsto_compose [OF this z]
                have lim: (λw. norm (A i *v ((γ w − x) /R norm (γ w − x)) −
                                A j *v ((γ w − x) /R norm (γ w − x))) − 2 / N)
                          ⟶ norm (A i *v z − A j *v z) − 2 / N
                  by auto
                have dist (A i *v z) (A j *v z) ≤ 2 / N
                      using tendsto_upperbound [OF lim even] by (auto simp:
dist_norm)
                with N show dist (A i *v z) (A j *v z) < ε
                  by linarith
              qed
            qed
            then have d • z = 0
              using CA_eq d orthogonal_def by auto
            then show False
              using ‹0 < ξ› ‹ξ ≤ |d • z|› by auto
          qed
          ultimately show ?thesis
            using dim_eq_full by fastforce
        qed
        finally have ?CA = UNIV .
        then have Cauchy (λn. (A n) *v axis j 1)
          by auto
        then obtain L where (λn. A n *v axis j 1) ⟶ L
          by (auto simp: Cauchy_convergent_iff convergent_def)
        then have (λx. (A x *v axis j 1) $ i) ⟶ L $ i
          by (rule tendsto_vec_nth)
        then show ∃ a. (λn. A n $ i $ j) ⟶ a
          by (force simp: vax)
      qed
      then obtain B where B: ⋀i j. (λn. A n $ i $ j) ⟶ B $ i $ j
        by (auto simp: lambda_skolem)
      have lin_df: linear (f' x)
           and lim_df: ((λy. (1 / norm (y − x)) *R (f y − (f x + f' x (y −
x)))) ⟶ 0) (at x within S)
      using ‹x ∈ S› assms by (auto simp: has_derivative_within linear_linear)
```

```
        moreover
        interpret linear f' x by fact
        have (matrix (f' x) - B) *v w = 0 for w
        proof (rule lemma_partial_derivatives [of (*v) (matrix (f' x) - B)])
          show linear ((*v) (matrix (f' x) - B))
            by (rule matrix_vector_mul_linear)
          have ((λy. ((f x + f' x (y - x)) - f y) /R norm (y - x)) ⟶ 0) (at
x within S)
            using tendsto_minus [OF lim_df] by (simp add: field_split_simps)
            then show ((λy. (matrix (f' x) - B) *v (y - x) /R norm (y - x))
⟶ 0) (at x within S)
          proof (rule Lim_transform)
            have ((λy. ((f y + B *v x - (f x + B *v y)) /R norm (y - x))) ⟶
0) (at x within S)
            proof (clarsimp simp add: Lim_within dist_norm)
              fix e :: real
              assume e > 0
              then obtain q::nat where q ≠ 0 and qe2: 1/q < e/2
                  by (metis divide_pos_pos inverse_eq_divide real_arch_inverse
zero_less_numeral)
              let ?g = λp. sum  (λi. sum (λj. abs((A p - B)$i$j)) UNIV) UNIV
              have (λk. onorm (λy. (A k - B) *v y)) ⟶ 0
              proof (rule Lim_null_comparison)
                show ∀F k in sequentially. norm (onorm (λy. (A k - B) *v y)) ≤
?g k
                proof (rule eventually_sequentiallyI)
                  fix k :: nat
                  assume 0 ≤ k
                  have 0 ≤ onorm ((*v) (A k - B))
                    using matrix_vector_mul_bounded_linear
                    by (rule onorm_pos_le)
                     then show norm (onorm ((*v) (A k - B))) ≤ (∑ i∈UNIV.
∑ j∈UNIV. |(A k - B) $ i $ j|)
                         by (simp add: onorm_le_matrix_component_sum del:
vector_minus_component)
                qed
              next
                show ?g ⟶ 0
                    using B Lim_null tendsto_rabs_zero_iff by (fastforce intro!:
tendsto_null_sum)
              qed
              with ‹e > 0› obtain p where ⋀n. n ≥ p ⟹ |onorm ((*v) (A n -
B))| < e/2
                  unfolding lim_sequentially by (metis diff_zero dist_real_def
divide_pos_pos zero_less_numeral)
              then have pqe2: |onorm ((*v) (A (p + q) - B))| < e/2
                using le_add1 by blast
              show ∃ d>0. ∀ y∈S. y ≠ x ∧ norm (y - x) < d ⟶
                    inverse (norm (y - x)) * norm (f y + B *v x - (f x + B *v
```

```
y)) < e
              proof (intro exI, safe)
                show 0 < δ(p + q)
                  by (simp add: δ)
              next
                fix y
                assume y: y ∈ S norm (y − x) < δ(p + q) and y ≠ x
               have ∗: ⟦norm(b − c) < e − d; norm(y − x − b) ≤ d⟧ ⟹ norm(y
− x − c) < e
                  for b c d e x and y:: real^'n
                  using norm_triangle_ineq2 [of y − x − c y − x − b] by simp
                have norm (f y − f x − B ∗v (y − x)) < e ∗ norm (y − x)
                proof (rule ∗)
                  show norm (f y − f x − A (p + q) ∗v (y − x)) ≤ norm (y − x)
/ (Suc (p + q))
                    using A [OF y] by simp
                 have norm (A (p + q) ∗v (y − x) − B ∗v (y − x)) ≤ onorm(λx.
(A(p + q) − B) ∗v x) ∗ norm(y − x)
                      by (metis linear_linear matrix_vector_mul_linear ma-
trix_vector_mult_diff_rdistrib onorm)
                  also have ... < (e/2) ∗ norm (y − x)
                    using ‹y ≠ x› pqe2 by auto
                  also have ... ≤ (e − 1 / (Suc (p + q))) ∗ norm (y − x)
                  proof (rule mult_right_mono)
                    have 1 / Suc (p + q) ≤ 1 / q
                      using ‹q ≠ 0› by (auto simp: field_split_simps)
                    also have ... < e/2
                      using qe2 by auto
                    finally show e / 2 ≤ e − 1 / real (Suc (p + q))
                      by linarith
                  qed auto
                 finally show norm (A (p + q) ∗v (y − x) − B ∗v (y − x)) < e ∗
norm (y − x) − norm (y − x) / real (Suc (p + q))
                    by (simp add: algebra_simps)
                qed
               then show inverse (norm (y − x)) ∗ norm (f y + B ∗v x − (f x +
B ∗v y)) < e
                  using ‹y ≠ x› by (simp add: field_split_simps algebra_simps)
              qed
            qed
            then show ((λy. (matrix (f' x) − B) ∗v (y − x) /R
                        norm (y − x) − (f x + f' x (y − x) − f y) /R norm (y −
x)) ⟶ 0)
                      (at x within S)
              by (simp add: algebra_simps diff lin_df scalar_mult_eq_scaleR)
          qed
        qed (use x in ‹simp; auto simp: not_less›)
        ultimately have f' x = (∗v) B
          by (force simp: algebra_simps scalar_mult_eq_scaleR)
```

```
      show matrix (f' x) $ m $ n ≤ b
      proof (rule tendsto_upperbound [of λi. (A i $ m $ n) _ sequentially])
        show (λi. A i $ m $ n) ⟶ matrix (f' x) $ m $ n
          by (simp add: B ‹f' x = (∗v) B›)
        show ∀F i in sequentially. A i $ m $ n ≤ b
          by (simp add: Ab less_eq_real_def)
      qed auto
    next
      fix e :: real
      assume x ∈ S and b: matrix (f' x) $ m $ n ≤ b and e > 0
      then obtain d where d>0
        and d: ⋀y. y∈S ⟹ 0 < dist y x ∧ dist y x < d ⟶ norm (f y − f x
− f' x (y − x)) / (norm (y − x))
            < e/2
        using f [OF ‹x ∈ S›]
        by (simp add: Deriv.has_derivative_at_within Lim_within)
          (auto simp add: field_simps dest: spec [of _ e/2])
      let ?A = matrix(f' x) − (χ i j. if i = m ∧ j = n then e / 4 else 0)
      obtain B where BRats: ⋀i j. B$i$j ∈ ℚ and Bo_e6: onorm((∗v) (?A −
B)) < e/6
        using matrix_rational_approximation ‹e > 0›
        by (metis zero_less_divide_iff zero_less_numeral)
      show ∃d>0. ∃A. A $ m $ n < b ∧ (∀i j. A $ i $ j ∈ ℚ) ∧
          (∀y∈S. norm (y − x) < d ⟶ norm (f y − f x − A ∗v (y − x)) ≤ e
∗ norm (y − x))
      proof (intro exI conjI ballI allI impI)
        show d>0
          by (rule ‹d>0›)
        show B $ m $ n < b
        proof −
          have |matrix ((∗v) (?A − B)) $ m $ n| ≤ onorm ((∗v) (?A − B))
            using component_le_onorm [OF matrix_vector_mul_linear, of _ m
n] by metis
          then show ?thesis
            using b Bo_e6 by simp
        qed
        show B $ i $ j ∈ ℚ for i j
          using BRats by auto
        show norm (f y − f x − B ∗v (y − x)) ≤ e ∗ norm (y − x)
          if y ∈ S and y: norm (y − x) < d for y
        proof (cases y = x)
          case True then show ?thesis
            by simp
        next
          case False
            have ∗: norm(d' − d) ≤ e/2 ⟹ norm(y − (x + d')) < e/2 ⟹
norm(y − x − d) ≤ e for d d' e and x y::real^'n
            using norm_triangle_le [of d' − d y − (x + d')] by simp
          show ?thesis
```

```
          proof (rule *)
            have split246: ⟦norm y ≤ e / 6; norm(x − y) ≤ e / 4⟧ ⟹ norm x
≤ e/2 if e > 0 for e and x y :: real^'n
              using norm_triangle_le [of y x−y e/2] ‹e > 0› by simp
            have linear (f' x)
              using True f has_derivative_linear by blast
           then have norm (f' x (y − x) − B *v (y − x)) = norm ((matrix (f'
x) − B) *v (y − x))
              by (simp add: matrix_vector_mult_diff_rdistrib)
            also have ... ≤ (e * norm (y − x)) / 2
            proof (rule split246)
              have norm ((?A − B) *v (y − x)) / norm (y − x) ≤ onorm(λx.
(?A − B) *v x)
                by (rule le_onorm) auto
              also have  ... < e/6
                by (rule Bo_e6)
              finally have norm ((?A − B) *v (y − x)) / norm (y − x) < e / 6 .
            then show norm ((?A − B) *v (y − x)) ≤ e * norm (y − x) / 6
              by (simp add: field_split_simps False)
            have norm ((matrix (f' x) − B) *v (y − x) − ((?A − B) *v (y −
x))) = norm ((χ i j. if i = m ∧ j = n then e / 4 else 0) *v (y − x))
              by (simp add: algebra_simps)
            also have ... = norm((e/4) *R (y − x)$n *R axis m (1::real))
            proof −
              have (∑j∈UNIV. (if i = m ∧ j = n then e / 4 else 0) * (y $ j
− x $ j)) * 4 = e * (y $ n − x $ n) * axis m 1 $ i for i
              proof (cases i=m)
                case True then show ?thesis
                  by (auto simp: if_distrib [of λz. z * _] cong: if_cong)
              next
                case False then show ?thesis
                  by (simp add: axis_def)
              qed
              then have (χ i j. if i = m ∧ j = n then e / 4 else 0) *v (y − x)
= (e/4) *R (y − x)$n *R axis m (1::real)
                by (auto simp: vec_eq_iff matrix_vector_mult_def)
              then show ?thesis
                by metis
            qed
            also have ... ≤ e * norm (y − x) / 4
            proof −
              have |y $ n − x $ n| ≤ norm (y − x)
                by (metis component_le_norm_cart vector_minus_component)
              with ‹e > 0› show ?thesis
                by (simp add: norm_mult abs_mult)
            qed
            finally show norm ((matrix (f' x) − B) *v (y − x) − ((?A − B)
*v (y − x))) ≤ e * norm (y − x) / 4 .
            show 0 < e * norm (y − x)
```

```
            by (simp add: False ‹e > 0›)
          qed
          finally show norm (f' x (y - x) - B *v (y - x)) ≤ (e * norm (y -
x)) / 2 .
          show norm (f y - (f x + f' x (y - x))) < (e * norm (y - x)) / 2
           using False d [OF ‹y ∈ S›] y by (simp add: dist_norm field_simps)
         qed
        qed
       qed
      qed
     qed auto
   qed
   show negligible ?M
    using negligible_subset [OF nN MN] .
  qed
  then show ?thesis
   by (simp add: borel_measurable_vimage_halfspace_component_le sets_restrict_space_iff
assms)
qed


theorem borel_measurable_det_Jacobian:
 fixes f :: real^'n::{finite,wellorder} ⇒ real^'n::_
  assumes S: S ∈ sets lebesgue and f: ⋀x. x ∈ S ⟹ (f has_derivative f' x) (at
x within S)
  shows (λx. det(matrix(f' x))) ∈ borel_measurable (lebesgue_on S)
  unfolding det_def
  by (intro measurable) (auto intro: f borel_measurable_partial_derivatives [OF
S])
```

The localisation wrt S uses the same argument for many similar results.

```
theorem borel_measurable_lebesgue_on_preimage_borel:
  fixes f :: 'a::euclidean_space ⇒ 'b::euclidean_space
  assumes S ∈ sets lebesgue
  shows f ∈ borel_measurable (lebesgue_on S) ⟷
        (∀ T. T ∈ sets borel ⟶ {x ∈ S. f x ∈ T} ∈ sets lebesgue)
proof −
  have {x. (if x ∈ S then f x else 0) ∈ T} ∈ sets lebesgue ⟷ {x ∈ S. f x ∈ T}
∈ sets lebesgue
        if T ∈ sets borel for T
    proof (cases 0 ∈ T)
      case True
      then have {x ∈ S. f x ∈ T} = {x. (if x ∈ S then f x else 0) ∈ T} ∩ S
                {x. (if x ∈ S then f x else 0) ∈ T} = {x ∈ S. f x ∈ T} ∪ −S
        by auto
      then show ?thesis
      by (metis (no_types, lifting) Compl_in_sets_lebesgue assms sets.Int sets.Un)
    next
      case False
```

then have $\{x.\ (\mathit{if}\ x \in S\ \mathit{then}\ f\ x\ \mathit{else}\ 0) \in T\} = \{x \in S.\ f\ x \in T\}$
by *auto*
then show *?thesis*
by *auto*
qed
then show *?thesis*
unfolding *borel_measurable_lebesgue_preimage_borel borel_measurable_if* [*OF assms*, *symmetric*]
by *blast*
qed

lemma *sets_lebesgue_almost_borel*:
assumes $S \in \mathit{sets\ lebesgue}$
obtains $B\ N$ **where** $B \in \mathit{sets\ borel}\ \mathit{negligible}\ N\ B \cup N = S$
by (*metis assms negligible_iff_null_sets negligible_subset null_sets_completionI sets_completionE sets_lborel*)

lemma *double_lebesgue_sets*:
assumes *S*: $S \in \mathit{sets\ lebesgue}$ **and** *T*: $T \in \mathit{sets\ lebesgue}$ **and** *fim*: $f\ `\ S \subseteq T$
shows $(\forall U.\ U \in \mathit{sets\ lebesgue} \wedge U \subseteq T \longrightarrow \{x \in S.\ f\ x \in U\} \in \mathit{sets\ lebesgue}) \longleftrightarrow$
$f \in \mathit{borel\_measurable}\ (\mathit{lebesgue\_on}\ S) \wedge$
$(\forall U.\ \mathit{negligible}\ U \wedge U \subseteq T \longrightarrow \{x \in S.\ f\ x \in U\} \in \mathit{sets\ lebesgue})$
(**is** *?lhs* $\longleftrightarrow$ _ $\wedge$ *?rhs*)
unfolding *borel_measurable_lebesgue_on_preimage_borel* [*OF S*]
proof (*intro iffI allI conjI impI*, *safe*)
fix $V :: 'b\ \mathit{set}$
assume $*$: $\forall U.\ U \in \mathit{sets\ lebesgue} \wedge U \subseteq T \longrightarrow \{x \in S.\ f\ x \in U\} \in \mathit{sets\ lebesgue}$
and $V \in \mathit{sets\ borel}$
then have *V*: $V \in \mathit{sets\ lebesgue}$
by *simp*
have $\{x \in S.\ f\ x \in V\} = \{x \in S.\ f\ x \in T \cap V\}$
using *fim* **by** *blast*
also have $\{x \in S.\ f\ x \in T \cap V\} \in \mathit{sets\ lebesgue}$
using $T\ V * \mathit{le\_inf\_iff}$ **by** *blast*
finally show $\{x \in S.\ f\ x \in V\} \in \mathit{sets\ lebesgue}$ **.**
next
fix $U :: 'b\ \mathit{set}$
assume $\forall U.\ U \in \mathit{sets\ lebesgue} \wedge U \subseteq T \longrightarrow \{x \in S.\ f\ x \in U\} \in \mathit{sets\ lebesgue}$
$\mathit{negligible}\ U\ U \subseteq T$
then show $\{x \in S.\ f\ x \in U\} \in \mathit{sets\ lebesgue}$
using *negligible_imp_sets* **by** *blast*
next
fix $U :: 'b\ \mathit{set}$
assume *1* [*rule_format*]: $(\forall T.\ T \in \mathit{sets\ borel} \longrightarrow \{x \in S.\ f\ x \in T\} \in \mathit{sets\ lebesgue})$
and *2* [*rule_format*]: $\forall U.\ \mathit{negligible}\ U \wedge U \subseteq T \longrightarrow \{x \in S.\ f\ x \in U\} \in \mathit{sets\ lebesgue}$
and $U \in \mathit{sets\ lebesgue}\ U \subseteq T$

```
  then obtain C N where C: C ∈ sets borel ∧ negligible N ∧ C ∪ N = U
    using sets_lebesgue_almost_borel by metis
  then have {x ∈ S. f x ∈ C} ∈ sets lebesgue
    by (blast intro: 1)
  moreover have {x ∈ S. f x ∈ N} ∈ sets lebesgue
    using C ‹U ⊆ T› by (blast intro: 2)
  moreover have {x ∈ S. f x ∈ C ∪ N} = {x ∈ S. f x ∈ C} ∪ {x ∈ S. f x ∈ N}
    by auto
  ultimately show {x ∈ S. f x ∈ U} ∈ sets lebesgue
    using C by auto
qed
```

6.41.3 Simplest case of Sard's theorem (we don't need continuity of derivative)

```
lemma Sard_lemma00:
  fixes P :: 'b::euclidean_space set
  assumes a ≥ 0 and a: a *R i ≠ 0 and i: i ∈ Basis
    and P: P ⊆ {x. a *R i • x = 0}
    and 0 ≤ m 0 ≤ e
 obtains S where S ∈ lmeasurable
          and {z. norm z ≤ m ∧ (∃t ∈ P. norm(z − t) ≤ e)} ⊆ S
          and measure lebesgue S ≤ (2 * e) * (2 * m) ^(DIM('b) − 1)
proof −
  have a > 0
    using assms by simp
  let ?v = (∑j∈Basis. (if j = i then e else m) *R j)
  show thesis
  proof
    have − e ≤ x • i x • i ≤ e
      if t ∈ P norm (x − t) ≤ e for x t
      using ‹a > 0› that Basis_le_norm [of i x−t] P i
      by (auto simp: inner_commute algebra_simps)
    moreover have − m ≤ x • j x • j ≤ m
      if norm x ≤ m t ∈ P norm (x − t) ≤ e j ∈ Basis and j ≠ i
      for x t j
      using that Basis_le_norm [of j x] by auto
    ultimately
    show {z. norm z ≤ m ∧ (∃t∈P. norm (z − t) ≤ e)} ⊆ cbox (−?v) ?v
      by (auto simp: mem_box)
    have *: ∀k∈Basis. − ?v • k ≤ ?v • k
      using ‹0 ≤ m› ‹0 ≤ e› by (auto simp: inner_Basis)
    have 2: 2 ^ DIM('b) = 2 * 2 ^(DIM('b) − Suc 0)
      by (metis DIM_positive Suc_pred power_Suc)
    show measure lebesgue (cbox (−?v) ?v) ≤ 2 * e * (2 * m) ^(DIM('b) − 1)
      using ‹i ∈ Basis›
     by (simp add: content_cbox [OF *] prod.distrib prod.If_cases Diff_eq [symmetric]
2)
  qed blast
```

qed

As above, but reorienting the vector (HOL Light's @textGEOM_BASIS_MULTIPLE_TAC)

lemma *Sard_lemma0*:
fixes $P :: (real\hat{}'n::\{finite,wellorder\})\ set$
assumes $a \neq 0$
and *P*: $P \subseteq \{x.\ a \cdot x = 0\}$ **and** $0 \leq m\ 0 \leq e$
obtains *S* **where** $S \in lmeasurable$
and $\{z.\ norm\ z \leq m \wedge (\exists t \in P.\ norm(z - t) \leq e)\} \subseteq S$
and *measure lebesgue* $S \leq (2 * e) * (2 * m)\ \hat{}\ (CARD('n) - 1)$
proof −
obtain *T* **and** *k*::*'n* **where** *T*: *orthogonal_transformation T* **and** *a*: $a = T\ (norm\ a *_R\ axis\ k\ (1::real))$
using *rotation_rightward_line* **by** *metis*
have *Tinv* [*simp*]: $T\ (inv\ T\ x) = x$ **for** *x*
by (*simp add*: *T orthogonal_transformation_surj surj_f_inv_f*)
obtain *S* **where** *S*: $S \in lmeasurable$
and *subS*: $\{z.\ norm\ z \leq m \wedge (\exists t \in T - `P.\ norm(z - t) \leq e)\} \subseteq S$
and *mS*: *measure lebesgue* $S \leq (2 * e) * (2 * m)\ \hat{}\ (CARD('n) - 1)$
proof (*rule Sard_lemma00* [*of norm a axis k* (*1*::*real*) *T*−*`P m e*])
have $norm\ a *_R\ axis\ k\ 1 \cdot x = 0$ **if** $T\ x \in P$ **for** *x*
by (*smt* (*verit*, *del_insts*) *P T a mem_Collect_eq orthogonal_transformation_def subset_eq that*)
then show $T - `P \subseteq \{x.\ norm\ a *_R\ axis\ k\ 1 \cdot x = 0\}$
by *auto*
qed (*use assms T* **in** *auto*)
show *thesis*
proof
show $T\ `\ S \in lmeasurable$
using *S measurable_orthogonal_image T* **by** *blast*
have $\{z.\ norm\ z \leq m \wedge (\exists t \in P.\ norm\ (z - t) \leq e)\} \subseteq T\ `\ \{z.\ norm\ z \leq m \wedge (\exists t \in T - `P.\ norm\ (z - t) \leq e)\}$
proof *clarsimp*
fix *x t*
assume §: $norm\ x \leq m\ t \in P\ norm\ (x - t) \leq e$
then have $norm\ (inv\ T\ x) \leq m$
using *orthogonal_transformation_inv* [*OF T*] **by** (*simp add*: *orthogonal_transformation_norm*)
moreover have $\exists t \in T - `P.\ norm\ (inv\ T\ x - t) \leq e$
by (*smt* (*verit*, *del_insts*) *T Tinv* § *linear_diff orthogonal_transformation_def orthogonal_transformation_norm vimage_eq*)
ultimately show $x \in T\ `\ \{z.\ norm\ z \leq m \wedge (\exists t \in T - `P.\ norm\ (z - t) \leq e)\}$
by *force*
qed
then show $\{z.\ norm\ z \leq m \wedge (\exists t \in P.\ norm\ (z - t) \leq e)\} \subseteq T\ `\ S$
using *image_mono* [*OF subS*] **by** (*rule order_trans*)
show *measure lebesgue* $(T\ `\ S) \leq 2 * e * (2 * m)\ \hat{}\ (CARD('n) - 1)$

```
    using mS T by (simp add: S measure_orthogonal_image)
  qed
qed
```

As above, but translating the sets (HOL Light's @textGEN_GEOM_ORIGIN_TAC)

```
lemma Sard_lemma1:
  fixes P :: (real^'n::{finite,wellorder}) set
   assumes P: dim P < CARD('n) and 0 ≤ m 0 ≤ e
 obtains S where S ∈ lmeasurable
          and {z. norm(z − w) ≤ m ∧ (∃t ∈ P. norm(z − w − t) ≤ e)} ⊆ S
          and measure lebesgue S ≤ (2 ∗ e) ∗ (2 ∗ m) ^ (CARD('n) − 1)
proof −
  obtain a where a ≠ 0 P ⊆ {x. a · x = 0}
    using lowdim_subset_hyperplane [of P] P span_base by auto
  then obtain S where S: S ∈ lmeasurable
    and subS: {z. norm z ≤ m ∧ (∃t ∈ P. norm(z − t) ≤ e)} ⊆ S
    and mS: measure lebesgue S ≤ (2 ∗ e) ∗ (2 ∗ m) ^ (CARD('n) − 1)
    by (rule Sard_lemma0 [OF _ _ ‹0 ≤ m› ‹0 ≤ e›])
  show thesis
  proof
    show (+)w ‘ S ∈ lmeasurable
      by (metis measurable_translation S)
    show {z. norm (z − w) ≤ m ∧ (∃t∈P. norm (z − w − t) ≤ e)} ⊆ (+)w ‘ S
      using subS by force
    show measure lebesgue ((+)w ‘ S) ≤ 2 ∗ e ∗ (2 ∗ m) ^ (CARD('n) − 1)
      by (metis measure_translation mS)
  qed
qed

lemma Sard_lemma2:
  fixes f :: real^'m::{finite,wellorder} ⇒ real^'n::{finite,wellorder}
  assumes mlen: CARD('m) ≤ CARD('n) (is ?m ≤ ?n)
    and B > 0 bounded S
    and derS: ⋀x. x ∈ S ⟹ (f has_derivative f' x) (at x within S)
    and rank: ⋀x. x ∈ S ⟹ rank(matrix(f' x)) < CARD('n)
    and B: ⋀x. x ∈ S ⟹ onorm(f' x) ≤ B
  shows negligible(f ‘ S)
proof −
  have lin_f': ⋀x. x ∈ S ⟹ linear(f' x)
    using derS has_derivative_linear by blast
  show ?thesis
  proof (clarsimp simp add: negligible_outer_le)
    fix e :: real
    assume e > 0
    obtain c where csub: S ⊆ cbox (− (vec c)) (vec c) and c > 0
    proof −
      obtain b where b: ⋀x. x ∈ S ⟹ norm x ≤ b
        using ‹bounded S› by (auto simp: bounded_iff)
```

```
    show thesis
    proof
      have - |b| - 1 ≤ x $ i ∧ x $ i ≤ |b| + 1 if x ∈ S for x i
        using component_le_norm_cart [of x i] b [OF that] by auto
      then show S ⊆ cbox (- vec (|b| + 1)) (vec (|b| + 1))
        by (auto simp: mem_box_cart)
    qed auto
  qed
  then have box_cc: box (- (vec c)) (vec c) ≠ {} and cbox_cc: cbox (- (vec
c)) (vec c) ≠ {}
    by (auto simp: interval_eq_empty_cart)
  obtain d where d > 0 d ≤ B
        and d: (d * 2) * (4 * B) ^ (?n - 1) ≤ e / (2*c) ^ ?m / ?m ^ ?m
    apply (rule that [of min B (e / (2*c) ^ ?m / ?m ^ ?m / (4 * B) ^ (?n -
1) / 2)])
    using ‹B > 0› ‹c > 0› ‹e > 0›
    by (simp_all add: divide_simps min_mult_distrib_right)
  have ∃ r. 0 < r ∧ r ≤ 1/2 ∧
          (x ∈ S
           ⟶ (∀ y. y ∈ S ∧ norm(y - x) < r
                 ⟶ norm(f y - f x - f' x (y - x)) ≤ d * norm(y - x))) for x
  proof (cases x ∈ S)
    case True
    then obtain r where r > 0
          and ⋀y. ⟦y ∈ S; norm (y - x) < r⟧
                ⟹ norm (f y - f x - f' x (y - x)) ≤ d * norm (y - x)
      using derS ‹d > 0› by (force simp: has_derivative_within_alt)
    then show ?thesis
      by (rule_tac x=min r (1/2) in exI) simp
  next
    case False
    then show ?thesis
      by (rule_tac x=1/2 in exI) simp
  qed
  then obtain r where r12: ⋀x. 0 < r x ∧ r x ≤ 1/2
        and r: ⋀x y. ⟦x ∈ S; y ∈ S; norm(y - x) < r x⟧
                   ⟹ norm(f y - f x - f' x (y - x)) ≤ d * norm(y - x)
    by metis
  then have ga: gauge (λx. ball x (r x))
    by (auto simp: gauge_def)
  obtain 𝒟 where 𝒟: countable 𝒟 and sub_cc: ⋃𝒟 ⊆ cbox (- vec c) (vec c)
    and cbox: ⋀K. K ∈ 𝒟 ⟹ interior K ≠ {} ∧ (∃ u v. K = cbox u v)
    and djointish: pairwise (λA B. interior A ∩ interior B = {}) 𝒟
    and covered: ⋀K. K ∈ 𝒟 ⟹ ∃ x ∈ S ∩ K. K ⊆ ball x (r x)
    and close: ⋀u v. cbox u v ∈ 𝒟 ⟹ ∃ n. ∀ i::'m. v $ i - u $ i = 2*c / 2^n
    and covers: S ⊆ ⋃𝒟
    apply (rule covering_lemma [OF csub box_cc ga])
    apply (auto simp: Basis_vec_def cart_eq_inner_axis [symmetric])
    done
```

```
    let ?μ = measure lebesgue
    have ∃ T. T ∈ lmeasurable ∧ f ‘ (K ∩ S) ⊆ T ∧ ?μ T ≤ e / (2*c) ^ ?m * ?μ
K
      if K ∈ 𝒟 for K
    proof −
      obtain u v where uv: K = cbox u v
        using cbox ‹K ∈ 𝒟› by blast
      then have uv_ne: cbox u v ≠ {}
        using cbox that by fastforce
      obtain x where x: x ∈ S ∩ cbox u v cbox u v ⊆ ball x (r x)
        using ‹K ∈ 𝒟› covered uv by blast
      then have dim (range (f′ x)) < ?n
        using rank_dim_range [of matrix (f′ x)] x rank[of x]
        by (auto simp: matrix_works scalar_mult_eq_scaleR lin_f′)
      then obtain T where T: T ∈ lmeasurable
            and subT: {z. norm(z − f x) ≤ (2 * B) * norm(v − u) ∧ (∃ t ∈ range
(f′ x). norm(z − f x − t) ≤ d * norm(v − u))} ⊆ T
            and measT: ?μ T ≤ (2 * (d * norm(v − u))) * (2 * ((2 * B) * norm(v
− u))) ^ (?n − 1)
                       (is _ ≤ ?DVU)
        using Sard_lemma1 [of range (f′ x) (2 * B) * norm(v − u) d * norm(v −
u)]
        using ‹B > 0› ‹d > 0› by auto
      show ?thesis
      proof (intro exI conjI)
        have f ‘ (K ∩ S) ⊆ {z. norm(z − f x) ≤ (2 * B) * norm(v − u) ∧ (∃ t ∈
range (f′ x). norm(z − f x − t) ≤ d * norm(v − u))}
          unfolding uv
        proof (clarsimp simp: mult.assoc, intro conjI)
          fix y
          assume y: y ∈ cbox u v and y ∈ S
          then have norm (y − x) < r x
            by (metis dist_norm mem_ball norm_minus_commute subsetCE x(2))
          then have le_dyx: norm (f y − f x − f′ x (y − x)) ≤ d * norm (y − x)
            using r [of x y] x ‹y ∈ S› by blast
          have yx_le: norm (y − x) ≤ norm (v − u)
          proof (rule norm_le_componentwise_cart)
            show norm ((y − x) $ i) ≤ norm ((v − u) $ i) for i
              using x y by (force simp: mem_box_cart dest!: spec [where x=i])
          qed
          have ∗: ⟦norm(y − x − z) ≤ d; norm z ≤ B; d ≤ B⟧ ⟹ norm(y − x)
≤ 2 * B
            for x y z :: real^′n::_ and d B
            using norm_triangle_ineq2 [of y − x z] by auto
          show norm (f y − f x) ≤ 2 * (B * norm (v − u))
          proof (rule ∗ [OF le_dyx])
            have norm (f′ x (y − x)) ≤ onorm (f′ x) * norm (y − x)
              using onorm [of f′ x y−x] by (meson IntE lin_f′ linear_linear x(1))
            also have ... ≤ B * norm (v − u)
```

```
            by (meson B IntE lin_f' linear_linear mult_mono' norm_ge_zero
onorm_pos_le x(1) yx_le)
          finally show norm (f' x (y − x)) ≤ B * norm (v − u) .
          show d * norm (y − x) ≤ B * norm (v − u)
            using ‹B > 0› by (auto intro: mult_mono [OF ‹d ≤ B› yx_le])
        qed
        show ∃ t. norm (f y − f x − f' x t) ≤ d * norm (v − u)
          by (smt (verit, best) ‹0 < d› le_dyx mult_le_cancel_left_pos yx_le)
      qed
      with subT show f ‘ (K ∩ S) ⊆ T by blast
      show ?μ T ≤ e / (2*c) ^ ?m * ?μ K
      proof (rule order_trans [OF measT])
        have ?DVU = (d * 2 * (4 * B) ^ (?n − 1)) * norm (v − u)^?n
          using ‹c > 0›
          apply (simp add: algebra_simps)
          by (metis Suc_pred power_Suc zero_less_card_finite)
        also have ... ≤ (e / (2*c) ^ ?m / (?m ^ ?m)) * norm(v − u) ^ ?n
          by (rule mult_right_mono [OF d]) auto
        also have ... ≤ e / (2*c) ^ ?m * ?μ K
        proof −
          have u ∈ ball (x) (r x) v ∈ ball x (r x)
          using box_ne_empty(1) contra_subsetD [OF x(2)] mem_box(2) uv_ne
by fastforce+
          moreover have r x ≤ 1/2
            using r12 by auto
          ultimately have norm (v − u) ≤ 1
            using norm_triangle_half_r [of x u 1 v]
                by (metis (no_types, opaque_lifting) dist_commute dist_norm
less_eq_real_def less_le_trans mem_ball)
          then have norm (v − u) ^ ?n ≤ norm (v − u) ^ ?m
            by (simp add: power_decreasing [OF mlen])
          also have ... ≤ ?μ K * real (?m ^ ?m)
          proof −
            obtain n where n: ⋀i. v$i − u$i = 2 * c / 2^n
              using close [of u v] ‹K ∈ 𝒟› uv by blast
            have norm (v − u) ^ ?m ≤ (∑ i∈UNIV. |(v − u) $ i|) ^ ?m
              by (intro norm_le_l1_cart power_mono) auto
             also have ... ≤ (∏ i∈UNIV. v $ i − u $ i) * real CARD('m) ^
CARD('m)
              by (simp add: n field_simps ‹c > 0› less_eq_real_def)
            also have ... = ?μ K * real (?m ^ ?m)
              by (simp add: uv uv_ne content_cbox_cart)
            finally show ?thesis .
          qed
          finally have *: 1 / real (?m ^ ?m) * norm (v − u) ^ ?n ≤ ?μ K
            by (simp add: field_split_simps)
          show ?thesis
            using mult_left_mono [OF *, of e / (2*c) ^ ?m] ‹c > 0› ‹e > 0› by
auto
```

```
        qed
        finally show ?DVU ≤ e / (2*c) ^ ?m * ?μ K .
      qed
    qed (use T in auto)
  qed
  then obtain g where meas_g: ⋀K. K ∈ 𝒟 ⟹ g K ∈ lmeasurable
                and sub_g: ⋀K. K ∈ 𝒟 ⟹ f ' (K ∩ S) ⊆ g K
                and le_g: ⋀K. K ∈ 𝒟 ⟹ ?μ (g K) ≤ e / (2*c)^?m * ?μ K
    by metis
  have le_e: ?μ (⋃i∈ℱ. g i) ≤ e
    if ℱ ⊆ 𝒟 finite ℱ for ℱ
  proof −
    have ?μ (⋃i∈ℱ. g i) ≤ (∑i∈ℱ. ?μ (g i))
      using meas_g ‹ℱ ⊆ 𝒟› by (auto intro: measure_UNION_le [OF ‹finite
ℱ›])
    also have ... ≤ (∑K∈ℱ. e / (2*c) ^ ?m * ?μ K)
     using ‹ℱ ⊆ 𝒟› sum_mono [OF le_g] by (meson le_g subsetCE sum_mono)
    also have ... = e / (2*c) ^ ?m * (∑K∈ℱ. ?μ K)
      by (simp add: sum_distrib_left)
    also have ... ≤ e
    proof −
      have ℱ division_of ⋃ℱ
      proof (rule division_ofI)
        show K ⊆ ⋃ℱ  K ≠ {} ∃a b. K = cbox a b if K ∈ ℱ for K
          using ‹K ∈ ℱ› covered cbox ‹ℱ ⊆ 𝒟› by (auto simp: Union_upper)
        show interior K ∩ interior L = {} if K ∈ ℱ and L ∈ ℱ and K ≠ L for
K L
       by (metis (mono_tags, lifting) ‹ℱ ⊆ 𝒟› pairwiseD djointish pairwise_subset
that)
      qed (use that in auto)
      then have sum ?μ ℱ ≤ ?μ (⋃ℱ)
        by (simp add: content_division)
      also have ... ≤ ?μ (cbox (− vec c) (vec c) :: (real, 'm) vec set)
      proof (rule measure_mono_fmeasurable)
        show ⋃ℱ ⊆ cbox (− vec c) (vec c)
          by (meson Sup_subset_mono sub_cc order_trans ‹ℱ ⊆ 𝒟›)
      qed (use ‹ℱ division_of ⋃ℱ› lmeasurable_division in auto)
      also have ... = content (cbox (− vec c) (vec c) :: (real, 'm) vec set)
        by simp
      also have ... ≤ (2 ^ ?m * c ^ ?m)
        using ‹c > 0› by (simp add: content_cbox_if_cart)
      finally have sum ?μ ℱ ≤ (2 ^ ?m * c ^ ?m) .
      then show ?thesis
        using ‹e > 0› ‹c > 0› by (auto simp: field_split_simps)
    qed
    finally show ?thesis .
  qed
  show ∃T. f ' S ⊆ T ∧ T ∈ lmeasurable ∧ ?μ T ≤ e
  proof (intro exI conjI)
```

```
    show f ‘ S ⊆ ⋃ (g ‘ 𝒟)
      using covers sub_g by force
    show ⋃ (g ‘ 𝒟) ∈ lmeasurable
      by (rule fmeasurable_UN_bound [OF ‹countable 𝒟› meas_g le_e])
    show ?μ (⋃ (g ‘ 𝒟)) ≤ e
      by (rule measure_UN_bound [OF ‹countable 𝒟› meas_g le_e])
  qed
 qed
qed


theorem baby_Sard:
 fixes f :: real^'m::{finite,wellorder} ⇒ real^'n::{finite,wellorder}
 assumes mlen: CARD('m) ≤ CARD('n)
   and der: ⋀x. x ∈ S ⟹ (f has_derivative f' x) (at x within S)
   and rank: ⋀x. x ∈ S ⟹ rank(matrix(f' x)) < CARD('n)
 shows negligible(f ‘ S)
proof −
 let ?U = λn. {x ∈ S. norm(x) ≤ n ∧ onorm(f' x) ≤ real n}
 have ⋀x. x ∈ S ⟹ ∃n. norm x ≤ real n ∧ onorm (f' x) ≤ real n
   by (meson linear order_trans real_arch_simple)
 then have eq: S = (⋃n. ?U n)
   by auto
 have negligible (f ‘ ?U n) for n
 proof (rule Sard_lemma2 [OF mlen])
   show 0 < real n + 1
     by auto
   show bounded (?U n)
     using bounded_iff by blast
   show (f has_derivative f' x) (at x within ?U n) if x ∈ ?U n for x
     using der that by (force intro: has_derivative_subset)
 qed (use rank in auto)
 then show ?thesis
   by (subst eq) (simp add: image_Union negligible_Union_nat)
qed
```

6.41.4 A one-way version of change-of-variables not assuming injectivity.

```
lemma integral_on_image_ubound_weak:
 fixes f :: real^'n::{finite,wellorder} ⇒ real
 assumes S: S ∈ sets lebesgue
     and f: f ∈ borel_measurable (lebesgue_on (g ‘ S))
     and nonneg_fg: ⋀x. x ∈ S ⟹ 0 ≤ f(g x)
     and der_g: ⋀x. x ∈ S ⟹ (g has_derivative g' x) (at x within S)
     and det_int_fg: (λx. |det (matrix (g' x))| * f(g x)) integrable_on S
     and meas_gim: ⋀T. ⟦T ⊆ g ‘ S; T ∈ sets lebesgue⟧ ⟹ {x ∈ S. g x ∈ T}
∈ sets lebesgue
   shows f integrable_on (g ‘ S) ∧
```

```
        integral (g ‘ S) f ≤ integral S (λx. |det (matrix (g′ x))| ∗ f(g x))
      (is _ ∧ _ ≤ ?b)
proof −
  let ?D = λx. |det (matrix (g′ x))|
  have cont_g: continuous_on S g
    using der_g has_derivative_continuous_on by blast
  have [simp]: space (lebesgue_on S) = S
    by (simp add: S)
  have gS_in_sets_leb: g ‘ S ∈ sets lebesgue
    apply (rule differentiable_image_in_sets_lebesgue)
    using der_g by (auto simp: S differentiable_def differentiable_on_def)
  obtain h where nonneg_h: ⋀n x. 0 ≤ h n x
    and h_le_f: ⋀n x. x ∈ S ⟹ h n (g x) ≤ f (g x)
    and h_inc: ⋀n x. h n x ≤ h (Suc n) x
    and h_meas: ⋀n. h n ∈ borel_measurable lebesgue
    and fin_R: ⋀n. finite(range (h n))
    and lim: ⋀x. x ∈ g ‘ S ⟹ (λn. h n x) ⟶ f x
  proof −
    let ?f = λx. if x ∈ g ‘ S then f x else 0
    have ?f ∈ borel_measurable lebesgue ∧ (∀ x. 0 ≤ ?f x)
      by (auto simp: gS_in_sets_leb f nonneg_fg measurable_restrict_space_iff
[symmetric])
    then show ?thesis
    apply (clarsimp simp add: borel_measurable_simple_function_limit_increasing)
      apply (rename_tac h)
      by (rule_tac h=h in that) (auto split: if_split_asm)
  qed
  have h_lmeas: {t. h n (g t) = y} ∩ S ∈ sets lebesgue for y n
  proof −
    have space (lebesgue_on (UNIV::(real,'n) vec set)) = UNIV
      by simp
    then have ((h n) −‘{y} ∩ g ‘ S) ∈ sets (lebesgue_on (g ‘ S))
    by (metis Int_commute borel_measurable_vimage h_meas image_eqI inf_top.right_neutral
sets_restrict_space space_borel space_completion space_lborel)
    then have ({u. h n u = y} ∩ g ‘ S) ∈ sets lebesgue
      using gS_in_sets_leb
      by (simp add: integral_indicator fmeasurableI2 sets_restrict_space_iff vim-
age_def)
    then have {x ∈ S. g x ∈ ({u. h n u = y} ∩ g ‘ S)} ∈ sets lebesgue
      using meas_gim[of ({u. h n u = y} ∩ g ‘ S)] by force
    moreover have {t. h n (g t) = y} ∩ S = {x ∈ S. g x ∈ ({u. h n u = y} ∩ g ‘
S)}
      by blast
    ultimately show ?thesis
      by auto
  qed
  have hint: h n integrable_on g ‘ S ∧ integral (g ‘ S) (h n) ≤ integral S (λx. ?D
x ∗ h n (g x))
        (is ?INT ∧ ?lhs ≤ ?rhs) for n
```

```
  proof −
    let ?R = range (h n)
    have hn_eq: h n = (λx. ∑ y∈?R. y * indicat_real {x. h n x = y} x)
      by (simp add: indicator_def if_distrib fin_R cong: if_cong)
    have yind: (λt. y * indicator{x. h n x = y} t) integrable_on (g ‘ S) ∧
              (integral (g ‘ S) (λt. y * indicator {x. h n x = y} t))
              ≤ integral S (λt. |det (matrix (g′ t))| * y * indicator {x. h n x = y}
(g t))
       if y: y ∈ ?R for y::real
    proof (cases y=0)
      case True
      then show ?thesis using gS_in_sets_leb integrable_0 by force
    next
      case False
      with that have y > 0
        using less_eq_real_def nonneg_h by fastforce
      have (λx. if x ∈ {t. h n (g t) = y} then ?D x else 0) integrable_on S
      proof (rule measurable_bounded_by_integrable_imp_integrable)
        have (λx. ?D x) ∈ borel_measurable (lebesgue_on ({t. h n (g t) = y} ∩ S))
        proof −
          have (λv. det (matrix (g′ v))) ∈ borel_measurable (lebesgue_on (S ∩ {v.
h n (g v) = y}))
            by (metis Int_lower1 S assms(4) borel_measurable_det_Jacobian
measurable_restrict_mono)
          then show ?thesis
            by (simp add: Int_commute)
        qed
          then have (λx. if x ∈ {t. h n (g t) = y} ∩ S then ?D x else 0) ∈
borel_measurable lebesgue
          by (rule borel_measurable_if_I [OF _ h_lmeas])
      then show (λx. if x ∈ {t. h n (g t) = y} then ?D x else 0) ∈ borel_measurable
(lebesgue_on S)
        by (simp add: if_if_eq_conj Int_commute borel_measurable_if [OF S,
symmetric])
      show (λx. ?D x *R f (g x) /R y) integrable_on S
        by (rule integrable_cmul) (use det_int_fg in auto)
      show norm (if x ∈ {t. h n (g t) = y} then ?D x else 0) ≤ ?D x *R f (g x)
/R y
        if x ∈ S for x
        using nonneg_h [of n x] ‹y > 0› nonneg_fg [of x] h_le_f [of x n] that
        by (auto simp: divide_simps mult_left_mono)
    qed (use S in auto)
    then have int_det: (λt. |det (matrix (g′ t))|) integrable_on ({t. h n (g t) =
y} ∩ S)
      using integrable_restrict_Int by force
    have (g ‘ ({t. h n (g t) = y} ∩ S)) ∈ lmeasurable
    by (blast intro: has_derivative_subset [OF der_g] measurable_differentiable_image
[OF h_lmeas] int_det)
    moreover have g ‘ ({t. h n (g t) = y} ∩ S) = {x. h n x = y} ∩ g ‘ S
```

```
      by blast
    moreover have measure lebesgue (g ‘ ({t. h n (g t) = y} ∩ S))
                  ≤ integral ({t. h n (g t) = y} ∩ S) (λt. |det (matrix (g′ t))|)
    by (blast intro: has_derivative_subset [OF der_g] measure_differentiable_image
[OF h_lmeas _ int_det])
    ultimately show ?thesis
     using ‹y > 0› integral_restrict_Int [of S {t. h n (g t) = y} λt. |det (matrix
(g′ t))| * y]
      apply (simp add: integrable_on_indicator integral_indicator)
      apply (simp add: indicator_def of_bool_def if_distrib cong: if_cong)
      done
  qed
  show ?thesis
  proof
    show h n integrable_on g ‘ S
      apply (subst hn_eq)
      using yind by (force intro: integrable_sum [OF fin_R])
    have ?lhs = integral (g ‘ S) (λx. ∑ y∈range (h n). y * indicat_real {x. h n
x = y} x)
      by (metis hn_eq)
    also have ... = (∑ y∈range (h n). integral (g ‘ S) (λx. y * indicat_real {x.
h n x = y} x))
      by (rule integral_sum [OF fin_R]) (use yind in blast)
    also have ... ≤ (∑ y∈range (h n). integral S (λu. |det (matrix (g′ u))| * y
* indicat_real {x. h n x = y} (g u)))
      using yind by (force intro: sum_mono)
    also have ... = integral S (λu. ∑ y∈range (h n). |det (matrix (g′ u))| * y *
indicat_real {x. h n x = y} (g u))
    proof (rule integral_sum [OF fin_R, symmetric])
      fix y assume y: y ∈ ?R
      with nonneg_h have y ≥ 0
        by auto
       show (λu. |det (matrix (g′ u))| * y * indicat_real {x. h n x = y} (g u))
integrable_on S
      proof (rule measurable_bounded_by_integrable_imp_integrable)
          have (λx. indicat_real {x. h n x = y} (g x)) ∈ borel_measurable
(lebesgue_on S)
            using h_lmeas S
        by (auto simp: indicator_vimage [symmetric] borel_measurable_indicator_iff
sets_restrict_space_iff)
        then show (λu. |det (matrix (g′ u))| * y * indicat_real {x. h n x = y} (g
u)) ∈ borel_measurable (lebesgue_on S)
        by (intro borel_measurable_times borel_measurable_abs borel_measurable_const
borel_measurable_det_Jacobian [OF S der_g])
      next
        fix x
        assume x ∈ S
        then have y * indicat_real {x. h n x = y} (g x) ≤ f (g x)
              by (metis (full_types) h_le_f indicator_simps mem_Collect_eq
```

```
mult.right_neutral mult_zero_right nonneg_fg)
        with ‹y ≥ 0› show norm (?D x * y * indicat_real {x. h n x = y} (g x))
≤ ?D x * f(g x)
          by (simp add: abs_mult mult.assoc mult_left_mono)
       qed (use S det_int_fg in auto)
     qed
     also have ... = integral S (λT. |det (matrix (g' T))| *
                                  (∑ y∈range (h n). y * indicat_real {x. h n x = y}
(g T)))
       by (simp add: sum_distrib_left mult.assoc)
     also have ... = ?rhs
       by (metis hn_eq)
     finally show integral (g ‘ S) (h n) ≤ ?rhs .
   qed
 qed
 have le: integral S (λT. |det (matrix (g' T))| * h n (g T)) ≤ ?b for n
 proof (rule integral_le)
   show (λT. |det (matrix (g' T))| * h n (g T)) integrable_on S
   proof (rule measurable_bounded_by_integrable_imp_integrable)
    have (λT. |det (matrix (g' T))| *R h n (g T)) ∈ borel_measurable (lebesgue_on
S)
    proof (intro borel_measurable_scaleR borel_measurable_abs borel_measurable_det_Jacobian
‹S ∈ sets lebesgue›)
      have eq: {x ∈ S. f x ≤ a} = (⋃ b ∈ (f ‘ S) ∩ atMost a. {x. f x = b} ∩ S)
for f and a::real
        by auto
      have finite ((λx. h n (g x)) ‘ S ∩ {..a}) for a
        by (force intro: finite_subset [OF _ fin_R])
      with h_lmeas [of n] show (λx. h n (g x)) ∈ borel_measurable (lebesgue_on
S)
        apply (simp add: borel_measurable_vimage_halfspace_component_le ‹S
∈ sets lebesgue› sets_restrict_space_iff eq)
        by (metis (mono_tags) SUP_inf sets.finite_UN)
     qed (use der_g in blast)
      then show (λT. |det (matrix (g' T))| * h n (g T)) ∈ borel_measurable
(lebesgue_on S)
       by simp
     show norm (?D x * h n (g x)) ≤ ?D x *R f (g x)
       if x ∈ S for x
       by (simp add: h_le_f mult_left_mono nonneg_h that)
   qed (use S det_int_fg in auto)
   show ?D x * h n (g x) ≤ ?D x * f (g x) if x ∈ S for x
     by (simp add: ‹x ∈ S› h_le_f mult_left_mono)
   show (λx. ?D x * f (g x)) integrable_on S
     using det_int_fg by blast
 qed
 have f integrable_on g ‘ S ∧ (λk. integral (g ‘ S) (h k)) ⟶ integral (g ‘ S) f
 proof (rule monotone_convergence_increasing)
   have |integral (g ‘ S) (h n)| ≤ integral S (λx. ?D x * f (g x)) for n
```

```
    proof −
      have |integral (g ‘ S) (h n)| = integral (g ‘ S) (h n)
        using hint by (simp add: integral_nonneg nonneg_h)
      also have ... ≤ integral S (λx. ?D x ∗ f (g x))
        using hint le by (meson order_trans)
      finally show ?thesis .
    qed
    then show bounded (range (λk. integral (g ‘ S) (h k)))
      by (force simp: bounded_iff)
  qed (use h_inc lim hint in auto)
  moreover have integral (g ‘ S) (h n) ≤ integral S (λx. ?D x ∗ f (g x)) for n
    using hint by (blast intro: le order_trans)
  ultimately show ?thesis
    by (auto intro: Lim_bounded)
qed


lemma integral_on_image_ubound_nonneg:
  fixes f :: real^'n::{finite,wellorder} ⇒ real
  assumes nonneg_fg: ⋀x. x ∈ S ⟹ 0 ≤ f(g x)
      and der_g:  ⋀x. x ∈ S ⟹ (g has_derivative g' x) (at x within S)
      and intS: (λx. |det (matrix (g' x))| ∗ f(g x)) integrable_on S
  shows f integrable_on (g ‘ S) ∧ integral (g ‘ S) f ≤ integral S (λx. |det (matrix
(g' x))| ∗ f(g x))
        (is _ ∧ _ ≤ ?b)
proof −
  let ?D = λx. det (matrix (g' x))
  define S' where S' ≡ {x ∈ S. ?D x ∗ f(g x) ≠ 0}
  then have der_gS': ⋀x. x ∈ S' ⟹ (g has_derivative g' x) (at x within S')
    by (metis (mono_tags, lifting) der_g has_derivative_subset mem_Collect_eq
subset_iff)
  have (λx. if x ∈ S then |?D x| ∗ f (g x) else 0) integrable_on UNIV
    by (simp add: integrable_restrict_UNIV intS)
 then have Df_borel: (λx. if x ∈ S then |?D x| ∗ f (g x) else 0) ∈ borel_measurable
lebesgue
    using integrable_imp_measurable lebesgue_on_UNIV_eq by force
  have S': S' ∈ sets lebesgue
  proof −
    from Df_borel borel_measurable_vimage_open [of _ UNIV]
    have {x. (if x ∈ S then |?D x| ∗ f (g x) else 0) ∈ T} ∈ sets lebesgue
      if open T for T
      using that unfolding lebesgue_on_UNIV_eq
      by (fastforce simp add: dest!: spec)
    then have {x. (if x ∈ S then |?D x| ∗ f (g x) else 0) ∈ −{0}} ∈ sets lebesgue
      using open_Compl by blast
    then show ?thesis
      by (simp add: S'_def conj_ac split: if_split_asm cong: conj_cong)
  qed
  then have gS': g ‘ S' ∈ sets lebesgue
```

```
  proof (rule differentiable_image_in_sets_lebesgue)
    show g differentiable_on S'
      using der_g unfolding S'_def differentiable_def differentiable_on_def
      by (blast intro: has_derivative_subset)
  qed auto
  have f: f ∈ borel_measurable (lebesgue_on (g ` S'))
  proof (clarsimp simp add: borel_measurable_vimage_open)
    fix T :: real set
    assume open T
    have {x ∈ g ` S'. f x ∈ T} = g ` {x ∈ S'. f(g x) ∈ T}
      by blast
    moreover have g ` {x ∈ S'. f(g x) ∈ T} ∈ sets lebesgue
    proof (rule differentiable_image_in_sets_lebesgue)
      let ?h = λx. |?D x| * f (g x) /R |?D x|
      have (λx. if x ∈ S' then |?D x| * f (g x) else 0) = (λx. if x ∈ S then |?D x|
* f (g x) else 0)
        by (auto simp: S'_def)
      also have ... ∈ borel_measurable lebesgue
        by (rule Df_borel)
      finally have *: (λx. |?D x| * f (g x)) ∈ borel_measurable (lebesgue_on S')
        by (simp add: borel_measurable_if_D)
      have (λv. det (matrix (g' v))) ∈ borel_measurable (lebesgue_on S')
        using S' borel_measurable_det_Jacobian der_gS' by blast
      then have ?h ∈ borel_measurable (lebesgue_on S')
      using * borel_measurable_abs borel_measurable_inverse borel_measurable_scaleR
by blast
      moreover have ?h x = f(g x) if x ∈ S' for x
        using that by (auto simp: S'_def)
      ultimately have (λx. f(g x)) ∈ borel_measurable (lebesgue_on S')
        by (metis (no_types, lifting) measurable_lebesgue_cong)
      then show {x ∈ S'. f (g x) ∈ T} ∈ sets lebesgue
       by (simp add: ‹S' ∈ sets lebesgue› ‹open T› borel_measurable_vimage_open
sets_restrict_space_iff)
      show g differentiable_on {x ∈ S'. f (g x) ∈ T}
        using der_g unfolding S'_def differentiable_def differentiable_on_def
        by (blast intro: has_derivative_subset)
    qed auto
    ultimately have {x ∈ g ` S'. f x ∈ T} ∈ sets lebesgue
      by metis
    then show {x ∈ g ` S'. f x ∈ T} ∈ sets (lebesgue_on (g ` S'))
      by (simp add: ‹g ` S' ∈ sets lebesgue› sets_restrict_space_iff)
  qed
  have intS': (λx. |?D x| * f (g x)) integrable_on S'
    using intS
   by (rule integrable_spike_set) (auto simp: S'_def intro: empty_imp_negligible)
  have lebS': {x ∈ S'. g x ∈ T} ∈ sets lebesgue if T ⊆ g ` S' T ∈ sets lebesgue
for T
  proof −
    have g ∈ borel_measurable (lebesgue_on S')
```

```
    using der_gS' has_derivative_continuous_on S'
    by (blast intro: continuous_imp_measurable_on_sets_lebesgue)
  moreover have {x ∈ S'. g x ∈ U} ∈ sets lebesgue if negligible U U ⊆ g ` S'
for U
  proof (intro negligible_imp_sets negligible_differentiable_vimage that)
    fix x
    assume x: x ∈ S'
    then have linear (g' x)
      using der_gS' has_derivative_linear by blast
    with x show inj (g' x)
      by (auto simp: S'_def det_nz_iff_inj)
  qed (use der_gS' in auto)
  ultimately show ?thesis
    using double_lebesgue_sets [OF S' gS' order_refl] that by blast
qed
have int_gS': f integrable_on g ` S' ∧ integral (g ` S') f ≤ integral S' (λx. |?D
x| * f(g x))
  using integral_on_image_ubound_weak [OF S' f nonneg_fg der_gS' intS'
lebS'] S'_def by blast
have negligible (g ` {x ∈ S. det(matrix(g' x)) = 0})
proof (rule baby_Sard, simp_all)
  fix x
  assume x: x ∈ S ∧ det (matrix (g' x)) = 0
  then show (g has_derivative g' x) (at x within {x ∈ S. det (matrix (g' x)) =
0})
    by (metis (no_types, lifting) der_g has_derivative_subset mem_Collect_eq
subsetI)
  then show rank (matrix (g' x)) < CARD('n)
    using det_nz_iff_inj matrix_vector_mul_linear x
    by (fastforce simp add: less_rank_noninjective)
qed
then have negg: negligible (g ` S − g ` {x ∈ S. ?D x ≠ 0})
  by (rule negligible_subset) (auto simp: S'_def)
have null: g ` {x ∈ S. ?D x ≠ 0} − g ` S = {}
  by (auto simp: S'_def)
let ?F = {x ∈ S. f (g x) ≠ 0}
have eq: g ` S' = g ` ?F ∩ g ` {x ∈ S. ?D x ≠ 0}
  by (auto simp: S'_def image_iff)
show ?thesis
proof
  have ((λx. if x ∈ g ` ?F then f x else 0) integrable_on g ` {x ∈ S. ?D x ≠ 0})
    using int_gS' eq integrable_restrict_Int [where f=f]
    by simp
  then have f integrable_on g ` {x ∈ S. ?D x ≠ 0}
    by (auto simp: image_iff elim!: integrable_eq)
  then show f integrable_on g ` S
    using negg null
    by (auto intro: integrable_spike_set [OF _ empty_imp_negligible negligi-
ble_subset])
```

```
    have integral (g ‘ S) f = integral (g ‘ {x ∈ S. ?D x ≠ 0}) f
      using negg by (auto intro: negligible_subset integral_spike_set)
    also have ... = integral (g ‘ {x ∈ S. ?D x ≠ 0}) (λx. if x ∈ g ‘ ?F then f x
else 0)
      by (auto simp: image_iff intro!: integral_cong)
    also have ... = integral (g ‘ S′) f
      using  eq integral_restrict_Int by simp
    also have ... ≤ integral S′ (λx. |?D x| * f(g x))
      by (metis int_gS′)
    also have ... ≤ ?b
      by (rule integral_subset_le [OF _ intS′ intS]) (use nonneg_fg S′_def in
auto)
    finally show integral (g ‘ S) f ≤ ?b .
  qed
qed


lemma absolutely_integrable_on_image_real:
  fixes f :: real^′n::{finite,wellorder} ⇒ real and g :: real^′n::_ ⇒ real^′n::_
  assumes der_g: ⋀x. x ∈ S ⟹ (g has_derivative g′ x) (at x within S)
    and intS: (λx. |det (matrix (g′ x))| * f(g x)) absolutely_integrable_on S
  shows f absolutely_integrable_on (g ‘ S)
proof −
  let ?D = λx. |det (matrix (g′ x))| * f (g x)
  let ?N = {x ∈ S. f (g x) < 0} and ?P = {x ∈ S. f (g x) > 0}
  have eq: {x. (if x ∈ S then ?D x else 0) > 0} = {x ∈ S. ?D x > 0}
           {x. (if x ∈ S then ?D x else 0) < 0} = {x ∈ S. ?D x < 0}
    by auto
  have ?D integrable_on S
    using intS absolutely_integrable_on_def by blast
  then have (λx. if x ∈ S then ?D x else 0) integrable_on UNIV
    by (simp add: integrable_restrict_UNIV)
 then have D_borel: (λx. if x ∈ S then ?D x else 0) ∈ borel_measurable (lebesgue_on
UNIV)
    using integrable_imp_measurable lebesgue_on_UNIV_eq by blast
  then have Dlt: {x ∈ S. ?D x < 0} ∈ sets lebesgue
    unfolding borel_measurable_vimage_halfspace_component_lt
    by (drule_tac x=0 in spec) (auto simp: eq)
  from D_borel have Dgt: {x ∈ S. ?D x > 0} ∈ sets lebesgue
    unfolding borel_measurable_vimage_halfspace_component_gt
    by (drule_tac x=0 in spec) (auto simp: eq)

  have dfgbm: ?D ∈ borel_measurable (lebesgue_on S)
    using intS absolutely_integrable_on_def integrable_imp_measurable by blast
  have der_gN: (g has_derivative g′ x) (at x within ?N) if x ∈ ?N for x
     using der_g has_derivative_subset that by force
  have (λx. − f x) integrable_on g ‘ ?N ∧
        integral (g ‘ ?N) (λx. − f x) ≤ integral ?N (λx. |det (matrix (g′ x))| * − f
(g x))
```

```
  proof (rule integral_on_image_ubound_nonneg [OF _ der_gN])
    have 1: ?D integrable_on {x ∈ S. ?D x < 0}
      using Dlt
     by (auto intro: set_lebesgue_integral_eq_integral [OF set_integrable_subset]
intS)
    have uminus ∘ (λx. |det (matrix (g' x))| ∗ − f (g x)) integrable_on ?N
    by (simp add: o_def mult_less_0_iff empty_imp_negligible integrable_spike_set
[OF 1])
    then show (λx. |det (matrix (g' x))| ∗ − f (g x)) integrable_on ?N
      by (simp add: integrable_neg_iff o_def)
  qed auto
  then have f integrable_on g ‘ ?N
    by (simp add: integrable_neg_iff)
  moreover have g ‘ ?N = {y ∈ g ‘ S. f y < 0}
    by auto
  ultimately have f integrable_on {y ∈ g ‘ S. f y < 0}
    by simp
  then have N: f absolutely_integrable_on {y ∈ g ‘ S. f y < 0}
    by (rule absolutely_integrable_absolutely_integrable_ubound) auto

  have der_gP: (g has_derivative g' x) (at x within ?P) if x ∈ ?P for x
      using der_g has_derivative_subset that by force
    have f integrable_on g ‘ ?P ∧ integral (g ‘ ?P) f ≤ integral ?P ?D
    proof (rule integral_on_image_ubound_nonneg [OF _ der_gP])
      show ?D integrable_on ?P
      proof (rule integrable_spike_set)
        show ?D integrable_on {x ∈ S. 0 < ?D x}
          using Dgt
       by (auto intro: set_lebesgue_integral_eq_integral [OF set_integrable_subset]
intS)
      qed (auto simp: zero_less_mult_iff empty_imp_negligible)
    qed auto
  then have f integrable_on g ‘ ?P
    by metis
  moreover have g ‘ ?P = {y ∈ g ‘ S. f y > 0}
    by auto
  ultimately have f integrable_on {y ∈ g ‘ S. f y > 0}
    by simp
  then have P: f absolutely_integrable_on {y ∈ g ‘ S. f y > 0}
    by (rule absolutely_integrable_absolutely_integrable_lbound) auto
  have (λx. if x ∈ g ‘ S ∧ f x < 0 ∨ x ∈ g ‘ S ∧ 0 < f x then f x else 0) = (λx.
if x ∈ g ‘ S then f x else 0)
    by auto
  then show ?thesis
   using absolutely_integrable_Un [OF N P] absolutely_integrable_restrict_UNIV
[symmetric, where f=f]
    by simp
qed
```

proposition *absolutely_integrable_on_image*:
fixes $f :: real^{\frown\prime}m::\{finite,wellorder\} \Rightarrow real^{\frown\prime}n$ **and** $g :: real^{\frown\prime}m::\_ \Rightarrow real^{\frown\prime}m::\_$
assumes *der_g*: $\bigwedge x.\ x \in S \Longrightarrow (g\ has\_derivative\ g'\ x)\ (at\ x\ within\ S)$
and *intS*: $(\lambda x.\ |det\ (matrix\ (g'\ x))| *_R f(g\ x))\ absolutely\_integrable\_on\ S$
shows $f\ absolutely\_integrable\_on\ (g\ `\ S)$
apply (*rule absolutely_integrable_componentwise* [*OF absolutely_integrable_on_image_real* [*OF der_g*]])
using *absolutely_integrable_component* [*OF intS*] **by** *auto*

proposition *integral_on_image_ubound*:
fixes $f :: real^{\frown\prime}n::\{finite,wellorder\} \Rightarrow real$ **and** $g :: real^{\frown\prime}n::\_ \Rightarrow real^{\frown\prime}n::\_$
assumes$\bigwedge x.\ x \in S \Longrightarrow 0 \le f(g\ x)$
and $\bigwedge x.\ x \in S \Longrightarrow (g\ has\_derivative\ g'\ x)\ (at\ x\ within\ S)$
and $(\lambda x.\ |det\ (matrix\ (g'\ x))| * f(g\ x))\ integrable\_on\ S$
shows $integral\ (g\ `\ S)\ f \le integral\ S\ (\lambda x.\ |det\ (matrix\ (g'\ x))| * f(g\ x))$
using *integral_on_image_ubound_nonneg* [*OF assms*] **by** *simp*

6.41.5 Change-of-variables theorem

The classic change-of-variables theorem. We have two versions with quite general hypotheses, the first that the transforming function has a continuous inverse, the second that the base set is Lebesgue measurable.

lemma *cov_invertible_nonneg_le*:
fixes $f :: real^{\frown\prime}n::\{finite,wellorder\} \Rightarrow real$ **and** $g :: real^{\frown\prime}n::\_ \Rightarrow real^{\frown\prime}n::\_$
assumes *der_g*: $\bigwedge x.\ x \in S \Longrightarrow (g\ has\_derivative\ g'\ x)\ (at\ x\ within\ S)$
and *der_h*: $\bigwedge y.\ y \in T \Longrightarrow (h\ has\_derivative\ h'\ y)\ (at\ y\ within\ T)$
and *f0*: $\bigwedge y.\ y \in T \Longrightarrow 0 \le f\ y$
and *hg*: $\bigwedge x.\ x \in S \Longrightarrow g\ x \in T \wedge h(g\ x) = x$
and *gh*: $\bigwedge y.\ y \in T \Longrightarrow h\ y \in S \wedge g(h\ y) = y$
and *id*: $\bigwedge y.\ y \in T \Longrightarrow h'\ y \circ g'(h\ y) = id$
shows $f\ integrable\_on\ T \wedge (integral\ T\ f) \le b \longleftrightarrow$
$(\lambda x.\ |det\ (matrix\ (g'\ x))| * f(g\ x))\ integrable\_on\ S \wedge$
$integral\ S\ (\lambda x.\ |det\ (matrix\ (g'\ x))| * f(g\ x)) \le b$
(**is** *?lhs = ?rhs*)
proof −
have *Teq*: $T = g`S$ **and** *Seq*: $S = h`T$
using *hg gh image_iff* **by** *fastforce+*
have *gS*: *g differentiable_on S*
by (*meson der_g differentiable_def differentiable_on_def*)
let *?D* $= \lambda x.\ |det\ (matrix\ (g'\ x))| * f\ (g\ x)$
show *?thesis*
proof
assume *?lhs*
then have *fT*: *f integrable_on T* **and** *intf*: *integral T f* $\le b$
by *blast+*
show *?rhs*
proof
let *?fgh* $= \lambda x.\ |det\ (matrix\ (h'\ x))| * (|det\ (matrix\ (g'\ (h\ x)))| * f\ (g\ (h\ x)))$

```
    have ddf: ?fgh x = f x
      if x ∈ T for x
    proof −
      have matrix (h' x) ** matrix (g' (h x)) = mat 1
        by (metis der_g der_h gh has_derivative_linear local.id matrix_compose
matrix_id_mat_1 that)
      then have |det (matrix (h' x))| * |det (matrix (g' (h x)))| = 1
        by (metis abs_1 abs_mult det_I det_mul)
      then show ?thesis
        by (simp add: gh that)
    qed
    have ?D integrable_on (h ` T)
   proof (intro set_lebesgue_integral_eq_integral absolutely_integrable_on_image_real)
      show (λx. ?fgh x) absolutely_integrable_on T
        by (smt (verit, del_insts) abs_absolutely_integrableI_1 ddf f0 fT inte-
grable_eq)
    qed (use der_h in auto)
    with Seq show (λx. ?D x) integrable_on S
      by simp
    have integral S (λx. ?D x) ≤ integral T (λx. ?fgh x)
      unfolding Seq
    proof (rule integral_on_image_ubound)
      show (λx. ?fgh x) integrable_on T
      using ddf fT integrable_eq by force
    qed (use f0 gh der_h in auto)
    also have ... = integral T f
      by (force simp: ddf intro: integral_cong)
    finally show integral S (λx. ?D x) ≤ b
      using intf by linarith
  qed
 next
  assume R: ?rhs
  then have f integrable_on g ` S
    using der_g f0 hg integral_on_image_ubound_nonneg by blast
  moreover have integral (g ` S) f ≤ integral S (λx. ?D x)
    by (rule integral_on_image_ubound [OF f0 der_g]) (use R Teq in auto)
  ultimately show ?lhs
    using R by (simp add: Teq)
 qed
qed


lemma cov_invertible_nonneg_eq:
 fixes f :: real^'n::{finite,wellorder} ⇒ real and g :: real^'n::_ ⇒ real^'n::_
 assumes ⋀x. x ∈ S ⟹ (g has_derivative g' x) (at x within S)
    and ⋀y. y ∈ T ⟹ (h has_derivative h' y) (at y within T)
    and ⋀y. y ∈ T ⟹ 0 ≤ f y
    and ⋀x. x ∈ S ⟹ g x ∈ T ∧ h(g x) = x
    and ⋀y. y ∈ T ⟹ h y ∈ S ∧ g(h y) = y
```

and $\bigwedge y.\ y \in T \Longrightarrow h'\ y \circ g'(h\ y) = id$
shows $((\lambda x.\ |det\ (matrix\ (g'\ x))| * f(g\ x))\ has\_integral\ b)\ S \longleftrightarrow (f\ has\_integral\ b)\ T$
using *cov_invertible_nonneg_le* [*OF assms*]
by (*simp add*: *has_integral_iff*) (*meson eq_iff*)

lemma *cov_invertible_real*:
fixes $f :: real^{\prime}n::\{finite,wellorder\} \Rightarrow real$ **and** $g :: real^{\prime}n::\_ \Rightarrow real^{\prime}n::\_$
assumes *der_g*: $\bigwedge x.\ x \in S \Longrightarrow (g\ has\_derivative\ g'\ x)\ (at\ x\ within\ S)$
and *der_h*: $\bigwedge y.\ y \in T \Longrightarrow (h\ has\_derivative\ h'\ y)\ (at\ y\ within\ T)$
and *hg*: $\bigwedge x.\ x \in S \Longrightarrow g\ x \in T \wedge h(g\ x) = x$
and *gh*: $\bigwedge y.\ y \in T \Longrightarrow h\ y \in S \wedge g(h\ y) = y$
and *id*: $\bigwedge y.\ y \in T \Longrightarrow h'\ y \circ g'(h\ y) = id$
shows $(\lambda x.\ |det\ (matrix\ (g'\ x))| * f(g\ x))\ absolutely\_integrable\_on\ S \wedge$
$integral\ S\ (\lambda x.\ |det\ (matrix\ (g'\ x))| * f(g\ x)) = b \longleftrightarrow$
$f\ absolutely\_integrable\_on\ T \wedge integral\ T\ f = b$
(**is** *?lhs* = *?rhs*)
proof −
have *Teq*: $T = g`S$ **and** *Seq*: $S = h`T$
using *hg gh image_iff* **by** *fastforce+*
let *?DP* = $\lambda x.\ |det\ (matrix\ (g'\ x))| * f(g\ x)$ **and** *?DN* = $\lambda x.\ |det\ (matrix\ (g'\ x))| * -f(g\ x)$
have +: (*?DP has_integral b*) $\{x \in S.\ f\ (g\ x) > 0\} \longleftrightarrow (f\ has\_integral\ b)\ \{y \in T.\ f\ y > 0\}$ **for** *b*
proof (*rule cov_invertible_nonneg_eq*)
have ∗: $(\lambda x.\ f\ (g\ x)) -` Y \cap \{x \in S.\ f\ (g\ x) > 0\}$
$= ((\lambda x.\ f\ (g\ x)) -` Y \cap S) \cap \{x \in S.\ f\ (g\ x) > 0\}$ **for** *Y*
by *auto*
show (*g has_derivative g' x*) (*at x within* $\{x \in S.\ f\ (g\ x) > 0\}$) **if** $x \in \{x \in S.\ f\ (g\ x) > 0\}$ **for** *x*
using *that der_g has_derivative_subset* **by** *fastforce*
show (*h has_derivative h' y*) (*at y within* $\{y \in T.\ f\ y > 0\}$) **if** $y \in \{y \in T.\ f\ y > 0\}$ **for** *y*
using *that der_h has_derivative_subset* **by** *fastforce*
qed (*use gh hg id* **in** *auto*)
have −: (*?DN has_integral b*) $\{x \in S.\ f\ (g\ x) < 0\} \longleftrightarrow ((\lambda x.\ -f\ x)\ has\_integral\ b)\ \{y \in T.\ f\ y < 0\}$ **for** *b*
proof (*rule cov_invertible_nonneg_eq*)
have ∗: $(\lambda x.\ -f\ (g\ x)) -` y \cap \{x \in S.\ f\ (g\ x) < 0\}$
$= ((\lambda x.\ f\ (g\ x)) -` uminus\ `\ y \cap S) \cap \{x \in S.\ f\ (g\ x) < 0\}$ **for** *y*
using *image_iff* **by** *fastforce*
show (*g has_derivative g' x*) (*at x within* $\{x \in S.\ f\ (g\ x) < 0\}$) **if** $x \in \{x \in S.\ f\ (g\ x) < 0\}$ **for** *x*
using *that der_g has_derivative_subset* **by** *fastforce*
show (*h has_derivative h' y*) (*at y within* $\{y \in T.\ f\ y < 0\}$) **if** $y \in \{y \in T.\ f\ y < 0\}$ **for** *y*
using *that der_h has_derivative_subset* **by** *fastforce*
qed (*use gh hg id* **in** *auto*)

show *?thesis*
proof
assume *LHS*: *?lhs*
have *eq*: $\{x.\ (\textit{if } x \in S \textit{ then } ?DP\ x \textit{ else } 0) > 0\} = \{x \in S.\ ?DP\ x > 0\}$
$\{x.\ (\textit{if } x \in S \textit{ then } ?DP\ x \textit{ else } 0) < 0\} = \{x \in S.\ ?DP\ x < 0\}$
by *auto*
have *?DP integrable_on S*
using *LHS absolutely_integrable_on_def* **by** *blast*
then have $(\lambda x.\ \textit{if } x \in S \textit{ then } ?DP\ x \textit{ else } 0)$ *integrable_on UNIV*
by (*simp add*: *integrable_restrict_UNIV*)
then have *D_borel*: $(\lambda x.\ \textit{if } x \in S \textit{ then } ?DP\ x \textit{ else } 0) \in$ *borel_measurable* (*lebesgue_on UNIV*)
using *integrable_imp_measurable lebesgue_on_UNIV_eq* **by** *blast*
then have *SN*: $\{x \in S.\ ?DP\ x < 0\} \in$ *sets lebesgue*
unfolding *borel_measurable_vimage_halfspace_component_lt*
by (*drule_tac x=0* **in** *spec*) (*auto simp*: *eq*)
from *D_borel* **have** *SP*: $\{x \in S.\ ?DP\ x > 0\} \in$ *sets lebesgue*
unfolding *borel_measurable_vimage_halfspace_component_gt*
by (*drule_tac x=0* **in** *spec*) (*auto simp*: *eq*)
have *?DP absolutely_integrable_on* $\{x \in S.\ ?DP\ x > 0\}$
using *LHS* **by** (*fast intro*!: *set_integrable_subset* [*OF _*, *of _ S*] *SP*)
then have *aP*: *?DP absolutely_integrable_on* $\{x \in S.\ f\ (g\ x) > 0\}$
by (*rule absolutely_integrable_spike_set*) (*auto simp*: *zero_less_mult_iff empty_imp_negligible*)
have *?DP absolutely_integrable_on* $\{x \in S.\ ?DP\ x < 0\}$
using *LHS* **by** (*fast intro*!: *set_integrable_subset* [*OF _*, *of _ S*] *SN*)
then have *aN*: *?DP absolutely_integrable_on* $\{x \in S.\ f\ (g\ x) < 0\}$
by (*rule absolutely_integrable_spike_set*) (*auto simp*: *mult_less_0_iff empty_imp_negligible*)
have *fN*: *f integrable_on* $\{y \in T.\ f\ y < 0\}$
integral $\{y \in T.\ f\ y < 0\}\ f =$ *integral* $\{x \in S.\ f\ (g\ x) < 0\}$ *?DP*
using $-$ [*of integral* $\{x \in S.\ f(g\ x) < 0\}$ *?DN*] *aN*
by (*auto simp*: *set_lebesgue_integral_eq_integral has_integral_iff integrable_neg_iff*)
have *faN*: *f absolutely_integrable_on* $\{y \in T.\ f\ y < 0\}$
proof (*rule absolutely_integrable_integrable_bound*)
show $(\lambda x.\ -\ f\ x)$ *integrable_on* $\{y \in T.\ f\ y < 0\}$
using *fN* **by** (*auto simp*: *integrable_neg_iff*)
qed (*use fN* **in** *auto*)
have *fP*: *f integrable_on* $\{y \in T.\ f\ y > 0\}$
integral $\{y \in T.\ f\ y > 0\}\ f =$ *integral* $\{x \in S.\ f\ (g\ x) > 0\}$ *?DP*
using $+$ [*of integral* $\{x \in S.\ f(g\ x) > 0\}$ *?DP*] *aP*
by (*auto simp*: *set_lebesgue_integral_eq_integral has_integral_iff integrable_neg_iff*)
have *faP*: *f absolutely_integrable_on* $\{y \in T.\ f\ y > 0\}$
using *fP*(*1*) *nonnegative_absolutely_integrable_1* **by** *fastforce*
have *fa*: *f absolutely_integrable_on* $(\{y \in T.\ f\ y < 0\} \cup \{y \in T.\ f\ y > 0\})$
by (*rule absolutely_integrable_Un* [*OF faN faP*])
show *?rhs*
proof
have *eq*: $((\textit{if } x \in T \wedge f\ x < 0 \vee x \in T \wedge 0 < f\ x \textit{ then } 1 \textit{ else } 0) * f\ x)$
$= (\textit{if } x \in T \textit{ then } 1 \textit{ else } 0) * f\ x$ **for** x

```
      by auto
    show f absolutely_integrable_on T
      using fa by (simp add: indicator_def of_bool_def set_integrable_def eq)
     have [simp]: {y ∈ T. f y < 0} ∩ {y ∈ T. 0 < f y} = {} for T and f ::
('real^'n::_) ⇒ real
      by auto
    have integral T f = integral ({y ∈ T. f y < 0} ∪ {y ∈ T. f y > 0}) f
      by (intro empty_imp_negligible integral_spike_set) (auto simp: eq)
    also have ... = integral {y ∈ T. f y < 0} f + integral {y ∈ T. f y > 0} f
      using fN fP by simp
    also have ... = integral {x ∈ S. f (g x) < 0} ?DP + integral {x ∈ S. 0 <
f (g x)} ?DP
      by (simp add: fN fP)
   also have ... = integral ({x ∈ S. f (g x) < 0} ∪ {x ∈ S. 0 < f (g x)}) ?DP
      using aP aN by (simp add: set_lebesgue_integral_eq_integral)
    also have ... = integral S ?DP
      by (intro empty_imp_negligible integral_spike_set) auto
    also have ... = b
      using LHS by simp
    finally show integral T f = b .
  qed
 next
  assume RHS: ?rhs
  have eq: {x. (if x ∈ T then f x else 0) > 0} = {x ∈ T. f x > 0}
          {x. (if x ∈ T then f x else 0) < 0} = {x ∈ T. f x < 0}
    by auto
  have f integrable_on T
    using RHS absolutely_integrable_on_def by blast
  then have (λx. if x ∈ T then f x else 0) integrable_on UNIV
    by (simp add: integrable_restrict_UNIV)
    then have D_borel: (λx. if x ∈ T then f x else 0) ∈ borel_measurable
(lebesgue_on UNIV)
    using integrable_imp_measurable lebesgue_on_UNIV_eq by blast
  then have TN: {x ∈ T. f x < 0} ∈ sets lebesgue
    unfolding borel_measurable_vimage_halfspace_component_lt
    by (drule_tac x=0 in spec) (auto simp: eq)
  from D_borel have TP: {x ∈ T. f x > 0} ∈ sets lebesgue
    unfolding borel_measurable_vimage_halfspace_component_gt
    by (drule_tac x=0 in spec) (auto simp: eq)
  have aint: f absolutely_integrable_on {y. y ∈ T ∧ 0 < (f y)}
          f absolutely_integrable_on {y. y ∈ T ∧ (f y) < 0}
      and intT: integral T f = b
    using set_integrable_subset [of _ T] TP TN RHS by blast+
  show ?lhs
  proof
    have fN: f integrable_on {v ∈ T. f v < 0}
      using absolutely_integrable_on_def aint by blast
   then have DN: (?DN has_integral integral {y ∈ T. f y < 0} (λx. − f x)) {x
∈ S. f (g x) < 0}
```

```
      using − [of integral {y ∈ T. f y < 0} (λx. − f x)]
      by (simp add: has_integral_neg_iff integrable_integral)
    have aDN: ?DP absolutely_integrable_on {x ∈ S. f (g x) < 0}
      apply (rule absolutely_integrable_integrable_bound [where g = ?DN])
      using DN hg by (fastforce simp: abs_mult integrable_neg_iff)+
    have fP: f integrable_on {v ∈ T. f v > 0}
      using absolutely_integrable_on_def aint by blast
    then have DP: (?DP has_integral integral {y ∈ T. f y > 0} f) {x ∈ S. f (g
x) > 0}
      using + [of integral {y ∈ T. f y > 0} f]
      by (simp add: has_integral_neg_iff integrable_integral)
    have aDP: ?DP absolutely_integrable_on {x ∈ S. f (g x) > 0}
      apply (rule absolutely_integrable_integrable_bound [where g = ?DP])
      using DP hg by (fastforce simp: integrable_neg_iff)+
    have eq: (if x ∈ S then 1 else 0) * ?DP x = (if x ∈ S ∧ f (g x) < 0 ∨ x ∈ S
∧ f (g x) > 0 then 1 else 0) * ?DP x for x
      by force
    have ?DP absolutely_integrable_on ({x ∈ S. f (g x) < 0} ∪ {x ∈ S. f (g x)
> 0})
      by (rule absolutely_integrable_Un [OF aDN aDP])
    then show I: ?DP absolutely_integrable_on S
      by (simp add: indicator_def of_bool_def eq set_integrable_def)
     have [simp]: {y ∈ S. f y < 0} ∩ {y ∈ S. 0 < f y} = {} for S and f ::
(real^'n::_) ⇒ real
      by auto
    have integral S ?DP = integral ({x ∈ S. f (g x) < 0} ∪ {x ∈ S. f (g x) >
0}) ?DP
      by (intro empty_imp_negligible integral_spike_set) auto
    also have ... = integral {x ∈ S. f (g x) < 0} ?DP + integral {x ∈ S. 0 <
f (g x)} ?DP
      using aDN aDP by (simp add: set_lebesgue_integral_eq_integral)
    also have ... = − integral {y ∈ T. f y < 0} (λx. − f x) + integral {y ∈ T.
f y > 0} f
      using DN DP by (auto simp: has_integral_iff)
    also have ... = integral ({x ∈ T. f x < 0} ∪ {x ∈ T. 0 < f x}) f
      by (simp add: fN fP)
    also have ... = integral T f
      by (intro empty_imp_negligible integral_spike_set) auto
    also have ... = b
      using intT by simp
    finally show integral S ?DP = b .
  qed
 qed
qed


lemma cv_inv_version3:
 fixes f :: real^'m::{finite,wellorder} ⇒ real^'n and g :: real^'m::_ ⇒ real^'m::_
 assumes der_g: ⋀x. x ∈ S ⟹ (g has_derivative g' x) (at x within S)
```

```
    and der_h: ⋀y. y ∈ T ⟹ (h has_derivative h' y) (at y within T)
    and hg: ⋀x. x ∈ S ⟹ g x ∈ T ∧ h(g x) = x
    and gh: ⋀y. y ∈ T ⟹ h y ∈ S ∧ g(h y) = y
    and id: ⋀y. y ∈ T ⟹ h' y ∘ g'(h y) = id
  shows (λx. |det (matrix (g' x))| *R f(g x)) absolutely_integrable_on S ∧
            integral S (λx. |det (matrix (g' x))| *R f(g x)) = b
        ⟷ f absolutely_integrable_on T ∧ integral T f = b
proof −
  let ?D = λx. |det (matrix (g' x))| *R f(g x)
  have ((λx. |det (matrix (g' x))| * f(g x) $ i) absolutely_integrable_on S ∧ integral
S (λx. |det (matrix (g' x))| * (f(g x) $ i)) = b $ i) ⟷
        ((λx. f x $ i) absolutely_integrable_on T ∧ integral T (λx. f x $ i) = b $
i) for i
    by (rule cov_invertible_real [OF der_g der_h hg gh id])
  then have ?D absolutely_integrable_on S ∧ (?D has_integral b) S ⟷
        f absolutely_integrable_on T ∧ (f has_integral b) T
    unfolding absolutely_integrable_componentwise_iff [where f=f] has_integral_componentwise_iff
[of f]
          absolutely_integrable_componentwise_iff [where f=?D] has_integral_componentwise_iff
[of ?D]
    by (auto simp: all_conj_distrib Basis_vec_def cart_eq_inner_axis [symmetric]
          has_integral_iff set_lebesgue_integral_eq_integral)
  then show ?thesis
    using absolutely_integrable_on_def by blast
qed


lemma cv_inv_version4:
  fixes f :: real^'m::{finite,wellorder} ⇒ real^'n and g :: real^'m::_ ⇒ real^'m::_
  assumes der_g: ⋀x. x ∈ S ⟹ (g has_derivative g' x) (at x within S) ∧ invert-
ible(matrix(g' x))
    and hg: ⋀x. x ∈ S ⟹ continuous_on (g ` S) h ∧ h(g x) = x
  shows (λx. |det (matrix (g' x))| *R f(g x)) absolutely_integrable_on S ∧
            integral S (λx. |det (matrix (g' x))| *R f(g x)) = b
        ⟷ f absolutely_integrable_on (g ` S) ∧ integral (g ` S) f = b
proof −
  have ∀x. ∃h'. x ∈ S
          ⟶ (g has_derivative g' x) (at x within S) ∧ linear h' ∧ g' x ∘ h' = id ∧
h' ∘ g' x = id
    using der_g matrix_invertible has_derivative_linear by blast
  then obtain h' where h':
    ⋀x. x ∈ S
          ⟹ (g has_derivative g' x) (at x within S) ∧
             linear (h' x) ∧ g' x ∘ (h' x) = id ∧ (h' x) ∘ g' x = id
    by metis
  show ?thesis
  proof (rule cv_inv_version3)
    show ⋀y. y ∈ g ` S ⟹ (h has_derivative h' (h y)) (at y within g ` S)
      using h' hg
```

```
    by (force simp: continuous_on_eq_continuous_within intro!: has_derivative_inverse_within)
  qed (use h' hg in auto)
qed


theorem has_absolute_integral_change_of_variables_invertible:
  fixes f :: real^'m::{finite,wellorder} ⇒ real^'n and g :: real^'m::_ ⇒ real^'m::_
  assumes der_g: ⋀x. x ∈ S ⟹ (g has_derivative g' x) (at x within S)
      and hg: ⋀x. x ∈ S ⟹ h(g x) = x
      and conth: continuous_on (g ` S) h
  shows (λx. |det (matrix (g' x))| *R f(g x)) absolutely_integrable_on S ∧ integral
S (λx. |det (matrix (g' x))| *R f(g x)) = b ⟷
        f absolutely_integrable_on (g ` S) ∧ integral (g ` S) f = b
    (is ?lhs = ?rhs)
proof −
  let ?S = {x ∈ S. invertible (matrix (g' x))} and ?D = λx. |det (matrix (g' x))|
*R f(g x)
  have *: ?D absolutely_integrable_on ?S ∧ integral ?S ?D = b
          ⟷ f absolutely_integrable_on (g ` ?S) ∧ integral (g ` ?S) f = b
  proof (rule cv_inv_version4)
    show (g has_derivative g' x) (at x within ?S) ∧ invertible (matrix (g' x))
      if x ∈ ?S for x
      using der_g that has_derivative_subset that by fastforce
    show continuous_on (g ` ?S) h ∧ h (g x) = x
      if x ∈ ?S for x
     using that continuous_on_subset [OF conth] by (simp add: hg image_mono)
  qed
   have (g has_derivative g' x) (at x within {x ∈ S. rank (matrix (g' x)) <
CARD('m)}) if x ∈ S for x
     by (metis (no_types, lifting) der_g has_derivative_subset mem_Collect_eq
subsetI that)
  then have negligible (g ` {x ∈ S. ¬ invertible (matrix (g' x))})
    by (auto simp: invertible_det_nz det_eq_0_rank intro: baby_Sard)
  then have neg: negligible {x ∈ g ` S. x ∉ g ` ?S ∧ f x ≠ 0}
    by (auto intro: negligible_subset)
  have [simp]: {x ∈ g ` ?S. x ∉ g ` S ∧ f x ≠ 0} = {}
    by auto
  have ?D absolutely_integrable_on ?S ∧ integral ?S ?D = b
    ⟷ ?D absolutely_integrable_on S ∧ integral S ?D = b
    apply (intro conj_cong absolutely_integrable_spike_set_eq)
     apply(auto simp: integral_spike_set invertible_det_nz empty_imp_negligible
neg)
    done
  moreover
  have f absolutely_integrable_on (g ` ?S) ∧ integral (g ` ?S) f = b
    ⟷ f absolutely_integrable_on (g ` S) ∧ integral (g ` S) f = b
   by (auto intro!: conj_cong absolutely_integrable_spike_set_eq integral_spike_set
neg)
  ultimately
```

show *?thesis*
using ∗ **by** *blast*
qed

theorem *has_absolute_integral_change_of_variables_compact*:
fixes *f* :: *real^'m*::{*finite*,*wellorder*} ⇒ *real^'n* **and** *g* :: *real^'m*::_ ⇒ *real^'m*::_
assumes *compact S*
and *der_g*: ⋀*x*. *x* ∈ *S* ⟹ (*g has_derivative g' x*) (*at x within S*)
and *inj*: *inj_on g S*
shows ((λ*x*. |*det* (*matrix* (*g' x*))| ∗*R* *f*(*g x*)) *absolutely_integrable_on S* ∧
integral S (λ*x*. |*det* (*matrix* (*g' x*))| ∗*R* *f*(*g x*)) = *b*)
⟷ *f absolutely_integrable_on* (*g* ‘ *S*) ∧ *integral* (*g* ‘ *S*) *f* = *b*)
proof −
obtain *h* **where** *hg*: ⋀*x*. *x* ∈ *S* ⟹ *h*(*g x*) = *x*
using *inj* **by** (*metis the_inv_into_f_f*)
have *conth*: *continuous_on* (*g* ‘ *S*) *h*
by (*metis* ‹*compact S*› *continuous_on_inv der_g has_derivative_continuous_on hg*)
show *?thesis*
by (*rule has_absolute_integral_change_of_variables_invertible* [*OF der_g hg conth*])
qed

lemma *has_absolute_integral_change_of_variables_compact_family*:
fixes *f* :: *real^'m*::{*finite*,*wellorder*} ⇒ *real^'n* **and** *g* :: *real^'m*::_ ⇒ *real^'m*::_
assumes *compact*: ⋀*n*::*nat*. *compact* (*F n*)
and *der_g*: ⋀*x*. *x* ∈ (⋃*n*. *F n*) ⟹ (*g has_derivative g' x*) (*at x within* (⋃*n*. *F n*))
and *inj*: *inj_on g* (⋃*n*. *F n*)
shows ((λ*x*. |*det* (*matrix* (*g' x*))| ∗*R* *f*(*g x*)) *absolutely_integrable_on* (⋃*n*. *F n*) ∧
integral (⋃*n*. *F n*) (λ*x*. |*det* (*matrix* (*g' x*))| ∗*R* *f*(*g x*)) = *b*)
⟷ *f absolutely_integrable_on* (*g* ‘ (⋃*n*. *F n*)) ∧ *integral* (*g* ‘ (⋃*n*. *F n*)) *f* = *b*)
proof −
let *?D* = λ*x*. |*det* (*matrix* (*g' x*))| ∗*R* *f* (*g x*)
let *?U* = λ*n*. ⋃*m*≤*n*. *F m*
let *?lift* = *vec*::*real*⇒*real^1*
have *F_leb*: *F m* ∈ *sets lebesgue* **for** *m*
by (*simp add*: *compact borel_compact*)
have *iff*: (λ*x*. |*det* (*matrix* (*g' x*))| ∗*R* *f* (*g x*)) *absolutely_integrable_on* (*?U n*) ∧
integral (*?U n*) (λ*x*. |*det* (*matrix* (*g' x*))| ∗*R* *f* (*g x*)) = *b*
⟷ *f absolutely_integrable_on* (*g* ‘ (*?U n*)) ∧ *integral* (*g* ‘ (*?U n*)) *f* = *b*
for *n b* **and** *f* :: *real^'m*::_ ⇒ *real^'k*
proof (*rule has_absolute_integral_change_of_variables_compact*)

```
    show compact (?U n)
      by (simp add: compact compact_UN)
    show (g has_derivative g' x) (at x within (?U n))
      if x ∈ ?U n for x
      using that by (blast intro!: has_derivative_subset [OF der_g])
    show inj_on g (?U n)
      using inj by (auto simp: inj_on_def)
  qed
  show ?thesis
    unfolding image_UN
  proof safe
    assume DS: ?D absolutely_integrable_on (⋃ n. F n)
      and b: b = integral (⋃ n. F n) ?D
    have DU: ⋀n. ?D absolutely_integrable_on (?U n)
          (λn. integral (?U n) ?D) ⟶ integral (⋃ n. F n) ?D
      using integral_countable_UN [OF DS F_leb] by auto
    with iff have fag: f absolutely_integrable_on g ` (?U n)
      and fg_int: integral (⋃ m≤n. g ` F m) f = integral (?U n) ?D for n
      by (auto simp: image_UN)
    let ?h = λx. if x ∈ (⋃ m. g ` F m) then norm(f x) else 0
    have (λx. if x ∈ (⋃ m. g ` F m) then f x else 0) absolutely_integrable_on UNIV
    proof (rule dominated_convergence_absolutely_integrable)
      show (λx. if x ∈ (⋃ m≤k. g ` F m) then f x else 0) absolutely_integrable_on
UNIV for k
        unfolding absolutely_integrable_restrict_UNIV
        using fag by (simp add: image_UN)
      let ?nf = λn x. if x ∈ (⋃ m≤n. g ` F m) then norm(f x) else 0
      show ?h integrable_on UNIV
      proof (rule monotone_convergence_increasing [THEN conjunct1])
        show ?nf k integrable_on UNIV for k
          using fag
            unfolding integrable_restrict_UNIV absolutely_integrable_on_def by
(simp add: image_UN)
        { fix n
          have (norm ∘ ?D) absolutely_integrable_on ?U n
            by (intro absolutely_integrable_norm DU)
          then have integral (g ` ?U n) (norm ∘ f) = integral (?U n) (norm ∘ ?D)
            using iff [of n vec ∘ norm ∘ f integral (?U n) (λx. |det (matrix (g' x))|
*R (?lift ∘ norm ∘ f) (g x))]
            unfolding absolutely_integrable_on_1_iff integral_on_1_eq by (auto
simp: o_def)
        }
        moreover have bounded (range (λk. integral (?U k) (norm ∘ ?D)))
          unfolding bounded_iff
        proof (rule exI, clarify)
          fix k
          show norm (integral (?U k) (norm ∘ ?D)) ≤ integral (⋃ n. F n) (norm ∘
?D)
            unfolding integral_restrict_UNIV [of _ norm ∘ ?D, symmetric]
```

```
        proof (rule integral_norm_bound_integral)
         show (λx. if x ∈ ⋃ (F ‘ {..k}) then (norm ∘ ?D) x else 0) integrable_on
UNIV
           (λx. if x ∈ (⋃ n. F n) then (norm ∘ ?D) x else 0) integrable_on UNIV
            using DU(1) DS
         unfolding absolutely_integrable_on_def o_def integrable_restrict_UNIV
by auto
        qed auto
      qed
      ultimately show bounded (range (λk. integral UNIV (?nf k)))
        by (simp add: integral_restrict_UNIV image_UN [symmetric] o_def)
    next
      show (λk. if x ∈ (⋃ m≤k. g ‘ F m) then norm (f x) else 0)
            ⟶ (if x ∈ (⋃ m. g ‘ F m) then norm (f x) else 0) for x
        by (force intro: tendsto_eventually eventually_sequentiallyI)
    qed auto
  next
    show (λk. if x ∈ (⋃ m≤k. g ‘ F m) then f x else 0)
          ⟶ (if x ∈ (⋃ m. g ‘ F m) then f x else 0) for x
    proof clarsimp
      fix m y
      assume y ∈ F m
      show (λk. if ∃ x∈{..k}. g y ∈ g ‘ F x then f (g y) else 0) ⟶ f (g y)
     using ‹y ∈ F m› by (force intro: tendsto_eventually eventually_sequentiallyI
[of m])
    qed
  qed auto
  then show fai: f absolutely_integrable_on (⋃ m. g ‘ F m)
    using absolutely_integrable_restrict_UNIV by blast
  show integral ((⋃ x. g ‘ F x)) f = integral (⋃ n. F n) ?D
  proof (rule LIMSEQ_unique)
    show (λn. integral (?U n) ?D) ⟶ integral (⋃ x. g ‘ F x) f
      unfolding fg_int [symmetric]
    proof (rule integral_countable_UN [OF fai])
      show g ‘ F m ∈ sets lebesgue for m
      proof (rule differentiable_image_in_sets_lebesgue [OF F_leb])
        show g differentiable_on F m
          by (meson der_g differentiableI UnionI differentiable_on_def differen-
tiable_on_subset rangeI subsetI)
      qed auto
    qed
  qed (use DU in metis)
 next
  assume fs: f absolutely_integrable_on (⋃ x. g ‘ F x)
    and b: b = integral ((⋃ x. g ‘ F x)) f
  have gF_leb: g ‘ F m ∈ sets lebesgue for m
  proof (rule differentiable_image_in_sets_lebesgue [OF F_leb])
    show g differentiable_on F m
      using der_g unfolding differentiable_def differentiable_on_def
```

```
      by (meson Sup_upper UNIV_I UnionI has_derivative_subset image_eqI)
    qed auto
    have fgU: ⋀n. f absolutely_integrable_on (⋃m≤n. g ‘ F m)
     (λn. integral (⋃m≤n. g ‘ F m) f) ⟶ integral (⋃m. g ‘ F m) f
     using integral_countable_UN [OF fs gF_leb] by auto
    with iff have DUn: ?D absolutely_integrable_on ?U n
     and D_int: integral (?U n) ?D = integral (⋃m≤n. g ‘ F m) f for n
     by (auto simp: image_UN)
    let ?h = λx. if x ∈ (⋃n. F n) then norm(?D x) else 0
    have (λx. if x ∈ (⋃n. F n) then ?D x else 0) absolutely_integrable_on UNIV
    proof (rule dominated_convergence_absolutely_integrable)
     show (λx. if x ∈ ?U k then ?D x else 0) absolutely_integrable_on UNIV for
k
      unfolding absolutely_integrable_restrict_UNIV using DUn by simp
     let ?nD = λn x. if x ∈ ?U n then norm(?D x) else 0
     show ?h integrable_on UNIV
     proof (rule monotone_convergence_increasing [THEN conjunct1])
      show ?nD k integrable_on UNIV for k
       using DUn
        unfolding integrable_restrict_UNIV absolutely_integrable_on_def by
(simp add: image_UN)
      { fix n::nat
       have (norm ∘ f) absolutely_integrable_on (⋃m≤n. g ‘ F m)
        using absolutely_integrable_norm fgU by blast
       then have integral (?U n) (norm ∘ ?D) = integral (g ‘ ?U n) (norm ∘ f)
        using iff [of n ?lift ∘ norm ∘ f integral (g ‘ ?U n) (?lift ∘ norm ∘ f)]
        unfolding absolutely_integrable_on_1_iff integral_on_1_eq image_UN
by (auto simp: o_def)
      }
      moreover have bounded (range (λk. integral (g ‘ ?U k) (norm ∘ f)))
       unfolding bounded_iff
      proof (rule exI, clarify)
       fix k
        show norm (integral (g ‘ ?U k) (norm ∘ f)) ≤ integral (g ‘ (⋃n. F n))
(norm ∘ f)
         unfolding integral_restrict_UNIV [of _ norm ∘ f, symmetric]
       proof (rule integral_norm_bound_integral)
        show (λx. if x ∈ g ‘ ?U k then (norm ∘ f) x else 0) integrable_on UNIV
             (λx. if x ∈ g ‘ (⋃n. F n) then (norm ∘ f) x else 0) integrable_on
UNIV
          using fgU fs
        unfolding absolutely_integrable_on_def o_def integrable_restrict_UNIV
          by (auto simp: image_UN)
       qed auto
      qed
      ultimately show bounded (range (λk. integral UNIV (?nD k)))
       unfolding integral_restrict_UNIV image_UN [symmetric] o_def by simp
     next
      show (λk. if x ∈ ?U k then norm (?D x) else 0) ⟶ (if x ∈ (⋃n. F n)
```

then norm (?D x) else 0) **for** *x*
by (*force intro*: *tendsto_eventually eventually_sequentiallyI*)
qed *auto*
next
show (*λk. if x ∈ ?U k then ?D x else 0*) ⟶ (*if x ∈ (⋃ n. F n) then ?D x else 0*) **for** *x*
proof *clarsimp*
fix *n*
assume *x ∈ F n*
show (*λm. if ∃ j∈{..m}. x ∈ F j then ?D x else 0*) ⟶ *?D x*
using ‹*x ∈ F n*› **by** (*auto intro*!: *tendsto_eventually eventually_sequentiallyI* [*of n*])
qed
qed *auto*
then show *Dai*: *?D absolutely_integrable_on* (⋃ *n. F n*)
unfolding *absolutely_integrable_restrict_UNIV* **by** *simp*
show *integral* (⋃ *n. F n*) *?D* = *integral* ((⋃ *x. g ‘ F x*)) *f*
proof (*rule LIMSEQ_unique*)
show (*λn. integral* (⋃ *m≤n. g ‘ F m*) *f*) ⟶ *integral* (⋃ *n. F n*) *?D*
unfolding *D_int* [*symmetric*] **by** (*rule integral_countable_UN* [*OF Dai F_leb*])
qed (*use fgU* **in** *metis*)
qed
qed

theorem *has_absolute_integral_change_of_variables*:
fixes *f* :: *real^'m::*{*finite,wellorder*} ⇒ *real^'n* **and** *g* :: *real^'m::_* ⇒ *real^'m::_*
assumes *S*: *S ∈ sets lebesgue*
and *der_g*: ⋀*x. x ∈ S* ⟹ (*g has_derivative g' x*) (*at x within S*)
and *inj*: *inj_on g S*
shows (*λx. |det* (*matrix* (*g' x*))| ∗*R f*(*g x*)) *absolutely_integrable_on S* ∧
integral S (*λx. |det* (*matrix* (*g' x*))| ∗*R f*(*g x*)) = *b*
⟷ *f absolutely_integrable_on* (*g ‘ S*) ∧ *integral* (*g ‘ S*) *f* = *b*
proof −
obtain *C N* **where** *fsigma C* **and** *N*: *N ∈ null_sets lebesgue* **and** *CNS*: *C* ∪ *N* = *S* **and** *disjnt C N*
using *lebesgue_set_almost_fsigma* [*OF S*] **.**
then obtain *F* :: *nat* ⇒ (*real^'m::_*) *set*
where *F*: *range F* ⊆ *Collect compact* **and** *Ceq*: *C* = *Union*(*range F*)
using *fsigma_Union_compact* **by** *metis*
have *negligible N*
using *N* **by** (*simp add*: *negligible_iff_null_sets*)
let *?D* = *λx. |det* (*matrix* (*g' x*))| ∗*R f* (*g x*)
have *?D absolutely_integrable_on C* ∧ *integral C ?D* = *b*
⟷ *f absolutely_integrable_on* (*g ‘ C*) ∧ *integral* (*g ‘ C*) *f* = *b*
unfolding *Ceq*
proof (*rule has_absolute_integral_change_of_variables_compact_family*)
fix *n x*

```
    assume x ∈ ⋃(F ‘ UNIV)
    then show (g has_derivative g' x) (at x within ⋃(F ‘ UNIV))
      using Ceq ‹C ∪ N = S› der_g has_derivative_subset by blast
  next
    have ⋃(F ‘ UNIV) ⊆ S
      using Ceq ‹C ∪ N = S› by blast
    then show inj_on g (⋃(F ‘ UNIV))
      using inj by (meson inj_on_subset)
  qed (use F in auto)
  moreover
  have ?D absolutely_integrable_on C ∧ integral C ?D = b
    ⟷ ?D absolutely_integrable_on S ∧ integral S ?D = b
  proof (rule conj_cong)
    have neg: negligible {x ∈ C − S. ?D x ≠ 0} negligible {x ∈ S − C. ?D x ≠
0}
      using CNS by (blast intro: negligible_subset [OF ‹negligible N›])+
    then show (?D absolutely_integrable_on C) = (?D absolutely_integrable_on
S)
      by (rule absolutely_integrable_spike_set_eq)
    show (integral C ?D = b) ⟷ (integral S ?D = b)
      using integral_spike_set [OF neg] by simp
  qed
  moreover
  have f absolutely_integrable_on (g ‘ C) ∧ integral (g ‘ C) f = b
    ⟷ f absolutely_integrable_on (g ‘ S) ∧ integral (g ‘ S) f = b
  proof (rule conj_cong)
    have g differentiable_on N
    by (metis CNS der_g differentiable_def differentiable_on_def differentiable_on_subset
sup.cobounded2)
    with ‹negligible N›
    have neg_gN: negligible (g ‘ N)
      by (blast intro: negligible_differentiable_image_negligible)
    have neg: negligible {x ∈ g ‘ C − g ‘ S. f x ≠ 0}
              negligible {x ∈ g ‘ S − g ‘ C. f x ≠ 0}
      using CNS by (blast intro: negligible_subset [OF neg_gN])+
    then show (f absolutely_integrable_on g ‘ C) = (f absolutely_integrable_on g
‘ S)
      by (rule absolutely_integrable_spike_set_eq)
    show (integral (g ‘ C) f = b) ⟷ (integral (g ‘ S) f = b)
      using integral_spike_set [OF neg] by simp
  qed
  ultimately show ?thesis
    by simp
qed


corollary absolutely_integrable_change_of_variables:
  fixes f :: real^'m::{finite,wellorder} ⇒ real^'n and g :: real^'m::_ ⇒ real^'m::_
  assumes S ∈ sets lebesgue
```

```
    and ⋀x. x ∈ S ⟹ (g has_derivative g' x) (at x within S)
    and inj_on g S
  shows f absolutely_integrable_on (g ` S)
    ⟷ (λx. |det (matrix (g' x))| *R f(g x)) absolutely_integrable_on S
  using assms has_absolute_integral_change_of_variables by blast

corollary integral_change_of_variables:
  fixes f :: real^'m::{finite,wellorder} ⇒ real^'n and g :: real^'m::_ ⇒ real^'m::_
  assumes S: S ∈ sets lebesgue
    and der_g: ⋀x. x ∈ S ⟹ (g has_derivative g' x) (at x within S)
    and inj: inj_on g S
    and disj: (f absolutely_integrable_on (g ` S) ∨
        (λx. |det (matrix (g' x))| *R f(g x)) absolutely_integrable_on S)
  shows integral (g ` S) f = integral S (λx. |det (matrix (g' x))| *R f(g x))
  using has_absolute_integral_change_of_variables [OF S der_g inj] disj
  by blast

lemma has_absolute_integral_change_of_variables_1:
  fixes f :: real ⇒ real^'n::{finite,wellorder} and g :: real ⇒ real
  assumes S: S ∈ sets lebesgue
    and der_g: ⋀x. x ∈ S ⟹ (g has_vector_derivative g' x) (at x within S)
    and inj: inj_on g S
  shows (λx. |g' x| *R f(g x)) absolutely_integrable_on S ∧
        integral S (λx. |g' x| *R f(g x)) = b
    ⟷ f absolutely_integrable_on (g ` S) ∧ integral (g ` S) f = b
proof −
  let ?lift = vec :: real ⇒ real^1
  let ?drop = (λx::real^1. x $ 1)
  have S': ?lift ` S ∈ sets lebesgue
   by (auto intro: differentiable_image_in_sets_lebesgue [OF S] differentiable_vec)
  have ((λx. vec (g (x $ 1))) has_derivative (*R) (g' z)) (at (vec z) within ?lift `
S)
    if z ∈ S for z
    using der_g [OF that]
    by (simp add: has_vector_derivative_def has_derivative_vector_1)
  then have der': ⋀x. x ∈ ?lift ` S ⟹
      (?lift ∘ g ∘ ?drop has_derivative (*R) (g' (?drop x))) (at x within ?lift ` S)
    by (auto simp: o_def)
  have inj': inj_on (vec ∘ g ∘ ?drop) (vec ` S)
    using inj by (simp add: inj_on_def)
  let ?fg = λx. |g' x| *R f(g x)
  have ((λx. ?fg x $ i) absolutely_integrable_on S ∧ ((λx. ?fg x $ i) has_integral
b $ i) S
    ⟷ (λx. f x $ i) absolutely_integrable_on g ` S ∧ ((λx. f x $ i) has_integral
b $ i) (g ` S)) for i
     using has_absolute_integral_change_of_variables [OF S' der' inj', of λx.
?lift(f (?drop x) $ i) ?lift (b$i)]
   unfolding integrable_on_1_iff integral_on_1_eq absolutely_integrable_on_1_iff
absolutely_integrable_drop absolutely_integrable_on_def
```

by (*auto simp*: *image_comp o_def integral_vec1_eq has_integral_iff*)
then have *?fg absolutely_integrable_on S* ∧ (*?fg has_integral b*) *S*
⟷ *f absolutely_integrable_on* (*g ' S*) ∧ (*f has_integral b*) (*g ' S*)
unfolding *has_integral_componentwise_iff* [**where** *y*=*b*]
absolutely_integrable_componentwise_iff [**where** *f*=*f*]
absolutely_integrable_componentwise_iff [**where** *f* = *?fg*]
by (*force simp*: *Basis_vec_def cart_eq_inner_axis*)
then show *?thesis*
using *absolutely_integrable_on_def* **by** *blast*
qed

corollary *absolutely_integrable_change_of_variables_1*:
fixes *f* :: *real* ⇒ *real^'n*::{*finite*,*wellorder*} **and** *g* :: *real* ⇒ *real*
assumes *S*: *S* ∈ *sets lebesgue*
and *der_g*: ⋀*x*. *x* ∈ *S* ⟹ (*g has_vector_derivative g' x*) (*at x within S*)
and *inj*: *inj_on g S*
shows (*f absolutely_integrable_on g ' S* ⟷
(λ*x*. |*g' x*| *$_R$ f*(*g x*)) *absolutely_integrable_on S*)
using *has_absolute_integral_change_of_variables_1* [*OF assms*] **by** *auto*

when *n* = (*1*::*'a*)

lemma *has_absolute_integral_change_of_variables_1'*:
fixes *f* :: *real* ⇒ *real* **and** *g* :: *real* ⇒ *real*
assumes *S*: *S* ∈ *sets lebesgue*
and *der_g*: ⋀*x*. *x* ∈ *S* ⟹ (*g has_field_derivative g' x*) (*at x within S*)
and *inj*: *inj_on g S*
shows (λ*x*. |*g' x*| *$_R$ f*(*g x*)) *absolutely_integrable_on S* ∧
integral S (λ*x*. |*g' x*| *$_R$ f*(*g x*)) = *b*
⟷ *f absolutely_integrable_on* (*g ' S*) ∧ *integral* (*g ' S*) *f* = *b*
proof −
have (λ*x*. |*g' x*| *$_R$ vec* (*f*(*g x*)) :: *real ^ 1*) *absolutely_integrable_on S* ∧
integral S (λ*x*. |*g' x*| *$_R$ vec* (*f*(*g x*))) = (*vec b* :: *real ^ 1*)
⟷ (λ*x*. *vec* (*f x*) :: *real ^ 1*) *absolutely_integrable_on* (*g ' S*) ∧
integral (*g ' S*) (λ*x*. *vec* (*f x*)) = (*vec b* :: *real ^ 1*)
using *assms* **unfolding** *has_real_derivative_iff_has_vector_derivative*
by (*intro has_absolute_integral_change_of_variables_1 assms*) *auto*
thus *?thesis*
by (*simp add*: *absolutely_integrable_on_1_iff integral_on_1_eq*)
qed

6.41.6 Change of variables for integrals: special case of linear function

lemma *has_absolute_integral_change_of_variables_linear*:
fixes *f* :: *real^'m*::{*finite*,*wellorder*} ⇒ *real^'n* **and** *g* :: *real^'m*::_ ⇒ *real^'m*::_
assumes *linear g*
shows (λ*x*. |*det* (*matrix g*)| *$_R$ f*(*g x*)) *absolutely_integrable_on S* ∧
integral S (λ*x*. |*det* (*matrix g*)| *$_R$ f*(*g x*)) = *b*

```
    ⟷ f absolutely_integrable_on (g ‘ S) ∧ integral (g ‘ S) f = b
proof (cases det(matrix g) = 0)
  case True
  then have negligible(g ‘ S)
    using assms det_nz_iff_inj negligible_linear_singular_image by blast
  with True show ?thesis
   by (auto simp: absolutely_integrable_on_def integrable_negligible integral_negligible)
next
  case False
  then obtain h where h: ⋀x. x ∈ S ⟹ h (g x) = x linear h
    using assms det_nz_iff_inj linear_injective_isomorphism by metis
  show ?thesis
  proof (rule has_absolute_integral_change_of_variables_invertible)
    show (g has_derivative g) (at x within S) for x
      by (simp add: assms linear_imp_has_derivative)
    show continuous_on (g ‘ S) h
      using continuous_on_eq_continuous_within has_derivative_continuous lin-
ear_imp_has_derivative h by blast
  qed (use h in auto)
qed

lemma absolutely_integrable_change_of_variables_linear:
  fixes f :: real^'m::{finite,wellorder} ⇒ real^'n and g :: real^'m::_ ⇒ real^'m::_
  assumes linear g
  shows (λx. |det (matrix g)| *R f(g x)) absolutely_integrable_on S
    ⟷ f absolutely_integrable_on (g ‘ S)
  using assms has_absolute_integral_change_of_variables_linear by blast

lemma absolutely_integrable_on_linear_image:
  fixes f :: real^'m::{finite,wellorder} ⇒ real^'n and g :: real^'m::_ ⇒ real^'m::_
  assumes linear g
  shows f absolutely_integrable_on (g ‘ S)
    ⟷ (f ∘ g) absolutely_integrable_on S ∨ det(matrix g) = 0
  unfolding assms absolutely_integrable_change_of_variables_linear [OF assms,
symmetric] absolutely_integrable_on_scaleR_iff
  by (auto simp: set_integrable_def)

lemma integral_change_of_variables_linear:
  fixes f :: real^'m::{finite,wellorder} ⇒ real^'n and g :: real^'m::_ ⇒ real^'m::_
  assumes linear g
     and f absolutely_integrable_on (g ‘ S) ∨ (f ∘ g) absolutely_integrable_on S
    shows integral (g ‘ S) f = |det (matrix g)| *R integral S (f ∘ g)
proof −
  have ((λx. |det (matrix g)| *R f (g x)) absolutely_integrable_on S) ∨ (f abso-
lutely_integrable_on g ‘ S)
    using absolutely_integrable_on_linear_image assms by blast
  moreover
  have ?thesis if ((λx. |det (matrix g)| *R f (g x)) absolutely_integrable_on S) (f
absolutely_integrable_on g ‘ S)
```

using *has_absolute_integral_change_of_variables_linear* [*OF* ‹*linear g*›] *that*
by (*auto simp*: *o_def*)
ultimately show *?thesis*
using *absolutely_integrable_change_of_variables_linear* [*OF* ‹*linear g*›]
by *blast*
qed

6.41.7 Change of variable for measure

lemma *has_measure_differentiable_image*:
fixes *f* :: *real^'n*::{*finite*,*wellorder*} ⇒ *real^'n*::_
assumes *S* ∈ *sets lebesgue*
and ⋀*x*. *x* ∈ *S* ⟹ (*f has_derivative f' x*) (*at x within S*)
and *inj_on f S*
shows *f* ‘ *S* ∈ *lmeasurable* ∧ *measure lebesgue* (*f* ‘ *S*) = *m*
⟷ ((λ*x*. |*det* (*matrix* (*f' x*))|) *has_integral m*) *S*
using *has_absolute_integral_change_of_variables* [*OF assms*, *of* λ*x*. (*1*::*real^1*) *vec m*]
unfolding *absolutely_integrable_on_1_iff integral_on_1_eq integrable_on_1_iff absolutely_integrable_on_def*
by (*auto simp*: *has_integral_iff lmeasurable_iff_integrable_on lmeasure_integral*)

lemma *measurable_differentiable_image_eq*:
fixes *f* :: *real^'n*::{*finite*,*wellorder*} ⇒ *real^'n*::_
assumes *S* ∈ *sets lebesgue*
and ⋀*x*. *x* ∈ *S* ⟹ (*f has_derivative f' x*) (*at x within S*)
and *inj_on f S*
shows *f* ‘ *S* ∈ *lmeasurable* ⟷ (λ*x*. |*det* (*matrix* (*f' x*))|) *integrable_on S*
using *has_measure_differentiable_image* [*OF assms*]
by *blast*

lemma *measurable_differentiable_image_alt*:
fixes *f* :: *real^'n*::{*finite*,*wellorder*} ⇒ *real^'n*::_
assumes *S* ∈ *sets lebesgue*
and ⋀*x*. *x* ∈ *S* ⟹ (*f has_derivative f' x*) (*at x within S*)
and *inj_on f S*
shows *f* ‘ *S* ∈ *lmeasurable* ⟷ (λ*x*. |*det* (*matrix* (*f' x*))|) *absolutely_integrable_on S*
using *measurable_differentiable_image_eq* [*OF assms*]
by (*simp only*: *absolutely_integrable_on_iff_nonneg*)

lemma *measure_differentiable_image_eq*:
fixes *f* :: *real^'n*::{*finite*,*wellorder*} ⇒ *real^'n*::_
assumes *S*: *S* ∈ *sets lebesgue*
and *der_f*: ⋀*x*. *x* ∈ *S* ⟹ (*f has_derivative f' x*) (*at x within S*)
and *inj*: *inj_on f S*
and *intS*: (λ*x*. |*det* (*matrix* (*f' x*))|) *integrable_on S*
shows *measure lebesgue* (*f* ‘ *S*) = *integral S* (λ*x*. |*det* (*matrix* (*f' x*))|)
using *measurable_differentiable_image_eq* [*OF S der_f inj*]

assms has_measure_differentiable_image **by** *blast*

end

6.42 Lipschitz Continuity

theory *Lipschitz*
imports
Derivative Abstract_Metric_Spaces
begin

definition *lipschitz_on*
where *lipschitz_on C U f* ⟷ (*0* ≤ *C* ∧ (∀ *x* ∈ *U*. ∀ *y*∈*U*. *dist* (*f x*) (*f y*) ≤ *C* ∗ *dist x y*))

bundle *lipschitz_syntax* **begin**
notation *lipschitz_on* (*_−lipschitz′_on* [*1000*])
end
bundle *no_lipschitz_syntax* **begin**
no_notation *lipschitz_on* (*_−lipschitz′_on* [*1000*])
end

unbundle *lipschitz_syntax*

lemma *lipschitz_onI*: *L−lipschitz_on X f*
if ⋀*x y*. *x* ∈ *X* ⟹ *y* ∈ *X* ⟹ *dist* (*f x*) (*f y*) ≤ *L* ∗ *dist x y* *0* ≤ *L*
using *that* **by** (*auto simp*: *lipschitz_on_def*)

lemma *lipschitz_onD*:
dist (*f x*) (*f y*) ≤ *L* ∗ *dist x y*
if *L−lipschitz_on X f x* ∈ *X y* ∈ *X*
using *that* **by** (*auto simp*: *lipschitz_on_def*)

lemma *lipschitz_on_nonneg*:
0 ≤ *L* **if** *L−lipschitz_on X f*
using *that* **by** (*auto simp*: *lipschitz_on_def*)

lemma *lipschitz_on_normD*:
norm (*f x* − *f y*) ≤ *L* ∗ *norm* (*x* − *y*)
if *lipschitz_on L X f x* ∈ *X y* ∈ *X*
using *lipschitz_onD*[*OF that*]
by (*simp add*: *dist_norm*)

lemma *lipschitz_on_mono*: *L−lipschitz_on D f* **if** *M−lipschitz_on E f D* ⊆ *E M* ≤ *L*
using *that*
by (*force simp*: *lipschitz_on_def intro*: *order_trans*[*OF _ mult_right_mono*])

lemmas *lipschitz_on_subset* = *lipschitz_on_mono*[*OF _ _ order_refl*]

and *lipschitz_on_le* = *lipschitz_on_mono*[*OF* _ *order_refl*]

lemma *lipschitz_on_leI*:
L−*lipschitz_on X f*
if ⋀*x y*. *x* ∈ *X* ⟹ *y* ∈ *X* ⟹ *x* ≤ *y* ⟹ *dist* (*f x*) (*f y*) ≤ *L* ∗ *dist x y*
0 ≤ *L*
for *f*::′*a*::{*linorder_topology*, *ordered_real_vector*, *metric_space*} ⇒ ′*b*::*metric_space*
proof (*rule lipschitz_onI*)
fix *x y* **assume** *xy*: *x* ∈ *X y* ∈ *X*
consider *y* ≤ *x* | *x* ≤ *y*
by (*rule le_cases*)
then show *dist* (*f x*) (*f y*) ≤ *L* ∗ *dist x y*
proof *cases*
case *1*
then have *dist* (*f y*) (*f x*) ≤ *L* ∗ *dist y x*
by (*auto intro*!: *that xy*)
then show *?thesis*
by (*simp add*: *dist_commute*)
qed (*auto intro*!: *that xy*)
qed *fact*

lemma *lipschitz_on_concat*:
fixes *a b c*::*real*
assumes *f*: *L*−*lipschitz_on* {*a* .. *b*} *f*
assumes *g*: *L*−*lipschitz_on* {*b* .. *c*} *g*
assumes *fg*: *f b* = *g b*
shows *lipschitz_on L* {*a* .. *c*} (λ*x*. *if x* ≤ *b then f x else g x*)
(**is** *lipschitz_on* _ _ *?f*)
proof (*rule lipschitz_on_leI*)
fix *x y*
assume *x*: *x* ∈ {*a*..*c*} **and** *y*: *y* ∈ {*a*..*c*} **and** *xy*: *x* ≤ *y*
consider *x* ≤ *b* ∧ *b* < *y* | *x* ≥ *b* ∨ *y* ≤ *b* **by** *arith*
then show *dist* (*?f x*) (*?f y*) ≤ *L* ∗ *dist x y*
proof *cases*
case *1*
have *dist* (*f x*) (*g y*) ≤ *dist* (*f x*) (*f b*) + *dist* (*g b*) (*g y*)
unfolding *fg* **by** (*rule dist_triangle*)
also have *dist* (*f x*) (*f b*) ≤ *L* ∗ *dist x b*
using *1 x*
by (*auto intro*!: *lipschitz_onD*[*OF f*])
also have *dist* (*g b*) (*g y*) ≤ *L* ∗ *dist b y*
using *1 x y*
by (*auto intro*!: *lipschitz_onD*[*OF g*] *lipschitz_onD*[*OF f*])
finally have *dist* (*f x*) (*g y*) ≤ *L* ∗ *dist x b* + *L* ∗ *dist b y*
by *simp*
also have ... = *L* ∗ (*dist x b* + *dist b y*)
by (*simp add*: *algebra_simps*)
also have *dist x b* + *dist b y* = *dist x y*
using *1 x y*

```
    by (auto simp: dist_real_def abs_real_def)
  finally show ?thesis
    using 1 by simp
 next
  case 2
  with lipschitz_onD[OF f, of x y] lipschitz_onD[OF g, of x y] x y xy
  show ?thesis
    by (auto simp: fg)
 qed
qed (rule lipschitz_on_nonneg[OF f])

lemma lipschitz_on_concat_max:
 fixes a b c::real
 assumes f: L−lipschitz_on {a .. b} f
 assumes g: M−lipschitz_on {b .. c} g
 assumes fg: f b = g b
 shows (max L M)−lipschitz_on {a .. c} (λx. if x ≤ b then f x else g x)
proof −
 have lipschitz_on (max L M) {a .. b} f lipschitz_on (max L M) {b .. c} g
  by (auto intro!: lipschitz_on_mono[OF f order_refl] lipschitz_on_mono[OF g
order_refl])
 from lipschitz_on_concat[OF this fg] show ?thesis .
qed
```

Equivalence between "abstract" and "type class" Lipschitz notions, for all types in the metric space class

```
lemma Lipschitz_map_euclidean [simp]:
 Lipschitz_continuous_map euclidean_metric euclidean_metric f
   ⟷ (∃ B. lipschitz_on B UNIV f)  (is ?lhs ⟷ ?rhs)
proof
 show ?lhs ⟹ ?rhs
  by (force simp add: Lipschitz_continuous_map_pos lipschitz_on_def less_le_not_le)
 show ?rhs ⟹ ?lhs
  by (auto simp: Lipschitz_continuous_map_def lipschitz_on_def)
qed
```

Continuity

```
proposition lipschitz_on_uniformly_continuous:
 assumes L−lipschitz_on X f
 shows uniformly_continuous_on X f
 unfolding uniformly_continuous_on_def
proof safe
 fix e::real
 assume 0 < e
 from assms have l: (L+1)−lipschitz_on X f
  by (rule lipschitz_on_mono) auto
 show ∃ d>0. ∀ x∈X. ∀ x′∈X. dist x′ x < d ⟶ dist (f x′) (f x) < e
  using lipschitz_onD[OF l] lipschitz_on_nonneg[OF assms] ‹0 < e›
```

by (*force intro*!: *exI*[**where** *x*=*e*/(*L* + *1*)] *simp*: *field_simps*)
qed

proposition *lipschitz_on_continuous_on*:
continuous_on X f **if** *L*−*lipschitz_on X f*
by (*rule uniformly_continuous_imp_continuous*[*OF lipschitz_on_uniformly_continuous*[*OF that*]])

lemma *lipschitz_on_continuous_within*:
continuous (*at x within X*) *f* **if** *L*−*lipschitz_on X f x* ∈ *X*
using *lipschitz_on_continuous_on*[*OF that*(*1*)] *that*(*2*)
by (*auto simp*: *continuous_on_eq_continuous_within*)

Differentiable functions

proposition *bounded_derivative_imp_lipschitz*:
assumes ⋀*x*. *x* ∈ *X* ⟹ (*f has_derivative f*′ *x*) (*at x within X*)
assumes *convex*: *convex X*
assumes ⋀*x*. *x* ∈ *X* ⟹ *onorm* (*f*′ *x*) ≤ *C* *0* ≤ *C*
shows *C*−*lipschitz_on X f*
proof (*rule lipschitz_onI*)
show ⋀*x y*. *x* ∈ *X* ⟹ *y* ∈ *X* ⟹ *dist* (*f x*) (*f y*) ≤ *C* ∗ *dist x y*
by (*auto intro*!: *assms differentiable_bound*[*unfolded dist_norm*[*symmetric*], *OF convex*])
qed *fact*

Structural introduction rules

named_theorems *lipschitz_intros structural introduction rules for Lipschitz controls*

lemma *lipschitz_on_compose* [*lipschitz_intros*]:
(*D* ∗ *C*)−*lipschitz_on U* (*g o f*)
if *f*: *C*−*lipschitz_on U f* **and** *g*: *D*−*lipschitz_on* (*f*‘*U*) *g*
proof (*rule lipschitz_onI*)
show *D*∗ *C* ≥ *0* **using** *lipschitz_on_nonneg*[*OF f*] *lipschitz_on_nonneg*[*OF g*]
by *auto*
fix *x y* **assume** *H*: *x* ∈ *U y* ∈ *U*
have *dist* (*g* (*f x*)) (*g* (*f y*)) ≤ *D* ∗ *dist* (*f x*) (*f y*)
apply (*rule lipschitz_onD*[*OF g*]) **using** *H* **by** *auto*
also have ... ≤ *D* ∗ *C* ∗ *dist x y*
using *mult_left_mono*[*OF lipschitz_onD*(*1*)[*OF f H*] *lipschitz_on_nonneg*[*OF g*]] **by** *auto*
finally show *dist* ((*g* ∘ *f*) *x*) ((*g* ∘ *f*) *y*) ≤ *D* ∗ *C*∗ *dist x y*
unfolding *comp_def* **by** (*auto simp add*: *mult.commute*)
qed

lemma *lipschitz_on_compose2*:
(*D* ∗ *C*)−*lipschitz_on U* (λ*x*. *g* (*f x*))
if *C*−*lipschitz_on U f D*−*lipschitz_on* (*f*‘*U*) *g*

using *lipschitz_on_compose*[*OF that*] **by** (*simp add*: *o_def*)

lemma *lipschitz_on_cong*[*cong*]:
C−*lipschitz_on U g* ⟷ *D*−*lipschitz_on V f*
if *C* = *D U* = *V* ⋀*x*. *x* ∈ *V* ⟹ *g x* = *f x*
using *that* **by** (*auto simp*: *lipschitz_on_def*)

lemma *lipschitz_on_transform*:
C−*lipschitz_on U g*
if *C*−*lipschitz_on U f*
⋀*x*. *x* ∈ *U* ⟹ *g x* = *f x*
using *that*
by *simp*

lemma *lipschitz_on_empty_iff*[*simp*]: *C*−*lipschitz_on* {} *f* ⟷ *C* ≥ *0*
by (*auto simp*: *lipschitz_on_def*)

lemma *lipschitz_on_insert_iff*[*simp*]:
C−*lipschitz_on* (*insert y X*) *f* ⟷
C−*lipschitz_on X f* ∧ (∀ *x* ∈ *X*. *dist* (*f x*) (*f y*) ≤ *C* ∗ *dist x y*)
by (*auto simp*: *lipschitz_on_def dist_commute*)

lemma *lipschitz_on_singleton* [*lipschitz_intros*]: *C* ≥ *0* ⟹ *C*−*lipschitz_on* {*x*} *f*
and *lipschitz_on_empty* [*lipschitz_intros*]: *C* ≥ *0* ⟹ *C*−*lipschitz_on* {} *f*
by *simp_all*

lemma *lipschitz_on_id* [*lipschitz_intros*]: *1*−*lipschitz_on U* (λ*x*. *x*)
by (*auto simp*: *lipschitz_on_def*)

lemma *lipschitz_on_constant* [*lipschitz_intros*]: *0*−*lipschitz_on U* (λ*x*. *c*)
by (*auto simp*: *lipschitz_on_def*)

lemma *lipschitz_on_add* [*lipschitz_intros*]:
fixes *f*::′*a*::*metric_space* ⇒′*b*::*real_normed_vector*
assumes *C*−*lipschitz_on U f*
D−*lipschitz_on U g*
shows (*C*+*D*)−*lipschitz_on U* (λ*x*. *f x* + *g x*)
proof (*rule lipschitz_onI*)
show *C* + *D* ≥ *0*
using *lipschitz_on_nonneg*[*OF assms*(*1*)] *lipschitz_on_nonneg*[*OF assms*(*2*)]
by *auto*
fix *x y* **assume** *H*: *x* ∈ *U y* ∈ *U*
have *dist* (*f x* + *g x*) (*f y* + *g y*) ≤ *dist* (*f x*) (*f y*) + *dist* (*g x*) (*g y*)
by (*simp add*: *dist_triangle_add*)
also have ... ≤ *C* ∗ *dist x y* + *D* ∗ *dist x y*
using *lipschitz_onD*(*1*)[*OF assms*(*1*) *H*] *lipschitz_onD*(*1*)[*OF assms*(*2*) *H*] **by** *auto*
finally show *dist* (*f x* + *g x*) (*f y* + *g y*) ≤ (*C*+*D*) ∗ *dist x y* **by** (*auto simp*

add: *algebra_simps*)
qed

lemma *lipschitz_on_cmult* [*lipschitz_intros*]:
fixes *f*::'*a*::*metric_space* ⇒ '*b*::*real_normed_vector*
assumes *C*−*lipschitz_on U f*
shows (*abs*(*a*) ∗ *C*)−*lipschitz_on U* (λ*x*. *a* ∗$_R$ *f x*)
proof (*rule lipschitz_onI*)
show *abs*(*a*) ∗ *C* ≥ *0* **using** *lipschitz_on_nonneg*[*OF assms*(*1*)] **by** *auto*
fix *x y* **assume** *H*: *x* ∈ *U y* ∈ *U*
have *dist* (*a* ∗$_R$ *f x*) (*a* ∗$_R$ *f y*) = *abs*(*a*) ∗ *dist* (*f x*) (*f y*)
by (*metis dist_norm norm_scaleR real_vector.scale_right_diff_distrib*)
also have ... ≤ *abs*(*a*) ∗ *C* ∗ *dist x y*
using *lipschitz_onD*(*1*)[*OF assms*(*1*) *H*] **by** (*simp add*: *Groups.mult_ac*(*1*) *mult_left_mono*)
finally show *dist* (*a* ∗$_R$ *f x*) (*a* ∗$_R$ *f y*) ≤ |*a*| ∗ *C* ∗ *dist x y* **by** *auto*
qed

lemma *lipschitz_on_cmult_real* [*lipschitz_intros*]:
fixes *f*::'*a*::*metric_space* ⇒ *real*
assumes *C*−*lipschitz_on U f*
shows (*abs*(*a*) ∗ *C*)−*lipschitz_on U* (λ*x*. *a* ∗ *f x*)
using *lipschitz_on_cmult*[*OF assms*] **by** *auto*

lemma *lipschitz_on_cmult_nonneg* [*lipschitz_intros*]:
fixes *f*::'*a*::*metric_space* ⇒ '*b*::*real_normed_vector*
assumes *C*−*lipschitz_on U f*
a ≥ *0*
shows (*a* ∗ *C*)−*lipschitz_on U* (λ*x*. *a* ∗$_R$ *f x*)
using *lipschitz_on_cmult*[*OF assms*(*1*), *of a*] *assms*(*2*) **by** *auto*

lemma *lipschitz_on_cmult_real_nonneg* [*lipschitz_intros*]:
fixes *f*::'*a*::*metric_space* ⇒ *real*
assumes *C*−*lipschitz_on U f*
a ≥ *0*
shows (*a* ∗ *C*)−*lipschitz_on U* (λ*x*. *a* ∗ *f x*)
using *lipschitz_on_cmult_nonneg*[*OF assms*] **by** *auto*

lemma *lipschitz_on_cmult_upper* [*lipschitz_intros*]:
fixes *f*::'*a*::*metric_space* ⇒ '*b*::*real_normed_vector*
assumes *C*−*lipschitz_on U f*
abs(*a*) ≤ *D*
shows (*D* ∗ *C*)−*lipschitz_on U* (λ*x*. *a* ∗$_R$ *f x*)
apply (*rule lipschitz_on_mono*[*OF lipschitz_on_cmult*[*OF assms*(*1*), *of a*], *of* _ *D* ∗ *C*])
using *assms*(*2*) *lipschitz_on_nonneg*[*OF assms*(*1*)] *mult_right_mono* **by** *auto*

lemma *lipschitz_on_cmult_real_upper* [*lipschitz_intros*]:
fixes *f*::'*a*::*metric_space* ⇒ *real*

assumes $C-lipschitz\_on\ U\ f$
$abs(a) \leq D$
shows $(D * C)-lipschitz\_on\ U\ (\lambda x.\ a * f\ x)$
using *lipschitz_on_cmult_upper*[*OF assms*] **by** *auto*

lemma *lipschitz_on_minus*[*lipschitz_intros*]:
fixes $f::'a::metric\_space \Rightarrow 'b::real\_normed\_vector$
assumes $C-lipschitz\_on\ U\ f$
shows $C-lipschitz\_on\ U\ (\lambda x.\ -\ f\ x)$
by (*metis* (*mono_tags*, *lifting*) *assms dist_minus lipschitz_on_def*)

lemma *lipschitz_on_minus_iff*[*simp*]:
$L-lipschitz\_on\ X\ (\lambda x.\ -\ f\ x) \longleftrightarrow L-lipschitz\_on\ X\ f$
$L-lipschitz\_on\ X\ (-\ f) \longleftrightarrow L-lipschitz\_on\ X\ f$
for $f::'a::metric\_space \Rightarrow 'b::real\_normed\_vector$
using *lipschitz_on_minus*[*of L X f*] *lipschitz_on_minus*[*of L X* $-f$]
by *auto*

lemma *lipschitz_on_diff*[*lipschitz_intros*]:
fixes $f::'a::metric\_space \Rightarrow 'b::real\_normed\_vector$
assumes $C-lipschitz\_on\ U\ f\ D-lipschitz\_on\ U\ g$
shows $(C + D)-lipschitz\_on\ U\ (\lambda x.\ f\ x - g\ x)$
using *lipschitz_on_add*[*OF assms*(*1*) *lipschitz_on_minus*[*OF assms*(*2*)]] **by** *auto*

lemma *lipschitz_on_closure* [*lipschitz_intros*]:
assumes $C-lipschitz\_on\ U\ f\ continuous\_on\ (closure\ U)\ f$
shows $C-lipschitz\_on\ (closure\ U)\ f$
proof (*rule lipschitz_onI*)
show $C \geq 0$ **using** *lipschitz_on_nonneg*[*OF assms*(*1*)] **by** *simp*
fix $x\ y$ **assume** $x \in closure\ U\ y \in closure\ U$
obtain $u\ v::nat \Rightarrow 'a$ **where** $*$: $\bigwedge n.\ u\ n \in U\ u \longrightarrow x$
$\bigwedge n.\ v\ n \in U\ v \longrightarrow y$
using ‹$x \in closure\ U$› ‹$y \in closure\ U$› **unfolding** *closure_sequential* **by** *blast*
have *a*: $(\lambda n.\ f\ (u\ n)) \longrightarrow f\ x$
using $*(1)$ $*(2)$ ‹$x \in closure\ U$› ‹$continuous\_on\ (closure\ U)\ f$›
unfolding *comp_def continuous_on_closure_sequentially*[*of U f*] **by** *auto*
have *b*: $(\lambda n.\ f\ (v\ n)) \longrightarrow f\ y$
using $*(3)$ $*(4)$ ‹$y \in closure\ U$› ‹$continuous\_on\ (closure\ U)\ f$›
unfolding *comp_def continuous_on_closure_sequentially*[*of U f*] **by** *auto*
have *l*: $(\lambda n.\ C * dist\ (u\ n)\ (v\ n) - dist\ (f\ (u\ n))\ (f\ (v\ n))) \longrightarrow C * dist\ x\ y - dist\ (f\ x)\ (f\ y)$
by (*intro tendsto_intros* $*$ *a b*)
have $C * dist\ (u\ n)\ (v\ n) - dist\ (f\ (u\ n))\ (f\ (v\ n)) \geq 0$ **for** n
using *lipschitz_onD*(*1*)[*OF assms*(*1*) ‹$u\ n \in U$› ‹$v\ n \in U$›] **by** *simp*
then have $C * dist\ x\ y - dist\ (f\ x)\ (f\ y) \geq 0$ **using** *LIMSEQ_le_const*[*OF l*, *of 0*] **by** *auto*
then show $dist\ (f\ x)\ (f\ y) \leq C * dist\ x\ y$ **by** *auto*
qed

lemma *lipschitz_on_Pair*[*lipschitz_intros*]:
assumes *f*: *L*−*lipschitz_on A f*
assumes *g*: *M*−*lipschitz_on A g*
shows (*sqrt* (*L*^{2} + *M*^{2}))−*lipschitz_on A* (λ*a*. (*f a*, *g a*))
proof (*rule lipschitz_onI*, *goal_cases*)
case (*1 x y*)
have *dist* (*f x*, *g x*) (*f y*, *g y*) = *sqrt* ((*dist* (*f x*) (*f y*))^2 + (*dist* (*g x*) (*g y*))^2)
by (*auto simp add*: *dist_Pair_Pair real_le_lsqrt*)
also have ... ≤ *sqrt* ((*L* ∗ *dist x y*)^2 + (*M* ∗ *dist x y*)^2)
by (*auto intro*!: *real_sqrt_le_mono add_mono power_mono 1 lipschitz_onD f g*)
also have ... ≤ *sqrt* (*L*^{2} + *M*^{2}) ∗ *dist x y*
by (*auto simp*: *power_mult_distrib ring_distribs*[*symmetric*] *real_sqrt_mult*)
finally show *?case* **.**
qed *simp*

lemma *lipschitz_extend_closure*:
fixes *f*::('*a*::*metric_space*) ⇒ ('*b*::*complete_space*)
assumes *C*−*lipschitz_on U f*
shows ∃ *g*. *C*−*lipschitz_on* (*closure U*) *g* ∧ (∀ *x*∈*U*. *g x* = *f x*)
proof −
obtain *g* **where** *g*: ⋀*x*. *x* ∈ *U* ⟹ *g x* = *f x uniformly_continuous_on* (*closure U*) *g*
using *uniformly_continuous_on_extension_on_closure*[*OF lipschitz_on_uniformly_continuous*[*OF assms*]] **by** *metis*
have *C*−*lipschitz_on* (*closure U*) *g*
apply (*rule lipschitz_on_closure*, *rule lipschitz_on_transform*[*OF assms*])
using *g uniformly_continuous_imp_continuous*[*OF g*(*2*)] **by** *auto*
then show *?thesis* **using** *g*(*1*) **by** *auto*
qed

lemma (**in** *bounded_linear*) *lipschitz_boundE*:
obtains *B* **where** *B*−*lipschitz_on A f*
proof −
from *nonneg_bounded*
obtain *B* **where** *B*: *B* ≥ *0* ⋀*x*. *norm* (*f x*) ≤ *B* ∗ *norm x*
by (*auto simp*: *ac_simps*)
have *B*−*lipschitz_on A f*
by (*auto intro*!: *lipschitz_onI B simp*: *dist_norm diff*[*symmetric*])
thus *?thesis* **..**
qed

6.42.1 Local Lipschitz continuity

Given a function defined on a real interval, it is Lipschitz-continuous if and only if it is locally so, as proved in the following lemmas. It is useful especially for piecewise-defined functions: if each piece is Lipschitz, then so is the whole function. The same goes for functions defined on geodesic spaces,

or more generally on geodesic subsets in a metric space (for instance convex subsets in a real vector space), and this follows readily from the real case, but we will not prove it explicitly.

We give several variations around this statement. This is essentially a connectedness argument.

lemma *locally_lipschitz_imp_lipschitz_aux*:
fixes f::*real* ⇒ ('*a*::*metric_space*)
assumes $a \leq b$
continuous_on $\{a..b\}$ f
$\bigwedge x.\ x \in \{a..<b\} \Longrightarrow \exists\, y \in \{x<..b\}.$ *dist* $(f\ y)\ (f\ x) \leq M * (y-x)$
shows *dist* $(f\ b)\ (f\ a) \leq M * (b-a)$
proof −
define A **where** $A = \{x \in \{a..b\}.$ *dist* $(f\ x)\ (f\ a) \leq M * (x-a)\}$
have ∗: $A = (\lambda x.\ M * (x-a) -$ *dist* $(f\ x)\ (f\ a))-`\{0..\} \cap \{a..b\}$
unfolding *A_def* **by** *auto*
have $a \in A$ **unfolding** *A_def* **using** ‹$a \leq b$› **by** *auto*
then have $A \neq \{\}$ **by** *auto*
moreover have *bdd_above A* **unfolding** *A_def* **by** *auto*
moreover have *closed A* **unfolding** ∗ **by** (*rule closed_vimage_Int*, *auto intro*!: *continuous_intros assms*)
ultimately have *Sup A* ∈ *A* **by** (*rule closed_contains_Sup*)
have *Sup A* = *b*
proof (*rule ccontr*)
assume *Sup A* ≠ *b*
define *x* **where** *x* = *Sup A*
have *I*: *dist* $(f\ x)\ (f\ a) \leq M * (x-a)$ **using** ‹*Sup A* ∈ *A*› *x_def A_def* **by** *auto*
have $x \in \{a..<b\}$ **unfolding** *x_def* **using** ‹*Sup A* ∈ *A*› ‹*Sup A* ≠ *b*› *A_def* **by** *auto*
then obtain *y* **where** *J*: $y \in \{x<..b\}$ *dist* $(f\ y)\ (f\ x) \leq M * (y-x)$ **using** *assms*(*3*) **by** *blast*
have *dist* $(f\ y)\ (f\ a) \leq$ *dist* $(f\ y)\ (f\ x) +$ *dist* $(f\ x)\ (f\ a)$ **by** (*rule dist_triangle*)
also have ... $\leq M * (y-x) + M * (x-a)$ **using** *I J*(*2*) **by** *auto*
finally have *dist* $(f\ y)\ (f\ a) \leq M * (y-a)$ **by** (*auto simp add*: *algebra_simps*)
then have $y \in A$ **unfolding** *A_def* **using** ‹$y \in \{x<..b\}$› ‹$x \in \{a..<b\}$› **by** *auto*
then have $y \leq$ *Sup A* **by** (*rule cSup_upper*, *auto simp*: *A_def*)
then show *False* **using** ‹$y \in \{x<..b\}$› *x_def* **by** *auto*
qed
then show *?thesis* **using** ‹*Sup A* ∈ *A*› *A_def* **by** *auto*
qed

lemma *locally_lipschitz_imp_lipschitz*:
fixes f::*real* ⇒ ('*a*::*metric_space*)
assumes *continuous_on* $\{a..b\}$ f
$\bigwedge x\ y.\ x \in \{a..<b\} \Longrightarrow y > x \Longrightarrow \exists\, z \in \{x<..y\}.$ *dist* $(f\ z)\ (f\ x) \leq M * (z-x)$
$M \geq 0$
shows *lipschitz_on M* $\{a..b\}$ f

proof (*rule lipschitz_onI*[*OF* _ ‹$M \geq 0$›])
have ∗: *dist* (*f t*) (*f s*) ≤ *M* ∗ (*t*−*s*) **if** *s* ≤ *t s* ∈ {*a*..*b*} *t* ∈ {*a*..*b*} **for** *s t*
proof (*rule locally_lipschitz_imp_lipschitz_aux*, *simp add*: ‹*s* ≤ *t*›)
show *continuous_on* {*s*..*t*} *f* **using** *continuous_on_subset*[*OF assms*(*1*)] *that*
by *auto*
fix *x* **assume** *x* ∈ {*s*..<*t*}
then have *x* ∈ {*a*..<*b*} **using** *that* **by** *auto*
show ∃ *z*∈{*x*<..*t*}. *dist* (*f z*) (*f x*) ≤ *M* ∗ (*z* − *x*)
using *assms*(*2*)[*OF* ‹*x* ∈ {*a*..<*b*}›, *of t*] ‹*x* ∈ {*s*..<*t*}› **by** *auto*
qed
fix *x y* **assume** *x* ∈ {*a*..*b*} *y* ∈ {*a*..*b*}
consider *x* ≤ *y* | *y* ≤ *x* **by** *linarith*
then show *dist* (*f x*) (*f y*) ≤ *M* ∗ *dist x y*
apply (*cases*)
using ∗[*OF* _ ‹*x* ∈ {*a*..*b*}› ‹*y* ∈ {*a*..*b*}›] ∗[*OF* _ ‹*y* ∈ {*a*..*b*}› ‹*x* ∈ {*a*..*b*}›]
by (*auto simp add*: *dist_commute dist_real_def*)
qed

We deduce that if a function is Lipschitz on finitely many closed sets on the real line, then it is Lipschitz on any interval contained in their union. The difficulty in the proof is to show that any point z in this interval (except the maximum) has a point arbitrarily close to it on its right which is contained in a common initial closed set. Otherwise, we show that there is a small interval $(z,\ T)$ which does not intersect any of the initial closed sets, a contradiction.

proposition *lipschitz_on_closed_Union*:
assumes ⋀*i*. *i* ∈ *I* ⟹ *lipschitz_on M* (*U i*) *f*
⋀*i*. *i* ∈ *I* ⟹ *closed* (*U i*)
finite I
M ≥ *0*
{*u*..(*v*::*real*)} ⊆ (⋃ *i*∈*I*. *U i*)
shows *lipschitz_on M* {*u*..*v*} *f*
proof (*rule locally_lipschitz_imp_lipschitz*[*OF* _ _ ‹*M* ≥ *0*›])
have ∗: *continuous_on* (*U i*) *f* **if** *i* ∈ *I* **for** *i*
by (*rule lipschitz_on_continuous_on*[*OF assms*(*1*)[*OF* ‹*i*∈ *I*›]])
have *continuous_on* (⋃ *i*∈*I*. *U i*) *f*
apply (*rule continuous_on_closed_Union*) **using** ‹*finite I*› ∗ *assms*(*2*) **by** *auto*
then show *continuous_on* {*u*..*v*} *f*
using ‹{*u*..(*v*::*real*)} ⊆ (⋃ *i*∈*I*. *U i*)› *continuous_on_subset* **by** *auto*

fix *z Z* **assume** *z*: *z* ∈ {*u*..<*v*} *z* < *Z*
then have *u* ≤ *v* **by** *auto*
define *T* **where** *T* = *min Z v*
then have *T*: *T* > *z T* ≤ *v T* ≥ *u T* ≤ *Z* **using** *z* **by** *auto*
define *A* **where** *A* = (⋃ *i*∈ *I* ∩ {*i*. *U i* ∩ {*z*<..*T*} ≠ {}}. *U i* ∩ {*z*..*T*})
have *a*: *closed A*
unfolding *A_def* **apply** (*rule closed_UN*) **using** ‹*finite I*› ‹⋀*i*. *i* ∈ *I* ⟹ *closed* (*U i*)› **by** *auto*

have *b*: *bdd_below A* **unfolding** *A_def* **using** ‹*finite I*› **by** *auto*
have $\exists i \in I.\ T \in U\ i$ **using** ‹$\{u..v\} \subseteq (\bigcup i{\in}I.\ U\ i)$› T **by** *auto*
then have *c*: $T \in A$ **unfolding** *A_def* **using** T **by** (*auto*, *fastforce*)
have $Inf\ A \geq z$
apply (*rule cInf_greatest*, *auto*) **using** *c* **unfolding** *A_def* **by** *auto*
moreover have $Inf\ A \leq z$
proof (*rule ccontr*)
assume $\neg(Inf\ A \leq z)$
then obtain *w* **where** *w*: $w > z\ w < Inf\ A$ **by** (*meson dense not_le_imp_less*)
have $Inf\ A \leq T$ **using** *a b c* **by** (*simp add*: *cInf_lower*)
then have $w \leq T$ **using** *w* **by** *auto*
then have $w \in \{u..v\}$ **using** *w* ‹$z \in \{u..<v\}$› T **by** *auto*
then obtain *j* **where** *j*: $j \in I\ w \in U\ j$ **using** ‹$\{u..v\} \subseteq (\bigcup i{\in}I.\ U\ i)$› **by** *fastforce*
then have $w \in U\ j \cap \{z..T\}\ U\ j \cap \{z<..T\} \neq \{\}$ **using** *j T w* ‹$w \leq T$› **by** *auto*
then have $w \in A$ **unfolding** *A_def* **using** ‹$j \in I$› **by** *auto*
then have $Inf\ A \leq w$ **using** *a b* **by** (*simp add*: *cInf_lower*)
then show *False* **using** *w* **by** *auto*
qed
ultimately have $Inf\ A = z$ **by** *simp*
moreover have $Inf\ A \in A$
apply (*rule closed_contains_Inf*) **using** *a b c* **by** *auto*
ultimately have $z \in A$ **by** *simp*
then obtain *i* **where** *i*: $i \in I\ U\ i \cap \{z<..T\} \neq \{\}\ z \in U\ i$ **unfolding** *A_def* **by** *auto*
then obtain *t* **where** $t \in U\ i \cap \{z<..T\}$ **by** *blast*
then have $dist\ (f\ t)\ (f\ z) \leq M * (t - z)$
using *lipschitz_onD*(1)[*OF assms*(1)[*of i*], *of t z*] *i dist_real_def* **by** *auto*
then show $\exists t{\in}\{z<..Z\}.\ dist\ (f\ t)\ (f\ z) \leq M * (t - z)$ **using** ‹$T \leq Z$› ‹$t \in U\ i \cap \{z<..T\}$› **by** *auto*
qed

6.42.2 Local Lipschitz continuity (uniform for a family of functions)

definition *local_lipschitz*::
$'a$::*metric_space set* ⇒ $'b$::*metric_space set* ⇒ ($'a$ ⇒ $'b$ ⇒ $'c$::*metric_space*) ⇒ *bool*
where
local_lipschitz T X f ≡ $\forall x \in X.\ \forall t \in T.$
$\exists u{>}0.\ \exists L.\ \forall t \in cball\ t\ u \cap T.\ L{-}lipschitz\_on\ (cball\ x\ u \cap X)\ (f\ t)$

lemma *local_lipschitzI*:
assumes $\bigwedge t\ x.\ t \in T \Longrightarrow x \in X \Longrightarrow \exists u{>}0.\ \exists L.\ \forall t \in cball\ t\ u \cap T.\ L{-}lipschitz\_on\ (cball\ x\ u \cap X)\ (f\ t)$
shows *local_lipschitz T X f*
using *assms*
unfolding *local_lipschitz_def*

by *auto*

lemma *local_lipschitzE*:
assumes *local_lipschitz*: *local_lipschitz T X f*
assumes $t \in T\ x \in X$
obtains *u L* **where** $u > 0\ \bigwedge s.\ s \in cball\ t\ u \cap T \Longrightarrow L\text{−}lipschitz\_on\ (cball\ x\ u \cap X)\ (f\ s)$
using *assms local_lipschitz_def*
by *metis*

lemma *local_lipschitz_continuous_on*:
assumes *local_lipschitz*: *local_lipschitz T X f*
assumes $t \in T$
shows *continuous_on X* (*f t*)
unfolding *continuous_on_def*
proof *safe*
fix *x* **assume** $x \in X$
from *local_lipschitzE*[*OF local_lipschitz* ‹$t \in T$› ‹$x \in X$›] **obtain** *u L*
where $0 < u$
and *L*: $\bigwedge s.\ s \in cball\ t\ u \cap T \Longrightarrow L\text{−}lipschitz\_on\ (cball\ x\ u \cap X)\ (f\ s)$
by *metis*
have $x \in ball\ x\ u$ **using** ‹$0 < u$› **by** *simp*
from *lipschitz_on_continuous_on*[*OF L*]
have *tendsto*: $(f\ t \longrightarrow f\ t\ x)\ (at\ x\ within\ cball\ x\ u \cap X)$
using ‹$0 < u$› ‹$x \in X$› ‹$t \in T$›
by (*auto simp*: *continuous_on_def*)
moreover have $\forall_F\ xa\ in\ at\ x.\ (xa \in cball\ x\ u \cap X) = (xa \in X)$
using *eventually_at_ball*[*OF* ‹$0 < u$›, *of x UNIV*]
by *eventually_elim auto*
ultimately show $(f\ t \longrightarrow f\ t\ x)\ (at\ x\ within\ X)$
by (*rule Lim_transform_within_set*)
qed

lemma
local_lipschitz_compose1:
assumes *ll*: *local_lipschitz* $(g\ `\ T)\ X\ (\lambda t.\ f\ t)$
assumes *g*: *continuous_on T g*
shows *local_lipschitz T X* $(\lambda t.\ f\ (g\ t))$
proof (*rule local_lipschitzI*)
fix *t x*
assume $t \in T\ x \in X$
then have $g\ t \in g\ `\ T$ **by** *simp*
from *local_lipschitzE*[*OF assms*(*1*) *this* ‹$x \in X$›]
obtain *u L* **where** $0 < u$ **and** *l*: $(\bigwedge s.\ s \in cball\ (g\ t)\ u \cap g\ `\ T \Longrightarrow L\text{−}lipschitz\_on\ (cball\ x\ u \cap X)\ (f\ s))$
by *auto*
from *g*[*unfolded continuous_on_eq_continuous_within*, *rule_format*, *OF* ‹$t \in T$›,
unfolded continuous_within_eps_delta, *rule_format*, *OF* ‹$0 < u$›]

```
  obtain d where d: d>0 ⋀x'. x'∈T ⟹ dist x' t < d ⟹ dist (g x') (g t) < u
    by (auto)
  show ∃u>0. ∃L. ∀t∈cball t u ∩ T. L−lipschitz_on  (cball x u ∩ X) (f (g t))
    using d ‹0 < u›
    by (fastforce intro: exI[where x=(min d u)/2] exI[where x=L]
      intro!: less_imp_le[OF d(2)] lipschitz_on_subset[OF l] simp: dist_commute)
qed

context
  fixes T::'a::metric_space set and X f
  assumes local_lipschitz: local_lipschitz T X f
begin

lemma continuous_on_TimesI:
  assumes y: ⋀x. x ∈ X ⟹ continuous_on T (λt. f t x)
  shows continuous_on (T × X) (λ(t, x). f t x)
  unfolding continuous_on_iff
proof (safe, simp)
  fix a b and e::real
  assume H: a ∈ T b ∈ X 0 < e
  hence 0 < e/2 by simp
  from y[unfolded continuous_on_iff, OF ‹b ∈ X›, rule_format, OF ‹a ∈ T› ‹0
< e/2›]
  obtain d where d: d > 0 ⋀t. t ∈ T ⟹ dist t a < d ⟹ dist (f t b) (f a b) <
e/2
    by auto

  from ‹a : T› ‹b ∈ X›
  obtain u L where u: 0 < u
    and L: ⋀t. t ∈ cball a u ∩ T ⟹ L−lipschitz_on  (cball b u ∩ X) (f t)
    by (erule local_lipschitzE[OF local_lipschitz])

  have a ∈ cball a u ∩ T by (auto simp: ‹0 < u› ‹a ∈ T› less_imp_le)
  from lipschitz_on_nonneg[OF L[OF ‹a ∈ cball _ _ ∩ _›]] have 0 ≤ L .

  let ?d = Min {d, u, (e/2/(L + 1))}
  show ∃d>0. ∀x∈T. ∀y∈X. dist (x, y) (a, b) < d ⟶ dist (f x y) (f a b) < e
  proof (rule exI[where x = ?d], safe)
    show 0 < ?d
      using ‹0 ≤ L› ‹0 < u› ‹0 < e› ‹0 < d›
      by (auto intro!: divide_pos_pos )
    fix x y
    assume x ∈ T y ∈ X
    assume dist_less: dist (x, y) (a, b) < ?d
    have dist y b ≤ dist (x, y) (a, b)
      using dist_snd_le[of (x, y) (a, b)]
      by auto
    also
    note dist_less
```

```
    also
    {
      note calculation
      also have ?d ≤ u by simp
      finally have dist y b < u .
    }
    have ?d ≤ e/2/(L + 1) by simp
    also have (L + 1) * ... ≤ e / 2
      using ‹0 < e› ‹L ≥ 0›
      by (auto simp: field_split_simps)
    finally have le1: (L + 1) * dist y b < e / 2 using ‹L ≥ 0› by simp

    have dist x a ≤ dist (x, y) (a, b)
      using dist_fst_le[of (x, y) (a, b)]
      by auto
    also note dist_less
    finally have dist x a < ?d .
    also have ?d ≤ d by simp
    finally have dist x a < d .
    note ‹dist x a < ?d›
    also have ?d ≤ u by simp
    finally have dist x a < u .
    then have x ∈ cball a u ∩ T
      using ‹x ∈ T›
      by (auto simp: dist_commute)
    have dist (f x y) (f a b) ≤ dist (f x y) (f x b) + dist (f x b) (f a b)
      by (rule dist_triangle)
    also have (L + 1)−lipschitz_on (cball b u ∩ X) (f x)
      using L[OF ‹x ∈ cball a u ∩ T›]
      by (rule lipschitz_on_le) simp
    then have dist (f x y) (f x b) ≤ (L + 1) * dist y b
      apply (rule lipschitz_onD)
      subgoal
        using ‹y ∈ X› ‹dist y b < u›
        by (simp add: dist_commute)
      subgoal
        using ‹0 < u› ‹b ∈ X›
        by simp
      done
    also have (L + 1) * dist y b ≤ e / 2
      using le1 ‹0 ≤ L› by simp
    also have dist (f x b) (f a b) < e / 2
      by (rule d; fact)
    also have e / 2 + e / 2 = e by simp
    finally show dist (f x y) (f a b) < e by simp
  qed
qed

lemma local_lipschitz_compact_implies_lipschitz:
```

```
  assumes compact X compact T
  assumes cont: ⋀x. x ∈ X ⟹ continuous_on T (λt. f t x)
  obtains L where ⋀t. t ∈ T ⟹ L−lipschitz_on X (f t)
proof −
  {
    assume ∗: ⋀n::nat. ¬(∀ t∈T. n−lipschitz_on X (f t))
    {
      fix n::nat
      from ∗[of n] have ∃ x y t. t ∈ T ∧ x ∈ X ∧ y ∈ X ∧ dist (f t y) (f t x) > n
∗ dist y x
        by (force simp: lipschitz_on_def)
    } then obtain t and x y::nat ⇒ 'b where xy: ⋀n. x n ∈ X ⋀n. y n ∈ X
      and t: ⋀n. t n ∈ T
      and d: ⋀n. dist (f (t n) (y n)) (f (t n) (x n)) > n ∗ dist (y n) (x n)
      by metis
    from xy assms obtain lx rx where lx': lx ∈ X strict_mono (rx :: nat ⇒ nat)
(x o rx) ⟶ lx
      by (metis compact_def)
    with xy have ⋀n. (y o rx) n ∈ X by auto
    with assms obtain ly ry where ly': ly ∈ X strict_mono (ry :: nat ⇒ nat) ((y
o rx) o ry) ⟶ ly
      by (metis compact_def)
    with t have ⋀n. ((t o rx) o ry) n ∈ T by simp
    with assms obtain lt rt where lt': lt ∈ T strict_mono (rt :: nat ⇒ nat) (((t
o rx) o ry) o rt) ⟶ lt
      by (metis compact_def)
    from lx' ly'
    have lx: (x o (rx o ry o rt)) ⟶ lx (is ?x ⟶ _)
      and ly: (y o (rx o ry o rt)) ⟶ ly (is ?y ⟶ _)
      and lt: (t o (rx o ry o rt)) ⟶ lt (is ?t ⟶ _)
      subgoal by (simp add: LIMSEQ_subseq_LIMSEQ o_assoc lt'(2))
      subgoal by (simp add: LIMSEQ_subseq_LIMSEQ ly'(3) o_assoc lt'(2))
      subgoal by (simp add: o_assoc lt'(3))
      done
    hence (λn. dist (?y n) (?x n)) ⟶ dist ly lx
      by (metis tendsto_dist)
    moreover
    let ?S = (λ(t, x). f t x) ` (T × X)
    have eventually (λn::nat. n > 0) sequentially
      by (metis eventually_at_top_dense)
    hence eventually (λn. norm (dist (?y n) (?x n)) ≤ norm (|diameter ?S| / n)
∗ 1) sequentially
    proof eventually_elim
      case (elim n)
      have 0 < rx (ry (rt n)) using ‹0 < n›
        by (metis dual_order.strict_trans1 lt'(2) lx'(2) ly'(2) seq_suble)
      have compact: compact ?S
          by (auto intro!: compact_continuous_image continuous_on_subset[OF
continuous_on_TimesI]
```

```
      compact_Times ‹compact X› ‹compact T› cont)
    have norm (dist (?y n) (?x n)) = dist (?y n) (?x n) by simp
    also
    from this elim d[of rx (ry (rt n))]
    have ... < dist (f (?t n) (?y n)) (f (?t n) (?x n)) / rx (ry (rt (n)))
      using lx'(2) ly'(2) lt'(2) ‹0 < rx _›
      by (auto simp add: field_split_simps strict_mono_def)
    also have ... ≤ diameter ?S / n
    proof (rule frac_le)
      show diameter ?S ≥ 0
        using compact compact_imp_bounded diameter_ge_0 by blast
       show dist (f (?t n) (?y n)) (f (?t n) (?x n)) ≤ diameter ((λ(t,x). f t x) ‘
(T × X))
      by (metis (no_types) compact compact_imp_bounded diameter_bounded_bound
image_eqI mem_Sigma_iff o_apply split_conv t xy(1) xy(2))
       show real n ≤ real (rx (ry (rt n)))
      by (meson le_trans lt'(2) lx'(2) ly'(2) of_nat_mono strict_mono_imp_increasing)
    qed (use ‹n > 0› in auto)
    also have ... ≤ abs (diameter ?S) / n
      by (auto intro!: divide_right_mono)
    finally show ?case by simp
  qed
  with _ have (λn. dist (?y n) (?x n)) ⟶ 0
    by (rule tendsto_0_le)
    (metis tendsto_divide_0[OF tendsto_const] filterlim_at_top_imp_at_infinity
       filterlim_real_sequentially)
  ultimately have lx = ly
    using LIMSEQ_unique by fastforce
  with assms lx' have lx ∈ X by auto
  from ‹lt ∈ T› this obtain u L where L: u > 0 ⋀t. t ∈ cball lt u ∩ T ⟹
L−lipschitz_on (cball lx u ∩ X) (f t)
    by (erule local_lipschitzE[OF local_lipschitz])
  hence L ≥ 0 by (force intro!: lipschitz_on_nonneg ‹lt ∈ T›)

  from L lt ly lx ‹lx = ly›
  have
    eventually (λn. ?t n ∈ ball lt u) sequentially
    eventually (λn. ?y n ∈ ball lx u) sequentially
    eventually (λn. ?x n ∈ ball lx u) sequentially
    by (auto simp: dist_commute Lim)
  moreover have eventually (λn. n > L) sequentially
    by (metis filterlim_at_top_dense filterlim_real_sequentially)
  ultimately
  have eventually (λ_. False) sequentially
  proof eventually_elim
    case (elim n)
    hence dist (f (?t n) (?y n)) (f (?t n) (?x n)) ≤ L * dist (?y n) (?x n)
      using assms xy t
      unfolding dist_norm[symmetric]
```

```
      by (intro lipschitz_onD[OF L(2)]) (auto)
    also have ... ≤ n * dist (?y n) (?x n)
      using elim by (intro mult_right_mono) auto
    also have ... ≤ rx (ry (rt n)) * dist (?y n) (?x n)
      by (intro mult_right_mono[OF _ zero_le_dist])
        (meson lt'(2) lx'(2) ly'(2) of_nat_le_iff order_trans seq_suble)
    also have ... < dist (f (?t n) (?y n)) (f (?t n) (?x n))
      by (auto intro!: d)
    finally show ?case by simp
  qed
  hence False
    by simp
 } then obtain L where ⋀t. t ∈ T ⟹ L−lipschitz_on X (f t)
  by metis
 thus ?thesis ..
qed

lemma local_lipschitz_subset:
 assumes S ⊆ T Y ⊆ X
 shows local_lipschitz S Y f
proof (rule local_lipschitzI)
 fix t x assume t ∈ S x ∈ Y
 then have t ∈ T x ∈ X using assms by auto
 from local_lipschitzE[OF local_lipschitz, OF this]
 obtain u L where u: 0 < u and L: ⋀s. s ∈ cball t u ∩ T ⟹ L−lipschitz_on
(cball x u ∩ X) (f s)
   by blast
 show ∃u>0. ∃L. ∀t∈cball t u ∩ S. L−lipschitz_on (cball x u ∩ Y) (f t)
   using assms
   by (auto intro: exI[where x=u] exI[where x=L]
       intro!: u lipschitz_on_subset[OF _ Int_mono[OF order_refl ‹Y ⊆ X›]] L)
qed

end

lemma local_lipschitz_minus:
 fixes f::'a::metric_space ⇒ 'b::metric_space ⇒ 'c::real_normed_vector
 shows local_lipschitz T X (λt x. − f t x) = local_lipschitz T X f
 by (auto simp: local_lipschitz_def lipschitz_on_minus)

lemma local_lipschitz_PairI:
 assumes f: local_lipschitz A B (λa b. f a b)
 assumes g: local_lipschitz A B (λa b. g a b)
 shows local_lipschitz A B (λa b. (f a b, g a b))
proof (rule local_lipschitzI)
 fix t x assume t ∈ A x ∈ B
 from local_lipschitzE[OF f this] local_lipschitzE[OF g this]
 obtain u L v M where 0 < u (⋀s. s ∈ cball t u ∩ A ⟹ L−lipschitz_on (cball
x u ∩ B) (f s))
```

```
    0 < v (⋀s. s ∈ cball t v ∩ A ⟹ M−lipschitz_on (cball x v ∩ B) (g s))
    by metis
  then show ∃u>0. ∃L. ∀t∈cball t u ∩ A. L−lipschitz_on (cball x u ∩ B) (λb.
(f t b, g t b))
    by (intro exI[where x=min u v])
      (force intro: lipschitz_on_subset intro!: lipschitz_on_Pair)
qed

lemma local_lipschitz_constI: local_lipschitz S T (λt x. f t)
  by (auto simp: intro!: local_lipschitzI lipschitz_on_constant intro: exI[where
x=1])

lemma (in bounded_linear) local_lipschitzI:
  shows local_lipschitz A B (λ_. f)
proof (rule local_lipschitzI, goal_cases)
  case (1 t x)
  from lipschitz_boundE[of (cball x 1 ∩ B)] obtain C where C−lipschitz_on
(cball x 1 ∩ B) f by auto
  then show ?case
    by (auto intro: exI[where x=1])
qed

proposition c1_implies_local_lipschitz:
  fixes T::real set and X::'a::{banach,heine_borel} set
    and f::real ⇒ 'a ⇒ 'a
  assumes f': ⋀t x. t ∈ T ⟹ x ∈ X ⟹ (f t has_derivative blinfun_apply (f'
(t, x))) (at x)
  assumes cont_f': continuous_on (T × X) f'
  assumes open T
  assumes open X
  shows local_lipschitz T X f
proof (rule local_lipschitzI)
  fix t x
  assume t ∈ T x ∈ X
  from open_contains_cball[THEN iffD1, OF ‹open X›, rule_format, OF ‹x ∈
X›]
  obtain u where u: u > 0 cball x u ⊆ X by auto
  moreover
  from open_contains_cball[THEN iffD1, OF ‹open T›, rule_format, OF ‹t ∈
T›]
  obtain v where v: v > 0 cball t v ⊆ T by auto
  ultimately
  have compact (cball t v × cball x u) cball t v × cball x u ⊆ T × X
    by (auto intro!: compact_Times)
  then have compact (f' ‘ (cball t v × cball x u))
   by (auto intro!: compact_continuous_image continuous_on_subset[OF cont_f'])
  then obtain B where B: B > 0 ⋀s y. s ∈ cball t v ⟹ y ∈ cball x u ⟹ norm
(f' (s, y)) ≤ B
    by (auto dest!: compact_imp_bounded simp: bounded_pos)
```

have *lipschitz*: *B−lipschitz_on* (*cball x* (*min u v*) ∩ *X*) (*f s*) **if** *s*: *s* ∈ *cball t v* **for** *s*
proof −
note *s*
also note ‹*cball t v* ⊆ *T*›
finally
have *deriv*: ⋀*y*. *y* ∈ *cball x u* ⟹ (*f s has_derivative blinfun_apply* (*f′* (*s*, *y*))) (*at y within cball x u*)
using ‹_ ⊆ *X*›
by (*auto intro*!: *has_derivative_at_withinI*[*OF f′*])
have *norm* (*f s y* − *f s z*) ≤ *B* ∗ *norm* (*y* − *z*)
if *y* ∈ *cball x u z* ∈ *cball x u*
for *y z*
using *s that*
by (*intro differentiable_bound*[*OF convex_cball deriv*])
(*auto intro*!: *B simp*: *norm_blinfun.rep_eq*[*symmetric*])
then show *?thesis*
using ‹0 < *B*›
by (*auto intro*!: *lipschitz_onI simp*: *dist_norm*)
qed
show ∃ *u*>0. ∃ *L*. ∀ *t*∈*cball t u* ∩ *T*. *L−lipschitz_on* (*cball x u* ∩ *X*) (*f t*)
by (*force intro*: *exI*[**where** *x*=*min u v*] *exI*[**where** *x*=*B*] *intro*!: *lipschitz simp*: *u v*)
qed

end
theory
Multivariate_Analysis
imports
Ordered_Euclidean_Space
Determinants
Cross3
Lipschitz
Starlike
begin

Entry point excluding integration and complex analysis.

end

6.43 Volume of a Simplex

theory *Simplex_Content*
imports *Change_Of_Vars*
begin

lemma *fact_neq_top_ennreal* [*simp*]: *fact n* ≠ (*top* :: *ennreal*)
by (*induction n*) (*auto simp*: *ennreal_mult_eq_top_iff*)

lemma *ennreal_fact*: *ennreal* (*fact n*) = *fact n*
by (*induction n*) (*auto simp*: *ennreal_mult algebra_simps ennreal_of_nat_eq_real_of_nat*)

context
fixes $S :: 'a\ set \Rightarrow real \Rightarrow ('a \Rightarrow real)\ set$
defines $S \equiv (\lambda A\ t.\ \{x.\ (\forall i{\in}A.\ 0 \le x\ i) \wedge sum\ x\ A \le t\})$
begin

lemma *emeasure_std_simplex_aux_step*:
assumes $b \notin A$ *finite A*
shows $x(b := y) \in S\ (insert\ b\ A)\ t \longleftrightarrow y \in \{0..t\} \wedge x \in S\ A\ (t - y)$
using *assms sum_nonneg*[*of A x*] **unfolding** *S_def*
by (*force simp*: *sum_delta_notmem algebra_simps*)

lemma *emeasure_std_simplex_aux*:
fixes $t :: real$
assumes *finite* $(A :: 'a\ set)$ $t \ge 0$
shows *emeasure* $(Pi_M\ A\ (\lambda\_.\ lborel))$
$(S\ A\ t \cap space\ (Pi_M\ A\ (\lambda\_.\ lborel))) = t\ \hat{}\ card\ A\ /\ fact\ (card\ A)$
using *assms*(*1*,*2*)
proof (*induction arbitrary*: *t rule*: *finite_induct*)
case (*empty t*)
thus *?case* **by** (*simp add*: *PiM_empty S_def*)
next
case (*insert b A t*)
define *n* **where** $n = Suc\ (card\ A)$
have *n_pos*: $n > 0$ **by** (*simp add*: *n_def*)
let *?M* $= \lambda A.\ (Pi_M\ A\ (\lambda\_.\ lborel))$
{
fix $A :: 'a\ set$ **and** $t :: real$ **assume** *finite A*
have $S\ A\ t \cap space\ (Pi_M\ A\ (\lambda\_.\ lborel)) =$
$Pi_E\ A\ (\lambda\_.\ \{0..\}) \cap (\lambda x.\ sum\ x\ A) -` \{..t\} \cap space\ (Pi_M\ A\ (\lambda\_.\ lborel))$
by (*auto simp*: *S_def space_PiM*)
also have $\ldots \in sets\ (Pi_M\ A\ (\lambda\_.\ lborel))$
using ‹*finite A*› **by** *measurable*
finally have $S\ A\ t \cap space\ (Pi_M\ A\ (\lambda\_.\ lborel)) \in sets\ (Pi_M\ A\ (\lambda\_.\ lborel))$ **.**
} note *meas* [*measurable*] = *this*

interpret *product_sigma_finite* $\lambda\_.\ lborel$
by *standard*
have *emeasure* (*?M* (*insert b A*)) (*S* (*insert b A*) *t* ∩ *space* (*?M* (*insert b A*)))
=
nn_integral (*?M* (*insert b A*))
$(\lambda x.\ indicator\ (S\ (insert\ b\ A)\ t \cap space\ (?M\ (insert\ b\ A)))\ x)$
using *insert.hyps* **by** (*subst nn_integral_indicator*) *auto*
also have $\ldots = (\int^+ y.\ \int^+ x.\ indicator\ (S\ (insert\ b\ A)\ t \cap space\ (?M\ (insert\ b\ A)))$
$(x(b := y))\ \partial ?M\ A\ \partial lborel)$
using *insert.prems insert.hyps* **by** (*intro product_nn_integral_insert_rev*) *auto*

```
  also have ... = (∫⁺ y. ∫⁺ x. indicator {0..t} y * indicator (S A (t - y) ∩
space (?M A)) x
                ∂?M A ∂lborel)
    using insert.hyps insert.prems emeasure_std_simplex_aux_step[of b A]
    by (intro nn_integral_cong)
       (auto simp: fun_eq_iff indicator_def space_PiM PiE_def extensional_def)
  also have ... = (∫⁺ y. indicator {0..t} y * (∫⁺ x. indicator (S A (t - y) ∩
space (?M A)) x
                ∂?M A) ∂lborel) using ‹finite A›
    by (subst nn_integral_cmult) auto
  also have ... = (∫⁺ y. indicator {0..t} y * emeasure (?M A) (S A (t - y) ∩
space (?M A)) ∂lborel)
    using ‹finite A› by (subst nn_integral_indicator) auto
  also have ... = (∫⁺ y. indicator {0..t} y * (t - y) ^ card A / ennreal (fact
(card A)) ∂lborel)
     using insert.IH by (intro nn_integral_cong) (auto simp: indicator_def di-
vide_ennreal)
  also have ... = (∫⁺ y. indicator {0..t} y * (t - y) ^ card A ∂lborel) / ennreal
(fact (card A))
    using ‹finite A› by (subst nn_integral_divide) auto
  also have (∫⁺ y. indicator {0..t} y * (t - y) ^ card A ∂lborel) =
              (∫⁺y∈{0..t}. ennreal ((t - y) ^ (n - 1)) ∂lborel)
    by (intro nn_integral_cong) (auto simp: indicator_def n_def)
  also have ((λx. - ((t - x) ^ n / n)) has_real_derivative (t - x) ^ (n - 1))
(at x)
    if x ∈ {0..t} for x by (rule derivative_eq_intros refl | simp add: n_pos)+
  hence (∫⁺y∈{0..t}. ennreal ((t - y) ^ (n - 1)) ∂lborel) =
          ennreal (-((t - t) ^ n / n) - (-((t - 0) ^ n / n)))
    using insert.prems insert.hyps by (intro nn_integral_FTC_Icc) auto
  also have ... = ennreal (t ^ n / n) using n_pos by (simp add: zero_power)
  also have ... / ennreal (fact (card A)) = ennreal (t ^ n / n / fact (card A))
    using n_pos ‹t ≥ 0› by (subst divide_ennreal) auto
  also have t ^ n / n / fact (card A) = t ^ n / fact n
    by (simp add: n_def)
  also have n = card (insert b A)
    using insert.hyps by (subst card.insert_remove) (auto simp: n_def)
  finally show ?case .
qed

end

lemma emeasure_std_simplex:
  emeasure lborel (convex hull (insert 0 Basis :: 'a :: euclidean_space set)) =
    ennreal (1 / fact DIM('a))
proof -
  have emeasure lborel {x::'a. (∀ i∈Basis. 0 ≤ x • i) ∧ sum ((•) x) Basis ≤ 1} =
            emeasure (distr (Pi_M Basis (λb. lborel)) borel (λf. ∑ b∈Basis. f b *_R
b))
                {x::'a. (∀ i∈Basis. 0 ≤ x • i) ∧ sum ((•) x) Basis ≤ 1}
```

```
    by (subst lborel_eq) simp
  also have ... = emeasure (Pi_M Basis (λb. lborel))
                  ({y::'a ⇒ real. (∀ i∈Basis. 0 ≤ y i) ∧ sum y Basis ≤ 1} ∩
                    space (Pi_M Basis (λb. lborel)))
    by (subst emeasure_distr) auto
  also have ... = ennreal (1 / fact DIM('a))
    by (subst emeasure_std_simplex_aux) auto
  finally show ?thesis by (simp only: std_simplex)
qed

theorem content_std_simplex:
  measure lborel (convex hull (insert 0 Basis :: 'a :: euclidean_space set)) =
    1 / fact DIM('a)
  by (simp add: measure_def emeasure_std_simplex)


proposition measure_lebesgue_linear_transformation:
  fixes A :: (real ^ 'n :: {finite, wellorder}) set
  fixes f :: _ ⇒ real ^ 'n :: {finite, wellorder}
  assumes bounded A A ∈ sets lebesgue linear f
  shows   measure lebesgue (f ` A) = |det (matrix f)| * measure lebesgue A
proof −
  from assms have [intro]: A ∈ lmeasurable
    by (intro bounded_set_imp_lmeasurable) auto
  hence [intro]: f ` A ∈ lmeasurable
    by (intro lmeasure_integral measurable_linear_image assms)
  have measure lebesgue (f ` A) = integral (f ` A) (λ_. 1)
    by (intro lmeasure_integral measurable_linear_image assms) auto
  also have ... = integral (f ` A) (λ_. 1 :: real ^ 1) $ 0
   by (subst integral_component_eq_cart [symmetric]) (auto intro: integrable_on_const)
  also have ... = |det (matrix f)| * integral A (λx. 1 :: real ^ 1) $ 0
    using assms
    by (subst integral_change_of_variables_linear)
      (auto simp: o_def absolutely_integrable_on_def intro: integrable_on_const)
  also have integral A (λx. 1 :: real ^ 1) $ 0 = integral A (λx. 1)
   by (subst integral_component_eq_cart [symmetric]) (auto intro: integrable_on_const)
  also have ... = measure lebesgue A
    by (intro lmeasure_integral [symmetric]) auto
  finally show ?thesis .
qed

theorem content_simplex:
  fixes X :: (real ^ 'n :: {finite, wellorder}) set and f :: 'n :: _ ⇒ real ^ ('n :: _)
  assumes finite X card X = Suc CARD('n) and x0: x0 ∈ X and bij: bij_betw f
UNIV (X − {x0})
  defines M ≡ (χ i. χ j. f j $ i − x0 $ i)
  shows content (convex hull X) = |det M| / fact (CARD('n))
proof −
  define g where g = (λx. M *v x)
```

```
  have [simp]: M *v axis i 1 = f i - x0 for i :: 'n
    by (simp add: M_def matrix_vector_mult_basis column_def vec_eq_iff)
  define std where std = (convex hull insert 0 Basis :: (real ^ 'n :: _) set)
  have compact: compact std unfolding std_def
    by (intro finite_imp_compact_convex_hull) auto

 have measure lebesgue (convex hull X) = measure lebesgue (((+) (-x0)) ` (convex
hull X))
    by (rule measure_translation [symmetric])
  also have ((+) (-x0)) ` (convex hull X) = convex hull (((+) (-x0)) ` X)
    by (rule convex_hull_translation [symmetric])
  also have ((+) (-x0)) ` X = insert 0 ((λx. x - x0) ` (X - {x0}))
    using x0 by (auto simp: image_iff)
  finally have eq: measure lebesgue (convex hull X) = measure lebesgue (convex
hull ...) .

  from compact have measure lebesgue (g ` std) = |det M| * measure lebesgue std
    by (subst measure_lebesgue_linear_transformation)
       (auto intro: finite_imp_bounded_convex_hull dest: compact_imp_closed
simp: g_def std_def)
  also have measure lebesgue std = content std using compact
    by (intro measure_completion) (auto dest: compact_imp_closed)
  also have content std = 1 / fact CARD('n) unfolding std_def
    by (simp add: content_std_simplex)
  also have g ` std = convex hull (g ` insert 0 Basis) unfolding std_def
    by (rule convex_hull_linear_image) (auto simp: g_def)
  also have g ` insert 0 Basis = insert 0 (g ` Basis)
    by (auto simp: g_def)
  also have g ` Basis = (λx. x - x0) ` range f
    by (auto simp: g_def Basis_vec_def image_iff)
  also have range f = X - {x0} using bij
    using bij_betw_imp_surj_on by blast
  also note eq [symmetric]
  finally show ?thesis
    using finite_imp_compact_convex_hull[OF ‹finite X›] by (auto dest: com-
pact_imp_closed)
qed

theorem content_triangle:
  fixes A B C :: real ^ 2
  shows content (convex hull {A, B, C}) =
        |(C $ 1 - A $ 1) * (B $ 2 - A $ 2) - (B $ 1 - A $ 1) * (C $ 2 - A
$ 2)| / 2
proof -
  define M :: real ^ 2 ^ 2 where M ≡ (χ i. χ j. (if j = 1 then B else C) $ i -
A $ i)
  define g where g = (λx. M *v x)
  define std where std = (convex hull insert 0 Basis :: (real ^ 2) set)
  have [simp]: M *v axis i 1 = (if i = 1 then B - A else C - A) for i
```

```
    by (auto simp: M_def matrix_vector_mult_basis column_def vec_eq_iff)
  have compact: compact std unfolding std_def
    by (intro finite_imp_compact_convex_hull) auto

  have measure lebesgue (convex hull {A, B, C}) =
        measure lebesgue (((+) (-A)) ` (convex hull {A, B, C}))
    by (rule measure_translation [symmetric])
  also have ((+) (-A)) ` (convex hull {A, B, C}) = convex hull (((+) (-A)) `
{A, B, C})
    by (rule convex_hull_translation [symmetric])
  also have ((+) (-A)) ` {A, B, C} = {0, B - A, C - A}
    by (auto simp: image_iff)
  finally have eq: measure lebesgue (convex hull {A, B, C}) =
                   measure lebesgue (convex hull {0, B - A, C - A}) .

  from compact have measure lebesgue (g ` std) = |det M| * measure lebesgue std
    by (subst measure_lebesgue_linear_transformation)
       (auto intro: finite_imp_bounded_convex_hull dest: compact_imp_closed
simp: g_def std_def)
  also have measure lebesgue std = content std using compact
    by (intro measure_completion) (auto dest: compact_imp_closed)
  also have content std = 1 / 2 unfolding std_def
    by (simp add: content_std_simplex)
  also have g ` std = convex hull (g ` insert 0 Basis) unfolding std_def
    by (rule convex_hull_linear_image) (auto simp: g_def)
  also have g ` insert 0 Basis = insert 0 (g ` Basis)
    by (auto simp: g_def)
  also have (2 :: 2) ≠ 1 by auto
  hence ¬(∀ y::2. y = 1) by blast
  hence g ` Basis = {B - A, C - A}
    by (auto simp: g_def Basis_vec_def image_iff)
  also note eq [symmetric]
  finally show ?thesis
    using finite_imp_compact_convex_hull[of {A, B, C}]
    by (auto dest!: compact_imp_closed simp: det_2 M_def)
qed

theorem heron:
  fixes A B C :: real ^ 2
  defines a ≡ dist B C and b ≡ dist A C and c ≡ dist A B
  defines s ≡ (a + b + c) / 2
  shows   content (convex hull {A, B, C}) = sqrt (s * (s - a) * (s - b) * (s -
c))
proof -
  have [simp]: (UNIV :: 2 set) = {1, 2}
    using exhaust_2 by auto
  have dist_eq: dist (A :: real ^ 2) B ^ 2 = (A $ 1 - B $ 1) ^ 2 + (A $ 2 - B
$ 2) ^ 2
    for A B by (simp add: dist_vec_def dist_real_def)
```

have *nonneg*: $s * (s - a) * (s - b) * (s - c) \geq 0$
using *dist_triangle*[*of A B C*] *dist_triangle*[*of A C B*] *dist_triangle*[*of B C A*]
by (*intro mult_nonneg_nonneg*) (*auto simp*: *s_def a_def b_def c_def dist_commute*)

have *16 * content* (*convex hull* {*A*, *B*, *C*}) ^ *2* =
*4 * ((C $ 1 - A $ 1) * (B $ 2 - A $ 2) - (B $ 1 - A $ 1) * (C $ 2 - A $ 2)) ^ 2*
by (*subst content_triangle*) (*simp add*: *power_divide*)
also have ... = (*2 * (dist A B ^ 2 * dist A C ^ 2 + dist A B ^ 2 * dist B C ^ 2 +*
*dist A C ^ 2 * dist B C ^ 2) - (dist A B ^ 2) ^ 2 - (dist A C ^ 2) ^ 2 - (dist B C ^ 2) ^ 2*)
unfolding *dist_eq* **unfolding** *power2_eq_square* **by** *algebra*
also have ... = (*a + b + c*) * ((*a + b + c*) - *2 * a*) * ((*a + b + c*) - *2 * b*) *
((*a + b + c*) - *2 * c*)
unfolding *power2_eq_square* **by** (*simp add*: *s_def a_def b_def c_def algebra_simps*)
also have ... = *16 * s * (s - a) * (s - b) * (s - c*)
by (*simp add*: *s_def field_split_simps*)
finally have *content* (*convex hull* {*A*, *B*, *C*}) ^ *2 = s * (s - a) * (s - b) * (s - c*)
by *simp*
also have ... = *sqrt* (*s * (s - a) * (s - b) * (s - c*)) ^ *2*
by (*intro real_sqrt_pow2* [*symmetric*] *nonneg*)
finally show *?thesis* **using** *nonneg*
by (*subst* (*asm*) *power2_eq_iff_nonneg*) *auto*
qed

end

6.44 Convergence of Formal Power Series

theory *FPS_Convergence*
imports
Generalised_Binomial_Theorem
HOL−Computational_Algebra.Formal_Power_Series
begin

In this theory, we will connect formal power series (which are algebraic objects) with analytic functions. This will become more important in complex analysis, and indeed some of the less trivial results will only be proven there.

6.44.1 Balls with extended real radius

The following is a variant of *ball* that also allows an infinite radius.

definition *eball* :: *'a* :: *metric_space* ⇒ *ereal* ⇒ *'a set* **where**
eball z r = {*z'*. *ereal* (*dist z z'*) < *r*}

lemma *in_eball_iff* [*simp*]: $z \in \mathit{eball}\ z0\ r \longleftrightarrow \mathit{ereal}\ (\mathit{dist}\ z0\ z) < r$
by (*simp add*: *eball_def*)

lemma *eball_ereal* [*simp*]: $\mathit{eball}\ z\ (\mathit{ereal}\ r) = \mathit{ball}\ z\ r$
by *auto*

lemma *eball_inf* [*simp*]: $\mathit{eball}\ z\ \infty = \mathit{UNIV}$
by *auto*

lemma *eball_empty* [*simp*]: $r \leq 0 \Longrightarrow \mathit{eball}\ z\ r = \{\}$
proof *safe*
fix x **assume** $r \leq 0\ x \in \mathit{eball}\ z\ r$
hence $\mathit{dist}\ z\ x < r$ **by** *simp*
also have $\ldots \leq \mathit{ereal}\ 0$ **using** ‹$r \leq 0$› **by** (*simp add*: *zero_ereal_def*)
finally show $x \in \{\}$ **by** *simp*
qed

lemma *eball_conv_UNION_balls*:
$\mathit{eball}\ z\ r = (\bigcup r' \in \{r'.\ \mathit{ereal}\ r' < r\}.\ \mathit{ball}\ z\ r')$
by (*cases r*) (*use dense gt_ex* **in** *force*)+

lemma *eball_mono*: $r \leq r' \Longrightarrow \mathit{eball}\ z\ r \leq \mathit{eball}\ z\ r'$
by *auto*

lemma *ball_eball_mono*: $\mathit{ereal}\ r \leq r' \Longrightarrow \mathit{ball}\ z\ r \leq \mathit{eball}\ z\ r'$
using *eball_mono*[*of ereal r r'*] **by** *simp*

lemma *open_eball* [*simp*, *intro*]: *open* (*eball z r*)
by (*cases r*) *auto*

6.44.2 Basic properties of convergent power series

definition *fps_conv_radius* :: $'a$ :: {*banach*, *real_normed_div_algebra*} *fps* ⇒ *ereal* **where**
fps_conv_radius f = *conv_radius* (*fps_nth f*)

definition *eval_fps* :: $'a$ :: {*banach*, *real_normed_div_algebra*} *fps* ⇒ $'a$ ⇒ $'a$ **where**
eval_fps f z = $(\sum n.\ \mathit{fps\_nth}\ f\ n * z\ \hat{}\ n)$

lemma *norm_summable_fps*:
fixes *f* :: $'a$ :: {*banach*, *real_normed_div_algebra*} *fps*
shows *norm z* < *fps_conv_radius f* ⟹ *summable* (λ*n*. *norm* (*fps_nth f n* ∗ *z* ^ *n*))
by (*rule abs_summable_in_conv_radius*) (*simp_all add*: *fps_conv_radius_def*)

lemma *summable_fps*:
fixes *f* :: $'a$ :: {*banach*, *real_normed_div_algebra*} *fps*
shows *norm z* < *fps_conv_radius f* ⟹ *summable* (λ*n*. *fps_nth f n* ∗ *z* ^ *n*)

by (*rule summable_in_conv_radius*) (*simp_all add*: *fps_conv_radius_def*)

theorem *sums_eval_fps*:
fixes $f :: 'a :: \{banach, real\_normed\_div\_algebra\}$ *fps*
assumes *norm z* < *fps_conv_radius f*
shows (λ*n*. *fps_nth f n* ∗ *z* ^ *n*) *sums eval_fps f z*
using *assms* **unfolding** *eval_fps_def fps_conv_radius_def*
by (*intro summable_sums summable_in_conv_radius*) *simp_all*

lemma *continuous_on_eval_fps*:
fixes $f :: 'a :: \{banach, real\_normed\_div\_algebra\}$ *fps*
shows *continuous_on* (*eball 0* (*fps_conv_radius f*)) (*eval_fps f*)
proof (*subst continuous_on_eq_continuous_at* [*OF open_eball*], *safe*)
fix *x* :: *'a* **assume** *x*: *x* ∈ *eball 0* (*fps_conv_radius f*)
define *r* **where** *r* = (*if fps_conv_radius f* = ∞ *then norm x* + *1 else*
(*norm x* + *real_of_ereal* (*fps_conv_radius f*)) / *2*)
have *r*: *norm x* < *r* ∧ *ereal r* < *fps_conv_radius f*
using *x* **by** (*cases fps_conv_radius f*)
(*auto simp*: *r_def eball_def split*: *if_splits*)

have *continuous_on* (*cball 0 r*) (λ*x*. ∑ *i*. *fps_nth f i* ∗ (*x* − *0*) ^ *i*)
by (*rule powser_continuous_suminf*) (*insert r*, *auto simp*: *fps_conv_radius_def*)
hence *continuous_on* (*cball 0 r*) (*eval_fps f*)
by (*simp add*: *eval_fps_def*)
thus *isCont* (*eval_fps f*) *x*
by (*rule continuous_on_interior*) (*use r* **in** *auto*)
qed

lemma *continuous_on_eval_fps'* [*continuous_intros*]:
assumes *continuous_on A g*
assumes *g* ‘ *A* ⊆ *eball 0* (*fps_conv_radius f*)
shows *continuous_on A* (λ*x*. *eval_fps f* (*g x*))
using *continuous_on_compose2*[*OF continuous_on_eval_fps assms*] **.**

lemma *has_field_derivative_powser*:
fixes $z :: 'a :: \{banach, real\_normed\_field\}$
assumes *ereal* (*norm z*) < *conv_radius f*
shows ((λ*z*. ∑ *n*. *f n* ∗ *z* ^ *n*) *has_field_derivative* (∑ *n*. *diffs f n* ∗ *z* ^ *n*))
(*at z within A*)
proof −
define *K* **where** *K* = (*if conv_radius f* = ∞ *then norm z* + *1*
else (*norm z* + *real_of_ereal* (*conv_radius f*)) / *2*)
have *K*: *norm z* < *K* ∧ *ereal K* < *conv_radius f*
using *assms* **by** (*cases conv_radius f*) (*auto simp*: *K_def*)
have *0* ≤ *norm z* **by** *simp*
also from *K* **have** ... < *K* **by** *simp*
finally have *K_pos*: *K* > *0* **by** *simp*

have *summable* (λ*n*. *f n* ∗ *of_real K* ^ *n*)

```
    using K and K_pos by (intro summable_in_conv_radius) auto
  moreover from K and K_pos have norm z < norm (of_real K :: 'a) by auto
  ultimately show ?thesis
    by (rule has_field_derivative_at_within [OF termdiffs_strong])
qed

lemma has_field_derivative_eval_fps:
  fixes z :: 'a :: {banach, real_normed_field}
  assumes norm z < fps_conv_radius f
  shows   (eval_fps f has_field_derivative eval_fps (fps_deriv f) z) (at z within
A)
proof -
  have (eval_fps f has_field_derivative eval_fps (Abs_fps (diffs (fps_nth f))) z)
(at z within A)
    using assms unfolding eval_fps_def fps_nth_Abs_fps fps_conv_radius_def
    by (intro has_field_derivative_powser) auto
  also have Abs_fps (diffs (fps_nth f)) = fps_deriv f
    by (simp add: fps_eq_iff diffs_def)
  finally show ?thesis .
qed

lemma holomorphic_on_eval_fps [holomorphic_intros]:
  fixes z :: 'a :: {banach, real_normed_field}
  assumes A ⊆ eball 0 (fps_conv_radius f)
  shows   eval_fps f holomorphic_on A
proof (rule holomorphic_on_subset [OF _ assms])
  show eval_fps f holomorphic_on eball 0 (fps_conv_radius f)
  proof (subst holomorphic_on_open [OF open_eball], safe, goal_cases)
    case (1 x)
    thus ?case
    by (intro exI[of _ eval_fps (fps_deriv f) x]) (auto intro: has_field_derivative_eval_fps)
  qed
qed

lemma analytic_on_eval_fps:
  fixes z :: 'a :: {banach, real_normed_field}
  assumes A ⊆ eball 0 (fps_conv_radius f)
  shows   eval_fps f analytic_on A
proof (rule analytic_on_subset [OF _ assms])
  show eval_fps f analytic_on eball 0 (fps_conv_radius f)
    using holomorphic_on_eval_fps[of eball 0 (fps_conv_radius f)]
    by (subst analytic_on_open) auto
qed

lemma continuous_eval_fps [continuous_intros]:
  fixes z :: 'a::{real_normed_field,banach}
  assumes norm z < fps_conv_radius F
  shows   continuous (at z within A) (eval_fps F)
proof -
```

from *ereal_dense2*[*OF assms*] **obtain** K :: *real* **where** K: *norm* $z < K$ $K <$ *fps_conv_radius* F
by *auto*
have $0 \leq$ *norm* z **by** *simp*
also have *norm* $z < K$ **by** *fact*
finally have $K > 0$ **.**
from K **and** ‹$K > 0$› **have** *summable* (λn. *fps_nth* F n $*$ *of_real* K ^ n)
by (*intro summable_fps*) *auto*
from *this* **have** *isCont* (*eval_fps* F) z **unfolding** *eval_fps_def*
by (*rule isCont_powser*) (*use* K **in** *auto*)
thus *continuous* (*at* z *within* A) (*eval_fps* F)
by (*simp add*: *continuous_at_imp_continuous_within*)
qed

6.44.3 Lower bounds on radius of convergence

lemma *fps_conv_radius_deriv*:
fixes f :: $'a$ :: {*banach*, *real_normed_field*} *fps*
shows *fps_conv_radius* (*fps_deriv* f) $\geq$ *fps_conv_radius* f
unfolding *fps_conv_radius_def*
proof (*rule conv_radius_geI_ex*)
fix r :: *real* **assume** r: $r > 0$ *ereal* $r <$ *conv_radius* (*fps_nth* f)
define K **where** K = (*if conv_radius* (*fps_nth* f) $= \infty$ *then* $r + 1$
else (*real_of_ereal* (*conv_radius* (*fps_nth* f)) $+$ r) / 2)
have K: $r < K \wedge$ *ereal* $K <$ *conv_radius* (*fps_nth* f)
using r **by** (*cases conv_radius* (*fps_nth* f)) (*auto simp*: K_*def*)
have *summable* (λn. *diffs* (*fps_nth* f) n $*$ *of_real* r ^ n)
proof (*rule termdiff_converges*)
fix x :: $'a$ **assume** *norm* $x <$ K
hence *ereal* (*norm* x) $<$ *ereal* K **by** *simp*
also have ... $<$ *conv_radius* (*fps_nth* f) **using** K **by** *simp*
finally show *summable* (λn. *fps_nth* f n $*$ x ^ n)
by (*intro summable_in_conv_radius*) *auto*
qed (*insert* K r, *auto*)
also have ... = (λn. *fps_nth* (*fps_deriv* f) n $*$ *of_real* r ^ n)
by (*simp add*: *fps_deriv_def diffs_def*)
finally show $\exists z$::$'a$. *norm* $z = r \wedge$ *summable* (λn. *fps_nth* (*fps_deriv* f) n $*$ z ^ n)
using r **by** (*intro exI*[*of* _ *of_real* r]) *auto*
qed

lemma *eval_fps_at_0*: *eval_fps* f 0 = *fps_nth* f 0
by (*simp add*: *eval_fps_def*)

lemma *fps_conv_radius_norm* [*simp*]:
fps_conv_radius (*Abs_fps* (λn. *norm* (*fps_nth* f n))) = *fps_conv_radius* f
by (*simp add*: *fps_conv_radius_def*)

lemma *fps_conv_radius_const* [*simp*]: *fps_conv_radius* (*fps_const* c) $= \infty$

proof −
have *fps_conv_radius* (*fps_const c*) = *conv_radius* (λ_. *0* :: *'a*)
unfolding *fps_conv_radius_def*
by (*intro conv_radius_cong eventually_mono*[*OF eventually_gt_at_top*[*of 0*]])
auto
thus *?thesis* **by** *simp*
qed

lemma *fps_conv_radius_0* [*simp*]: *fps_conv_radius 0* = ∞
by (*simp only*: *fps_const_0_eq_0* [*symmetric*] *fps_conv_radius_const*)

lemma *fps_conv_radius_1* [*simp*]: *fps_conv_radius 1* = ∞
by (*simp only*: *fps_const_1_eq_1* [*symmetric*] *fps_conv_radius_const*)

lemma *fps_conv_radius_numeral* [*simp*]: *fps_conv_radius* (*numeral n*) = ∞
by (*simp add*: *numeral_fps_const*)

lemma *fps_conv_radius_fps_X_power* [*simp*]: *fps_conv_radius* (*fps_X* ^ *n*) = ∞
proof −
have *fps_conv_radius* (*fps_X* ^ *n* :: *'a fps*) = *conv_radius* (λ_. *0* :: *'a*)
unfolding *fps_conv_radius_def*
by (*intro conv_radius_cong eventually_mono*[*OF eventually_gt_at_top*[*of n*]])

(*auto simp*: *fps_X_power_iff*)
thus *?thesis* **by** *simp*
qed

lemma *fps_conv_radius_fps_X* [*simp*]: *fps_conv_radius fps_X* = ∞
using *fps_conv_radius_fps_X_power*[*of 1*] **by** (*simp only*: *power_one_right*)

lemma *fps_conv_radius_shift* [*simp*]:
fps_conv_radius (*fps_shift n f*) = *fps_conv_radius f*
by (*simp add*: *fps_conv_radius_def fps_shift_def conv_radius_shift*)

lemma *fps_conv_radius_cmult_left*:
c ≠ *0* ⟹ *fps_conv_radius* (*fps_const c* ∗ *f*) = *fps_conv_radius f*
unfolding *fps_conv_radius_def* **by** (*simp add*: *conv_radius_cmult_left*)

lemma *fps_conv_radius_cmult_right*:
c ≠ *0* ⟹ *fps_conv_radius* (*f* ∗ *fps_const c*) = *fps_conv_radius f*
unfolding *fps_conv_radius_def* **by** (*simp add*: *conv_radius_cmult_right*)

lemma *fps_conv_radius_uminus* [*simp*]:
fps_conv_radius (−*f*) = *fps_conv_radius f*
using *fps_conv_radius_cmult_left*[*of* −*1 f*]
by (*simp flip*: *fps_const_neg*)

lemma *fps_conv_radius_add*: *fps_conv_radius* (*f* + *g*) ≥ *min* (*fps_conv_radius*

```
f) (fps_conv_radius g)
  unfolding fps_conv_radius_def using conv_radius_add_ge[of fps_nth f fps_nth
g]
  by simp

lemma fps_conv_radius_diff: fps_conv_radius (f − g) ≥ min (fps_conv_radius
f) (fps_conv_radius g)
  using fps_conv_radius_add[of f −g] by simp

lemma fps_conv_radius_mult: fps_conv_radius (f ∗ g) ≥ min (fps_conv_radius
f) (fps_conv_radius g)
  using conv_radius_mult_ge[of fps_nth f fps_nth g]
  by (simp add: fps_mult_nth fps_conv_radius_def atLeast0AtMost)

lemma fps_conv_radius_power: fps_conv_radius (f ^ n) ≥ fps_conv_radius f
proof (induction n)
  case (Suc n)
  hence fps_conv_radius f ≤ min (fps_conv_radius f) (fps_conv_radius (f ^ n))
    by simp
  also have ... ≤ fps_conv_radius (f ∗ f ^ n)
    by (rule fps_conv_radius_mult)
  finally show ?case by simp
qed simp_all

context
begin

lemma natfun_inverse_bound:
  fixes f :: 'a :: {real_normed_field} fps
  assumes fps_nth f 0 = 1 and δ > 0
      and summable: summable (λn. norm (fps_nth f (Suc n)) ∗ δ ^ Suc n)
      and le: (∑ n. norm (fps_nth f (Suc n)) ∗ δ ^ Suc n) ≤ 1
  shows   norm (natfun_inverse f n) ≤ inverse (δ ^ n)
proof (induction n rule: less_induct)
  case (less m)
  show ?case
  proof (cases m)
    case 0
      thus ?thesis using assms by (simp add: field_split_simps norm_inverse
norm_divide)
  next
    case [simp]: (Suc n)
    have norm (natfun_inverse f (Suc n)) =
           norm (∑ i = Suc 0..Suc n. fps_nth f i ∗ natfun_inverse f (Suc n − i))
      (is _ = norm ?S) using assms
      by (simp add: field_simps norm_mult norm_divide del: sum.cl_ivl_Suc)
   also have norm ?S ≤ (∑ i = Suc 0..Suc n. norm (fps_nth f i ∗ natfun_inverse
f (Suc n − i)))
      by (rule norm_sum)
```

```
    also have ... ≤ (∑ i = Suc 0..Suc n. norm (fps_nth f i) / δ ^ (Suc n − i))
    proof (intro sum_mono, goal_cases)
      case (1 i)
      have norm (fps_nth f i ∗ natfun_inverse f (Suc n − i)) =
            norm (fps_nth f i) ∗ norm (natfun_inverse f (Suc n − i))
        by (simp add: norm_mult)
      also have ... ≤ norm (fps_nth f i) ∗ inverse (δ ^ (Suc n − i))
        using 1 by (intro mult_left_mono less.IH) auto
      also have ... = norm (fps_nth f i) / δ ^ (Suc n − i)
        by (simp add: field_split_simps)
      finally show ?case .
    qed
    also have ... = (∑ i = Suc 0..Suc n. norm (fps_nth f i) ∗ δ ^ i) / δ ^ Suc n
      by (subst sum_divide_distrib, rule sum.cong)
         (insert ‹δ > 0›, auto simp: field_simps power_diff)
    also have (∑ i = Suc 0..Suc n. norm (fps_nth f i) ∗ δ ^ i) =
              (∑ i=0..n. norm (fps_nth f (Suc i)) ∗ δ ^ (Suc i))
      by (subst sum.atLeast_Suc_atMost_Suc_shift) simp_all
    also have {0..n} = {..<Suc n} by auto
    also have (∑ i< Suc n. norm (fps_nth f (Suc i)) ∗ δ ^ (Suc i)) ≤
              (∑ n. norm (fps_nth f (Suc n)) ∗ δ ^ (Suc n))
    using ‹δ > 0› by (intro sum_le_suminf ballI mult_nonneg_nonneg zero_le_power
summable) auto
    also have ... ≤ 1 by fact
    finally show ?thesis using ‹δ > 0›
      by (simp add: divide_right_mono field_split_simps)
  qed
qed

private lemma fps_conv_radius_inverse_pos_aux:
  fixes f :: 'a :: {banach, real_normed_field} fps
  assumes fps_nth f 0 = 1 fps_conv_radius f > 0
  shows   fps_conv_radius (inverse f) > 0
proof −
  let ?R = fps_conv_radius f
  define h where h = Abs_fps (λn. norm (fps_nth f n))
  have [simp]: fps_conv_radius h = ?R by (simp add: h_def)
  have continuous_on (eball 0 (fps_conv_radius h)) (eval_fps h)
    by (intro continuous_on_eval_fps)
  hence ∗: open (eval_fps h −‘ A ∩ eball 0 ?R) if open A for A
    using that by (subst (asm) continuous_on_open_vimage) auto
  have open (eval_fps h −‘ {..<2} ∩ eball 0 ?R)
    by (rule ∗) auto
  moreover have 0 ∈ eval_fps h −‘ {..<2} ∩ eball 0 ?R
    using assms by (auto simp: eball_def zero_ereal_def eval_fps_at_0 h_def)
  ultimately obtain ε where ε: ε > 0 ball 0 ε ⊆ eval_fps h −‘ {..<2} ∩ eball 0
?R
    by (subst (asm) open_contains_ball_eq) blast+
```

```
  define δ where δ = real_of_ereal (min (ereal ε / 2) (?R / 2))
  have δ: 0 < δ ∧ δ < ε ∧ ereal δ < ?R
    using ‹ε > 0› and assms by (cases ?R) (auto simp: δ_def min_def)

  have summable: summable (λn. norm (fps_nth f n) * δ ^ n)
   using δ by (intro summable_in_conv_radius) (simp_all add: fps_conv_radius_def)
  hence (λn. norm (fps_nth f n) * δ ^ n) sums eval_fps h δ
    by (simp add: eval_fps_def summable_sums h_def)
  hence (λn. norm (fps_nth f (Suc n)) * δ ^ Suc n) sums (eval_fps h δ - 1)
    by (subst sums_Suc_iff) (auto simp: assms)
  moreover {
    from δ have δ ∈ ball 0 ε by auto
    also have ... ⊆ eval_fps h -` {..<2} ∩ eball 0 ?R by fact
    finally have eval_fps h δ < 2 by simp
  }
  ultimately have le: (∑n. norm (fps_nth f (Suc n)) * δ ^ Suc n) ≤ 1
    by (simp add: sums_iff)
  from summable have summable: summable (λn. norm (fps_nth f (Suc n)) * δ
^ Suc n)
    by (subst summable_Suc_iff)

  have 0 < δ using δ by blast
  also have δ = inverse (limsup (λn. ereal (inverse δ)))
    using δ by (subst Limsup_const) auto
  also have ... ≤ conv_radius (natfun_inverse f)
    unfolding conv_radius_def
  proof (intro ereal_inverse_antimono Limsup_mono
          eventually_mono[OF eventually_gt_at_top[of 0]])
    fix n :: nat assume n: n > 0
    have root n (norm (natfun_inverse f n)) ≤ root n (inverse (δ ^ n))
      using n assms δ le summable
      by (intro real_root_le_mono natfun_inverse_bound) auto
    also have ... = inverse δ
      using n δ by (simp add: power_inverse [symmetric] real_root_pos2)
    finally show ereal (inverse δ) ≥ ereal (root n (norm (natfun_inverse f n)))
      by (subst ereal_less_eq)
  next
    have 0 = limsup (λn. 0::ereal)
      by (rule Limsup_const [symmetric]) auto
    also have ... ≤ limsup (λn. ereal (root n (norm (natfun_inverse f n))))
      by (intro Limsup_mono) (auto simp: real_root_ge_zero)
    finally show 0 ≤ ... by simp
  qed
  also have ... = fps_conv_radius (inverse f)
    using assms by (simp add: fps_conv_radius_def fps_inverse_def)
  finally show ?thesis by (simp add: zero_ereal_def)
qed

lemma fps_conv_radius_inverse_pos:
```

```
  fixes f :: 'a :: {banach, real_normed_field} fps
  assumes fps_nth f 0 ≠ 0 and fps_conv_radius f > 0
  shows   fps_conv_radius (inverse f) > 0
proof −
  let ?c = fps_nth f 0
  have fps_conv_radius (inverse f) = fps_conv_radius (fps_const ?c * inverse f)
    using assms by (subst fps_conv_radius_cmult_left) auto
  also have fps_const ?c * inverse f = inverse (fps_const (inverse ?c) * f)
    using assms by (simp add: fps_inverse_mult fps_const_inverse)
  also have fps_conv_radius ... > 0 using assms
    by (intro fps_conv_radius_inverse_pos_aux)
       (auto simp: fps_conv_radius_cmult_left)
  finally show ?thesis .
qed

end

lemma fps_conv_radius_exp [simp]:
  fixes c :: 'a :: {banach, real_normed_field}
  shows fps_conv_radius (fps_exp c) = ∞
  unfolding fps_conv_radius_def
proof (rule conv_radius_inftyI'')
  fix z :: 'a
  have (λn. norm (c * z) ^ n /R fact n) sums exp (norm (c * z))
    by (rule exp_converges)
  also have (λn. norm (c * z) ^ n /R fact n) = (λn. norm (fps_nth (fps_exp c)
n * z ^ n))
   by (rule ext) (simp add: norm_divide norm_mult norm_power field_split_simps)
  finally have summable ... by (simp add: sums_iff)
  thus summable (λn. fps_nth (fps_exp c) n * z ^ n)
    by (rule summable_norm_cancel)
qed
```

6.44.4 Evaluating power series

```
theorem eval_fps_deriv:
  assumes norm z < fps_conv_radius f
  shows   eval_fps (fps_deriv f) z = deriv (eval_fps f) z
  by (intro DERIV_imp_deriv [symmetric] has_field_derivative_eval_fps assms)

theorem fps_nth_conv_deriv:
  fixes f :: complex fps
  assumes fps_conv_radius f > 0
  shows   fps_nth f n = (deriv ^^ n) (eval_fps f) 0 / fact n
  using assms
proof (induction n arbitrary: f)
  case 0
  thus ?case by (simp add: eval_fps_def)
next
```

```
  case (Suc n f)
  have (deriv ^^ Suc n) (eval_fps f) 0 = (deriv ^^ n) (deriv (eval_fps f)) 0
    unfolding funpow_Suc_right o_def ..
  also have eventually (λz::complex. z ∈ eball 0 (fps_conv_radius f)) (nhds 0)
   using Suc.prems by (intro eventually_nhds_in_open) (auto simp: zero_ereal_def)
  hence eventually (λz. deriv (eval_fps f) z = eval_fps (fps_deriv f) z) (nhds 0)
    by eventually_elim (simp add: eval_fps_deriv)
  hence (deriv ^^ n) (deriv (eval_fps f)) 0 = (deriv ^^ n) (eval_fps (fps_deriv
f)) 0
    by (intro higher_deriv_cong_ev refl)
  also have ... / fact n = fps_nth (fps_deriv f) n
    using Suc.prems fps_conv_radius_deriv[of f]
    by (intro Suc.IH [symmetric]) auto
  also have ... / of_nat (Suc n) = fps_nth f (Suc n)
    by (simp add: fps_deriv_def del: of_nat_Suc)
  finally show ?case by (simp add: field_split_simps)
qed

theorem eval_fps_eqD:
  fixes f g :: complex fps
  assumes fps_conv_radius f > 0 fps_conv_radius g > 0
  assumes eventually (λz. eval_fps f z = eval_fps g z) (nhds 0)
  shows   f = g
proof (rule fps_ext)
  fix n :: nat
  have fps_nth f n = (deriv ^^ n) (eval_fps f) 0 / fact n
    using assms by (intro fps_nth_conv_deriv)
  also have (deriv ^^ n) (eval_fps f) 0 = (deriv ^^ n) (eval_fps g) 0
    by (intro higher_deriv_cong_ev refl assms)
  also have ... / fact n = fps_nth g n
    using assms by (intro fps_nth_conv_deriv [symmetric])
  finally show fps_nth f n = fps_nth g n .
qed

lemma eval_fps_const [simp]:
  fixes c :: 'a :: {banach, real_normed_div_algebra}
  shows eval_fps (fps_const c) z = c
proof -
  have (λn::nat. if n ∈ {0} then c else 0) sums (∑ n∈{0::nat}. c)
    by (rule sums_If_finite_set) auto
 also have ?this ⟷ (λn::nat. fps_nth (fps_const c) n * z ^ n) sums (∑ n∈{0::nat}.
c)
    by (intro sums_cong) auto
  also have (∑ n∈{0::nat}. c) = c
    by simp
  finally show ?thesis
    by (simp add: eval_fps_def sums_iff)
qed
```

lemma *eval_fps_0* [*simp*]:
eval_fps (*0* :: *'a* :: {*banach*, *real_normed_div_algebra*} *fps*) *z* = *0*
by (*simp only*: *fps_const_0_eq_0* [*symmetric*] *eval_fps_const*)

lemma *eval_fps_1* [*simp*]:
eval_fps (*1* :: *'a* :: {*banach*, *real_normed_div_algebra*} *fps*) *z* = *1*
by (*simp only*: *fps_const_1_eq_1* [*symmetric*] *eval_fps_const*)

lemma *eval_fps_numeral* [*simp*]:
eval_fps (*numeral n* :: *'a* :: {*banach*, *real_normed_div_algebra*} *fps*) *z* = *numeral n*
by (*simp only*: *numeral_fps_const eval_fps_const*)

lemma *eval_fps_X_power* [*simp*]:
eval_fps (*fps_X* ^ *m* :: *'a* :: {*banach*, *real_normed_div_algebra*} *fps*) *z* = *z* ^ *m*
proof −
have (λ*n*::*nat*. *if n* ∈ {*m*} *then z* ^ *n else 0* :: *'a*) *sums* (∑ *n*∈{*m*::*nat*}. *z* ^ *n*)
by (*rule sums_If_finite_set*) *auto*
also have *?this* ⟷ (λ*n*::*nat*. *fps_nth* (*fps_X* ^ *m*) *n* ∗ *z* ^ *n*) *sums* (∑ *n*∈{*m*::*nat*}. *z* ^ *n*)
by (*intro sums_cong*) (*auto simp*: *fps_X_power_iff*)
also have (∑ *n*∈{*m*::*nat*}. *z* ^ *n*) = *z* ^ *m*
by *simp*
finally show *?thesis*
by (*simp add*: *eval_fps_def sums_iff*)
qed

lemma *eval_fps_X* [*simp*]:
eval_fps (*fps_X* :: *'a* :: {*banach*, *real_normed_div_algebra*} *fps*) *z* = *z*
using *eval_fps_X_power*[*of 1 z*] **by** (*simp only*: *power_one_right*)

lemma *eval_fps_minus*:
fixes *f* :: *'a* :: {*banach*, *real_normed_div_algebra*} *fps*
assumes *norm z* < *fps_conv_radius f*
shows *eval_fps* (−*f*) *z* = −*eval_fps f z*
using *assms* **unfolding** *eval_fps_def*
by (*subst suminf_minus* [*symmetric*]) (*auto intro*!: *summable_fps*)

lemma *eval_fps_add*:
fixes *f g* :: *'a* :: {*banach*, *real_normed_div_algebra*} *fps*
assumes *norm z* < *fps_conv_radius f norm z* < *fps_conv_radius g*
shows *eval_fps* (*f* + *g*) *z* = *eval_fps f z* + *eval_fps g z*
using *assms* **unfolding** *eval_fps_def*
by (*subst suminf_add*) (*auto simp*: *ring_distribs intro*!: *summable_fps*)

lemma *eval_fps_diff*:
fixes *f g* :: *'a* :: {*banach*, *real_normed_div_algebra*} *fps*
assumes *norm z* < *fps_conv_radius f norm z* < *fps_conv_radius g*
shows *eval_fps* (*f* − *g*) *z* = *eval_fps f z* − *eval_fps g z*

```
  using assms unfolding eval_fps_def
  by (subst suminf_diff) (auto simp: ring_distribs intro!: summable_fps)

lemma eval_fps_mult:
  fixes f g :: 'a :: {banach, real_normed_div_algebra, comm_ring_1} fps
  assumes norm z < fps_conv_radius f norm z < fps_conv_radius g
  shows   eval_fps (f * g) z = eval_fps f z * eval_fps g z
proof -
  have eval_fps f z * eval_fps g z =
         (∑k. ∑i≤k. fps_nth f i * fps_nth g (k - i) * (z ^ i * z ^ (k - i)))
    unfolding eval_fps_def
  proof (subst Cauchy_product)
    show summable (λk. norm (fps_nth f k * z ^ k)) summable (λk. norm (fps_nth
g k * z ^ k))
      by (rule norm_summable_fps assms)+
  qed (simp_all add: algebra_simps)
  also have (λk. ∑i≤k. fps_nth f i * fps_nth g (k - i) * (z ^ i * z ^ (k - i))) =
              (λk. ∑i≤k. fps_nth f i * fps_nth g (k - i) * z ^ k)
    by (intro ext sum.cong refl) (simp add: power_add [symmetric])
  also have suminf ... = eval_fps (f * g) z
    by (simp add: eval_fps_def fps_mult_nth atLeast0AtMost sum_distrib_right)
  finally show ?thesis ..
qed

lemma eval_fps_shift:
  fixes f :: 'a :: {banach, real_normed_div_algebra, comm_ring_1} fps
  assumes n ≤ subdegree f norm z < fps_conv_radius f
  shows   eval_fps (fps_shift n f) z = (if z = 0 then fps_nth f n else eval_fps f z
/ z ^ n)
proof (cases z = 0)
  case False
  have eval_fps (fps_shift n f * fps_X ^ n) z = eval_fps (fps_shift n f) z * z ^ n
    using assms by (subst eval_fps_mult) simp_all
  also from assms have fps_shift n f * fps_X ^ n = f
    by (simp add: fps_shift_times_fps_X_power)
  finally show ?thesis using False by (simp add: field_simps)
qed (simp_all add: eval_fps_at_0)

lemma eval_fps_exp [simp]:
  fixes c :: 'a :: {banach, real_normed_field}
  shows eval_fps (fps_exp c) z = exp (c * z) unfolding eval_fps_def exp_def
  by (simp add: eval_fps_def exp_def scaleR_conv_of_real field_split_simps)
```

The case of division is more complicated and will therefore not be handled here. Handling division becomes much more easy using complex analysis, and we will do so once that is available.

6.44.5 Power series expansions of analytic functions

This predicate contains the notion that the given formal power series converges in some disc of positive radius around the origin and is equal to the given complex function there.

This relationship is unique in the sense that no complex function can have more than one formal power series to which it expands, and if two holomorphic functions that are holomorphic on a connected open set around the origin and have the same power series expansion, they must be equal on that set.

More concrete statements about the radius of convergence can usually be made, but for many purposes, the statment that the series converges to the function in some neighbourhood of the origin is enough, and that can be shown almost fully automatically in most cases, as there are straightforward introduction rules to show this.

In particular, when one wants to relate the coefficients of the power series to the values of the derivatives of the function at the origin, or if one wants to approximate the coefficients of the series with the singularities of the function, this predicate is enough.

definition
has_fps_expansion :: ('*a* :: {*banach*,*real_normed_div_algebra*} ⇒ '*a*) ⇒ '*a fps* ⇒ *bool*
(**infixl** *has'_fps'_expansion 60*)
where (*f has_fps_expansion F*) ⟷
fps_conv_radius F > *0* ∧ *eventually* (λ*z*. *eval_fps F z* = *f z*) (*nhds 0*)

named_theorems *fps_expansion_intros*

lemma *has_fps_expansion_schematicI*:
f has_fps_expansion A ⟹ *A* = *B* ⟹ *f has_fps_expansion B*
by *simp*

lemma *fps_nth_fps_expansion*:
fixes *f* :: *complex* ⇒ *complex*
assumes *f has_fps_expansion F*
shows *fps_nth F n* = (*deriv* ^^ *n*) *f 0* / *fact n*
proof −
have *fps_nth F n* = (*deriv* ^^ *n*) (*eval_fps F*) *0* / *fact n*
using *assms* **by** (*intro fps_nth_conv_deriv*) (*auto simp*: *has_fps_expansion_def*)
also have (*deriv* ^^ *n*) (*eval_fps F*) *0* = (*deriv* ^^ *n*) *f 0*
using *assms* **by** (*intro higher_deriv_cong_ev*) (*auto simp*: *has_fps_expansion_def*)
finally show *?thesis* **.**
qed

lemma *has_fps_expansion_imp_continuous*:
fixes *F* :: '*a*::{*real_normed_field*,*banach*} *fps*

assumes *f has_fps_expansion F*
shows *continuous* (*at 0 within A*) *f*
proof −
from *assms* **have** *isCont* (*eval_fps F*) *0*
by (*intro continuous_eval_fps*) (*auto simp*: *has_fps_expansion_def zero_ereal_def*)
also have *?this* ⟷ *isCont f 0* **using** *assms*
by (*intro isCont_cong*) (*auto simp*: *has_fps_expansion_def*)
finally have *isCont f 0* **.**
thus *continuous* (*at 0 within A*) *f*
by (*simp add*: *continuous_at_imp_continuous_within*)
qed

lemma *has_fps_expansion_const* [*simp*, *intro*, *fps_expansion_intros*]:
(λ_. *c*) *has_fps_expansion fps_const c*
by (*auto simp*: *has_fps_expansion_def*)

lemma *has_fps_expansion_0* [*simp*, *intro*, *fps_expansion_intros*]:
(λ_. *0*) *has_fps_expansion 0*
by (*auto simp*: *has_fps_expansion_def*)

lemma *has_fps_expansion_1* [*simp*, *intro*, *fps_expansion_intros*]:
(λ_. *1*) *has_fps_expansion 1*
by (*auto simp*: *has_fps_expansion_def*)

lemma *has_fps_expansion_numeral* [*simp*, *intro*, *fps_expansion_intros*]:
(λ_. *numeral n*) *has_fps_expansion numeral n*
by (*auto simp*: *has_fps_expansion_def*)

lemma *has_fps_expansion_fps_X_power* [*fps_expansion_intros*]:
(λ*x*. *x* ^ *n*) *has_fps_expansion* (*fps_X* ^ *n*)
by (*auto simp*: *has_fps_expansion_def*)

lemma *has_fps_expansion_fps_X* [*fps_expansion_intros*]:
(λ*x*. *x*) *has_fps_expansion fps_X*
by (*auto simp*: *has_fps_expansion_def*)

lemma *has_fps_expansion_cmult_left* [*fps_expansion_intros*]:
fixes *c* :: *'a* :: {*banach*, *real_normed_div_algebra*, *comm_ring_1*}
assumes *f has_fps_expansion F*
shows (λ*x*. *c* ∗ *f x*) *has_fps_expansion fps_const c* ∗ *F*
proof (*cases c* = *0*)
case *False*
from *assms* **have** *eventually* (λ*z*. *z* ∈ *eball 0* (*fps_conv_radius F*)) (*nhds 0*)
by (*intro eventually_nhds_in_open*) (*auto simp*: *has_fps_expansion_def zero_ereal_def*)
moreover from *assms* **have** *eventually* (λ*z*. *eval_fps F z* = *f z*) (*nhds 0*)
by (*auto simp*: *has_fps_expansion_def*)
ultimately have *eventually* (λ*z*. *eval_fps* (*fps_const c* ∗ *F*) *z* = *c* ∗ *f z*) (*nhds 0*)
by *eventually_elim* (*simp_all add*: *eval_fps_mult*)

```
  with assms and False show ?thesis
    by (auto simp: has_fps_expansion_def fps_conv_radius_cmult_left)
qed auto

lemma has_fps_expansion_cmult_right [fps_expansion_intros]:
  fixes c :: 'a :: {banach, real_normed_div_algebra, comm_ring_1}
  assumes f has_fps_expansion F
  shows   (λx. f x * c) has_fps_expansion F * fps_const c
proof -
  have F * fps_const c = fps_const c * F
    by (intro fps_ext) (auto simp: mult.commute)
  with has_fps_expansion_cmult_left [OF assms] show ?thesis
    by (simp add: mult.commute)
qed

lemma has_fps_expansion_minus [fps_expansion_intros]:
  assumes f has_fps_expansion F
  shows   (λx. - f x) has_fps_expansion -F
proof -
  from assms have eventually (λx. x ∈ eball 0 (fps_conv_radius F)) (nhds 0)
   by (intro eventually_nhds_in_open) (auto simp: has_fps_expansion_def zero_ereal_def)
  moreover from assms have eventually (λx. eval_fps F x = f x) (nhds 0)
    by (auto simp: has_fps_expansion_def)
  ultimately have eventually (λx. eval_fps (-F) x = -f x) (nhds 0)
    by eventually_elim (auto simp: eval_fps_minus)
  thus ?thesis using assms by (auto simp: has_fps_expansion_def)
qed

lemma has_fps_expansion_add [fps_expansion_intros]:
  assumes f has_fps_expansion F g has_fps_expansion G
  shows   (λx. f x + g x) has_fps_expansion F + G
proof -
  from assms have 0 < min (fps_conv_radius F) (fps_conv_radius G)
    by (auto simp: has_fps_expansion_def)
  also have ... ≤ fps_conv_radius (F + G)
    by (rule fps_conv_radius_add)
  finally have radius: ... > 0 .

  from assms have eventually (λx. x ∈ eball 0 (fps_conv_radius F)) (nhds 0)
                  eventually (λx. x ∈ eball 0 (fps_conv_radius G)) (nhds 0)
   by (intro eventually_nhds_in_open; force simp: has_fps_expansion_def zero_ereal_def)+
  moreover have eventually (λx. eval_fps F x = f x) (nhds 0)
           and eventually (λx. eval_fps G x = g x) (nhds 0)
    using assms by (auto simp: has_fps_expansion_def)
  ultimately have eventually (λx. eval_fps (F + G) x = f x + g x) (nhds 0)
    by eventually_elim (auto simp: eval_fps_add)
  with radius show ?thesis by (auto simp: has_fps_expansion_def)
qed
```

lemma *has_fps_expansion_diff* [*fps_expansion_intros*]:
assumes *f has_fps_expansion F g has_fps_expansion G*
shows (λ*x*. *f x* − *g x*) *has_fps_expansion F* − *G*
using *has_fps_expansion_add*[*of f F* λ*x*. − *g x* −*G*] *assms*
by (*simp add*: *has_fps_expansion_minus*)

lemma *has_fps_expansion_mult* [*fps_expansion_intros*]:
fixes *F G* :: *'a* :: {*banach*, *real_normed_div_algebra*, *comm_ring_1*} *fps*
assumes *f has_fps_expansion F g has_fps_expansion G*
shows (λ*x*. *f x* ∗ *g x*) *has_fps_expansion F* ∗ *G*
proof −
from *assms* **have** *0* < *min* (*fps_conv_radius F*) (*fps_conv_radius G*)
by (*auto simp*: *has_fps_expansion_def*)
also have ... ≤ *fps_conv_radius* (*F* ∗ *G*)
by (*rule fps_conv_radius_mult*)
finally have *radius*: ... > *0* **.**

from *assms* **have** *eventually* (λ*x*. *x* ∈ *eball 0* (*fps_conv_radius F*)) (*nhds 0*)
eventually (λ*x*. *x* ∈ *eball 0* (*fps_conv_radius G*)) (*nhds 0*)
by (*intro eventually_nhds_in_open*; *force simp*: *has_fps_expansion_def zero_ereal_def*)+
moreover have *eventually* (λ*x*. *eval_fps F x* = *f x*) (*nhds 0*)
and *eventually* (λ*x*. *eval_fps G x* = *g x*) (*nhds 0*)
using *assms* **by** (*auto simp*: *has_fps_expansion_def*)
ultimately have *eventually* (λ*x*. *eval_fps* (*F* ∗ *G*) *x* = *f x* ∗ *g x*) (*nhds 0*)
by *eventually_elim* (*auto simp*: *eval_fps_mult*)
with *radius* **show** *?thesis* **by** (*auto simp*: *has_fps_expansion_def*)
qed

lemma *has_fps_expansion_inverse* [*fps_expansion_intros*]:
fixes *F* :: *'a* :: {*banach*, *real_normed_field*} *fps*
assumes *f has_fps_expansion F*
assumes *fps_nth F 0* ≠ *0*
shows (λ*x*. *inverse* (*f x*)) *has_fps_expansion inverse F*
proof −
have *radius*: *fps_conv_radius* (*inverse F*) > *0*
using *assms* **unfolding** *has_fps_expansion_def*
by (*intro fps_conv_radius_inverse_pos*) *auto*
let *?R* = *min* (*fps_conv_radius F*) (*fps_conv_radius* (*inverse F*))
from *assms radius*
have *eventually* (λ*x*. *x* ∈ *eball 0* (*fps_conv_radius F*)) (*nhds 0*)
eventually (λ*x*. *x* ∈ *eball 0* (*fps_conv_radius* (*inverse F*))) (*nhds 0*)
by (*intro eventually_nhds_in_open*; *force simp*: *has_fps_expansion_def zero_ereal_def*)+
moreover have *eventually* (λ*z*. *eval_fps F z* = *f z*) (*nhds 0*)
using *assms* **by** (*auto simp*: *has_fps_expansion_def*)
ultimately have *eventually* (λ*z*. *eval_fps* (*inverse F*) *z* = *inverse* (*f z*)) (*nhds 0*)
proof *eventually_elim*
case (*elim z*)
hence *eval_fps* (*inverse F* ∗ *F*) *z* = *eval_fps* (*inverse F*) *z* ∗ *f z*

```
    by (subst eval_fps_mult) auto
  also have eval_fps (inverse F * F) z = 1
    using assms by (simp add: inverse_mult_eq_1)
  finally show ?case by (auto simp: field_split_simps)
 qed
 with radius show ?thesis by (auto simp: has_fps_expansion_def)
qed

lemma has_fps_expansion_exp [fps_expansion_intros]:
 fixes c :: 'a :: {banach, real_normed_field}
 shows (λx. exp (c * x)) has_fps_expansion fps_exp c
 by (auto simp: has_fps_expansion_def)

lemma has_fps_expansion_exp1 [fps_expansion_intros]:
 (λx::'a :: {banach, real_normed_field}. exp x) has_fps_expansion fps_exp 1
 using has_fps_expansion_exp[of 1] by simp

lemma has_fps_expansion_exp_neg1 [fps_expansion_intros]:
  (λx::'a :: {banach, real_normed_field}. exp (-x)) has_fps_expansion fps_exp
(-1)
 using has_fps_expansion_exp[of -1] by simp

lemma has_fps_expansion_deriv [fps_expansion_intros]:
 assumes f has_fps_expansion F
 shows   deriv f has_fps_expansion fps_deriv F
proof -
 have eventually (λz. z ∈ eball 0 (fps_conv_radius F)) (nhds 0)
   using assms by (intro eventually_nhds_in_open)
                  (auto simp: has_fps_expansion_def zero_ereal_def)
 moreover from assms have eventually (λz. eval_fps F z = f z) (nhds 0)
   by (auto simp: has_fps_expansion_def)
 then obtain s where open s 0 ∈ s and s: ⋀w. w ∈ s ⟹ eval_fps F w = f w
   by (auto simp: eventually_nhds)
 hence eventually (λw. w ∈ s) (nhds 0)
   by (intro eventually_nhds_in_open) auto
 ultimately have eventually (λz. eval_fps (fps_deriv F) z = deriv f z) (nhds 0)
 proof eventually_elim
   case (elim z)
   hence eval_fps (fps_deriv F) z = deriv (eval_fps F) z
     by (simp add: eval_fps_deriv)
   also have eventually (λw. w ∈ s) (nhds z)
     using elim and ‹open s› by (intro eventually_nhds_in_open) auto
   hence eventually (λw. eval_fps F w = f w) (nhds z)
     by eventually_elim (simp add: s)
   hence deriv (eval_fps F) z = deriv f z
     by (intro deriv_cong_ev refl)
   finally show ?case .
 qed
 with assms and fps_conv_radius_deriv[of F] show ?thesis
```

```
    by (auto simp: has_fps_expansion_def)
qed

lemma fps_conv_radius_binomial:
  fixes c :: 'a :: {real_normed_field,banach}
  shows fps_conv_radius (fps_binomial c) = (if c ∈ ℕ then ∞ else 1)
  unfolding fps_conv_radius_def by (simp add: conv_radius_gchoose)

lemma fps_conv_radius_ln:
  fixes c :: 'a :: {banach, real_normed_field, field_char_0}
  shows fps_conv_radius (fps_ln c) = (if c = 0 then ∞ else 1)
proof (cases c = 0)
  case False
  have conv_radius (λn. 1 / of_nat n :: 'a) = 1
  proof (rule conv_radius_ratio_limit_nonzero)
    show (λn. norm (1 / of_nat n :: 'a) / norm (1 / of_nat (Suc n) :: 'a)) ⟶
1
     using LIMSEQ_Suc_n_over_n by (simp add: norm_divide del: of_nat_Suc)
  qed auto
  also have conv_radius (λn. 1 / of_nat n :: 'a) =
              conv_radius (λn. if n = 0 then 0 else (- 1) ^ (n - 1) / of_nat n ::
'a)
    by (intro conv_radius_cong eventually_mono[OF eventually_gt_at_top[of 0]])
       (simp add: norm_mult norm_divide norm_power)
  finally show ?thesis using False unfolding fps_ln_def
    by (subst fps_conv_radius_cmult_left) (simp_all add: fps_conv_radius_def)
qed (auto simp: fps_ln_def)

lemma fps_conv_radius_ln_nonzero [simp]:
  assumes c ≠ (0 :: 'a :: {banach,real_normed_field,field_char_0})
  shows   fps_conv_radius (fps_ln c) = 1
  using assms by (simp add: fps_conv_radius_ln)

lemma fps_conv_radius_sin [simp]:
  fixes c :: 'a :: {banach, real_normed_field, field_char_0}
  shows fps_conv_radius (fps_sin c) = ∞
proof (cases c = 0)
  case False
  have ∞ = conv_radius (λn. of_real (sin_coeff n) :: 'a)
  proof (rule sym, rule conv_radius_inftyI', rule summable_norm_cancel, goal_cases)
    case (1 z)
    show ?case using summable_norm_sin[of z] by (simp add: norm_mult)
  qed
  also have ... / norm c = conv_radius (λn. c ^ n * of_real (sin_coeff n) :: 'a)
    using False by (subst conv_radius_mult_power) auto
  also have ... = fps_conv_radius (fps_sin c) unfolding fps_conv_radius_def
    by (rule conv_radius_cong_weak) (auto simp add: fps_sin_def sin_coeff_def)
  finally show ?thesis by simp
qed simp_all
```

lemma *fps_conv_radius_cos* [*simp*]:
fixes $c :: 'a :: \{banach, real\_normed\_field, field\_char\_0\}$
shows *fps_conv_radius* (*fps_cos c*) = ∞
proof (*cases c = 0*)
case *False*
have ∞ = *conv_radius* ($\lambda n.$ *of_real* (*cos_coeff n*) :: $'a$)
proof (*rule sym*, *rule conv_radius_inftyI''*, *rule summable_norm_cancel*, *goal_cases*)
case (*1 z*)
show *?case* **using** *summable_norm_cos*[*of z*] **by** (*simp add*: *norm_mult*)
qed
also have ... / *norm c* = *conv_radius* ($\lambda n.$ *c* ^ *n* * *of_real* (*cos_coeff n*) :: $'a$)
using *False* **by** (*subst conv_radius_mult_power*) *auto*
also have ... = *fps_conv_radius* (*fps_cos c*) **unfolding** *fps_conv_radius_def*
by (*rule conv_radius_cong_weak*) (*auto simp add*: *fps_cos_def cos_coeff_def*)
finally show *?thesis* **by** *simp*
qed *simp_all*

lemma *eval_fps_sin* [*simp*]:
fixes $z :: 'a :: \{banach, real\_normed\_field, field\_char\_0\}$
shows *eval_fps* (*fps_sin c*) *z* = *sin* (*c* * *z*)
proof −
have ($\lambda n.$ *sin_coeff n* *$_R$ (*c* * *z*) ^ *n*) *sums sin* (*c* * *z*) **by** (*rule sin_converges*)
also have ($\lambda n.$ *sin_coeff n* *$_R$ (*c* * *z*) ^ *n*) = ($\lambda n.$ *fps_nth* (*fps_sin c*) *n* * *z* ^ *n*)
by (*rule ext*) (*auto simp*: *sin_coeff_def fps_sin_def power_mult_distrib scaleR_conv_of_real*)
finally show *?thesis* **by** (*simp add*: *sums_iff eval_fps_def*)
qed

lemma *eval_fps_cos* [*simp*]:
fixes $z :: 'a :: \{banach, real\_normed\_field, field\_char\_0\}$
shows *eval_fps* (*fps_cos c*) *z* = *cos* (*c* * *z*)
proof −
have ($\lambda n.$ *cos_coeff n* *$_R$ (*c* * *z*) ^ *n*) *sums cos* (*c* * *z*) **by** (*rule cos_converges*)
also have ($\lambda n.$ *cos_coeff n* *$_R$ (*c* * *z*) ^ *n*) = ($\lambda n.$ *fps_nth* (*fps_cos c*) *n* * *z* ^ *n*)
by (*rule ext*) (*auto simp*: *cos_coeff_def fps_cos_def power_mult_distrib scaleR_conv_of_real*)
finally show *?thesis* **by** (*simp add*: *sums_iff eval_fps_def*)
qed

lemma *cos_eq_zero_imp_norm_ge*:
assumes *cos* (*z* :: *complex*) = *0*
shows *norm z* ≥ *pi* / *2*
proof −
from *assms* **obtain** *n* **where** *z* = *complex_of_real* ((*of_int n* + *1* / *2*) * *pi*)
by (*auto simp*: *cos_eq_0 algebra_simps*)
also have *norm* ... = |*real_of_int n* + *1* / *2*| * *pi*
by (*subst norm_of_real*) (*simp_all add*: *abs_mult*)
also have *real_of_int n* + *1* / *2* = *of_int* (*2* * *n* + *1*) / *2* **by** *simp*

also have $|\ldots| = \mathit{of\_int}\ |2 * n + 1|\ /\ 2$ **by** (*subst abs_divide*) *simp_all*
also have $\ldots * \mathit{pi} = \mathit{of\_int}\ |2 * n + 1| * (\mathit{pi}\ /\ 2)$ **by** *simp*
also have $\ldots \geq \mathit{of\_int}\ 1 * (\mathit{pi}\ /\ 2)$
by (*intro mult_right_mono*, *subst of_int_le_iff*) (*auto simp*: *abs_if*)
finally show *?thesis* **by** *simp*
qed

lemma *eval_fps_binomial*:
fixes $c :: \mathit{complex}$
assumes $\mathit{norm}\ z < 1$
shows $\mathit{eval\_fps}\ (\mathit{fps\_binomial}\ c)\ z = (1 + z)\ \mathit{powr}\ c$
using *gen_binomial_complex*[*OF assms*] **by** (*simp add*: *sums_iff eval_fps_def*)

lemma *has_fps_expansion_binomial_complex* [*fps_expansion_intros*]:
fixes $c :: \mathit{complex}$
shows $(\lambda x.\ (1 + x)\ \mathit{powr}\ c)\ \mathit{has\_fps\_expansion}\ \mathit{fps\_binomial}\ c$
proof −
have $*$: *eventually* $(\lambda z{::}\mathit{complex}.\ z \in \mathit{eball}\ 0\ 1)\ (\mathit{nhds}\ 0)$
by (*intro eventually_nhds_in_open*) *auto*
thus *?thesis*
by (*auto simp*: *has_fps_expansion_def eval_fps_binomial fps_conv_radius_binomial*
intro!: *eventually_mono* [*OF* $*$])
qed

lemma *has_fps_expansion_sin* [*fps_expansion_intros*]:
fixes $c :: {'a} :: \{\mathit{banach}, \mathit{real\_normed\_field}, \mathit{field\_char\_0}\}$
shows $(\lambda x.\ \mathit{sin}\ (c * x))\ \mathit{has\_fps\_expansion}\ \mathit{fps\_sin}\ c$
by (*auto simp*: *has_fps_expansion_def*)

lemma *has_fps_expansion_sin'* [*fps_expansion_intros*]:
$(\lambda x{::}{'a} :: \{\mathit{banach}, \mathit{real\_normed\_field}\}.\ \mathit{sin}\ x)\ \mathit{has\_fps\_expansion}\ \mathit{fps\_sin}\ 1$
using *has_fps_expansion_sin*[*of 1*] **by** *simp*

lemma *has_fps_expansion_cos* [*fps_expansion_intros*]:
fixes $c :: {'a} :: \{\mathit{banach}, \mathit{real\_normed\_field}, \mathit{field\_char\_0}\}$
shows $(\lambda x.\ \mathit{cos}\ (c * x))\ \mathit{has\_fps\_expansion}\ \mathit{fps\_cos}\ c$
by (*auto simp*: *has_fps_expansion_def*)

lemma *has_fps_expansion_cos'* [*fps_expansion_intros*]:
$(\lambda x{::}{'a} :: \{\mathit{banach}, \mathit{real\_normed\_field}\}.\ \mathit{cos}\ x)\ \mathit{has\_fps\_expansion}\ \mathit{fps\_cos}\ 1$
using *has_fps_expansion_cos*[*of 1*] **by** *simp*

lemma *has_fps_expansion_shift* [*fps_expansion_intros*]:
fixes $F :: {'a} :: \{\mathit{banach}, \mathit{real\_normed\_field}\}\ \mathit{fps}$
assumes $f\ \mathit{has\_fps\_expansion}\ F$ **and** $n \leq \mathit{subdegree}\ F$
assumes $c = \mathit{fps\_nth}\ F\ n$
shows $(\lambda x.\ \mathit{if}\ x = 0\ \mathit{then}\ c\ \mathit{else}\ f\ x\ /\ x\ \hat{}\ n)\ \mathit{has\_fps\_expansion}\ (\mathit{fps\_shift}\ n\ F)$

```
proof −
  have eventually (λx. x ∈ eball 0 (fps_conv_radius F)) (nhds 0)
   using assms by (intro eventually_nhds_in_open) (auto simp: has_fps_expansion_def
zero_ereal_def)
  moreover have eventually (λx. eval_fps F x = f x) (nhds 0)
    using assms by (auto simp: has_fps_expansion_def)
  ultimately have eventually (λx. eval_fps (fps_shift n F) x =
                    (if x = 0 then c else f x / x ^ n)) (nhds 0)
    by eventually_elim (auto simp: eval_fps_shift assms)
  with assms show ?thesis by (auto simp: has_fps_expansion_def)
qed

lemma has_fps_expansion_divide [fps_expansion_intros]:
  fixes F G :: 'a :: {banach, real_normed_field} fps
  assumes f has_fps_expansion F and g has_fps_expansion G and
          subdegree G ≤ subdegree F G ≠ 0
          c = fps_nth F (subdegree G) / fps_nth G (subdegree G)
  shows   (λx. if x = 0 then c else f x / g x) has_fps_expansion (F / G)
proof −
  define n where n = subdegree G
  define F' and G' where F' = fps_shift n F and G' = fps_shift n G
  have F = F' * fps_X ^ n G = G' * fps_X ^ n unfolding F'_def G'_def n_def

    by (rule fps_shift_times_fps_X_power [symmetric] le_refl | fact)+
  moreover from assms have fps_nth G' 0 ≠ 0
    by (simp add: G'_def n_def)
  ultimately have FG: F / G = F' * inverse G'
    by (simp add: fps_divide_unit)

  have (λx. (if x = 0 then fps_nth F n else f x / x ^ n) *
          inverse (if x = 0 then fps_nth G n else g x / x ^ n)) has_fps_expansion
F / G
    (is ?h has_fps_expansion _) unfolding FG F'_def G'_def n_def using ‹G
≠ 0›
    by (intro has_fps_expansion_mult has_fps_expansion_inverse
              has_fps_expansion_shift assms) auto
  also have ?h = (λx. if x = 0 then c else f x / g x)
    using assms(5) unfolding n_def
    by (intro ext) (auto split: if_splits simp: field_simps)
  finally show ?thesis .
qed

lemma has_fps_expansion_divide' [fps_expansion_intros]:
  fixes F G :: 'a :: {banach, real_normed_field} fps
  assumes f has_fps_expansion F and g has_fps_expansion G and fps_nth G 0
≠ 0
  shows   (λx. f x / g x) has_fps_expansion (F / G)
proof −
  have (λx. if x = 0 then fps_nth F 0 / fps_nth G 0 else f x / g x) has_fps_expansion
```

```
(F / G)
  (is ?h has_fps_expansion _) using assms(3) by (intro has_fps_expansion_divide
assms) auto
 also from assms have fps_nth F 0 = f 0 fps_nth G 0 = g 0
  by (auto simp: has_fps_expansion_def eval_fps_at_0 dest: eventually_nhds_x_imp_x)
 hence ?h = (λx. f x / g x) by auto
 finally show ?thesis .
qed

lemma has_fps_expansion_tan [fps_expansion_intros]:
 fixes c :: 'a :: {banach, real_normed_field, field_char_0}
 shows (λx. tan (c * x)) has_fps_expansion fps_tan c
proof −
 have (λx. sin (c * x) / cos (c * x)) has_fps_expansion fps_sin c / fps_cos c
  by (intro fps_expansion_intros) auto
 thus ?thesis by (simp add: tan_def fps_tan_def)
qed

lemma has_fps_expansion_tan' [fps_expansion_intros]:
 tan has_fps_expansion fps_tan (1 :: 'a :: {banach, real_normed_field, field_char_0})
 using has_fps_expansion_tan[of 1] by simp

lemma has_fps_expansion_imp_holomorphic:
 assumes f has_fps_expansion F
 obtains s where open s 0 ∈ s f holomorphic_on s ⋀z. z ∈ s ⟹ f z = eval_fps
F z
proof −
 from assms obtain s where s: open s 0 ∈ s ⋀z. z ∈ s ⟹ eval_fps F z = f z
  unfolding has_fps_expansion_def eventually_nhds by blast
 let ?s' = eball 0 (fps_conv_radius F) ∩ s
 have eval_fps F holomorphic_on ?s'
  by (intro holomorphic_intros) auto
 also have ?this ⟷ f holomorphic_on ?s'
  using s by (intro holomorphic_cong) auto
 finally show ?thesis using s assms
  by (intro that[of ?s']) (auto simp: has_fps_expansion_def zero_ereal_def)
qed

end
```

6.45 Smooth paths

```
theory Smooth_Paths
 imports
 Retracts
begin
```

6.45.1 Homeomorphisms of arc images

lemma *path_connected_arc_complement*:
fixes γ :: *real* $\Rightarrow$ *'a*::*euclidean_space*
assumes *arc* γ *2* $\leq$ *DIM*(*'a*)
shows *path_connected*(− *path_image* γ)
proof −
have *path_image* γ *homeomorphic* {*0..1*::*real*}
by (*simp add*: *assms homeomorphic_arc_image_interval*)
then show *?thesis*
by (*intro path_connected_complement_homeomorphic_convex_compact*) (*auto simp*: *assms*)
qed

lemma *connected_arc_complement*:
fixes γ :: *real* $\Rightarrow$ *'a*::*euclidean_space*
assumes *arc* γ *2* $\leq$ *DIM*(*'a*)
shows *connected*(− *path_image* γ)
by (*simp add*: *assms path_connected_arc_complement path_connected_imp_connected*)

lemma *inside_arc_empty*:
fixes γ :: *real* $\Rightarrow$ *'a*::*euclidean_space*
assumes *arc* γ
shows *inside*(*path_image* γ) = {}
proof (*cases DIM*(*'a*) = *1*)
case *True*
then show *?thesis*
using *assms connected_arc_image connected_convex_1_gen inside_convex* **by** *blast*
next
case *False*
then have *connected* (− *path_image* γ)
by (*metis DIM_ge_Suc0 One_nat_def Suc_1 antisym assms connected_arc_complement not_less_eq_eq*)
then
show *?thesis*
by (*simp add*: *assms bounded_arc_image inside_bounded_complement_connected_empty*)
qed

lemma *inside_simple_curve_imp_closed*:
fixes γ :: *real* $\Rightarrow$ *'a*::*euclidean_space*
shows $[\![$*simple_path* γ; *x* $\in$ *inside*(*path_image* γ)$]\!] \Longrightarrow$ *pathfinish* γ = *pathstart* γ
using *arc_simple_path inside_arc_empty* **by** *blast*

6.45.2 Piecewise differentiability of paths

lemma *continuous_on_joinpaths_D1*:
assumes *continuous_on* {*0..1*} (*g1* +++ *g2*)
shows *continuous_on* {*0..1*} *g1*

proof (*rule continuous_on_eq*)
have *continuous_on* {*0..1/2*} (*g1 +++ g2*)
using *assms continuous_on_subset split_01* **by** *auto*
then show *continuous_on* {*0..1*} (*g1 +++ g2 ∘ (∗) (inverse 2)*)
by (*intro continuous_intros*) *force*
qed (*auto simp*: *joinpaths_def*)

lemma *continuous_on_joinpaths_D2*:
⟦*continuous_on* {*0..1*} (*g1 +++ g2*); *pathfinish g1 = pathstart g2*⟧ ⟹ *continuous_on* {*0..1*} *g2*
using *path_def path_join* **by** *blast*

lemma *piecewise_differentiable_D1*:
assumes (*g1 +++ g2*) *piecewise_differentiable_on* {*0..1*}
shows *g1 piecewise_differentiable_on* {*0..1*}
proof −
obtain *S* **where** *cont*: *continuous_on* {*0..1*} *g1* **and** *finite S*
and *S*: ⋀*x*. *x* ∈ {*0..1*} − *S* ⟹ *g1 +++ g2 differentiable at x within* {*0..1*}
using *assms* **unfolding** *piecewise_differentiable_on_def*
by (*blast dest*!: *continuous_on_joinpaths_D1*)
show *?thesis*
unfolding *piecewise_differentiable_on_def*
proof (*intro exI conjI ballI cont*)
show *finite* (*insert 1* (((∗)*2*) ‘ *S*))
by (*simp add*: ‹*finite S*›)
show *g1 differentiable at x within* {*0..1*} **if** *x* ∈ {*0..1*} − *insert 1* ((∗) *2* ‘ *S*)
for *x*
proof (*rule_tac d*=*dist* (*x/2*) (*1/2*) **in** *differentiable_transform_within*)
have *g1 +++ g2 differentiable at* (*x / 2*) *within* {*0..1/2*}
by (*rule differentiable_subset* [*OF S* [*of x/2*]] | *use that* **in** *force*)+
then show *g1 +++ g2 ∘ (∗) (inverse 2) differentiable at x within* {*0..1*}
using *image_affinity_atLeastAtMost_div* [*of 2 0 0*::*real 1*]
by (*auto intro*: *differentiable_chain_within*)
qed (*use that* **in** ‹*auto simp*: *joinpaths_def*›)
qed
qed

lemma *piecewise_differentiable_D2*:
assumes (*g1 +++ g2*) *piecewise_differentiable_on* {*0..1*} **and** *eq*: *pathfinish g1 = pathstart g2*
shows *g2 piecewise_differentiable_on* {*0..1*}
proof −
have [*simp*]: *g1 1 = g2 0*
using *eq* **by** (*simp add*: *pathfinish_def pathstart_def*)
obtain *S* **where** *cont*: *continuous_on* {*0..1*} *g2* **and** *finite S*
and *S*: ⋀*x*. *x* ∈ {*0..1*} − *S* ⟹ *g1 +++ g2 differentiable at x within* {*0..1*}
using *assms* **unfolding** *piecewise_differentiable_on_def*
by (*blast dest*!: *continuous_on_joinpaths_D2*)
show *?thesis*

```
    unfolding piecewise_differentiable_on_def
  proof (intro exI conjI ballI cont)
    show finite (insert 0 ((λx. 2*x-1) ‘S))
      by (simp add: ‹finite S›)
     show g2 differentiable at x within {0..1} if x ∈ {0..1} - insert 0 ((λx.
2*x-1) ‘S) for x
    proof (rule_tac d=dist ((x+1)/2) (1/2) in differentiable_transform_within)
      have x2: (x + 1) / 2 ∉ S
        using that
        apply (clarsimp simp: image_iff)
       by (metis add.commute add_diff_cancel_left' mult_2 field_sum_of_halves)
      have g1 +++ g2 ∘ (λx. (x+1) / 2) differentiable at x within {0..1}
     by (rule differentiable_chain_within differentiable_subset [OF S [of (x+1)/2]]
| use x2 that in force)+
      then show g1 +++ g2 ∘ (λx. (x+1) / 2) differentiable at x within {0..1}
        by (auto intro: differentiable_chain_within)
      show (g1 +++ g2 ∘ (λx. (x + 1) / 2)) x' = g2 x' if x' ∈ {0..1} dist x' x
< dist ((x + 1) / 2) (1/2) for x'
      proof -
        have [simp]: (2*x'+2)/2 = x'+1
          by (simp add: field_split_simps)
        show ?thesis
          using that by (auto simp: joinpaths_def)
      qed
    qed (use that in ‹auto simp: joinpaths_def›)
  qed
qed

lemma piecewise_C1_differentiable_D1:
  fixes g1 :: real ⇒ 'a::real_normed_field
  assumes (g1 +++ g2) piecewise_C1_differentiable_on {0..1}
    shows g1 piecewise_C1_differentiable_on {0..1}
proof -
  obtain S where finite S
          and co12: continuous_on ({0..1} - S) (λx. vector_derivative (g1 +++
g2) (at x))
            and g12D: ∀x∈{0..1} - S. g1 +++ g2 differentiable at x
   using assms by (auto simp: piecewise_C1_differentiable_on_def C1_differentiable_on_eq)
  have g1D: g1 differentiable at x if x ∈ {0..1} - insert 1 ((*) 2 ‘ S) for x
  proof (rule differentiable_transform_within)
    show g1 +++ g2 ∘ (*) (inverse 2) differentiable at x
      using that g12D
      unfolding joinpaths_def
      by (intro differentiable_chain_at derivative_intros | force)+
    show ⋀x'. ⟦dist x' x < dist (x/2) (1/2)⟧
         ⟹ (g1 +++ g2 ∘ (*) (inverse 2)) x' = g1 x'
      using that by (auto simp: dist_real_def joinpaths_def)
  qed (use that in ‹auto simp: dist_real_def›)
  have [simp]: vector_derivative (g1 ∘ (*) 2) (at (x/2)) = 2 *R vector_derivative
```

```
g1 (at x)
            if x ∈ {0..1} − insert 1 ((∗) 2 ‘ S) for x
    apply (subst vector_derivative_chain_at)
    using that
    apply (rule derivative_eq_intros g1D | simp)+
    done
  have continuous_on ({0..1/2} − insert (1/2) S) (λx. vector_derivative (g1
+++ g2) (at x))
    using co12 by (rule continuous_on_subset) force
   then have coDhalf: continuous_on ({0..1/2} − insert (1/2) S) (λx. vec-
tor_derivative (g1 ∘ (∗)2) (at x))
  proof (rule continuous_on_eq [OF _ vector_derivative_at])
    show (g1 +++ g2 has_vector_derivative vector_derivative (g1 ∘ (∗) 2) (at
x)) (at x)
      if x ∈ {0..1/2} − insert (1/2) S for x
    proof (rule has_vector_derivative_transform_within)
      show (g1 ∘ (∗) 2 has_vector_derivative vector_derivative (g1 ∘ (∗) 2) (at
x)) (at x)
       using that
       by (force intro: g1D differentiable_chain_at simp: vector_derivative_works
[symmetric])
     show ⋀x′. ⟦dist x′ x < dist x (1/2)⟧ ⟹ (g1 ∘ (∗) 2) x′ = (g1 +++ g2) x′
       using that by (auto simp: dist_norm joinpaths_def)
    qed (use that in ‹auto simp: dist_norm›)
  qed
  have continuous_on ({0..1} − insert 1 ((∗) 2 ‘ S))
                     ((λx. 1/2 ∗ vector_derivative (g1 ∘ (∗)2) (at x)) ∘ (∗)(1/2))
    using coDhalf
    apply (intro continuous_intros)
    by (simp add: scaleR_conv_of_real image_set_diff image_image)
   then have con_g1: continuous_on ({0..1} − insert 1 ((∗) 2 ‘ S)) (λx. vec-
tor_derivative g1 (at x))
    by (rule continuous_on_eq) (simp add: scaleR_conv_of_real)
  have continuous_on {0..1} g1
   using continuous_on_joinpaths_D1 assms piecewise_C1_differentiable_on_def
by blast
  with ‹finite S› show ?thesis
   apply (clarsimp simp add: piecewise_C1_differentiable_on_def C1_differentiable_on_eq)
    apply (rule_tac x=insert 1 (((∗)2)‘S) in exI)
    apply (simp add: g1D con_g1)
  done
qed

lemma piecewise_C1_differentiable_D2:
  fixes g2 :: real ⇒ 'a::real_normed_field
  assumes (g1 +++ g2) piecewise_C1_differentiable_on {0..1} pathfinish g1 =
pathstart g2
    shows g2 piecewise_C1_differentiable_on {0..1}
proof −
```

```
  obtain S where finite S
          and co12: continuous_on ({0..1} - S) (λx. vector_derivative (g1 +++
g2) (at x))
          and g12D: ∀x∈{0..1} - S. g1 +++ g2 differentiable at x
   using assms by (auto simp: piecewise_C1_differentiable_on_def C1_differentiable_on_eq)
  have g2D: g2 differentiable at x if x ∈ {0..1} - insert 0 ((λx. 2*x-1) ` S) for
x
  proof (rule differentiable_transform_within)
    show g1 +++ g2 ∘ (λx. (x + 1) / 2) differentiable at x
      using g12D that
      unfolding joinpaths_def
      apply (drule_tac x=(x+1) / 2 in bspec, force simp: field_split_simps)
      apply (rule differentiable_chain_at derivative_intros | force)+
      done
    show ⋀x'. dist x' x < dist ((x + 1) / 2) (1/2) ⟹ (g1 +++ g2 ∘ (λx. (x +
1) / 2)) x' = g2 x'
      using that by (auto simp: dist_real_def joinpaths_def field_simps)
    qed (use that in ‹auto simp: dist_norm›)
   have [simp]: vector_derivative (g2 ∘ (λx. 2*x-1)) (at ((x+1)/2)) = 2 *R
vector_derivative g2 (at x)
             if x ∈ {0..1} - insert 0 ((λx. 2*x-1) ` S) for x
    using that by (auto simp: vector_derivative_chain_at field_split_simps g2D)
   have continuous_on ({1/2..1} - insert (1/2) S) (λx. vector_derivative (g1
+++ g2) (at x))
    using co12 by (rule continuous_on_subset) force
   then have coDhalf: continuous_on ({1/2..1} - insert (1/2) S) (λx. vec-
tor_derivative (g2 ∘ (λx. 2*x-1)) (at x))
  proof (rule continuous_on_eq [OF _ vector_derivative_at])
    show (g1 +++ g2 has_vector_derivative vector_derivative (g2 ∘ (λx. 2 * x
- 1)) (at x))
         (at x)
      if x ∈ {1 / 2..1} - insert (1 / 2) S for x
   proof (rule_tac f=g2 ∘ (λx. 2*x-1) and d=dist (3/4) ((x+1)/2) in has_vector_derivative_transform_within)
      show (g2 ∘ (λx. 2 * x - 1) has_vector_derivative vector_derivative (g2 ∘
(λx. 2 * x - 1)) (at x))
           (at x)
     using that by (force intro: g2D differentiable_chain_at simp: vector_derivative_works
[symmetric])
      show ⋀x'. ⟦dist x' x < dist (3 / 4) ((x + 1) / 2)⟧ ⟹ (g2 ∘ (λx. 2 * x -
1)) x' = (g1 +++ g2) x'
       using that by (auto simp: dist_norm joinpaths_def add_divide_distrib)
    qed (use that in ‹auto simp: dist_norm›)
  qed
  have [simp]: ((λx. (x+1) / 2) ` ({0..1} - insert 0 ((λx. 2 * x - 1) ` S))) =
({1/2..1} - insert (1/2) S)
    apply (simp add: image_set_diff inj_on_def image_image)
    apply (auto simp: image_affinity_atLeastAtMost_div add_divide_distrib)
    done
  have continuous_on ({0..1} - insert 0 ((λx. 2*x-1) ` S))
```

((λx. 1/2 ∗ vector_derivative (g2 ∘ (λx. 2∗x−1)) (at x)) ∘ (λx. (x+1)/2))
by (*rule continuous_intros* | *simp add*: *coDhalf*)+
then have *con_g2*: *continuous_on* ({*0..1*} − *insert 0* ((λx. *2∗x−1*) ‘ *S*)) (λx. *vector_derivative g2* (*at x*))
by (*rule continuous_on_eq*) (*simp add*: *scaleR_conv_of_real*)
have *continuous_on* {*0..1*} *g2*
using *continuous_on_joinpaths_D2 assms piecewise_C1_differentiable_on_def*
by *blast*
with ‹*finite S*› **show** *?thesis*
by (*meson C1_differentiable_on_eq con_g2 finite_imageI finite_insert g2D piecewise_C1_differentiable_on_def*)
qed

6.45.3 Valid paths, and their start and finish

definition *valid_path* :: (*real* ⇒ *'a* :: *real_normed_vector*) ⇒ *bool*
where *valid_path f* ≡ *f piecewise_C1_differentiable_on* {*0..1*::*real*}

definition *closed_path* :: (*real* ⇒ *'a* :: *real_normed_vector*) ⇒ *bool*
where *closed_path g* ≡ *g 0* = *g 1*

In particular, all results for paths apply

lemma *valid_path_imp_path*: *valid_path g* ⟹ *path g*
by (*simp add*: *path_def piecewise_C1_differentiable_on_def valid_path_def*)

lemma *connected_valid_path_image*: *valid_path g* ⟹ *connected*(*path_image g*)
by (*metis connected_path_image valid_path_imp_path*)

lemma *compact_valid_path_image*: *valid_path g* ⟹ *compact*(*path_image g*)
by (*metis compact_path_image valid_path_imp_path*)

lemma *bounded_valid_path_image*: *valid_path g* ⟹ *bounded*(*path_image g*)
by (*metis bounded_path_image valid_path_imp_path*)

lemma *closed_valid_path_image*: *valid_path g* ⟹ *closed*(*path_image g*)
by (*metis closed_path_image valid_path_imp_path*)

lemma *valid_path_translation_eq*: *valid_path* ((+)*d* ∘ *p*) ⟷ *valid_path p*
by (*simp add*: *valid_path_def piecewise_C1_differentiable_on_translation_eq*)

lemma *valid_path_compose*:
assumes *valid_path g*
and *der*: ⋀*x*. *x* ∈ *path_image g* ⟹ *f field_differentiable* (*at x*)
and *con*: *continuous_on* (*path_image g*) (*deriv f*)
shows *valid_path* (*f* ∘ *g*)
proof −
obtain *S* **where** *finite S* **and** *g_diff*: *g C1_differentiable_on* {*0..1*} − *S*
using ‹*valid_path g*› **unfolding** *valid_path_def piecewise_C1_differentiable_on_def*

by *auto*
have $f \circ g$ *differentiable at* t **when** $t \in \{0..1\} - S$ **for** t
proof (*rule differentiable_chain_at*)
show g *differentiable at* t **using** ‹*valid_path g*›
by (*meson C1_differentiable_on_eq* ‹g *C1_differentiable_on* $\{0..1\} - S$› *that*)
next
have $g\ t \in$*path_image* g **using** *that DiffD1 image_eqI path_image_def* **by** *metis*
then show f *differentiable at* $(g\ t)$
using *der*[*THEN field_differentiable_imp_differentiable*] **by** *auto*
qed
moreover have *continuous_on* $(\{0..1\} - S)$ $(\lambda x.$ *vector_derivative* $(f \circ g)$ $(at\ x))$
proof (*rule continuous_on_eq* [**where** $f = \lambda x.$ *vector_derivative* g $(at\ x) *$ *deriv* f $(g\ x)$],
rule continuous_intros)
show *continuous_on* $(\{0..1\} - S)$ $(\lambda x.$ *vector_derivative* g $(at\ x))$
using *g_diff C1_differentiable_on_eq* **by** *auto*
next
have *continuous_on* $\{0..1\}$ $(\lambda x.$ *deriv* f $(g\ x))$
using *continuous_on_compose*[*OF _ con*[*unfolded path_image_def*],*unfolded comp_def*]
‹*valid_path g*› *piecewise_C1_differentiable_on_def valid_path_def*
by *blast*
then show *continuous_on* $(\{0..1\} - S)$ $(\lambda x.$ *deriv* f $(g\ x))$
using *continuous_on_subset* **by** *blast*
next
show *vector_derivative* g $(at\ t) *$ *deriv* f $(g\ t) =$ *vector_derivative* $(f \circ g)$ $(at\ t)$
when $t \in \{0..1\} - S$ **for** t
by (*metis C1_differentiable_on_eq DiffD1 der g_diff imageI path_image_def that*
vector_derivative_chain_at_general)
qed
ultimately have $f \circ g$ *C1_differentiable_on* $\{0..1\} - S$
using *C1_differentiable_on_eq* **by** *blast*
moreover have *path* $(f \circ g)$
using *der*
by (*simp add*: *path_continuous_image*[*OF valid_path_imp_path*[*OF* ‹*valid_path g*›]] *continuous_at_imp_continuous_on field_differentiable_imp_continuous_at*)
ultimately show *?thesis* **unfolding** *valid_path_def piecewise_C1_differentiable_on_def path_def*
using ‹*finite S*› **by** *auto*
qed

lemma *valid_path_uminus_comp*[*simp*]:
fixes g::*real* $\Rightarrow$ *'a* ::*real_normed_field*
shows *valid_path* $(uminus \circ g) \longleftrightarrow$ *valid_path* g

proof
show *valid_path g* $\Longrightarrow$ *valid_path* (*uminus* $\circ$ *g*) **for** *g*::*real* $\Rightarrow$ *'a*
by (*auto intro*!: *valid_path_compose derivative_intros*)
then show *valid_path g* **when** *valid_path* (*uminus* $\circ$ *g*)
by (*metis fun.map_comp group_add_class.minus_comp_minus id_comp that*)
qed

lemma *valid_path_offset*[*simp*]:
shows *valid_path* ($\lambda t.\ g\ t - z$) $\longleftrightarrow$ *valid_path g*
proof
show $*$: *valid_path* (*g*::*real*$\Rightarrow$*'a*) $\Longrightarrow$ *valid_path* ($\lambda t.\ g\ t - z$) **for** *g z*
unfolding *valid_path_def*
by (*fastforce intro*:*derivative_intros C1_differentiable_imp_piecewise piecewise_C1_differentiable_diff*)
show *valid_path* ($\lambda t.\ g\ t - z$) $\Longrightarrow$ *valid_path g*
using $*$[*of* $\lambda t.\ g\ t - z$ $-z$,*simplified*] **.**
qed

lemma *valid_path_imp_reverse*:
assumes *valid_path g*
shows *valid_path*(*reversepath g*)
proof −
obtain *S* **where** *finite S* **and** *S*: *g C1_differentiable_on* ({*0..1*} − *S*)
using *assms* **by** (*auto simp*: *valid_path_def piecewise_C1_differentiable_on_def*)
then have *finite* ((−) *1* ‘ *S*)
by *auto*
moreover have (*reversepath g C1_differentiable_on* ({*0..1*} − (−) *1* ‘ *S*))
unfolding *reversepath_def*
apply (*rule C1_differentiable_compose* [*of* λx::*real*. *1*−*x* _ *g*, *unfolded o_def*])
using *S*
by (*force simp*: *finite_vimageI inj_on_def C1_differentiable_on_eq elim*!: *continuous_on_subset*)+
ultimately show *?thesis* **using** *assms*
by (*auto simp*: *valid_path_def piecewise_C1_differentiable_on_def path_def* [*symmetric*])
qed

lemma *valid_path_reversepath* [*simp*]: *valid_path*(*reversepath g*) $\longleftrightarrow$ *valid_path g*
using *valid_path_imp_reverse* **by** *force*

lemma *valid_path_join*:
assumes *valid_path g1 valid_path g2 pathfinish g1* = *pathstart g2*
shows *valid_path*(*g1* +++ *g2*)
proof −
have *g1 1* = *g2 0*
using *assms* **by** (*auto simp*: *pathfinish_def pathstart_def*)
moreover have (*g1* $\circ$ ($\lambda x.\ 2*x$)) *piecewise_C1_differentiable_on* {*0..1/2*}
apply (*rule piecewise_C1_differentiable_compose*)

using *assms*
apply (*auto simp*: *valid_path_def piecewise_C1_differentiable_on_def continuous_on_joinpaths*)
apply (*force intro*: *finite_vimageI* [**where** $h = (*)2$] *inj_onI*)
done
moreover have (*g2* ∘ (λ*x*. *2***x*−*1*)) *piecewise_C1_differentiable_on* {*1/2..1*}
apply (*rule piecewise_C1_differentiable_compose*)
using *assms* **unfolding** *valid_path_def piecewise_C1_differentiable_on_def*
by (*auto intro*!: *continuous_intros finite_vimageI* [**where** *h* = (λ*x*. *2***x* − *1*)] *inj_onI*
simp: *image_affinity_atLeastAtMost_diff continuous_on_joinpaths*)
ultimately show *?thesis*
unfolding *valid_path_def continuous_on_joinpaths joinpaths_def*
by (*intro piecewise_C1_differentiable_cases*) (*auto simp*: *o_def*)
qed

lemma *valid_path_join_D1*:
fixes *g1* :: *real* ⇒ *'a*::*real_normed_field*
shows *valid_path* (*g1* +++ *g2*) ⟹ *valid_path g1*
unfolding *valid_path_def*
by (*rule piecewise_C1_differentiable_D1*)

lemma *valid_path_join_D2*:
fixes *g2* :: *real* ⇒ *'a*::*real_normed_field*
shows ⟦*valid_path* (*g1* +++ *g2*); *pathfinish g1* = *pathstart g2*⟧ ⟹ *valid_path g2*
unfolding *valid_path_def*
by (*rule piecewise_C1_differentiable_D2*)

lemma *valid_path_join_eq* [*simp*]:
fixes *g2* :: *real* ⇒ *'a*::*real_normed_field*
shows *pathfinish g1* = *pathstart g2* ⟹ (*valid_path*(*g1* +++ *g2*) ⟷ *valid_path g1* ∧ *valid_path g2*)
using *valid_path_join_D1 valid_path_join_D2 valid_path_join* **by** *blast*

lemma *valid_path_shiftpath* [*intro*]:
assumes *valid_path g pathfinish g* = *pathstart g a* ∈ {*0..1*}
shows *valid_path*(*shiftpath a g*)
using *assms*
unfolding *valid_path_def shiftpath_alt_def*
apply (*intro piecewise_C1_differentiable_cases*)
apply (*simp_all add*: *add.commute*)
apply (*rule piecewise_C1_differentiable_affine* [*of g 1 a*, *simplified o_def scaleR_one*])
apply (*force simp*: *pathfinish_def pathstart_def elim*: *piecewise_C1_differentiable_on_subset*)
apply (*rule piecewise_C1_differentiable_affine* [*of g 1 a*−*1*, *simplified o_def scaleR_one algebra_simps*])
apply (*auto simp*: *pathfinish_def pathstart_def elim*: *piecewise_C1_differentiable_on_subset*)
done

lemma *vector_derivative_linepath_within*:
  $x \in \{0..1\} \Longrightarrow$ *vector_derivative* (*linepath a b*) (*at x within* $\{0..1\}$) $= b - a$
 by (*simp add*: *has_vector_derivative_linepath_within vector_derivative_at_within_ivl*)

lemma *vector_derivative_linepath_at* [*simp*]: *vector_derivative* (*linepath a b*) (*at x*) $= b - a$
 by (*simp add*: *has_vector_derivative_linepath_within vector_derivative_at*)

lemma *valid_path_linepath* [*iff*]: *valid_path* (*linepath a b*)
 using *C1_differentiable_on_eq piecewise_C1_differentiable_on_def valid_path_def*
by *fastforce*

lemma *valid_path_subpath*:
 fixes *g* :: *real* $\Rightarrow$ *'a* :: *real_normed_vector*
 assumes *valid_path g* $u \in \{0..1\}$ $v \in \{0..1\}$
  shows *valid_path*(*subpath u v g*)
proof (*cases v=u*)
 case *True*
 then show *?thesis*
  unfolding *valid_path_def subpath_def*
  by (*force intro*: *C1_differentiable_on_const C1_differentiable_imp_piecewise*)
next
 case *False*
 let *?f* $= \lambda x.\ ((v-u) * x + u)$
 have (*g* $\circ$ *?f*) *piecewise_C1_differentiable_on* $\{0..1\}$
 proof (*rule piecewise_C1_differentiable_compose*)
  show *?f piecewise_C1_differentiable_on* $\{0..1\}$
   by (*simp add*: *C1_differentiable_imp_piecewise*)
  have *g piecewise_C1_differentiable_on* (*if* $u \le v$ *then* $\{u..v\}$ *else* $\{v..u\}$)
   using *assms piecewise_C1_differentiable_on_subset valid_path_def* **by** *force*
  then show *g piecewise_C1_differentiable_on ?f* ‘ $\{0..1\}$
   by (*simp add*: *image_affinity_atLeastAtMost split*: *if_split_asm*)
  show $\bigwedge x.$ *finite* ($\{0..1\} \cap$ *?f* $-$‘ $\{x\}$)
   using *False*
    by (*simp add*: *Int_commute* [*of* $\{0..1\}$] *inj_on_def crossproduct_eq finite_vimage_IntI*)
 qed
 then show *?thesis*
  by (*auto simp*: *o_def valid_path_def subpath_def*)
qed

lemma *valid_path_rectpath* [*simp*, *intro*]: *valid_path* (*rectpath a b*)
 by (*simp add*: *Let_def rectpath_def*)

end

6.46 Metrics on product spaces

theory *Function_Metric*
imports
Function_Topology
Elementary_Metric_Spaces
begin

In general, the product topology is not metrizable, unless the index set is countable. When the index set is countable, essentially any (convergent) combination of the metrics on the factors will do. We use below the simplest one, based on L^1, but L^2 would also work, for instance.

What is not completely trivial is that the distance thus defined induces the same topology as the product topology. This is what we have to prove to show that we have an instance of *metric_space*.

The proofs below would work verbatim for general countable products of metric spaces. However, since distances are only implemented in terms of type classes, we only develop the theory for countable products of the same space.

instantiation *fun* :: (*countable*, *metric_space*) *metric_space*
begin

definition *dist_fun_def*:
dist x y = (∑ *n*. (*1*/*2*)^*n* * *min* (*dist* (*x* (*from_nat n*)) (*y* (*from_nat n*))) *1*)

definition *uniformity_fun_def*:
(*uniformity*::(('*a* ⇒ '*b*) × ('*a* ⇒ '*b*)) *filter*) = (*INF e*∈{*0*<..}. *principal* {(*x*, *y*). *dist* (*x*::('*a*⇒'*b*)) *y* < *e*})

Except for the first one, the auxiliary lemmas below are only useful when proving the instance: once it is proved, they become trivial consequences of the general theory of metric spaces. It would thus be desirable to hide them once the instance is proved, but I do not know how to do this.

lemma *dist_fun_le_dist_first_terms*:
dist x y ≤ *2* * *Max* {*dist* (*x* (*from_nat n*)) (*y* (*from_nat n*))|*n*. *n* ≤ *N*} + (*1*/*2*)^*N*
proof −
have (∑ *n*. (*1* / *2*) ^ (*n*+*Suc N*) * *min* (*dist* (*x* (*from_nat* (*n*+*Suc N*))) (*y* (*from_nat* (*n*+*Suc N*)))) *1*)
= (∑ *n*. (*1* / *2*) ^ (*Suc N*) * ((*1*/*2*) ^ *n* * *min* (*dist* (*x* (*from_nat* (*n*+*Suc N*))) (*y* (*from_nat* (*n*+*Suc N*)))) *1*))
by (*rule suminf_cong*, *simp add*: *power_add*)
also have ... = (*1*/*2*)^(*Suc N*) * (∑ *n*. (*1* / *2*) ^ *n* * *min* (*dist* (*x* (*from_nat* (*n*+*Suc N*))) (*y* (*from_nat* (*n*+*Suc N*)))) *1*)
apply (*rule suminf_mult*)
by (*rule summable_comparison_test'*[*of* λ*n*. (*1*/*2*)^*n*], *auto simp add*: *summable_geometric_iff*)
also have ... ≤ (*1*/*2*)^(*Suc N*) * (∑ *n*. (*1* / *2*) ^ *n*)

```
    apply (simp, rule suminf_le, simp)
   by (rule summable_comparison_test'[of λn. (1/2)^n], auto simp add: summable_geometric_iff)
  also have ... = (1/2)^(Suc N) * 2
    using suminf_geometric[of 1/2] by auto
  also have ... = (1/2)^N
    by auto
  finally have *: (∑ n. (1 / 2) ^ (n+Suc N) * min (dist (x (from_nat (n+Suc
N))) (y (from_nat (n+Suc N)))) 1) ≤ (1/2)^N
    by simp

  define M where M = Max {dist (x (from_nat n)) (y (from_nat n))|n. n ≤ N}
  have dist (x (from_nat 0)) (y (from_nat 0)) ≤ M
    unfolding M_def by (rule Max_ge, auto)
  then have [simp]: M ≥ 0 by (meson dual_order.trans zero_le_dist)
  have dist (x (from_nat n)) (y (from_nat n)) ≤ M if n≤N for n
    unfolding M_def apply (rule Max_ge) using that by auto
  then have i: min (dist (x (from_nat n)) (y (from_nat n))) 1 ≤ M if n≤N for
n
    using that by force
  have (∑ n< Suc N. (1 / 2) ^ n * min (dist (x (from_nat n)) (y (from_nat
n))) 1) ≤
      (∑ n< Suc N. M * (1 / 2) ^ n)
    by (rule sum_mono, simp add: i)
  also have ... = M * (∑ n<Suc N. (1 / 2) ^ n)
    by (rule sum_distrib_left[symmetric])
  also have ... ≤ M * (∑ n. (1 / 2) ^ n)
   by (rule mult_left_mono, rule sum_le_suminf, auto simp add: summable_geometric_iff)
  also have ... = M * 2
    using suminf_geometric[of 1/2] by auto
  finally have **: (∑ n< Suc N. (1 / 2) ^ n * min (dist (x (from_nat n)) (y
(from_nat n))) 1) ≤ 2 * M
    by simp

  have dist x y = (∑ n. (1 / 2) ^ n * min (dist (x (from_nat n)) (y (from_nat
n))) 1)
    unfolding dist_fun_def by simp
  also have ... = (∑ n. (1 / 2) ^ (n+Suc N) * min (dist (x (from_nat (n+Suc
N))) (y (from_nat (n+Suc N)))) 1)
                + (∑ n<Suc N. (1 / 2) ^ n * min (dist (x (from_nat n)) (y
(from_nat n))) 1)
    apply (rule suminf_split_initial_segment)
   by (rule summable_comparison_test'[of λn. (1/2)^n], auto simp add: summable_geometric_iff)
  also have ... ≤ 2 * M + (1/2)^N
    using * ** by auto
  finally show ?thesis unfolding M_def by simp
qed

lemma open_fun_contains_ball_aux:
  assumes open (U::(('a ⇒ 'b) set))
```

```
        x ∈ U
  shows ∃ e>0. {y. dist x y < e} ⊆ U
proof −
  have ∗: openin (product_topology (λi. euclidean) UNIV) U
    using open_fun_def assms by auto
  obtain X where H: Pi_E UNIV X ⊆ U
                   ⋀i. openin euclidean (X i)
                   finite {i. X i ≠ topspace euclidean}
                   x ∈ Pi_E UNIV X
    using product_topology_open_contains_basis[OF ∗ ‹x ∈ U›] by auto
  define I where I = {i. X i ≠ topspace euclidean}
  have finite I unfolding I_def using H(3) by auto
  {
    fix i
    have x i ∈ X i using ‹x ∈ U› H by auto
    then have ∃ e. e>0 ∧ ball (x i) e ⊆ X i
      using ‹openin euclidean (X i)› open_openin open_contains_ball by blast
    then obtain e where e>0 ball (x i) e ⊆ X i by blast
    define f where f = min e (1/2)
    have f>0 f<1 unfolding f_def using ‹e>0› by auto
    moreover have ball (x i) f ⊆ X i unfolding f_def using ‹ball (x i) e ⊆ X
i› by auto
    ultimately have ∃ f. f > 0 ∧ f < 1 ∧ ball (x i) f ⊆ X i by auto
  } note ∗ = this
  have ∃ e. ∀ i. e i > 0 ∧ e i < 1 ∧ ball (x i) (e i) ⊆ X i
    by (rule choice, auto simp add: ∗)
  then obtain e where ⋀i. e i > 0 ⋀i. e i < 1 ⋀i. ball (x i) (e i) ⊆ X i
    by blast
  define m where m = Min {(1/2)^(to_nat i) ∗ e i|i. i ∈ I}
  have m > 0 if I≠{}
    unfolding m_def Min_gr_iff using ‹finite I› ‹I ≠ {}› ‹⋀i. e i > 0› by auto
  moreover have {y. dist x y < m} ⊆ U
  proof (auto)
    fix y assume dist x y < m
    have y i ∈ X i if i ∈ I for i
    proof −
      have ∗: summable (λn. (1/2)^n ∗ min (dist (x (from_nat n)) (y (from_nat
n))) 1)
        by (rule summable_comparison_test'[of λn. (1/2)^n], auto simp add:
summable_geometric_iff)
      define n where n = to_nat i
      have (1/2)^n ∗ min (dist (x (from_nat n)) (y (from_nat n))) 1 < m
        using ‹dist x y < m› unfolding dist_fun_def using sum_le_suminf[OF
∗, of {n}] by auto
      then have (1/2)^(to_nat i) ∗ min (dist (x i) (y i)) 1 < m
        using ‹n = to_nat i› by auto
      also have ... ≤ (1/2)^(to_nat i) ∗ e i
        unfolding m_def apply (rule Min_le) using ‹finite I› ‹i ∈ I› by auto
      ultimately have min (dist (x i) (y i)) 1 < e i
```

```
      by (auto simp add: field_split_simps)
    then have dist (x i) (y i) < e i
      using ‹e i < 1› by auto
    then show y i ∈ X i using ‹ball (x i) (e i) ⊆ X i› by auto
  qed
  then have y ∈ Pi_E UNIV X using H(1) unfolding I_def topspace_euclidean
by (auto simp add: PiE_iff)
  then show y ∈ U using ‹Pi_E UNIV X ⊆ U› by auto
 qed
 ultimately have ∗: ∃ m>0. {y. dist x y < m} ⊆ U if I ≠ {} using that by
auto

 have U = UNIV if I = {}
   using that H(1) unfolding I_def topspace_euclidean by (auto simp add:
PiE_iff)
 then have ∃ m>0. {y. dist x y < m} ⊆ U if I = {} using that zero_less_one
by blast
 then show ∃ m>0. {y. dist x y < m} ⊆ U using ∗ by blast
qed

lemma ball_fun_contains_open_aux:
 fixes x::('a ⇒ 'b) and e::real
 assumes e>0
 shows ∃ U. open U ∧ x ∈ U ∧ U ⊆ {y. dist x y < e}
proof −
 have ∃ N::nat. 2^N > 8/e
  by (simp add: real_arch_pow)
 then obtain N::nat where 2^N > 8/e by auto
 define f where f = e/4
 have [simp]: e>0 f > 0 unfolding f_def using assms by auto
 define X::('a ⇒ 'b set) where X = (λi. if (∃ n≤N. i = from_nat n) then {z.
dist (x i) z < f} else UNIV)
 have {i. X i ≠ UNIV} ⊆ from_nat‘{0..N}
  unfolding X_def by auto
 then have finite {i. X i ≠ topspace euclidean}
  unfolding topspace_euclidean using finite_surj by blast
 have ⋀i. open (X i)
  unfolding X_def using metric_space_class.open_ball open_UNIV by auto
 then have ⋀i. openin euclidean (X i)
  using open_openin by auto
 define U where U = Pi_E UNIV X
 have open U
  unfolding open_fun_def product_topology_def apply (rule topology_generated_by_Basis)
  unfolding U_def using ‹⋀i. openin euclidean (X i)› ‹finite {i. X i ≠ topspace
euclidean}›
  by auto
 moreover have x ∈ U
  unfolding U_def X_def by (auto simp add: PiE_iff)
 moreover have dist x y < e if y ∈ U for y
```

```
  proof −
    have ∗: dist (x (from_nat n)) (y (from_nat n)) ≤ f if n ≤ N for n
      using ‹y ∈ U› unfolding U_def apply (auto simp add: PiE_iff)
      unfolding X_def using that by (metis less_imp_le mem_Collect_eq)
    have ∗∗: Max {dist (x (from_nat n)) (y (from_nat n))|n. n ≤ N} ≤ f
      apply (rule Max.boundedI) using ∗ by auto

    have dist x y ≤ 2 ∗ Max {dist (x (from_nat n)) (y (from_nat n))|n. n ≤ N}
+ (1/2)^N
      by (rule dist_fun_le_dist_first_terms)
    also have ... ≤ 2 ∗ f + e / 8
    apply (rule add_mono) using ∗∗ ‹2^N > 8/e› by(auto simp add: field_split_simps)
    also have ... ≤ e/2 + e/8
      unfolding f_def by auto
    also have ... < e
      by auto
    finally show dist x y < e by simp
  qed
  ultimately show ?thesis by auto
qed

lemma fun_open_ball_aux:
  fixes U::('a ⇒ 'b) set
  shows open U ⟷ (∀ x∈U. ∃ e>0. ∀ y. dist x y < e ⟶ y ∈ U)
proof (auto)
  assume open U
  fix x assume x ∈ U
  then show ∃ e>0. ∀ y. dist x y < e ⟶ y ∈ U
    using open_fun_contains_ball_aux[OF ‹open U› ‹x ∈ U›] by auto
next
  assume H: ∀ x∈U. ∃ e>0. ∀ y. dist x y < e ⟶ y ∈ U
  define K where K = {V. open V ∧ V ⊆ U}
  {
    fix x assume x ∈ U
    then obtain e where e: e>0 {y. dist x y < e} ⊆ U using H by blast
    then obtain V where V: open V x ∈ V V ⊆ {y. dist x y < e}
      using ball_fun_contains_open_aux[OF ‹e>0›, of x] by auto
    have V ∈ K
      unfolding K_def using e(2) V(1) V(3) by auto
    then have x ∈ (⋃ K) using ‹x ∈ V› by auto
  }
  then have (⋃ K) = U
    unfolding K_def by auto
  moreover have open (⋃ K)
    unfolding K_def by auto
  ultimately show open U by simp
qed

instance proof
```

```
  fix x y::'a ⇒ 'b show (dist x y = 0) = (x = y)
  proof
    assume x = y
    then show dist x y = 0 unfolding dist_fun_def using ‹x = y› by auto
  next
    assume dist x y = 0
    have ∗: summable (λn. (1/2)^n ∗ min (dist (x (from_nat n)) (y (from_nat
n))) 1)
    by (rule summable_comparison_test'[of λn. (1/2)^n], auto simp add: summable_geometric_iff)
    have (1/2)^n ∗ min (dist (x (from_nat n)) (y (from_nat n))) 1 = 0 for n
        using ‹dist x y = 0› unfolding dist_fun_def by (simp add: ∗ sum-
inf_eq_zero_iff)
    then have dist (x (from_nat n)) (y (from_nat n)) = 0 for n
      by (metis div_0 min_def nonzero_mult_div_cancel_left power_eq_0_iff
            zero_eq_1_divide_iff zero_neq_numeral)
    then have x (from_nat n) = y (from_nat n) for n
      by auto
    then have x i = y i for i
      by (metis from_nat_to_nat)
    then show x = y
      by auto
  qed
next
```

The proof of the triangular inequality is trivial, modulo the fact that we are dealing with infinite series, hence we should justify the convergence at each step. In this respect, the following simplification rule is extremely handy.

```
  have [simp]: summable (λn. (1/2)^n ∗ min (dist (u (from_nat n)) (v (from_nat
n))) 1) for u v::'a ⇒ 'b
   by (rule summable_comparison_test'[of λn. (1/2)^n], auto simp add: summable_geometric_iff)
  fix x y z::'a ⇒ 'b
  {
    fix n
    have ∗: dist (x (from_nat n)) (y (from_nat n)) ≤
              dist (x (from_nat n)) (z (from_nat n)) + dist (y (from_nat n)) (z
(from_nat n))
      by (simp add: dist_triangle2)
    have 0 ≤ dist (y (from_nat n)) (z (from_nat n))
      using zero_le_dist by blast
    moreover
    {
     assume min (dist (y (from_nat n)) (z (from_nat n))) 1 ≠ dist (y (from_nat
n)) (z (from_nat n))
      then have 1 ≤ min (dist (x (from_nat n)) (z (from_nat n))) 1 + min (dist
(y (from_nat n)) (z (from_nat n))) 1
      by (metis (no_types) diff_le_eq diff_self min_def zero_le_dist zero_le_one)
    }
    ultimately have min (dist (x (from_nat n)) (y (from_nat n))) 1 ≤
          min (dist (x (from_nat n)) (z (from_nat n))) 1 + min (dist (y (from_nat
```

```
n)) (z (from_nat n))) 1
      using * by linarith
  } note ineq = this
  have dist x y = (∑ n. (1/2)^n * min (dist (x (from_nat n)) (y (from_nat n)))
1)
    unfolding dist_fun_def by simp
  also have ... ≤ (∑ n. (1/2)^n * min (dist (x (from_nat n)) (z (from_nat n)))
1
                      + (1/2)^n * min (dist (y (from_nat n)) (z (from_nat n))) 1)
    apply (rule suminf_le)
     using ineq apply (metis (no_types, opaque_lifting) add.right_neutral dis-
trib_left
    le_divide_eq_numeral1(1) mult_2_right mult_left_mono zero_le_one zero_le_power)
    by (auto simp add: summable_add)
  also have ... = (∑ n. (1/2)^n * min (dist (x (from_nat n)) (z (from_nat n)))
1)
                + (∑ n. (1/2)^n * min (dist (y (from_nat n)) (z (from_nat n)))
1)
    by (rule suminf_add[symmetric], auto)
  also have ... = dist x z + dist y z
    unfolding dist_fun_def by simp
  finally show dist x y ≤ dist x z + dist y z
    by simp
next
```

Finally, we show that the topology generated by the distance and the product topology coincide. This is essentially contained in Lemma *fun_open_ball_aux*, except that the condition to prove is expressed with filters. To deal with this, we copy some mumbo jumbo from Lemma *eventually_uniformity_metric* in `~~/src/HOL/Real_Vector_Spaces.thy`

```
  fix U::('a ⇒ 'b) set
  have eventually P uniformity ⟷ (∃ e>0. ∀ x (y::('a ⇒ 'b)). dist x y < e ⟶
P (x, y)) for P
  unfolding uniformity_fun_def apply (subst eventually_INF_base)
    by (auto simp: eventually_principal subset_eq intro: bexI[of _ min _ _])
  then show open U = (∀ x∈U. ∀ F (x', y) in uniformity. x' = x ⟶ y ∈ U)
    unfolding fun_open_ball_aux by simp
qed (simp add: uniformity_fun_def)

end
```

Nice properties of spaces are preserved under countable products. In addition to first countability, second countability and metrizability, as we have seen above, completeness is also preserved, and therefore being Polish.

```
instance fun :: (countable, complete_space) complete_space
proof
  fix u::nat ⇒ ('a ⇒ 'b) assume Cauchy u
  have Cauchy (λn. u n i) for i
```

```
    unfolding Cauchy_def
  proof (intro strip)
    fix e::real assume e>0
    obtain k where i = from_nat k
      using from_nat_to_nat[of i] by metis
    have (1/2)^k * min e 1 > 0 using ‹e>0› by auto
    then have ∃N. ∀m n. N ≤ m ∧ N ≤ n ⟶ dist (u m) (u n) < (1/2)^k *
min e 1
      using ‹Cauchy u› by (meson Cauchy_def)
    then obtain N::nat where C: ∀m n. N ≤ m ∧ N ≤ n ⟶ dist (u m) (u n)
< (1/2)^k * min e 1
      by blast
    have ∀m n. N ≤ m ∧ N ≤ n ⟶ dist (u m i) (u n i) < e
    proof (auto)
      fix m n::nat assume m ≥ N n ≥ N
      have (1/2)^k * min (dist (u m i) (u n i)) 1
            = sum (λp. (1/2)^p * min (dist (u m (from_nat p)) (u n (from_nat
p))) 1) {k}
        using ‹i = from_nat k› by auto
     also have ... ≤ (∑p. (1/2)^p * min (dist (u m (from_nat p)) (u n (from_nat
p))) 1)
        apply (rule sum_le_suminf)
          by (rule summable_comparison_test'[of λn. (1/2)^n], auto simp add:
summable_geometric_iff)
      also have ... = dist (u m) (u n)
        unfolding dist_fun_def by auto
      also have ... < (1/2)^k * min e 1
        using C ‹m ≥ N› ‹n ≥ N› by auto
      finally have min (dist (u m i) (u n i)) 1 < min e 1
        by (auto simp add: field_split_simps)
      then show dist (u m i) (u n i) < e by auto
    qed
    then show ∃M. ∀m≥M. ∀n≥M. dist (u m i) (u n i) < e
      by blast
  qed
  then have ∃x. (λn. u n i) ⟶ x for i
    using Cauchy_convergent convergent_def by auto
  then have ∃x. ∀i. (λn. u n i) ⟶ x i
    using choice by force
  then obtain x where *: ⋀i. (λn. u n i) ⟶ x i by blast
  have u ⟶ x
  proof (rule metric_LIMSEQ_I)
    fix e assume [simp]: e>(0::real)
    have i: ∃K. ∀n≥K. dist (u n i) (x i) < e/4 for i
      by (rule metric_LIMSEQ_D, auto simp add: *)
    have ∃K. ∀i. ∀n≥K i. dist (u n i) (x i) < e/4
      apply (rule choice) using i by auto
    then obtain K where K: ⋀i n. n ≥ K i ⟹ dist (u n i) (x i) < e/4
      by blast
```

```
    have ∃N::nat. 2^N > 4/e
      by (simp add: real_arch_pow)
    then obtain N::nat where 2^N > 4/e by auto
    define L where L = Max {K (from_nat n)|n. n ≤ N}
    have dist (u k) x < e if k ≥ L for k
    proof −
      have ∗: dist (u k (from_nat n)) (x (from_nat n)) ≤ e / 4 if n ≤ N for n
      proof −
        have K (from_nat n) ≤ L
          unfolding L_def apply (rule Max_ge) using ‹n ≤ N› by auto
        then have k ≥ K (from_nat n) using ‹k ≥ L› by auto
        then show ?thesis using K less_imp_le by auto
      qed
      have ∗∗: Max {dist (u k (from_nat n)) (x (from_nat n)) |n. n ≤ N} ≤ e/4
        apply (rule Max.boundedI) using ∗ by auto
      have dist (u k) x ≤ 2 ∗ Max {dist (u k (from_nat n)) (x (from_nat n)) |n.
n ≤ N} + (1/2)^N
        using dist_fun_le_dist_first_terms by auto
      also have ... ≤ 2 ∗ e/4 + e/4
        apply (rule add_mono)
        using ∗∗ ‹2^N > 4/e› less_imp_le by (auto simp add: field_split_simps)
      also have ... < e by auto
      finally show ?thesis by simp
    qed
    then show ∃L. ∀k≥L. dist (u k) x < e by blast
  qed
  then show convergent u using convergent_def by blast
qed

instance fun :: (countable, polish_space) polish_space
  by standard

end
theory Analysis
  imports

  Convex
  Determinants

  FSigma
  Sum_Topology
  Abstract_Topological_Spaces
  Abstract_Metric_Spaces
  Urysohn
  Connected
  Abstract_Limits
  Isolated
```

Elementary_Normed_Spaces
Norm_Arith

Convex_Euclidean_Space
Operator_Norm

Line_Segment
Derivative
Cartesian_Euclidean_Space
Weierstrass_Theorems

Ball_Volume
Integral_Test
Improper_Integral
Equivalence_Measurable_On_Borel
Lebesgue_Integral_Substitution
Embed_Measure
Complete_Measure
Radon_Nikodym
Fashoda_Theorem
Cross3
Homeomorphism
Bounded_Continuous_Function
Abstract_Topology
Product_Topology
Lindelof_Spaces
Infinite_Products
Infinite_Sum
Infinite_Set_Sum
Polytope
Jordan_Curve
Poly_Roots
Generalised_Binomial_Theorem
Gamma_Function
Change_Of_Vars
Multivariate_Analysis
Simplex_Content
FPS_Convergence
Smooth_Paths
Abstract_Euclidean_Space
Function_Metric
begin

end

Bibliography

[1]

[2] J. B. Conway. *A course in functional analysis*, volume 96. Springer Science & Business Media, 2013.

[3] J. Dugundji. An extension of Tietze's theorem. *Pacific J. Math.*, 1(3):353–367, 1951.

[4] M. Maggesi. A formalization of metric spaces in HOL light. *J. Autom. Reasoning*, 60(2):237–254, 2018.